

Universidade do Minho
Escola de Engenharia

Nuno Ernesto Salgado Oliveira

Architectural Reconfiguration of Interacting Services

The MAP-i Doctoral Program of the Universities of Minho, Aveiro and Porto





Universidade do Minho



April 2015



Universidade do Minho
Escola de Engenharia

Nuno Ernesto Salgado Oliveira

Architectural Reconfiguration of Interacting Services

The MAP-i Doctoral Program of the Universities of Minho, Aveiro and Porto



Universidade do Minho



supervisors:
Professor Luís Soares Barbosa

April 2015

Nome: Nuno Ernesto Salgado Oliveira

Endereço electrónico: nuno.s.oliveira@insectec.pt

Título tese: Architectural Reconfiguration of Interacting Services

Orientador: Luís Soares Barbosa

Ano de conclusão: 2015

Designação do Doutoramento: Doutoramento em Informática

É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA TESE APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

Universidade do Minho, 2015

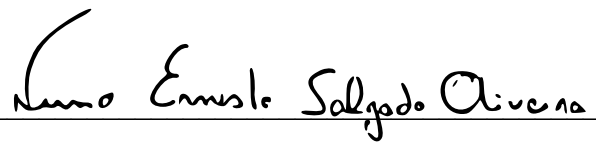
Assinatura: Nuno Ernesto Salgado Oliveira

STATEMENT OF INTEGRITY

I hereby declare having conducted my thesis with integrity. I confirm that I have not used plagiarism or any form of falsification of results in the process of the thesis elaboration.
I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

University of Minho, 2015

Full name: Nuno Ernesto Salgado Oliveira

Signature: 

Acknowledgements

This is the last page of my Ph.D. dissertation, but for its value and importance, I made it appear before everything else. The following text is just a piece of me to you in appreciation for what you have given me during this adventure. I am in debt to all of you!

I thank Professor Luís Soares Barbosa for his great motivational, comforting, words and his tireless effort in leading me always into the right direction. His constant availability for receiving me, even when his schedule was tight, was priceless to me. Most of the times, I must confess, I was just looking for his hypnotic conversation in order to regain the necessary motivation to go through this process. Less couldn't I expect from such a wise human being, who knows his trade better than anyone else. More than a supervisor, a friend.

I thank Professor Pedro Rangel Henriques for being one of the most admirable men I know. I appreciate his cheerful conversations, jokes and important advices even through his dark and heavy moments in life. All the things he *pushed* me to do, even though not related with my Ph.D., have made me more aware of my work and served as a refuge to be in contact with other research interests, some of them, clearly present in this dissertation. I am happy I was able to absorb, from Pedro, values and behaviours that have reconfigured me and, for sure, I will carry throughout my life. He is the truth supervisor for life.

I thank Professor Farhad Arbab for receiving me so well at CWI. I felt like I was at home. During my stays in Amsterdam we had fruitful conversations and discussions that have made me understand better some of the things I was doing. Of course, I would like also to thank his team at CWI, in particular, Sung, Kasper and Enric, for being great colleagues and bringing up nice discussions. Warm regards, guys. A special thank goes to Minnie, for arranging everything I need so I could be comfortably installed at the guest house.

I thank Alexandra Silva for bringing into discussion a research topic that, certainly, changed the course of my thesis. I would be forever in debt to her for the support she gave me to carry this research both remotely and closely at Nijmegen.

I appreciate the conversations and the amazing moments in Brazil with Renato Neves and Alexandre Madeira. Paulo Novais, César Analide, David, Marco, Isabel, Fábio, André, thank you so much for the good times we had together. Ângelo Costa and Tiago Oliveira, how can I appreciate their friendship? Two different people with whom I have learned so many things; two different people that made me aware of a whole different set of concerns and values in life. I make them responsible for a great part of the success of this Ph.D.. Thank you very very much! José João, Nuno Carvalho, Maria João, Ricardo Martini, Daniel Cerejo and Dima, another thank

you for all the conversations and conferences we had and the work we have achieved together.

A special acknowledgement to Flávio Rodrigues, my first M.Sc. student. His co-supervision, with professor Luís Barbosa, was an amazing experience. For all his commitment to the work and for all the discussions we had professionally and personally, a huge hug and thank you.

I thank my MAP-i colleagues and friends Carlos Pereira, Nuno Luz, Ricardo Anacleto and João Freitas, for their presence since the first moment of our doctoral program, whose end is fast approaching. For sure, friendship will remain after such an adventure.

I thank Daniela da Cruz for her support when I needed to hear those motivational words. I cannot forget the opportunity she offered me to join her team at Checkmarx, and for being so flexible and comprehensive. Thank you very much, my dear friend!

Of course, I cannot forget Daniela Fonte and Ismael Vilas Boas for their friendship and support whenever I need. For being so attentive people, with always a nice word to say and cheer one up, I really thank you.

Agradeço aos meus pais, Arlindo e Teresa, por tudo, e tudo será pouco! Agradeço em especial a paciência para ouvirem os meus desabafos sobre o trabalho. Agradeço ainda as palavras de motivação e apoio quando realmente as precisei de ouvir, ou as palavras de desespero por eu nunca mais terminar a empreitada. Esta tese não é de ouro, nem poderia ser, mas para mim vale isso e muito mais, e vocês sentem e sabem. E por esse saber especial, que é o saber de pais, vos agradeço mais uma vez.

Agradeço à minha irmã, Patrícia, e seu marido, Ibraim, pelo apoio incondicional que sempre me deram. As conversas sobre o andamento da tese e a propulsão emocional que me forneceram de forma a atingir os meus objetivos, como se fossem os deles, são impagáveis. Por isso, e pela referência que são para mim, muito obrigado!

Finalmente agradeço à Ângela, a minha namorada, que personifica toda esta dissertação. O meu percurso com ela iniciou com o doutoramento; e como no doutoramento, houve altos e baixos, certezas e incertezas, vencidas da melhor maneira. Agora termino o doutoramento, mas felizmente continuo o percurso com ela. Por toda a paciência em esperar, apoiar, motivar, eu lhe agradeço. Pela presença constante mesmo na minha ausência, eu lhe agradeço. Por fazer parte de mim, eu lhe agradeço!

O trabalho apresentado nesta dissertação foi suportado pela Fundação para a Ciência e Tecnologia (FCT) através de uma Bolsa de Doutoramento com referência SFRH/BD/71475/2010.

Architectural Reconfiguration of Interacting Services
(Abstract)

The exponential growth of information technology users and the rising of their expectations imposed a paradigmatic change in the way software systems are developed. From monolithic to modular, from centralised to distributed, from static to dynamic. Software systems are nowadays regarded as coordinated compositions of several computational blocks, distributed by different execution nodes, within flexible and dynamic architectures. They are not flawless, though. Moreover, execution nodes may fail, new requirements may become necessary, or the deployment environment may evolve in such a way that measures of quality of service of the system become degraded. Reconfiguring, repairing, adapting, preferably in a dynamic way, became, thus, relevant issues for the software architect.

But, developing such systems right is still a challenge. In particular, current (formal) methods for characterising and analysing contextual changes and reconfiguration strategies fall behind the engineering needs.

This thesis formalises a framework, referred to as ARIS*, for modelling and analysing architectural reconfigurations. The focus is set on the coordination layer, understood in the context of the Reo model, as it plays the key role in defining the behaviour of compositional systems. Therefore, it proposes a notion of a Coordination Pattern, as a graph-based model of the coordination layer; and of Reconfiguration Patterns, as parametric operations inducing topological changes in coordination patterns.*

Properties of reconfigurations can be stated and evaluated from two different perspectives: behavioural and structural. The former compares the behavioural semantics of the reconfigured system based on whatever semantic model one associates to coordination patterns. The latter focuses on the graph topology of the coordination pattern. Properties are expressed in a propositional hybrid logic, referring to the actual connectivity expressed in that graph.

To bring quality of service into the picture, the thesis also contributes with a new semantic model for stochastic Reo, based on interactive Markov chains. This opens new possibilities for analysis of both coordination patterns and reconfigurations. In particular for inspecting the effects of reconfigurations in the system's quality of service, or determining reconfiguration triggers, based on the variations of the latter.

Another contribution of the thesis is the integration of ARIS *in a monitoring strategy that enables self-adaptation and attempts to deliver it as a service in a cloud environment.*

Tools are delivered to support ARIS*. In particular, language-based technology to encode, transform and analyse coordination and reconfiguration patterns, materialises it in a dedicated editor.*

All the above mentioned contributions are assessed through a case study where a static system is worked out to support self-adaptation.

Reconfiguração Arquitetural da Interação de Serviços
(Resumo)

O crescimento exponencial de utilizadores de tecnologias de informação e o aumento das suas expetativas, impuseram uma mudamça paradigmática na maneira como os sistemas de software são desenvolvidos. De monolíticos para modulares, de centralizados para distribuídos, de estáticos para dinâmicos. Os sistemas de software são, hoje, tidos como a composição coordenada de vários blocos de computação, distribuídos por diferentes nodos de execução, com arquiteturas flexíveis e dinâmicas. Não são infalíveis, porém. Nodos de execução podem falhar, novos requisitos podem tornar-se necessários, ou o ambiente de produção pode evoluir de tal modo que medidas de qualidade de serviço se podem degradar. Reconfigurar, reparar, adaptar, de preferência dinamicamente, tornaram-se assim conceitos importantes para o arquiteto de software.

Mas desenvolver estes sistemas corretamente é ainda um desafio. Em particular, os atuais métodos (formais) para a caraterização e análise contextual, e para a estratégias de reconfiguração estão aquém das necessidades da engenharia.

Esta tese formaliza uma framework*, denominada* ARIS*, para a modelação e análise de reconfigurações arquiteturais. A atenção é focada na camada de coordenação, vista sob o prisma do modelo de coordenação Reo, dado que esta desempenha um papel chave na definição do comportamento de sistemas compostos. Assim, é proposta uma noção de Padrão de Coordenação, como sendo um modelo da camada de coordenação baseado em grafos; e outra de Padrão de Reconfiguração, como sendo operações parametrizáveis que produzem alterações topológicas nos padrões de coordenação.*

Propriedades das reconfigurações podem ser expressas e avaliadas sob duas perspetivas diferentes: comportamental e estrutural. A primeira compara a semântica de comportamento do sistema reconfigurado com base num modelo semântico qualquer escolhido para associar aos padrões de coordenação. O último foca-se na estrutura topológica do padrão de coordenação. Propriedades são expressas numa lógica híbrida proposicional, referindo-se à conetividade capturada em tal grafo de estrutura.

Fazendo a qualidade de serviço entrar em cena, a tese contribui também com um novo modelo semântico para Reo estocástico, baseado em cadeias de Markov interativas. Este modelo proporciona novas possibilidades para análise de padrões de coordenação e reconfiguração. Em particular, para investigar os efeitos das reconfigurações na qualidade de serviço do sistema, ou para determinar pontos de reconfiguração com base nas variações da mesma.

Outro contributo da tese é a integração de ARIS *numa estratégia de monitorização que habilita a auto-adaptação, e define um marco na tentativa de a entregar como um serviço em ambientes* Cloud*.*

Ferramentas são disponibilizadas para suporte a ARIS*. Em particular, tecnologia baseada em linguagens para descrever, transformar e analisar padrões de coordenação e reconfiguração, materializa a* framework *num editor dedicado.*

Todas as contribuições mencionadas acima são avaliadas através de um caso de estudo, onde um sistema estático é trabalhado para suportar a sua auto-adaptação.

To the three women who architected my life:
Mãe Teresa, who designed my structure;
Avó Maria, who established my behaviour;
Ângela, who reconfigured my qualities.

Para as três mulheres que arquitetaram a minha vida:
Mãe Teresa, que desenhou a minha estrutura;
Avó Maria, que estabeleceu o meu comportamento;
Ângela, que reconfigurou as minhas qualidades.

Contents

Acronyms **xvii**

Figures **xxii**

Tables **xxiii**

Listings **xxv**

1 Introduction **1**
1.1 Context . . . 2
1.2 Motivation . . . 5
1.2.1 A scenario . . . 5
1.2.2 Modelling and analysis . . . 6
1.2.3 Reconfigurations . . . 7
1.3 Aims and contributions . . . 7
1.3.1 Thesis . . . 7
1.3.2 Goals . . . 8
1.3.3 Contributions . . . 9
1.4 Document organisation . . . 11

2 Background **13**
2.1 The Reo coordination model . . . 13
2.1.1 Channels, nodes and connectors . . . 13
2.1.2 Semantic models . . . 15
2.1.3 Stochastic Reo . . . 19
2.2 Interactive Markov chains . . . 19
2.2.1 Process algebra and *labelled transition systems* . . . 20
2.2.2 Markov process and *continuous-time Markov chains* . . . 20
2.2.3 *Interactive Markov chains* . . . 23
2.3 Hybrid Logic . . . 25
2.3.1 Modal Logic . . . 26
2.3.2 Hybrid extension to modal logic . . . 27

I A stochastic model for software coordination **29**

3 State of the Art: Models for Performance Evaluation **31**

3.1 Algebraic stochastic models . . . 31
3.1.1 Stochastic process algebras . . . 31
3.1.2 Stochastic Petri nets . . . 32
3.1.3 Stochastic automata networks . . . 33
3.1.4 Markovian-based models . . . 34
3.2 Queueing networks . . . 34
3.3 Component-based performance evaluation . . . 36
3.4 Coordination-oriented stochastic approaches . . . 38
3.4.1 Generic coordination . . . 38
3.4.2 Workflow, choreography and orchestration . . . 40

4 Interactive Markov chains for stochastic Reo 43
4.1 $\mathsf{IMC}_{\mathsf{Reo}}$ . . . 43
4.2 $\mathsf{IMC}_{\mathsf{Reo}}$ composition . . . 46
4.2.1 Parallel composition . . . 46
4.2.2 Synchronisation . . . 48
4.2.3 Properties . . . 50
4.2.4 Cleaning up unintended transitions . . . 53
4.2.5 Composition idiosyncrasies . . . 55
4.3 Distilled $\mathsf{IMC}_{\mathsf{Reo}}$ . . . 58
4.3.1 The writer, the reader, the channel and the node . . . 60
4.3.2 $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$: the distilled $\mathsf{IMC}_{\mathsf{Reo}}$ . . . 62
4.3.3 Composition in $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ . . . 66
4.4 Summary . . . 70

II Reconfiguration of interacting services 73

5 State of the Art: Software Reconfiguration 75
5.1 Architectural reconfigurations . . . 75
5.1.1 Algebraic approaches . . . 76
5.1.2 Pattern-based approaches . . . 79
5.1.3 Coordination-targeting approaches . . . 81
5.2 Languages for reconfiguration . . . 85
5.2.1 Languages for architecture description . . . 85
5.2.2 Languages for reflective adaptations . . . 88
5.3 Self-adaptation . . . 90
5.3.1 Approaches . . . 90
5.3.2 Adaptation specification . . . 94

6 Modelling Reconfigurations 97
6.1 Coordination patterns . . . 97
6.2 Reconfigurations . . . 101
6.2.1 Primitive reconfiguration operations . . . 101
6.2.2 Composing reconfigurations . . . 104
6.2.3 Reconfiguration patterns . . . 108
6.3 Supporting dynamic reconfigurations via consistent state transfer . . . 114

6.3.1 Symbolic states . . . 114
6.3.2 State transfer . . . 116
6.4 Reconfigurations on the stochastic setting . . . 118
6.4.1 Stochastic coordination patterns . . . 118
6.4.2 Reconfigurations revisited . . . 121
6.5 Summary . . . 123

7 Reasoning about reconfigurations 125
7.1 Behavioural reasoning . . . 125
7.1.1 Comparing reconfigurations . . . 126
7.1.2 A behavioural classification of reconfigurations . . . 129
7.2 Structural reasoning . . . 131
7.2.1 A hybrid logic . . . 131
7.2.2 Bisimulation for $Hp\mathcal{E}$ . . . 134
7.2.3 Expressing 'long scope' properties . . . 138
7.2.4 Comparing reconfigurations . . . 139
7.2.5 A structural classification of reconfigurations . . . 141
7.3 The stochastic case . . . 142
7.3.1 Quantitative reasoning . . . 143
7.3.2 A quantitative classification of reconfigurations . . . 147
7.4 Summary . . . 148

8 Self-adaptation of Architectures 151
8.1 A self-adaptation approach . . . 151
8.1.1 The offline phase: planning reconfigurations . . . 152
8.1.2 The online phase: the feedback loop . . . 154
8.2 Triggering of reconfigurations . . . 157
8.3 Adaptation as a Service . . . 159
8.3.1 Architecture and main workflow . . . 160
8.3.2 Discussion . . . 164
8.4 Summary . . . 164

III Tool support and case study 167

9 Tool Support 169
9.1 CooPLa . . . 169
9.1.1 Channels . . . 170
9.1.2 Patterns . . . 172
9.1.3 Stochastic extension . . . 174
9.2 ReCooPLa . . . 176
9.2.1 Reconfigurations . . . 177
9.2.2 Application of reconfigurations . . . 179
9.3 The CooPLa Editor . . . 180
9.3.1 Editor Overview . . . 181
9.3.2 IMCREOtool . . . 184
9.3.3 Reconfiguration engine . . . 187

9.3.4 Importer and exporter 188
9.4 Summary 191

10 Case Study: towards an adaptable system **193**
10.1 The ASK system 193
10.1.1 The architecture 194
10.1.2 The static performance on a dynamic environment 195
10.2 Planning adaptations 196
10.2.1 Adaptable-ASK design 196
10.2.2 Analysis of RTS configurations 199
10.2.3 Objectives, constraints and filters 201
10.3 Runtime situation 203
10.4 Summary 205

11 Conclusion **207**
11.1 Retrospective 207
11.2 Related work discussion 211
11.2.1 Models for performance evaluation 212
11.2.2 Software reconfiguration 213
11.3 Future work and research directions 216

A Deriving $\mathsf{IMC}_{\mathsf{Reo}}$ from Stochatic Coordination Patterns **221**

B CooPLa Grammar **223**

C ReCooPLa Grammar **225**
C.1 The Grammar 225
C.2 Reconfiguration Patterns in ReCooPLa 226

References **228**

Acronyms

A

AaaS adaptation as a service.

ACID atomicity, consistency, isolation and durability.

ADL architecture description language.

ASK Access Society's Knowledge.

B

BPMN business process modelling notation.

C

CA constraint automaton.

CCA continuous-time constraint automaton.

CLI command line interface.

CSP communicating sequential processes.

CTL computation tree logic.

CTMC continuous-time Markov chain.

D

DSL domain-specific language.

DTMC discrete-time Markov chain.

E

eBNF extended Backus-Naur form.

ECT extensible coordination tools.

EMPA extended Markovian process algebra.

F

FOL first order logic.

G

GPS global position system.

GUI graphical user interface.

H

HL hybrid logic.

I

IDE integrated development environment.

IMC interactive Markov chain.

IO input/output.

IT information technology.

J

JVM Java virtual machine.

L

LTL linear temporal logic.

LTS labelled transition system.

M

ML modal logic.

O

OCL object constraint language.

P

PA port automaton.

PEPA performance evaluation process algebra.

Q

QIA quantitative intensional automaton.

QoS quality of service.

R

RA Reo automaton.

REST representational state transfer.

RTS reconfiguration transition system.

S

GSPN generalised stochastic Petri net.

SaaS software as a service.

SAN stochastic automata network.

SLA service level agreement.

SOAP simple object access protocol.

SOA service-oriented architecture.

SOC service-oriented computing.

SPA stochastic process algebra.

SPN stochastic Petri net.

SRA stochastic Reo automaton.

U

UDDI universal description discovery and integration.

UML unified modelling language.

W

WSDL web service description language.

WS web service.

X

XML extensible markup language.

Figures

2.1 Primitive Reo channels and Reo connectors 14
2.2 Constraint automata for the primitive Reo channels and connectors. . 16
2.3 Reo automata for primitive Reo channels and connectors. 18
2.4 Primitive stochastic Reo channels. 19

4.1 IMC for the basic stochastic Reo channels. 45
4.2 Fragment of the parallel composition of a lossy and a fifo_e channel. . . 47
4.3 Composing two $\mathsf{2fifo}_\mathsf{e}$ connectors. 48
4.4 Fragment of the parallel composition of two $\mathsf{2fifo}_\mathsf{e}$ 48
4.5 Parallel composition after synchronisation and cleaning 54
4.6 Fragments of two equivalent synchronised $\mathsf{IMC}_\mathsf{Reo}$ models 55
4.7 Composing a lossy and a sync channel. 56
4.8 Composing a lossy and a fifo_e channel. 57
4.9 Composition of two fifo_e channels (fragment). 57
4.10 three-port basic connectors and corresponding $\mathsf{IMC}_\mathsf{Reo}$ models. 59
4.11 Connector refactoring for composition via two channel end nodes. . . 59
4.12 The essential components of stochastic Reo 61
4.13 The two-pase component-based stochastic Reo model of a lossyfifo. . . 62
4.14 The $\mathcal{D}\mathsf{IMC}_\mathsf{Reo}$ for the basic stochastic Reo channels 63
4.15 The $\mathsf{IMC}_\mathsf{Reo}$ for the reader and writer components 63
4.16 Different Reo node configurations. 64
4.17 $\mathcal{D}\mathsf{IMC}_\mathsf{Reo}$ models for merger–replicator and merger–router nodes. 65
4.18 The $\mathsf{IMC}_\mathsf{Reo}$ models for the node configurations in Figure 4.16. 66
4.19 Design-phase $\mathcal{D}\mathsf{IMC}_\mathsf{Reo}$ model for the lossysync connector. 68
4.20 Deployment-phase $\mathcal{D}\mathsf{IMC}_\mathsf{Reo}$ model for lossysync connector. 69

6.1 The Sequencer coordination pattern. 100
6.2 Reconfigurations of the Sequencer coordination pattern 111
6.3 Step-by-step example of moveP reconfiguration. 112
6.4 Example of dynamic reconfigurations and state transfer. 117

7.1 The ProActiveDependentSequencer pattern and its semantics. 127
7.2 Semantics of Sequencer pattern variants (1). 128
7.3 Semantics of Sequencer pattern variants (2). 129
7.4 Behavioural taxonomy for reconfigurations. 130
7.5 Example of a *displaced* invariant. 140
7.6 Structural taxonomy for reconfigurations. 142

7.7 Quantitative taxonomy for reconfigurations. 147
7.8 An ontology base for reconfigurations 150

8.1 Feedback loop based on a reconfiguration transition system. 154
8.2 Sequence diagram for the *Monitor* component. 155
8.3 Sequence diagram for the *Planner* component. 156
8.4 Sequence diagram for the *Executor* component. 157
8.5 Adaptation as a Service architecture overview. 161

9.1 CooPLa description of some Reo channels. 171
9.2 CooPLa description of the router channel. 172
9.3 CooPLa specification of the Sequencer. 174
9.4 Channels added of stochastic labels 175
9.5 Stochastic instance `sseq` of the Sequencer coordination pattern. . . . 176
9.6 ReCooPLa implementation of the OverlapP reconfiguration. 179
9.7 Reconfiguration script for updating the Sequencer. 180
9.8 A toolchain for the CooPLa Editor. 181
9.10 IMCREOTool triggering button. 184
9.11 IMCREOTool wizard pages. 185
9.12 Working labels 187
9.13 Reconfiguration engine triggering button. 187
9.14 The application of a reconfiguration. 188
9.15 The importing and exporting features. 188
9.16 Exported coordination pattern to Vereofy notation. 189
9.17 Using the ECT and the CooPLa Editor together. 190

10.1 The architecture of ASK. 194
10.2 The *Executer* component model with stochastic information. 195
10.3 Configurations for the Adaptable-ASK system. 198
10.4 RTS for the Adaptable-ASK system. 199
10.5 The PRISM model for the *scaled out* coordination pattern. 200
10.6 Throughput ratio values for the Adaptable-ASK configurations. . . . 201
10.7 Performance of Adaptable-ASK 204

Tables

7.1 Values for Latency, Throughput and Blocking quality dimensions . . 146

9.1 Primitive reconfigurations and their counterpart formal names 178
9.2 Working label values and meaning. 186

10.1 Requests to the ASK system during a day 195
10.2 Adaptable-ASK properties. 197
10.3 Steady-state throughput ratio analysis for RTS configurations. 200
10.4 Predicted configurations for filters. 202

Listings

A.1 Converting stochastic coordination pattern into $\mathsf{IMC}_{\mathsf{Reo}}$. 221
A.2 Converting stochastic coordination pattern into $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$. 222
B.1 CooPLa main grammar 223
B.2 CooPLa channels grammar 223
B.3 CooPLa coordination patterns grammar 223
B.4 CooPLa stochastic instances grammar 224
C.1 ReCooPLa main grammar 225
C.2 ReCooPLa reconfiguration grammar 225
C.3 ReCooPLa application grammar 226

Chapter 1

Introduction

That is what I find so wonderful [Pause] The way man adapts himself [Pause] To changing conditions.

– *Samuel Beckett, Happy Days*

Along the last decade, the world has witnessed a revolution in the *information technology (IT)* area. Software systems, once monolithic and centralised, became distributed, modular, performant and multi-platform. For this revolution contributed the evolution and emergence of web and infrastructure technologies, programming languages and (software) architecture paradigms.

Software systems are, now, required to perform seamlessly in a variety of platforms (*e.g.,* web, mobile, desktop, *etc.*) and environments (*e.g.,* hardware architecture, operating system, *etc.*) under a certain context (*e.g.,* user requests, type of network, memory availability, processing power, *etc.*), each of which setting forward a number of challenges on development, deployment and maintenance of such systems. The performance of a system is characterised by several dimensions also referred to as the systems' non-functional or *quality of service (QoS)* requirements. These include scalability, availability, security, dependability, reliability and adaptability, to mention but a few. Although some of these are qualitative dimensions, the majority delivers a quantitative perspective to the system, enabling their measuring and analysis.

The correct development of these systems postulate, thus, the existence of (formal) models and techniques that allow for their modelling and analysis. In particular, the analysis shall traverse the quantitative aspects of these systems, so that QoS requirements are also inspected. Actually, in ever-evolving contexts, these requirements soon start degrading, possibly leading the system into failure. To avoid this,

systems shall be able to adapt to changes, by reconfiguring themselves. Assuming that systems' architectures are flexible enough, and practice already came up with solutions for dynamically applying the required changes, how can one predict that after such reconfigurations, the system will response as expected both quantitatively and qualitatively? How can reconfigurations be formally modelled and analysed? Are formal methods prepared to deal with these hot topics?

These questions establish the starting point for this Ph.D. thesis. In the context of new trends of software development, in the era of cloud computing, new challenges arrive to the formal instance of software engineering that are still remaining to be tackled.

In this chapter. The context and basic concepts on software composition and coordination is presented. Then, a motivation introduction along with open issues in the framed context open way to the definition of the thesis statement, its goals and concrete contributions.

1.1 Context

The development of software has evolved hand in hand with the exponential growth of information technology users and the rising of their expectations. Reutilisation, performance and ubiquitousness became, thus, central concerns in new software development trends. The advent of cloud computing confirmed the development paradigm change and made possible these three pillars to stand out.

Cloud computing [72], is the ultimate technological advance to support the needs of large scale, highly demanding, computing. Clouds are unanimously defined as large distributed systems consisting of pools of virtualised computers, that dynamically provide computational resources depending on the real-time needs of hosted software systems [73, 248, 239]. A cloud hides from its users all the infrastructural complexity, as much as electricity grids do; being, therefore, exploited as a pay-per-use model. The dynamic provision of resources (the elasticity of the cloud) offer (the feeling of) infinite computation resources. This, together with powerful load-balancing mechanisms, contributes to the desired minimisation of QoS degradation. The principle of cloud computing is to offer computing, storage and software, in a stack of services: infrastructure, platform and software. The first, provides computation, storage and communication resources through virtual machines managed by the users as much as in a normal server or grid. Platforms are delivered as services where the practitioners can develop and deploy their software solutions without the need for managing infrastructural issues. Software applications constitute the top

layer of the cloud stack; this enables the access to services offered by software applications, that otherwise would have to be installed and managed by users on their local machines. The term *software as a service (SaaS)* was coined to refer to this cloud layer, where software runs taking advantage of potentially infinite computational resources. The easy access to these services and their confirmed performance led to the adoption of emerging architectural styles such as *service-oriented architecture (SOA)*.

SOA [116, 158, 115, 154], as opposed to other software architectural styles [130, 237, 244, 243], is completely in line with both the cloud and business environments. Enterprise software solutions greatly benefit from using such architectural paradigm as business concerns are modularly separated and organised into services. Services are reusable loosely-coupled *loci* of computation offering a designated behaviour via their public interfaces. They may be owned by different organisations, run on different physical locations and written in different programming languages. New SOA systems are constructed by composing services' interfaces, in order to achieve some business objective. Concretely, SOA empowered the integration and interoperability of systems, including legacy ones, by resorting to standard web technologies like *web services (WSs)*, which constitute the *de facto* realisation of services. A protocol for the standardised development of WSs and, consequently, SOA systems assumes that service providers describe their services' interface and behaviour in *web service description language (WSDL)*, and make such descriptions public in a *universal description discovery and integration (UDDI)* repository; in turn, a service consumer accesses those services via messaging protocols like *simple object access protocol (SOAP)* or *representational state transfer (REST)*, in order to integrate them into a new system, in accordance to the system (or company) goals.

Consequently, the development of SOA systems, referred to as *service-oriented computing (SOC)*, sets its emphasis, not only on the implementation of services as reusable building blocks, but also (and primarily) on the designing of the global system architecture. Such an architectural design includes taking decisions ranging *from*: which services are to be selected and reused as part of the new system; *to*: how they are composed together to fulfil some business goal, while maximising the coverage of agreed levels of QoS.

A multitude of languages, traditionally referred to as *architecture description languages (ADLs)*, have been proposed to aid on the task of designing software architectures. Some representatives include ACME [126, 128], Wright [9, 7, 8], Darwin [181], Rapide [178, 177], ArchJava [5, 4], $\mathcal{P}i\mathcal{L}ar$ [102, 103] or Archery [234, 233]. An ADL provides syntactic constructs for the characterisation of the software sys-

tem's structure and behaviour [126], taking into account three main concepts (ADL elements): *component*, *connector* and *configuration*. Components abstract the notion of a computation unit (*e.g.,* procedure, service, *etc.*); connectors abstract the notion of interaction policies among components; and configurations define the topology of the architecture, *i.e.,* how components are connected to other components via connectors. The concrete description of components and connectors in ADLs typically include the definition of an interface, for communication with components or connectors; a semantics, usually based on algebraic semantic models, for general analysis or requirement consistency check; and constraints, for defining invariant properties of the element. The description of the configuration element varies from ADL to ADL; it may include only the definition of the topology, but some languages go forward, providing features to describe non-functional properties of the whole architecture or how it evolves to cope with scalability and performance threats.

However, classical ADLs usually do not capture all notions that are essential in SOC, such as the coordination of services (*i.e.,* the interaction constraints between services), which in particular subsumes workflow, control flow, synchronisation, mutual exclusion, non-determinism or context-dependency. Coordination provides the necessary *glue code* to compose the building blocks into a complete system. Notably, it plays an important role in the description of a system's architecture and global behaviour, in order to meet the system function and non-functional requirements.

In the context of SOA, coordination of services is commonly referred to as orchestration or choreography [71]. The former defines the system behaviour based on a central process (*i.e.,* the coordinator) which coordinates the actions of the participating services. Such services just provide their functionalities and are not aware of both other participants and its participation on a coordinated process. The latter is otherwise: the system behaviour is defined from the set of all participant services, which are aware of each other and know when to provide functionality and synchronise with their peers. Coupling and decoupling of coordinated entities (*i.e.,* the degree to which they are dependent on each other) or their reutilisation is usually imposed by the kind of coordination practiced. It can be classified as endogenous [214] or exogenous [11, 12]. The former incorporates/hardcodes the coordination strategies (by means of primitives) within the computation of the coordinated entities. The latter defines the coordination strategies outside of the coordinated entities, in a completely separated layer of code.

Exogenous models of coordination that cater for SOC within cloud environments, provide the essential focal point for this thesis's work, as discussed in the sequel.

1.2 Motivation

In this section, the motivation for this thesis is exploited. In particular, a scenario and an overview of the main issues in the research area underlying this thesis are discussed.

1.2.1 A scenario

Picture a system for elderly surveillance. Its main feature is to capture vital information from the elderly, and send it back to a central (established in a cloud environment), from where healthcare professionals *read* it and act accordingly. It also serves as agenda (*e.g.,* for medical appointments), reminder (*e.g.,* for medicine taking), telephone (*e.g.,* for direct contact with healthcare assistant). When outside, it still works as a *global position system (GPS)* with the ability of both recording important path points (*e.g.,* where the car was parked) and taking random pictures for later remembrance (*e.g.,* for aiding people with Alzheimer's disease).

This is a typical intelligent system in the ambient intelligence research area [22, 97, 23, 99], which during the last decade has gained relevance within the research community.

It goes without saying that this system works on a mobile platform, being dependent on internet signal (wifi, 3G/4G). Moreover, it takes into consideration *wearable* sensors for acquiring vital information, which ship it to the mobile device via bluetooth, and from the device into the central, with a flavour of real-timeliness. Services (agenda, reminders, GPS, photos, among others) are coordinated and composed together to fulfil system's functional requirements. The computational resources and battery power of these systems are limited, while bluetooth and 3G/4G technologies consume considerable amounts of energy. In this sense, the system should act (and react) in accordance to both maximise the battery life when working under such contexts, and minimise resources usage to avoid bottlenecks, failures and similar problems. In general, to cope with non-functional (QoS) requirements.

In order to always meet the requirements in contexts where change is the norm rather than the exception, it is necessary that these systems are able to evaluate themselves on their overall behaviour and performance. This is usually achieved by considering faithful models of the system that allow for checking the desired properties and reasoning about resource allocation and overall performance.

Whenever requirements are not met (which may happen for a plethora of reasons), these systems should be able to evolve at runtime, entailing the need for dynamic adaptation [133]. Adaptation implicitly demands a change in the internals

of the system (given that the context and the environment is not controlled).

Modifications at the architectural level, in particular in the coordination layer, and how they can be correctly designed and applied, play the essential motivation for the work in this thesis. Let's, then, detail briefly the major issues involved.

1.2.2 Modelling and analysis

The practical reach of formal methods is leaning towards software composition. It is an endeavour to providing compositional models, where properties of the composed system can be derived from the properties of its building blocks and the coordination *glueing* them. Several formalisms and models have been proposed to achieve the requirements of software coordination. Valuable representatives include Linda [83], CoLaS [101], LGI [192], PICCOLA [2], BPEL (*a.k.a* WS-BPEL or BPEL4WS) [104], Join Calculus [124], Reo [14], CommUnity [119], ROAD [96], ARC [226], PBRD [242], Orc [193] and BIP [35, 34]. Although sharing philosophy, they are different in nature; and not all of them meet the coordination and compositionality requirements. From the presented set, Reo gains relevance because it is one of the few that comprise notions of synchronisation, mutual exclusion, non-determinism or context-dependency. Moreover, it is an exogenous channel-based approach, which allows for complex coordination structures to be obtained by the composition of simpler ones. Also, it approximates high-level models like BPEL, that allow for encoding long-studied business work- and control-flow patterns [1, 231, 257, 116] into coordination policies [26].

The underlying semantics of (some of) these formalisms form the basis of behaviourally reasoning about the coordination layer, which subsumes the overall analysis of system functional requirements. But behavioural analysis is just the *tip of the iceberg*. In addition, as motivated early on, the fulfilment of non-functional requirements is also of ultimate importance for this kind of systems. Maintaining QoS requirements above certain levels and deciding resource allocation, demands techniques for analysing systems in a quantitative setting. At this point, formal methods fall short in providing coordination models with both a notion of their own QoS and a suitable compositional semantic model for their quantitative analysis. There are, nonetheless, efforts to shorten this gap; even if failing in their conjugation. On the one hand, for instance, stochastic Reo [16] embodies the necessary notion of QoS in the Reo coordination model, but no quantitative semantic model with practical evidence is associated to it. On the other hand, a plethora of stochastic semantic models exist [172, 24, 28, 143, 110, 86, 141] but most of them are not compositional, do not provide necessary tool support for practitioners, or are not able to capture the complex semantics of existing coordination models.

1.2.3 Reconfigurations

Traditionally, an architectural reconfiguration mainly targets the manipulation (*e.g.,* substitution, update or removal) of components, often disregarding connectors, or, otherwise, abstracting them as yet another component [225, 211, 148, 182, 236]. SOAs provide the necessary flexibility to easily achieve such changes. In fact, service discovery, and their binding and unbinding to the architecture is ideally deferred to runtime, in this paradigm [120]. This brings multiple advantages, one of them being the possibility of discovering and binding services that will ensure better global system performance for the current context. But this may not be enough. Sometimes, it is the interaction policy between these services (*i.e.,* the structure of the coordination code) that must be reconfigured, as discussed, for instance, in [170]. Reconfigurations may, thus, substitute, add or remove parts of the coordination layer, in order to restructure the encoded policies, in accordance to changes.

This suggests that coordination models should provide mechanisms for their construction, deconstruction and mobility. Fortunately, this is already contemplated in some of the formalisms considered above; and there are already approaches that formalise reconfigurations on top of these formalisms, ranging over varied techniques [91, 170, 55].

What is still lacking, however, is a formal notion of reconfigurations with appropriate support for their analysis and comparison as a way of (*i*) perceiving the consequences of applying these reconfigurations; (*ii*) guaranteeing that after a reconfiguration, the requirements hold; and (*iii*) the system preserves its integrity and consistency on transferring execution states. Moreover, a connection between reconfigurations and QoS is missing. This would allow for determining how and when a reconfiguration takes place.

1.3 Aims and contributions

1.3.1 Thesis

> *The correct reconfiguration of (the coordination between) interacting services, in a quantitative setting, is possible once suitable formal mechanisms (*e.g., *calculus, languages, semantic models) for its modelling and analysis are provided.*

1.3.2 Goals

> *Set up a conceptual and formal framework for modelling and analysing (behaviourally, structurally and quantitatively) reconfigurations.*

This constitutes the overall goal of this thesis. Concretely, the study of these reconfigurations is narrowed down to software connectors in the context of exogenous coordination. The break down of this main goal is summarised as follows:

Analysing coordination structures. Current practice in formal methods counts on semantic models for coordination formalisms that restrict analysis to a behavioural perspective. It is objective of this thesis to push this subject forward by offering models that enable the analysis of both structural and performance properties of coordination structures.

Modelling reconfigurations. This topic plays the central role in this thesis. In concrete, it is one's objective to formalise a notion of reconfigurations, where the semantics of applying them is defined upon the mathematical structure of coordination elements.

Analysing reconfigurations. With the existing semantic models, and those developed in the context of these thesis, it becomes interesting to observe how different reconfigurations compare when applied to coordination structures. To know the benefits and drawbacks in behavioural, structural and performative aspects of coordination structures after applying reconfigurations is another goal of this thesis.

Triggering reconfigurations. After studying the coordination structures of a system, and deciding how to reconfigure them, it is necessary to decide when and why these reconfigurations shall be applied. Triggering reconfigurations is a well-studied topic, but with little effort put on its formalisation. Formalising QoS-aware triggering of reconfigurations constitutes an endeavour of bridging theory and practice in what concerns self-adaptive software.

Tool support. To wrap up the research work and endow software architects with means for designing and analysing evolvable systems, it is also an objective of this thesis to devise suitable tool support. This includes an integrated editor featuring both domain-specific languages for the specification of coordination structures, reconfigurations and their actual application. Upon this, converters are envisioned

that transform such specifications and make them available as models used by well-established tools for model-checking and analysis in general.

1.3.3 Contributions

The main contributions of this thesis are the following:

- a quantitative semantic model for stochastic Reo;
- a framework, referred to as ARIS (expanding to the tile of this thesis), for modelling and analyse reconfigurations of coordination structures, including,
  - an abstract model for modelling coordination structures;
  - a notion of reconfigurations for coordination structures;
  - a methodology for reasoning about reconfigurations under three different perspectives: behaviour, structure and performance;
- a strategy for self-adaptation of software architectures on top of the ARIS framework;
- a set of support tools for ARIS.

Most of these results appeared in the following list of publications.

- Nuno Oliveira, Nuno Rodrigues, and Pedro R. Henriques. "Domain-Specific Language for Coordination Patterns". In: *Computer Science and Information Systems* 8.2 (May 2011), pp. 343–359.
- Nuno Rodrigues, Nuno Oliveira, and Luís Soares Barbosa. "The role of coordination analysis in software integration projects". In: *On the Move to Meaningful Internet Systems: OTM 2011 Workshops*. Ed. by P. Herrero R. Meersman T. Dillon. Vol. 7046. Lecture Notes in Computer Science. Berlin Heidelberg: Springer-Verlag, Oct. 2011, pp. 83–92.
- Nuno Oliveira and Luís S. Barbosa. "On the reconfiguration of software connectors". In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. Vol. 2. SAC '13. Coimbra, Portugal: ACM, Mar. 2013, pp. 1885–1892.
- Nuno Oliveira and Luís S. Barbosa. "Reconfiguration Mechanisms for Service Coordination". In: *Web Services and Formal Methods*. Ed. by Maurice H. ter Beek and Niels Lohmann. Vol. 7843. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 134–149.
- Nuno Oliveira and Luís S. Barbosa. "A self-adaptation strategy for service-based architectures". In: *Proceedings of the 8th Brazilian Symposium on Software Components, Architectures and Reuse*. SBCARS'2014. Distinguished with Best Paper Award. Maceió, Alagoas, Brasil: IEEE, Sept. 2014, pp. 1–10.
- Nuno Oliveira, Alexandra Silva, and Luís S. Barbosa. "Quantitative Analysis of Reo-based Service Coordination". In: *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. Vol. 2. SAC'14. Gyeongju, Korea: ACM, Mar. 2014, pp. 1247–1254.
- Nuno Oliveira, Alexandra Silva, and Luís S. Barbosa. "$\mathsf{IMC}_{\mathsf{Reo}}$: interactive Markov chains for stochastic Reo". In: *Journal of Internet Services and Information Security* 5.1 (Feb. 2015), pp. 3–28.
- Alejandro Sanchez, Nuno Oliveira, Luis S. Barbosa, and Pedro Henriques. "A perspective on architectural re-engineering". In: *Science of Computer Programming* 98 (Jan. 2015), pp. 764–784.
- Flávio Rodrigues, Nuno Oliveira, and Luís S. Barbosa. "ReCooPLa: a DSL for Coordination-based Reconfiguration of Software Architectures". In: *3rd Symposium on Languages, Applications and Technologies*. Ed. by Maria J. V. Pereira, José P. Leal, and Alberto Simões. Vol. 38. OpenAccess Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, June 2014, pp. 61–76.

The last publication reports on results partially obtained in collaboration with Flávio Rodrigues, whose M.Sc. thesis with bibliographic reference

- Flávio Rodrigues. "An Engine for Coordination-based Architectural Reconfigurations". M.Sc. thesis. Braga, Portugal: Departamento de Informática, Universidade do Minho, Dec. 2014.

was co-supervised by the author.

The tools developed and companion documentation is available at

`coopla.di.uminho.pt`

Moreover, a stand-alone command-line interface for the generation of analysable assets from the design of QoS-augmented coordination pieces is available at

`http://reo.project.cwi.nl/reo/wiki/ImcReo`.

For pretty printing Reo-like circuits, a LaTeX package was developed, documented and made available on the CTAN repository, at

`http://www.ctan.org/pkg/reotex`.

Along the last four years, the author made also a number of contributions in areas not directly relevant for the thesis, but witnessing his collaboration in the research dynamics of the university. The following list presents all such contributions that mainly result from co-orientations of master theses and collaboration with fellow research groups.

- Ines Čeh, Matej Črepinšek, Tomaž Kosar, Marjan Mernik, Pedro R. Henriques, Maria J. V. Pereira, Daniela da Cruz, and Nuno Oliveira. "Tool-Supported Building of DSLs from OWL Ontologies". In: *INForum'11 — III Simpósio de Informática: 5th Compilers, Programming Languages, Related Technologies and Applications (CoRTA'2011)*. Ed. by Raul Barrosa and Luís Caires. Universidade de Coimbra, Sept. 2011, pp. 210–221.
- Ivan Luković, Maria João Varanda Pereira, Nuno Oliveira, Daniela da Cruz, and Pedro R. Henriques. "A DSL for PIM Specifications: Design and Attribute Grammar based Implementation". In: *Computer Science and Information Systems* 8.2 (May 2011), pp. 379–403.
- Nuno Oliveira, Maria J. V. Pereira, Alda L. Gancarski, and Pedro R. Henriques. "Learning Spaces for Knowledge Generation". In: *1st Symposium on Languages, Applications and Technologies, SLATE 2012*. Ed. by Alberto Simões, Ricardo Queirós, and Daniela da Cruz. Vol. 21. OASIcs. Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, June 2012, pp. 175–184.
- Maria J. V. Pereira, Mario Berón, Daniela da Cruz, Nuno Oliveira, and Pedro R. Henriques. "Problem Domain Oriented Approach for Program Comprehension". In: *1st Symposium on Languages, Applications and Technologies, SLATE 2012*. Ed. by Alberto Simões, Ricardo Queirós, and Daniela da Cruz. Vol. 21. OASIcs. Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, June 2012, pp. 91–105.
- Ismael Vilas Boas, Nuno Oliveira, Pedro Rangel Henriques, and Daniela da Cruz. "Agile development for education effectiveness improvement". In: *Proceedings of the XV international symposium on computers in education (SIIE'2013)*. Viseu, Portugal, Nov. 2013, pp. 299–304.
- Maria João Varanda Pereira, Nuno Oliveira, Daniela da Cruz, and Pedro Rangel Henriques. "Choosing Grammars to Support Language Processing Courses". In: *2nd Symposium on Languages, Applications and Technologies*. Ed. by José Paulo Leal, Ricardo Rocha, and Alberto Simões. Vol. 29. OpenAccess Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, June 2013, pp. 155–168.

- Pedro Carvalho, Nuno Oliveira, and Pedro Rangel Henriques. "Unfuzzying Fuzzy Parsing". In: *3rd Symposium on Languages, Applications and Technologies*. Ed. by Maria João Varanda Pereira, José Paulo Leal, and Alberto Simões. Vol. 38. OpenAccess Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, June 2014, pp. 101–108.
- Daniela Fonte, Ismael Vilas Boas, Nuno Oliveira, Daniela da Cruz, Alda Lopes Gançarski, and Pedro Rangel Henriques. "Partial Correctness and Continuous Integration in Computer Supported Education". In: *CSEDU 2014 — Proceedings of the 6th International Conference on Computer Supported Education*. Ed. by Susan Zvacek, Maria Teresa Restivo, James Uhomoibhi, and Markus Helfert. Vol. 2. SciTePress — Science and Technology Publications, Apr. 2014, pp. 205–212.
- Maria João Varanda Pereira, Nuno Oliveira, Daniela da Cruz, and Pedro Henriques. "An effective Way to Teach Language Processing Courses". In: *Innovative Teaching Strategies and New Learning Paradigms in Computer Programming*. Ed. by Ricardo Queirós. Hershey, PA, USA: IGI Global, Nov. 2014.

1.4 Document organisation

The thesis is organised into three parts.

Part I deals with stochastic semantic models for coordination structures. Chapter 3 is devoted to a literature review on models, tools and techniques for performance evaluation of software systems. Chapter 4 introduces a new stochastic model for capturing the semantics of stochastic Reo.

Part II develops ARIS, a framework for modelling and reasoning about architectural reconfiguration of interacting services. State of the art in architectural reconfiguration is reviewed in Chapter 5. Chapter 6 formalises notions of coordination and reconfiguration patterns in either static, dynamic and stochastic settings. Chapter 7 proposes a methodology for reasoning about, comparing and classifying reconfigurations. Chapter 8 presents a strategy for self-adaptation of software systems taking advantage of the techniques developed in the previous chapters; it provides also an approach for delivering adaptation as a service.

Part III reports on tool support for ARIS, in Chapter 9, and includes an illustrative case study, in Chapter 10.

Not included in either of the three mentioned parts, Chapter 2 presents background references to relevant areas underlying this thesis and, finally, Chapter 11 concludes the thesis with a retrospective of the work done and a proposal of research topics for further work.

Chapter 2

Background

A little knowledge is a dangerous thing. So is a lot.

– *Albert Einstein*

In this chapter. The Reo coordination model is recalled along with its most renowned semantic models, as it is preponderant in the illustration of coordination based reconfigurations in the context of this thesis. Stochastic Reo is also addressed, as it plays a central role in the definition of stochastic notions for coordination and reconfigurations. Additionally, Interactive Markov Chains are introduced as they were chosen as the target quantitative model for the analysis of reconfigurations. To finish the chapter, the basics of Hybrid Logic are briefly recalled, as such logic is used for expressing structure architectural constraints.

2.1 The Reo coordination model

Reo [12, 18, 14, 15] is a popular model for exogenous coordination of software components based on channels. It is compositional, in the sense that complex coordination structures are achieved from the combination of channels. Moreover, it has well-established semantics that determine the constrained behaviour of such complex structures.

2.1.1 Channels, nodes and connectors

A Reo channel is a point-to-point communication device with exactly two directed ends and a behaviour (*i.e.,* coordination policy) defined by a semantic model. A

channel *end* may accept or dispense data, in which cases it is said to be a *source*, or a *sink*, respectively. Normally, a channel has one source and one sink end, but Reo also allows channels to have two source or two sink ends. Channels are the primitive construct for coordination in Reo. Channel ends can be joined into *nodes* to assembly complex connectors. Nodes may be of three distinct types: (*i*) a *source* node connects only source channel ends; (*ii*) a *sink* node connects only sink channel ends and (*iii*) a *mixed* node which connects both source and sink ends. Source and sink nodes are also referred to as the *boundary nodes* or *ports* of a connector. Figure 2.1 depicts a basic set of primitive Reo channels, and four connectors built from the composition of some of these primitives.

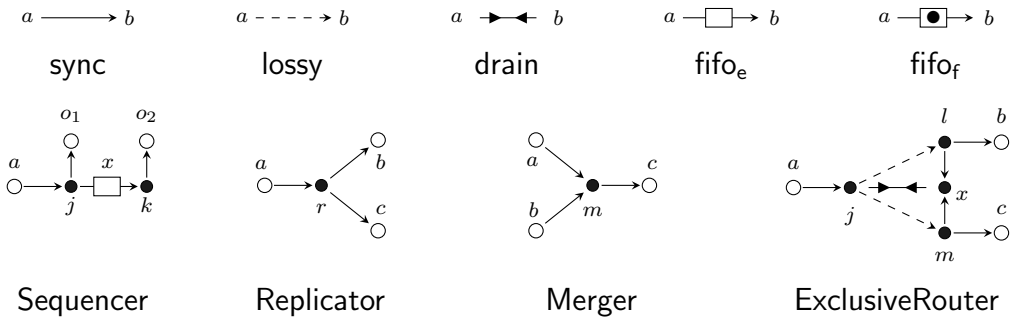


Figure 2.1: Primitive Reo channels (first row) and Reo connectors (second row).

In general, channels (and connectors) are activated by the presence of *input/output (IO)* requests (also referred to as read and write operations) pending at their boundaries. Each channel presents a different behaviour.

Informally, the sync channel consumes data at its source end and transmits it to its sink end, provided that there are IO requests at both ends at the same logical time. Otherwise, the channel blocks until communication is allowed to proceed. The lossy channel behaves likewise, but, instead of blocking, it loses data taken from its source end, whenever there is not a *matching* read operation pending at the sink end. The drain channel takes data from its two source ends, consuming it synchronously. Differently, a fifo channel has a buffering capacity of one memory position, allowing for asynchronous communication between its ends. Qualifiers e and f refers to the channel internal state (either *empty* or *full*, respectively).

The Sequencer and the ExclusiveRouter are examples of Reo connectors built from the composition of several primitive Reo channels, by joining their ends into nodes. A mixed node behaves as a Replicator (respectively, a Merger) when composed of a source (respectively, a sink) channel end and two or more sink (respectively, source) ends. In the first case it accepts data at the source end and replicates it to all connected sink ends. In the second, it merges non-deterministically to the sink

end data selected from the connected source ends. Clearly, when a mixed node is composed of multiple source and sink ends, data is simultaneously merged and replicated. This behaviour is referred in the Reo literature as the *pumping station*.

The Sequencer connector takes data from node a and transmits it to node o_1 and buffer x in a first synchronous step. Then it takes data from the buffer to node o_2 in a second synchronous step. The net effect is that nodes o_1 and o_2 receive data in sequence. The ExclusiveRouter connector takes data from node a and transmits it either to b or to c. In detail, data is transmitted to b (respectively, c) when this node has pending requests, but there are no requests at node c (respectively, b). When there are pending requests at both b and c, the merger node x chooses non-deterministically one of these nodes to receive data.

Graphically, the white circles represent the boundaries of the connectors, *i.e.,* source and sink nodes (used to link the connector to services or other connectors), while the black ones represent mixed (internal) nodes. The ExclusiveRouter node that assumes the behaviour of its homonym connector is usually represented as $\otimes$.

From the set of primitive channels and connectors depicted in Figure 2.1, attentions shall be drawn to the lossy channel as it presents a *context-dependent* behaviour. This is, its behaviour (more precisely, whether it loses data or not) depends exclusively on the environment issuing or not IO requests to its boundary nodes. This feature clearly deviates lossy channel from nondeterministic channels or connectors like the Merger. The ExclusiveRouter is another interesting case of a connector, as it presents both context-dependent and non-deterministic behaviour. The context-dependency feature is, therefore, a desired characteristic of some Reo channels, and as such, it is required that Reo semantic models correctly capture it.

2.1.2 Semantic models

Most of the work on Reo concerns the definition of a precise semantics for connectors and their composition. It is without surprise, then, that a multitude of models to formally describe the semantics of Reo have been proposed since its emergence [156]. Some semantic models describe data flow through timed data streams [13, 19]; others introduce colours to describe how and why data flows through nodes and channels [100, 93]; others still resort to some form of generalised automata. In this section the two most prominent automata-based models are recalled: *constraint automata (CA)* [29, 27] and *Reo automata (RA)* [54].

Constraint Automata

Constraint automata [29, 27] are defined over a set Σ of nodes representing the connector ports, and data constraints over Σ. Data constraints, collected in set DC, are given by the grammar:

$$g \ni \texttt{true} \mid d_A = d_B \mid g_1 \vee g_2 \mid \neg g,$$

where $A, B \in \Sigma$, d_A, d_B represent data items and the atomic proposition $d_A = d_B$ holds when node A is assigned the datum coming from B. Formally,

Definition 2.1. *A constraint automaton $\mathcal{A}$ is a tuple $(Q, \Sigma, \longrightarrow, Q_0)$, where Q is a set of states, $Q_0 \subseteq Q$, is the set of initial states, Σ is a (finite) set of ports, and $\longrightarrow \subseteq Q \times 2^{\Sigma} \times DC \times Q$, is the transition relation: each state transition is labelled by the set of ports which become active on its firing and a set of data constraints.*

Two binary relations, $\equiv$ and $\leq$, are defined over DC as follows: $g_1 \equiv g_2$ if g_1 and g_2 define equal data assignments; $g_1 \leq g_2$ if data assignments in g_1 imply those in g_2. Additionally, $dc(q, N, P) = \bigvee \{g : q \xrightarrow{N,g} p \wedge p \in P\}$ is the weakest data constraint that ensures the existence of a transition from q to any state in P, via a set of names N.

Figure 2.2 depicts the constraint automata for each of the primitive Reo channels and connectors shown in Figure 2.1. Consider, for instance, the sync channel: label $\{a, b\}$ captures the fact that both ends a and b are synchronously activated, while the constraint $d_b = d_a$ specifies that data present in a is transmitted to b.

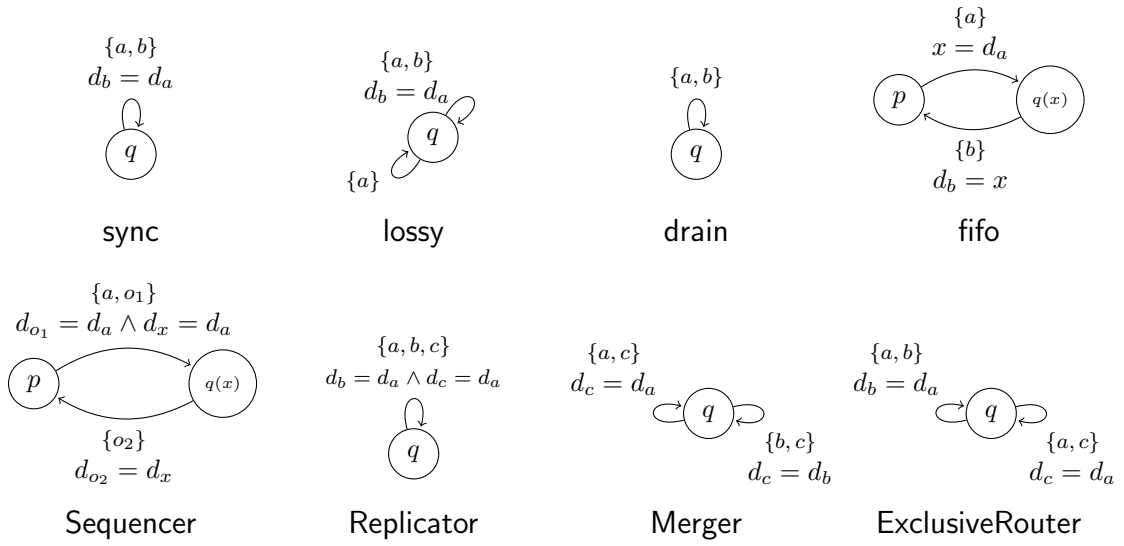


Figure 2.2: Constraint automata for the primitive Reo channels and connectors.

Constraint automata compose (composition is, as usual in automata theory, a

combination of product and hide [29]). However, the model is unable to capture context dependent behaviour. For example, a constraint automaton corresponding to a lossy channel models nondeterministically the choice between data flow and data loss, a decision which is intended to be (deterministically) made by the environment.

Definition 2.2 (Bisimulation [29, 50]). *Let* $\mathcal{A}_1 = (Q_1, \mathcal{N}, \longrightarrow, Q_{0_1})$ *and* $\mathcal{A}_2 = (Q_2, \mathcal{N}, \longrightarrow, Q_{0_2})$ *be two constraint automata. An equivalence relation* $R \subseteq Q_1 \times Q_2$ *is a bisimulation if and only if for any* $(q_1, q_2) \in R$*, the following holds:*

$$dc(q_1, N, P) \equiv dc(q_2, N, P) \tag{2.1}$$

$$q_1 \xrightarrow{N,} q_1' \Rightarrow \exists_{q_2'} \,.\, q_2 \xrightarrow{N,} q_2' \wedge (q_1', p_2') \in R \text{ (and vice versa)} \tag{2.2}$$

where $N \subseteq \mathcal{N}$ *is the set of names through which data may flow and* $P \subseteq Q_1 \cup Q_2$ *is the set of states to where it is possible to transit via names in* N*.*

Two constraint automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are bisimilar (denoted $\mathcal{A}_1 \sim \mathcal{A}_2$) if and only if there exists a bisimulation R such that for all initial states $q_{0_1} \in Q_{0_1}$ of $\mathcal{A}_1$ there is an initial state $q_{0_2} \in Q_{0_2}$ of $\mathcal{A}_2$ such that $(q_{0_1}, q_{0_2}) \in R$.

If equation (2.1) is replaced by $dc(q_1, N, P) \leq dc(q_2, N, P)$ and in equation (2.2) the *vice-versa* condition is dropped, then R is a simulation. Accordingly, two constraint automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are similar (denoted $\mathcal{A}_1 \preceq \mathcal{A}_2$) if and only if there exists a simulation R such that for all initial states Q_{0_1} of $\mathcal{A}_1$ there is an initial state $q_{0_2} \in Q_{0_2}$ of $\mathcal{A}_2$ such that $(q_{0_1}, q_{0_2}) \in R$.

Reo Automata

Reo automata [54], on the other hand, are context-sensitive and act as acceptors of *guarded strings*. Formally, let $\Sigma = \{\sigma_1, \ldots, \sigma_2\}$ be a set of ports. The set of guards is the free Boolean algebra $\mathcal{B}_\Sigma$ over Σ generated by the following grammar

$$g \ni \sigma \in \Sigma \mid \top \mid \bot \mid g \vee g \mid g \wedge g \mid \overline{g}.$$

and represent constraints on the firing of a transition. Atomic guards, collected in set $\mathbf{At}_\Sigma$, are conjunctions of $p, \overline{p}$, for $p \in \Sigma$. Intuitively they specify which ports are and are not enabled (*i.e.,* exhibiting pending requests or their absence). A guarded string over Σ is a sequence $\langle \alpha_1, f_1 \rangle ... \langle \alpha_n, f_n \rangle$, for $n \geq 0$, $f_i \subseteq \Sigma$, where each α_i is a guard and each f_i stands for the ports that synchronously fire read/write operations. Relation $\leq$ on guarded strings is defined as $g_1 \leq g_2 \iff g_1 \wedge g_2 = g_1$, thus expressing logic implication.

Definition 2.3. *A Reo automaton $\mathcal{A}_{Reo}$ is a tuple (Σ, Q, δ), where Σ is a set of ports, Q is a set of states and $\delta \subseteq Q \times \mathcal{B}_\Sigma \times 2^\Sigma \times Q$ is the transition function which satisfies the* reactivity *and* uniformity *conditions.*

A transition (q, g, f, q'), typically represented as $q \xrightarrow{g|f} q'$, says that if the connector is in state q and the port requests present at the moment, encoded as an atomic guard α, are such that $\alpha \leq g$, then the ports in f will fire and the connector will evolve to state q'. Intuitively, reactivity ensures that data flows through ports with pending requests, and uniformity enforces that the firing set of a port is a subset of its request set (see [54] for the formal definition).

Figure 2.3 depicts the RA for each of the primitive Reo channels and connectors considered in Figure 2.1.

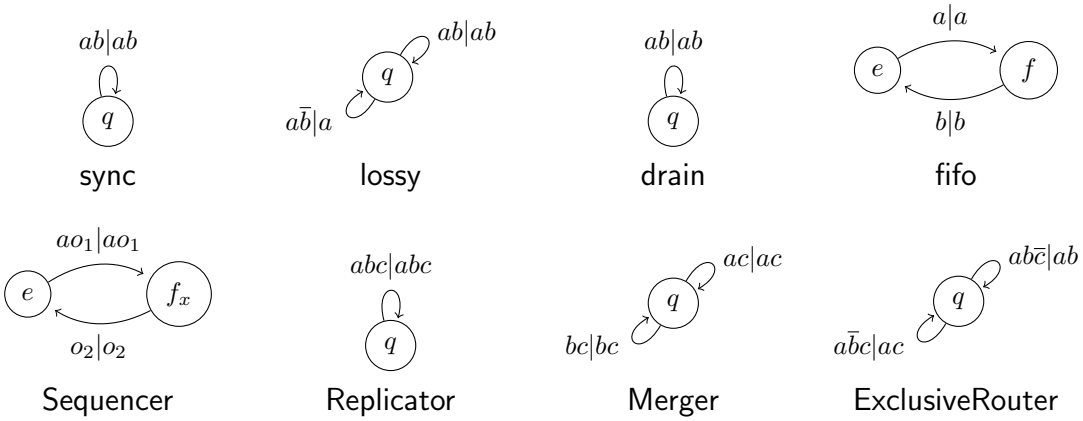


Figure 2.3: Reo automata for primitive Reo channels and connectors.

As shown in the lossy channel example, the context-awareness is well captured through the use of *negative* information in Reo automata. In this example, the operation in a fires (without synchronisation with b) when there is a request in a but not in b (represented by $\overline{b}$), as expressed in the guard.

Definition 2.4 (Bisimulation [54])**.** *Let $\mathcal{A}_1 = (\Sigma, Q_1, \delta_1)$ and $\mathcal{A}_2 = (\Sigma, Q_2, \delta_2)$ be two reo automata. An equivalence relation $R \subseteq Q_1 \times Q_2$ is a bisimulation* iff *for all $(q_1, q_2) \in R$ the following (and vice versa) hold:*

$$\forall_{q_1 \xrightarrow{g|f} q_1' \in \delta_1 \ \wedge \ \alpha \in \boldsymbol{At}_\Sigma} \ . \ \alpha \leq g \ , \exists_{q_2 \xrightarrow{g'|f} q_2' \in \delta_2} \ . \ \alpha \leq \ g' \ \wedge \ (q_1', q_2') \in R \qquad (2.3)$$

If equation (2.3) holds only in one direction, then R is a simulation. Accordingly, a Reo automaton $\mathcal{A}_2$ simulates another Reo automaton $\mathcal{A}_1$ (denoted $\mathcal{A}_1 \preceq \mathcal{A}_2$) iff all states of q_1 of $\mathcal{A}_1$ are in relation R with some state q_2 of $\mathcal{A}_2$, and R is a simulation. Similarly, $\mathcal{A}_1$ and $\mathcal{A}_2$ are bisimilar (denoted $\mathcal{A}_1 \sim \mathcal{A}_2$) iff for all states q_1 of $\mathcal{A}_1$ there is a state q_2 of $\mathcal{A}_2$ such that $(q_1, q_2) \in R$, with R a bisimulation.

2.1.3 Stochastic Reo

Stochastic Reo [16, 200] extends Reo by modelling coordination from a quantitative perspective. Non-negative real values are added both to channels and to their ends to represent, respectively, *processing delays* and *arrival rates* of IO requests. The former models the time needed for the channel to process data from one point to another, where point refers to a channel end, a buffer or a null space where data is lost or automatically produced. One channel may be annotated with more than one processing delay, depending on their operational behaviour. The latter models the time interval between consecutive arrivals of environment-issued IO requests to channel ends.

Figure 2.4 shows the basic channels used in stochastic Reo. In essence, they are represented as normal Reo channels, but annotated with stochastic values (rates and delays).

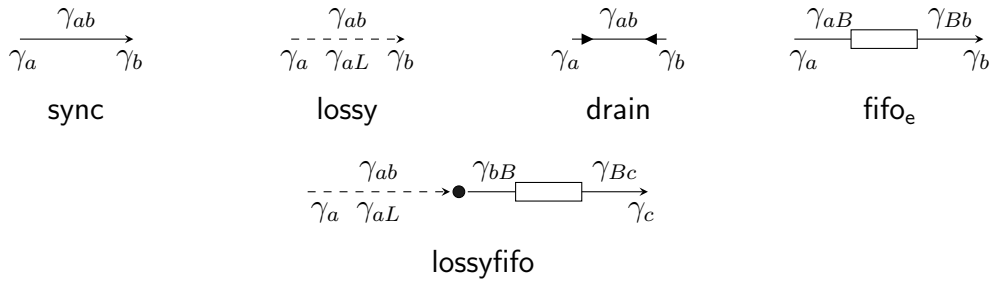


Figure 2.4: Primitive stochastic Reo channels.

Stochastic Reo is still compositional. Each composed channel retains its processing delay stochastic value. The request arrival rates, however, are only preserved for the boundary nodes of the connector. Mixed nodes cease their communication with the environment and are always ready to read/write data from/to the channels; therefore the arrival rates associated to the constituent channel ends are ignored (*c.f.,* lossyfifo connector in Figure 2.4). This behaviour is known as the *self-contained pumping station*, firstly referred in [14].

2.2 Interactive Markov chains

Interactive Markov chains (IMCs) [143, 144, 58, 258] were proposed as a model for performance evaluation of distributed (communicating) systems. IMCs extend process algebra [149, 190, 191, 25, 42] by modelling systems from a quantitative perspective framed in continuous time. IMCs address, then, qualitative and quantitative (probabilistic) behaviour of systems in a combination of process algebra and *continuous-time Markov chains (CTMCs)* [247, 28, 24].

2.2.1 Process algebra and *labelled transition systems*

The expression 'process algebra' refers to a mathematical way of describing the observable (possibly nondeterministic) behaviour of a distributed (communicating) system. In this regard, it presents (variants of) three operators for the sequential, alternative and parallel composition of processes (via their atomic actions), as well as fundamental laws on such operators. Process algebra, thus, enables the study of the behaviour of distributed system in what respects the interaction between its parts and their environments. Key studies include the verification that the specified system satisfies certain qualitative properties. Such verification (done through model checking techniques [94, 224, 95, 189, 138]) require, however, more abstract representations. Therefore, it is usual to formulate processes as automata-like structures: the *labelled transition systems (LTSs)*.

Definition 2.5 (Labelled transition system)**.** *A LTS is a triple* $(S, Act, \dashrightarrow)$*, where* S *is a set of states;* Act *is a set of actions and* $\dashrightarrow \subseteq S \times Act \times S$ *is a set of* interactive *transitions.*

If, in addition, an initial state is added to the definition of LTS, then it is called an interactive process [143]. In the context of process algebra, transitions in a LTS are labelled with the atomic actions with which the modelled process interacts with other processes, in particular, and the system environment, in general.

Process algebra and LTSs are not tailored, however, to model and study concurrent systems that present probabilistic behaviour. The specification of these systems requires their combination with other mathematical structures, namely, Markov processes.

2.2.2 Markov process and continuous-time Markov chains

A Markov process is a stochastic process (*i.e.,* a collection of random variables $\{X_t |\ t \in T\}$ in a discrete state space S, with T a total ordered set representing time) that fulfils the *Markov property*, which says:

$$\begin{aligned} &Prob\{X_{t_{n+1}} = j |\ X_{t_n} = i_{t_n}, X_{t_{n-1}} = i_{t_{n-1}}, \cdots, X_{t_0} = i_{t_0}\} \\ &\quad = Prob\{X_{t_n+1} = j |\ X_{t_n} = i_{t_n}\} \end{aligned} \tag{2.4}$$

That is, at a given time instance t_n, the proceeding behaviour of variable $X_{t_{n+1}}$ depends only on the current state (i_{t_n}), and not on its history. Markov processes can be *homogeneous*, if each variable X_t is completely independent of time t, or

inhomogeneous, if they depend on that time instance. Time can be regarded as discrete or continuous, in which cases it is assumed $T = \mathbb{N}$ or $T = \mathbb{R}^+$, respectively. The time domain defines classes of Markov processes: *discrete-time Markov chains (DTMCs)* [172] for the discrete-time domain and CTMCs for the continuous-time domain.

A consequence of the Markov property is the *memoryless property*, which states that the sojourn time (*i.e.,* the waiting time) at a state is independent of the time already spent in that state. Therefore, it is required that the random variables of the subjacent Markov process follow a distribution that observes this *memoryless property*. Only geometric and exponential probability distributions [117, 221] present such a property. The former is suitable for the discrete domain while the latter capture the continuous case. These are, therefore, the distributions associated to DTMCs and CTMCs, respectively.

Since performance of distributed systems is measured in the continuous-time domain, CTMCs are the suitable Markov processes to model the quantitative part of such systems. As a consequence, DTMCs are not recalled here.

A CTMC is, then, a Markov process with a discrete state space and continuous time range. It meets the Markov property (2.4). If the time is regarded as $t_0 < \cdots < t_{n-2} < t_{n-1} < t_n < t_n + \Delta t$, for some small time variation Δt, we obtain:

$$\begin{aligned} &Prob\{X_{t_n+\Delta t} = j |\ X_{t_n} = i_{t_n}, X_{t_{n-1}} = i_{t_{n-1}}, \cdots, X_{t_0} = i_{t_0}\} \\ &\quad = Prob\{X_{t_n+\Delta t} = j |\ X_{t_n} = i_{t_n}\} \\ &\quad = Prob\{X_{\Delta t} = j |\ X_{t_0} = i_{t_0}\}. \end{aligned}$$

This means that in the continuous time, the probability of being in a state $j \in S$ is only dependent on the initial state (or its distribution) and on the time Δt remaining to elapse on the current state $i_{t_n} \in S$. For each Δt and each pair of states i and j, there is a $\gamma \in \mathbb{R}^+$ that defines how the probability of going from i to j increases with time, making clear that, eventually, a transition occurs. For this reason, a CTMC is better formulated as a transition system.

Definition 2.6 (Continuous-time Markov chain)**.** *A CTMC is a tuple $(S, \longrightarrow, s_i)$, where S is a set of states, $s_i \in S$ is the initial state, and $\longrightarrow \subseteq S \times \mathbb{R}^+ \times S$ is a set of markovian transitions, satisfying the following:*

1. $|\longrightarrow \cap (\{i\} \times \mathbb{R}^+ \times \{j\})| \leq 1$, *for all* $i, j \in S$;

2. $(i, \gamma, i) \notin \longrightarrow$, *for all* $i \in S$ *and* $\gamma \in \mathbb{R}^+$.

Informally, these two points mean, respectively, that, at most, one transition mediates two consecutive states; and that loops are not considered, as the probability of staying in a state decreases with time.

Each markovian transition (i, γ, j) contributes to a delay of leaving state i. In concrete, γ is called the rate of a markovian transition and it models an exponential distribution that defines the sojourn time of a state. The probability of being less than t time units in a state i with a unique outgoing transition is given by $Prob\{sojourn(i) \leq t\} = 1 - e^{-\gamma t}$. In general, when two or more transitions share the same source state, a race condition occurs. If such is the case, a transition executes with delay determined by the minimum of the rates of the transitions, which is exponentially distributed on the sum of these rates. In the same conditions, to know the probability of going to a specific state in t time units, the following is used: $Prob(i, j) = \frac{\mathbf{R}(i,j)}{\mathbf{E}(i)} \left(1 - e^{-\mathbf{E}(i)t}\right)$, where $\mathbf{R}(i, j)$ is the rate of the transition from i to j and $\mathbf{E}(i) = \sum_{j' \in S} \mathbf{R}(i, j')$ is the exit rate of state i, given as the sum of the rates of all transitions leaving that state. In general, for $C \subseteq S$, $\mathbf{E}(i, C) = \sum_{j' \in C} \mathbf{R}(i, j')$ and $Prob(i, C) = \sum_{j' \in C} Prob(i, j')$. These formulas come from the essential properties of the exponential distribution:

- An exponential distribution $Prob\{delay \leq t\} = 1 - e^{-\gamma t}$ is characterised by a parameter $\gamma \in \mathbb{R}^+$ (the rate of the distribution). The mean duration of *delay* is given by $1/\gamma$.

- Exponential distributions observe the memoryless property. Consequently, the remaining delay after some delay t_0 has elapsed, is exponentially distributed.

$$Prob\{delay \leq t + t_0 | delay > t_0\} = Prob\{delay \leq t\}$$

- Exponential distributions are closed under the minimum. The minimum of two exponential distributions with rates γ_1 and γ_2, respectively, exponentially distributed with parameter $\gamma_1 + \gamma_2$:

$$Prob\{min(delay_1, delay_2) \leq t\} = 1 - e^{-(\gamma_1 + \gamma_2)t}.$$

- The probability that a delay (with rate γ_1) is smaller than another (with rate γ_2) is given by their rates as follows:

$$Prob\{delay_1 < delay_2\} = \frac{\gamma_1}{\gamma_1 + \gamma_2},$$

  and vice versa.

- The continuous time domain of exponential distributions ensures that no two delays elapse at the same time.

Although making part of the most adopted stochastic models for modern systems analysis, CTMCs are not compositional and do not allow for specifying observable nondeterministic behaviour of system, as process algebras do. Therefore, CTMCs are not enough to represent all aspects of distributed communicating systems with probabilistic behaviour. A more comprehensive model is in order.

2.2.3 Interactive Markov chains

IMCs combine process algebra (or LTSs, to be concrete) and CTMCs to model systems that present both nondeterministic and probabilistic behaviour. Consequently, they are regarded as transition systems that evolve through two kinds of transitions: *interactive* and *Markovian*.

Definition 2.7 (Interactive Markov chain)**.** *An IMC is a tuple* $(S, Act, \dashrightarrow, \longrightarrow, s)$*, where* S *is a nonempty set of states;* Act *is a set of actions;* $\dashrightarrow \subseteq S \times Act \times S$ *is the set of interactive transitions;* $\longrightarrow \subseteq S \times \mathbb{R}^+ \times S$ *is the set of Markovian transitions and* $s \in S$ *is the initial state of the chain.*

As expected, Markovian transitions (s, γ, s'), denoted by $s \xrightarrow{\gamma} s'$, model a random delay in the system's (internal) evolution governed by an exponential distribution parameter $\gamma \in \mathbb{R}^+$. Interactive transitions capture the system's interaction with the environment by means of action labels. Each transition (s, a, s'), denoted as $s \overset{a}{\dashrightarrow} s'$, represent a change in the system from state s to state s' via an external (single) action a that is executed atomically either immediately or blocked until the environment triggers it.

Operations

The combination of LTSs with CTMCs brings composition to stochastic models that, for instance, CTMCs do not observe alone. It is then feasible to define a composition operation for IMCs — the parallel composition or product — that allows for the modular combination of simple chains into more complex structures.

Definition 2.8 (Parallel composition of IMCs [143])**.** *Let* $I = (S_I, Act_I, \dashrightarrow_I, \longrightarrow_I, s_i)$ *and* $J = (S_J, Act_J, \dashrightarrow_J, \longrightarrow_J, s_j)$ *be two IMCs. The parallel composition of* I *and* J *with respect to a set of actions* M *is an IMC* $(S, Act, \dashrightarrow, \longrightarrow, (s_i, s_j))$*, where* $S = S_I \times S_J$*,* $Act = Act_I \cup Act_J$*, and* $\dashrightarrow$ *and* $\longrightarrow$ *are the smallest relations satisfying*

1. *If* $i_1 \dashrightarrow_I i_2$ *and* $a_I \notin M$, *then* $(i_1, j) \dashrightarrow (i_2, j)$, *for* $j \in S_J$;
2. *If* $j_1 \dashrightarrow_J j_2$ *and* $a_J \notin M$, *then* $(i, j_1) \dashrightarrow (i, j_2)$, *for* $i \in S_I$;
3. *If* $i_1 \dashrightarrow_I i_2$, $j_1 \dashrightarrow_J j_2$ *and* $a \in M$, *then* $(i_1, j_1) \dashrightarrow (i_2, j_2)$;
4. *If* $i_1 \longrightarrow_I i_2$, *then* $(i_1, j) \longrightarrow (i_2, j)$, *for* $j \in S_J$;
5. *If* $j_1 \longrightarrow_J j_2$, *then* $(i, j_1) \longrightarrow (i, j_2)$, *for* $i \in S_I$.

After composing two IMCs, they *execute* in parallel, and synchronise on shared external actions. This way, two systems only synchronise after a stimulus from the environment. But, often, it is desired that this synchronisation occurs internally, making the whole system a black box. Accordingly, IMCs adopt such a notion and define the abstraction — or hiding — operation as follows.

Definition 2.9 (Hiding of IMCs [143])**.** *Let* $I = (S, Act, \dashrightarrow_I, \longrightarrow_I, s)$ *be an IMC. The abstraction of* I *with respect to a set of actions* M *is an IMC* $(S, Act \setminus M, \dashrightarrow, \longrightarrow_I, s)$, *where* $\dashrightarrow$ *is the smallest relations satisfying*

1. *If* $i_1 \dashrightarrow_I i_2$ *and* $a \notin M$, *then* $i_1 \dashrightarrow i_2$;
2. *If* $i_1 \dashrightarrow_I i_2$ *and* $a \in M$, *then* $(i_1) \dashrightarrow (i_2)$.

The interactive transitions with the special action τ are referred to as τ-transitions. As expected, such transitions represent internal (unobservable) activities. For further reference, the source states of τ-transitions are said *unstable*; otherwise they are said *stable*.

Since τ-transitions do not interact with the environment, they are assumed to take place immediately. Consequently, τ-transitions always precede Markovian ones, when their source states coincide. This owes to the fact that the probability of leaving a state with a 0-time units delay is always null: $1 - e^{-\gamma \cdot 0} = 1 - e^0 = 1 - 1 = 0$. This is known as the *maximal progress assumption* and, in practice, has the effect of reducing the size of IMCs. Notice, however, that this only concerns Markovian transitions; interactive transitions may as well execute immediately.

Comparing IMCs

Internal transitions play an important role in the behaviour of an IMC. Due to the presence of such transitions, two IMCs may present equivalent observable behaviour, but being different when regarded in a step-wise perspective, this is, when compared through weak or strong bisimulation, respectively.

Definition 2.10 (Strong bisimulation [144]). *Let $I = (S, Act, \dashrightarrow, \longrightarrow, s)$ be an IMC. An equivalence relation $\mathcal{R} \subseteq S \times S$ is a strong bisimulation on I if for any $(i,j) \in \mathcal{R}$ and equivalence class $C \in S/\mathcal{R}$ the following holds:*

1. *for any $a \in Act$, if $i \dashrightarrow^{a} i'$, then there exists a state $j' \in S$ such that $j \dashrightarrow^{a} j'$, and $(i', j') \in \mathcal{R}$ (and vice-versa);*
2. *if i is a* stable *state, then $\mathbf{E}(i, C) = \mathbf{E}(j, C)$;*

Two IMCs I and J with disjoint state spaces S_I and S_J and initial states i and j, respectively, are *strong bisimilar* (denoted $I \sim J$) if there exists a strong bisimulation $\mathcal{R} \subseteq S_I \cup S_J$ such that $(i,j) \in \mathcal{R}$. As expected, strong bisimilarity is the largest strong bisimulation and it is a congruence *w.r.t.* parallel composition and hiding operations [143].

Intuitively, strong bisimulation looks at all interactive transitions in a step-wise way (as traditional bisimulation in LTSs) and require that the cumulative rates of moving from bisimilar states to the same equivalence class is equal, unless the states are unstable. Differently, weak bisimulation regards visible interactive transitions in a step-wise way, but sees no difference between one τ-transition or a sequence of them; completely assuming, thus, the maximal progress notion.

Definition 2.11 (Weak bisimulation [144]). *Let $I = (S, Act, \dashrightarrow, \longrightarrow, s)$ be an IMC. An equivalence relation $\mathcal{R} \subseteq S \times S$ is a weak bisimulation on I if for any $(i,j) \in \mathcal{R}$ and equivalence class $C \in S/\mathcal{R}$ the following holds:*

1. *for any $a \in Act$, if $i \dashrightarrow^{a} i'$, then there exists a state $j' \in S$ such that $j \dashrightarrow^{a} j'$, and $(i', j') \in \mathcal{R}$ (and vice-versa);*
2. *if i is stable then $\mathbf{E}(i', C) = \mathbf{E}(j', C)$ for some state j' stable such that $j \dashrightarrow^{\tau^*} j'$ and $(i, j') \in \mathcal{R}$.*

Two IMCs I and J with disjoint state spaces S_I and S_J and initial states i and j, respectively, are *weak bisimilar* (denoted $I \approx J$) if there exists a weak bisimulation $\mathcal{R} \subseteq S_I \cup S_J$ such that $(i,j) \in \mathcal{R}$. As expected, weak bisimilarity is the largest strong bisimulation and it is a congruence *w.r.t.* parallel composition and hiding operations [143].

2.3 Hybrid Logic

Hybrid logic (HL) [49, 56, 47, 174, 57] extends *modal logic (ML)* [48] by adding to the latter the possibility of referring to specific states in the underlying Kripke

structure. This new expressive power is brought by the introduction of both a new sort of propositional symbols, called *nominals*; and the *satisfaction* operator, @. These extensions are addressed in detail later on in this section. Before, however, one shall recall the basic notions of ML.

2.3.1 Modal Logic

ML (or propositional ML) springs from classic propositional logic and it allows for talking about relational structures from a local and internal perspective, which is not offered by other logics.

Syntax

As a derivation from propositional logic, ML language resorts to a set of propositional variables $\Phi = \{p, q, r, \ldots\}$, the boolean connectives $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \top, \bot\}$ and adds two new unary operators: the *diamond*, $\Diamond$, and the *box*, $\Box$. Sentences (or formulas) of this modal language respect the following grammar:

$$\phi, \psi \ni p \mid \neg\phi \mid \psi \vee \phi \mid \Diamond\phi,$$

where $p \in \Phi$. The other boolean connectives are obtained as usually: $\top = \phi \vee \neg\phi$ $\bot = \neg\top$; $\psi \wedge \phi = \neg(\neg\psi \vee \neg\phi)$; $\psi \rightarrow \phi = \neg\psi \vee \phi$ and $\psi \leftrightarrow \phi = (\psi \rightarrow \phi) \wedge (\phi \rightarrow \psi)$. Finally, the box operator is obtained from the diamond: $\Box\phi = \neg\Diamond\neg\phi$ (as much as the existential quantifier operator, $\exists$, is obtained from the universal quantifier operator, $\forall$, in *first order logic (FOL)*).

Semantics

Relational structures or, in general, transition systems, are the object of study of ML. ML formulas are analysed from a precise point or state of a relational structure. Intuitively, formulas of the form $\Diamond\phi$ and $\Box\phi$ express that ϕ *possibly* holds in some of the states in relation with the original one; and ϕ *necessarily* holds in all the states in relation with the original one, respectively.

A relational structure, in this context, is often referred to as a Kripke structure and its states are called worlds.

To formally define the semantics of the modal language trough a satisfaction relation, it is necessary, first, to recall the notion of model (of the modal language).

Definition 2.12 (Model)**.** *A Model (of a modal language) is a pair* $\mathfrak{M} = (\mathfrak{F}, \nu)$*, where* $\mathfrak{F} = (W, \mathcal{R})$ *is Kripke structure with* W *a non-empty set of worlds and* $\mathcal{R}$ *a*

binary relation on W; and $\nu : \Phi \to 2^W$ is a valuation function that assigns to each $p \in \Phi$ a subset of W where p holds.

Therefore,

Definition 2.13 (ML Satisfaction)**.** *Let $\mathfrak{M} = (W, \mathcal{R}, \nu)$ be a model, $w \in W$ a state and ϕ a modal formula. The satisfaction of ϕ at state w in model $\mathfrak{M}$, denoted $\mathfrak{M}, w \models \phi$, is inductively defined as follows:*

$$
\begin{array}{lll}
\mathfrak{M}, w \models p & \text{iff} & w \in \nu(p), \quad \text{for } p \in \Phi \\
\mathfrak{M}, w \models \neg\phi & \text{iff} & \mathfrak{M}, w \not\models \phi \\
\mathfrak{M}, w \models \psi \vee \phi & \text{iff} & \mathfrak{M}, w \models \psi \text{ or } \mathfrak{M}, w \models \phi \\
\mathfrak{M}, w \models \Diamond\phi & \text{iff} & \exists_{w' \in W} \cdot (w, w') \in \mathcal{R} \wedge \mathfrak{M}, w' \models \phi
\end{array}
$$

Again, the semantics of $\Box\phi$ and of the remaining boolean connectives is obtained from this minimal set of inductive rules.

Clearly, the satisfaction relation $\models$ can be lift to *global* satisfaction in a model $\mathfrak{M}$ by quantifying over the states of such a model. Thus, ϕ is *globally satisfied* in a model $\mathfrak{M}$ if it is satisfied in all the states of $\mathfrak{M}$, written $\mathfrak{M} \models \phi$. In practice it may be useful to lift the satisfaction relation to a state in a Kripke structure, to a Kripke structure or even to a set of such structures [48].

Definition 2.13, however, is for the basic ML presented. Variants of this logic exist, where the modalities $\Diamond$ and $\Box$ are further extended with parameters that provide them different meanings. For instance, temporal logic takes into consideration time as a linear relational structure and uses modalities with parameters F and P (*c.f.,* $\langle F \rangle$, $\langle P \rangle$, $[F]$ and $[P]$) to refer to the future and the past, respectively. Upon this one, *linear temporal logic (LTL)* [95] and *computation tree logic (CTL)* [245] arose, that introduce even more modalities to refer to properties checked on the Kripke structure. Despite their crescent complexity, these modal languages fail to provide concrete syntactic ways to directly address states of relational structures. This motivates the hybrid extension to ML.

2.3.2 Hybrid extension to modal logic

As mentioned above, HL adds to a modal language the ability to name, or to explicitly refer to, specific states of the underlying Kripke structure. This extension is partially done through the introduction of a new sort of propositional variables,

called *nominals*: $\Lambda = \{i, j, k, \ldots\}$. The particular difference between nominals and propositional variables is that the former hold in exactly one state, instead of in a set of states. The satisfaction operator, @, is introduced to complete the hybrid extension to modal logic, and being able to directly refer to an exact point in the Kripke structure. The @ operator is used to formalise that some sentence ϕ holds at exactly some point w of the Kripke structure, named by a nominal i: $@_i\,\phi$.

Syntax

Naturally, the hybrid language of HL builds on top of the modal language recalled before. Thus, for $i \in \Lambda$ and $p \in \Phi$, sentences or formulas on HL respect the following grammar:

$$\phi, \psi \ni i \mid p \mid \neg\phi \mid \psi \vee \phi \mid \Diamond\phi \mid @_i\,\phi.$$

Semantics

HL is, as expected, interpreted in a model $\mathfrak{M} = (W, \mathcal{R}, \nu)$, over the satisfaction relation $\models$. But differently to the ML models, the valuation function, ν, in HL has domain in $\Phi \cup \Lambda$, range in 2^W and counts with the restriction $|\nu(i \in \Lambda)| = 1$. Accordingly, the satisfaction relation, $\models$, for this hybrid extension of the basic ML is formalised as follows:

Definition 2.14 (HL Satisfaction)**.** *Let $\mathfrak{M} = (W, \mathcal{R}, \nu)$ be a model, $w \in W$ a state and ϕ an hybrid formula. The satisfaction of ϕ at state w in model $\mathfrak{M}$, denoted $\mathfrak{M}, w \models \phi$ is inductively defined as follows:*

$$\begin{array}{lll} \mathfrak{M}, w \models i & \mathsf{iff} & w = \nu(i), \quad \mathsf{for}\ i \in \Lambda \\ \mathfrak{M}, w \models @_i\,\phi & \mathsf{iff} & \mathfrak{M}, \nu(i) \models \phi \end{array}$$

The semantics for the other formulas is as presented in Definition 2.13. As usually, $\mathfrak{M} \models \phi$ defines the lift to *global* satisfaction in a model. Note that all the formulas of the form $@_i\,\phi$ are globally satisfied in a model. Actually, the introduction of the @ operator breaks the need to analyse a formula in the context of a specific state w. This is because w does not interfere in the satisfaction of $@_i\,\phi$, since ϕ is always verified at the state referred by i, irrespective of w.

Part I

A stochastic model for software coordination

Chapter 3

State of the Art: Models for Performance Evaluation

I will never believe that God plays dice with the universe.

– *Albert Einstein*

In this chapter. The state-of-the-art models and techniques for performance evaluation are reviewed. In concrete, models are addressed that sprang from algebraic counterparts, such as process algebra, Petri nets, automata or Markov chains. Models based on queueing theory and those specific to capture component-based software performance are also addressed. Finally, models conferring stochastic semantics to concrete (stochastic-based) coordination models are discussed.

3.1 Algebraic stochastic models

A number of algebraic models have been extended to support stochastic information, in an attempt to go along with the crescent quantitative demands of software systems. In this section, relevant contributes that define the state of the art of such models are reviewed.

3.1.1 Stochastic process algebras

Stochastic process algebras (SPAs) [145, 89] are a stochastic extension to process algebra that integrate stochastic processes into the former. As such, they are intended to serve as models of performance analysis for (typically concurrent) systems that

communicate and interact with each other. SPAs present multiple characteristics, among which composition, formality and abstraction are highlighted. The first allows for modelling a complex system from the specification and interaction of its parts; the second endows the formalism with the definition of precise semantics to all its constructs; and finally, the third enables regarding systems as black boxes, by allowing to hide internal specificities, as desired.

Several SPA representatives arose with the intent of capturing different aspects in the quantitative modelling of concurrent systems. The *performance evaluation process algebra (PEPA)* [146] formalism was the first such representative to deliver Markovian support for performance calculation. In this formalism, the specification of a system is described by the interaction of its components. Components are able to perform actions in order to synchronise with other components. Each action is assumed to be associated with a random variable, modelling its duration. Durations are exponentially distributed, and hence, the relation to Markovian processes. Synchronisation in PEPA allows for more than one action in a component to be carried out, given that they do not exceed the bounded capacity assumed for the component. *Extended Markovian process algebra (EMPA)* [44] is another SPA representative also based in Markovian processes. It differs from PEPA in the sense that during synchronisation, only one action can be carried out. More representatives have been proposed. Among them are *interactive Markov chain (IMC)*, already detailed in Section 2.2, stochastic π calculus [223] or MoDeST [105], to mention but a few. These formalisms intend to overcome SPA limitations like, for instance, the specific use of exponential distributions to model time. MoDeST, is one such representative that accepts general distributions to that end.

3.1.2 Stochastic Petri nets

Stochastic Petri nets (SPNs) [194] were introduced as stochastic extensions to classic Petri nets. They associate exponential distributions to model firing delays of each transition in a Petri net. The introduction of these random variables made possible to prove that the reachability graph of the SPN is isomorphic to a Markov process. *Continuous-time Markov chains (CTMCs)* may, then, be generated from the SPNs, enabling the computation of performance measures. This formalism excels in modelling computer systems that use multiple resources. However, it is not compositional and does not take into account the global synchrony of events that are mostly required in concurrent systems. Moreover, it presents both a large number of reachability markings and complex solutions. It does not allow to model internal behaviour disregarding time.

In order to partially solve these problems, M. Marsan [183, 184] proposed the *generalised stochastic Petri nets (GSPNs)*. In this model, two sets of transitions are considered: timed and immediate transitions. The former associate firing delays to transitions as in SPNs and the latter have no delays associated, having, for that reason, priority over the timed transitions. Moreover probabilistic behaviour can also be associated to immediate transitions by adding priority levels to them. Summing up, this enables the reachability graph of a SPN to be partitioned into two markings that decrease the complexity of the solution model. Nevertheless, these are still non compositional models.

3.1.3 Stochastic automata networks

Stochastic automata networks (SANs) [240, 118] became popular in the late nineties to formalise parallel and distributed systems. In this formalism, each component is modelled as a stochastic automaton that may either run independently or synchronously interact with other such automaton. It is, thus, a natural compositional model.

Each stochastic automaton has its own probability transition matrix and is considered to have two types of transition rates. In one, rates are constants and are independent of any other automaton in the network; in the other, rates are functions from the global state space to the non negative reals. Moreover these automata are assumed to have two types of interaction: functional and synchronising. In the first type of interaction, only the local state of the automaton performing the interaction is affected. Thus, no global information of the system is required, besides the knowledge about the global state of the network. For the second sort of interaction, though, it is necessary to compute a global matrix of possible transitions. This may force a change in the states of the automata in the network, changing its global state.

In summa, the model mitigates the state space explosion problem of Markovian models as they are represented by small transition matrixes. Relevant properties of the system may be inspected without the need for computing a global matrix. But a requirement of these systems is that they operate almost independently and very rarely synchronise. Otherwise, state space may explode as much as in other state-based formalisms.

3.1.4 Markovian-based models

Markovian-based models are the *de facto* formalisms to measuring performance of computational systems. With effect, most of the reviewed stochastic models have an underlying counterpart Markovian model, that is actually used for obtaining the desired quantitative results. *Discrete-time Markov chains (DTMCs)* and CTMCs are two of the most used models.

However, DTMCs model time as discrete events, and thus have limited applicability. CTMCs, although observing time continuously, which makes them suitable for modelling real-time systems, they are not naturally compositional. They fail to deliver means for correctly modelling interacting systems. Contrarily, IMCs are compositional and have underlying support for continuous-time modelling. It supports non-determinism and delays. Recently, Markov automata [114, 110] were proposed as a model able to support non-determinism, probabilistic behaviour or time delays. This is, in fact, a combination of the theories behind IMCs and probabilistic automata.

As mature models for performance evaluation, tools abound to support reasoning about and analysing systems modelled in such formalisms. PRISM [173] and CADP [125] are two of the most popular and generic ones.

3.2 Queueing networks

Queueing theory [247, 61, 141] also plays an important role in the stochastic modelling of systems. In particular, the components of a system can be reduced to queues and the system itself as a network of queues.

A queue is a system to serve customers. Customers arrive to be served by a service station, and when served leave the system. A queue is endowed with one or more servers in its service station, and a waiting room with possibly unbounded capacity. Customers *wait* in the waiting room when all the servers are busy. A discipline is associated to queues for selecting the next customer to be served. Usually, first-come-first-served discipline is adopted, but more complex models may be adopted, for instance, one that prioritise customers.

A queue is a high-level model describing a probabilistic system with underlying semantics of a CTMC. The model incorporates stochastic processes for customer arrival and service delay times. It counts on other parameters like, for instance, the number of servers and the size of the waiting room. Given these parameters, a queue may be represented as, for instance $M/M/1$. This is the simplest queue representation considered in queueing theory. The first M means that the arrival of

customers is modelled as a Poisson process; the second M informs that the service time is modelled by a Markovian process lead by an exponential distribution; and finally, 1 means that the queue counts on a unique server. Moreover, this queue is unbounded and follows a first-come-first-served serving discipline.

In turn, a queueing network is regarded as a graph, where nodes are queues and edges are connections between them. Consequently, queueing networks have a routing matrix associated that define the probabilities with which a customer goes from a node into another. Special probabilities are assumed to exist to determine entering and leaving the network. Moreover, the network still adopts the notion of state, which is given by a vector, where each element is the number of clients in each station. This state is useful to describe state-dependent routing probabilities.

Performance analysis requires to know the stationary probabilities of the network (that hold in equilibrium). Although finding such stationary probabilities is a complex problem, these can be computed whenever the networks are separable. When such is the case, the stationary probability of the whole is given by the product of the individual queueing stations.

Classic queueing theory define important properties and formulas for queueing networks, in order to obtain desired performance evaluations. But usually, the underlying networks assume queues with unbounded waiting rooms. This is a main drawback in modelling real systems since resources (*e.g.,* memory) are always limited.

Queueing networks with finite capacity [32, 31, 213] can more realistically represent software systems. In these queues, when the servers are busy and the waiting room reaches its capacity, the queue prevents the flow of customers. This makes other queues (that flush customers into the busy and full one), and possibly the surrounding environment, to block.

Research in queueing networks also includes the study of losses in the context of finite capacity networks of queues [163]. The results of such studies become interesting for modelling systems that present such loss behaviour. In fact, loss-related performance measures are necessary for intensively accessed systems.

However, finite capacity queueing networks do not have, in general, a closed product-form. That is, the global analysis of the network cannot result from the individual analysis of each node. Clearly, this adds complexity to performance analysis. Approximation and simulation solutions are proposed for these queues. But this is not always desired when modelling, for instance, critical software systems. Moreover, although queueing networks are a compositional model, theory for fully supporting synchronisation is still an open issue.

3.3 Component-based performance evaluation

Models for the evaluation of component-based software performance usually focus on the components operational behaviour and disregard important information. This includes, for instance, development processes, specificities of component model paradigms or the context in which the system is to be deployed [168]. In this context, several approaches have been proposed that rely on high-level models of components and define, upon these, specific models for measuring quality attributes, excelling on performance measuring. In the end, these approaches result in a more human-friendly way of designing systems with non-functional requirements and obtain their performative measures.

A. Bertolino and R. Mirandola [45] present the CB-SPE framework, a compositional approach for performance evaluation of component-based software. It considers the entire life-cycle of software development, allowing for assessing quality attributes from requirement elicitation to deployment and maintenance.

The companion tool resorts to the *unified modelling language (UML)* notation for modelling components and their predicted performance attributes. The approach considers two modelling layers. One is a component layer, where architects create a repository of components. The components present predicted performance properties that are platform (deployment context) independent. The other is an application layer, where architects select the required components to compose a concrete architecture. Use cases are associated with some probability in order to define usage profiles of the application, and sequence diagrams model the control flow between the components. Moreover, by this time, the architect is in possession of deployment information and, thus, is able to associate to the context-dependent performance attributes of the components. This is done by producing UML deployment diagrams that describe resources and communication means.

Once the complete performance model is established, the CB-SPE tool is able to convert the underlying UML diagrams into a suitable queueing network. The latter is, in fact, the responsible for computing the global performance properties of the application.

E. Bondarev *et al* [53, 52] devise an approach for component-based performance evaluation on top of the ROBOCOP component model [132]. ROBOCOP components are modelled with resource consumptions, functional and behavioural specificities and an execution implementation. Each component has performance measures for each task they can execute, which are independent from the deployment environment. These components are then assembled together in an application architecture, which combines the behaviour and the resource consumption specifications.

The application is then simulated for critical scenarios, outputting result concerning execution times, latency and resource utilisation of each task.

The authors defend this approach as useful for predicting timing properties at early stages of development. Simulation avoids computation complexity by discarding full-state analysis of the system. Moreover, synchronisation and scheduling aspects are considered in this model, as desired in interacting systems.

V. Grassi *et al* [136, 137, 135, 134] introduce an approach for performance prediction of component-based software systems, centred on a kernel language named KLAPPER. The approach aims at hiding knowledge about performance analysis methodologies from the architects. This is achieved by using model transformation techniques that convert system models (*e.g.,* UML) into analysis models (*e.g.,* Markov chains or queueing networks).

The KLAPPER language establishes this desired bridge between design and analysis. It captures the necessary information for the performance evaluation from the design models, and converts them into stochastic models. The architects just need to define transformation rules that convert their component models into KLAPPER.

S. Becker *et al* [38] present an approach to performance evaluation of distributed systems upon a model of components called Palladio. The Palladio component model is used to specify architectures in a parametric way. The underlying development methodology offers support for different development roles. Concretely, it separates concerns for component developers, software architects, system deployers and domain experts. The component developers model components by using control flow graphs annotated with information about resource consumption. This information is parametrised so that it is independent of the deployment environment specificities. The software architects select components and glue them together in an application architecture. The system deployers are responsible for modelling the deployment environment and associate components demands to the environment resources. Lastly, domain experts model usage profiles that influence variables like, for instance, workload.

Tool support exists for this approach, endowing the developers with means for modelling a component-based system at a high-level of abstraction. The tool is responsible for weaving the parts concerning each development phase into a complete model. The latter is then translated into suitable queueing networks for performance analysis either via simulation or numerical analysis.

J. Karim *et al* [159] define an approach for performance assessment of component-based software systems, that targets the early phases of the software life-cycle. The architecture of the system (the first product) is regarded as the main piece for this

approach. It is from its description that the overall system performance is evaluated. Such architecture is designed by using UML diagrams. In particular, component diagrams are considered to specify the structure of the system; and sequence diagrams to describe both component behaviour and component interaction. The approach takes each component sequence diagram and converts it into an interface automaton [6], via an algorithm devised by the authors. The set of all interface automata are combined, defining the formal foundation for evaluation. It is in this combination that performance values are added. Synchronisation and processing delays are considered in the edges of the interface automata and combined conveniently. Finally, queueing theory is used to compute several performance attributes based on the obtained automata.

3.4 Coordination-oriented stochastic approaches

In the last few years, research on coordination models redirected attention to the quantitative area, as imposed by the performative demands of new software systems. This led to improving and extending already existing models of coordination, by incorporating quantitative information therein. In the sequel, models for generic coordination and also those centred on choreography and orchestration of services are considered.

3.4.1 Generic coordination

SBIP [40] is a stochastic extension of the BIP coordination model [35, 34]. In short, the BIP model specifies the static coordination of component-based architectures based on three layers: a behavioural, an interaction and a priorities layer. In the behavioural layer, the operational semantics for atomic components is laid out as finite state automata. States of the automata refer to locations where the components wait for synchronisation and store data in variables. Transitions are labelled with actions/ports and are constrained by guards on data and executions, which are provided in external functions. In the interaction layer, the actual communication policies (connectors) are defined for the component interaction. They are regarded as sets of ports that shall synchronise. In the priorities layer, interactions are prioritised taking into account conditions that are used, for instance, to solve non-determinism. The semantics of BIP is obtained by the composition (formally defined as the cartesian product) of the automata corresponding to each atomic component. Synchronisation is performed on the labels of the automata transitions as defined in the interaction protocols.

SBIP allows for the specification of stochastic aspects of atomic components in BIP. Stochastic behaviour is brought into scene by the use of stochastic variables in the automata states. These variables are associated with some probabilistic distribution which can be updated on transitions. The combination of BIP components with stochastic variables creates a stochastic semantics based on transition systems, capable of modelling both stochastic and non-deterministic behaviour. Non-determinism (which may remain after applying priorities) is removed from the resulting transition system. This is essential in order to obtain purely stochastic semantics. The construction of the overall transition system for the set of all atomic components and their interactions is made as in BIP.

SBIP is supported by suitable tools and algorithms that cater for statistical model checking of quantitative and qualitative properties expressed in (probabilistic) bounded linear temporal logic. Contrary to many other stochastic models, SBIP is not restricted to exponential distributions. C functions are assumed to be attached to the model that deliver support for other probabilistic distributions.

Several approaches were also attempted in the Reo community. In particular, such efforts centred on extending and incorporating stochastic behaviour into the essential semantic models of Reo. In the following paragraphs, the most representative quantitative semantic models are addressed.

C. Baier and V. Wolf [30] proposed *continuous-time constraint automata (CCA)* as an extension to *constraint automata*. It incorporates quantitative information on transitions of the latter, representing a (high-level) delay on the continuous-time setting. The CCA model follows closely the IMCs model in the sense that it presents two types of transitions: one is for interaction with the environment and the other is for delaying the evolution between states. The former behave as the transitions on *constraint automaton (CA)*; the latter (also referred to as Markovian transitions) take a positive real value (the parameter of an exponential distribution), which governs a stochastic process modelling the average delay between the occurrence of consecutive actions.

As a descendant of the CA model, it inherits relevant properties, namely composition, however, it fails to capture context-dependency. Moreover, the state-space increases to aport states where the coordination structure is waiting for interaction with the environment; and it is not consistent, as there is not a unique model for each Reo channel.

F. Arbab *et al* [16] introduce *quantitative intensional automata (QIA)*. This model spring from intensional automata [100] by adding data constraints, as in CA, and quantitative information to model processing delays and arrival rates of

interaction actions with the environment. Its states model the internal (memory) configuration of the connector together with the configuration of its boundary nodes, *i.e.,* the existence/absence of pending *input/output (IO)* requests.

QIA correctly captures context dependency, but it suffers from state explosion even when representing simple connectors, restricting scalability. Moreover, it is not a compositional model, because of its unstructured composition operator [198, 16].

Y.-J. Moon [199, 198, 200] proposed *stochastic Reo automata (SRA)*. It extends *Reo automata (RA)* by incorporating processing delays into transitions, and arrival rates of IO requests into the nodes of the Reo connector. In fact, this is the semantic model underlying the stochastic Reo model as presented in Section 2.1.3.

The resulting models are compositional (as inherited from RA), compact, and capture context-dependency. The practical use of stochastic Reo automata is, however, constrained by the lack of tool support. In an attempt to bridge this gap, partial translations were provided of these automata into CTMC and IMC, so that tool support from these standard models could be used [200]. The translation into IMC was, however, concluded not to correctly capture stochastic Reo semantics [200]; the other provided, indeed, concrete models for performance computation. Unfortunately, CTMC are not to compositional models, which enforces to recalculate the SRA model and translate it into CTMC every time the underlying Reo connector changes.

3.4.2 Workflow, choreography and orchestration

Workflow, choreography or orchestration notations like BPEL (and variants), *business process modelling notation (BPMN)* or Orc are extensively used for the high-level definition of business processes. In particular, they have a preponderant role in the specification of both workflows and interaction of system components and services. In the sequel, some state-of-the-art representatives of research on these models with respect to performance evaluation are mentioned.

D. Bruneo *et al* [63] propose a method for studying the *quality of service (QoS)* of composed *web services (WSs)* at design time. The underlying technique is to derive non-Markovian stochastic Petri net models from WS-BPEL models that are defined during application design. QoS information is added as annotation to the activities in the WS-BPEL models.

Y. Xia *et al* [256] proposed a similar technique that transforms BPEL models into GSPNs. A set of transformation rules addressing important BPEL constructs like activities or scopes, are used to that end. The approach only considers the elements directly related to the flow of activity execution as well as the most relevant aspects

of service composition. In particular, the authors propose a state-space method for computing the *expected-process-normal-completion-time* as the main performance estimate.

L. Herbert and R. Sharp [142] present a framework for quantitative analysis of business workflows in a slightly changed subset of BPMN. This extension to BPMN adds to it stochastic information as well as non-deterministic branching and reward annotations. The approach is similar to those presented before, in the sense that the designed workflows are translated into a target stochastic model; in this case, Markov decision processes. The latter are then submitted to the PRISM model checker in order to obtain relevant qualitative and quantitative information about the business process.

A. Benveniste *et al* [41] provide a complete framework for QoS-aware management of monotonic service orchestration. Monotonicity, in this study, refers to the property that orchestration performance augments as the performance of individual services augments too. This is not always verified, and thus the authors consider it a relevant property for the management of service orchestration, making it the base for a contract-based design of applications. For this, the framework includes an algebraic calculus for QoS composition, that captures how QoS of the orchestration is obtained from the individual QoS of the considered services.

In order to concretise performance evaluation, a model of stochastic orchestration, Probabilistic OrchNets, is devised based on Petri nets. The overall framework is implemented on top of the Orc orchestration language, but the authors defend that it is also applicable to BPEL. The use of OrchNets for modelling QoS and Orc for modelling the operational behaviour of the orchestration, implements the desired separation of concerns. Weaving techniques are proposed to combine the two in order to obtain a stochastic orchestration model, from where measures of performance are obtained.

Chapter 4

Interactive Markov chains for stochastic Reo

We can't solve problems by using the same kind of thinking we used when we created them.

– *Albert Einstein*

In this chapter. A quantitative semantic model for stochastic Reo, based on IMCs, is formalised. Important properties of this model are demonstrated: (*i*) it is compositional and (*ii*) its behavioural equivalence relation is a congruence with respect to the composition operations. Stochastic Reo is extended to more faithfully approximate the modelling of real-world software connectors. The latter implies to revisit the formalised quantitative semantic model over a more modular perspective.

Part of this chapter's content was previously published, by the author, in:

- Nuno Oliveira, Alexandra Silva, and Luís S. Barbosa. "Quantitative Analysis of Reo-based Service Coordination". In: *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. Vol. 2. SAC'14. Gyeongju, Korea: ACM, Mar. 2014, pp. 1247–1254.
- Nuno Oliveira, Alexandra Silva, and Luís S. Barbosa. "$\mathsf{IMC}_{\mathsf{Reo}}$: interactive Markov chains for stochastic Reo". In: *Journal of Internet Services and Information Security* 5.1 (Feb. 2015), pp. 3–28.

4.1 $\mathsf{IMC}_{\mathsf{Reo}}$

This section introduces $\mathsf{IMC}_{\mathsf{Reo}}$, a semantic model for stochastic Reo, based on the IMC formalism (*c.f.,* Section 2.2). In a nutshell, $\mathsf{IMC}_{\mathsf{Reo}}$ is an instance of a classical IMC with a different (structured) set of states and labels. Its composition builds

on the usual parallel composition for IMC, but discharges, through a synchronisation operator, the transitions that are not compliant with Reo semantics. These modifications to the standard definition are imposed so that the behaviour of Reo is correctly captured in the model.

Before diving into the *nitty-gritty* of $\mathsf{IMC}_{\mathsf{Reo}}$, some remarks on what transitions and states represent in this context, are in order to build intuition.

As expected, states capture the possible behaviour of a connector: data arrivals and data flowing through ports. Consider sets $\mathcal{N}$ and $\mathfrak{Q}$ as a set of port names, and a set of internal state names, respectively. Each state in $\mathsf{IMC}_{\mathsf{Reo}}$ is a triple (R, T, Q), where $R, T \in 2^{\mathcal{N}}$ denote sets of ports/nodes with, respectively, pending requests and data being transmitted; and $Q \in \mathfrak{Q}$ is an internal state identifier. The latter is used to distinguish between control states in state-based connectors. For example, in the fifo channel it may indicate whether the buffer is empty or full, by taking $\mathfrak{Q} = \{\mathsf{empty}, \mathsf{full}\}$. The selectors for each of the three components of an $\mathsf{IMC}_{\mathsf{Reo}}$ state s are denoted as R_s, T_s and Q_s, respectively.

Markovian transitions are labelled by $\gamma \in \mathbb{R}^+$. The negative exponential distribution parameter, γ, encodes, in each case, the connector processing delays and the rates of data arrival at its ports.

Interactive transitions, on their turn, are labelled with a set F of ports which, when firing, allow data to flow through them. Such ports correspond to the set of actions observable at the relevant $\mathsf{IMC}_{\mathsf{Reo}}$ state. In the sequel, this set is referred to as *actions*, for simplicity. The decision to take sets of actions (rather than a single action) to label interactive transitions is crucial to correctly capture (atomic) synchrony in the semantics of Reo. In fact, ports firing synchronously to enable data flow are the rule rather than the exception in Reo.

Intuition being built, $\mathsf{IMC}_{\mathsf{Reo}}$ can now be formally introduced.

Definition 4.1 ($\mathsf{IMC}_{\mathsf{Reo}}$)**.** *An* $\mathsf{IMC}_{\mathsf{Reo}}$ *is a tuple (S, Act,* $\dashrightarrow$*,* $\longrightarrow$*, s), where* $S \subseteq Act \times Act \times \mathfrak{Q}$ *is a nonempty set of states;* $Act \subseteq 2^{\mathcal{N}}$ *is a set of actions (the alphabet of the* $\mathsf{IMC}_{\mathsf{Reo}}$*);* $\dashrightarrow \subseteq S \times Act \times S$ *is the interactive transition relation;* $\longrightarrow \subseteq S \times \mathbb{R}^+ \times S$ *is the Markovian transition relation; and* $s \in S$ *is the initial state.*

Markovian transitions (s, γ, s') are written as $s \xrightarrow{\gamma} s'$; whereas notation $s \dashrightarrow^{a_1 a_2 \ldots} s'$ is used for interactive transitions $(s, \{a_1, a_2, ...\}, s')$. An interactive transition with an empty set of actions is said to be unobservable and is denoted by $s \dashrightarrow^{\tau} s'$.

States of the form $(R, \emptyset, Q)$ are referred to as *request* states and depicted as $\boxed{R}_Q$; states of the form $(\emptyset, T, Q)$ are referred to as *transmission* states and depicted as $\{T\}_Q$; states of the form (R, T, Q) are called *mixed* states and are depicted as

$\boxed{R}$ $\{T\}_Q$; finally, states of the form $(\emptyset, \emptyset, Q)$ are represented as $\emptyset_Q$ and denote the absence of both requests and data transmissions. For all representations, the buffer qualifier Q may be omitted, whenever clear from the context.

Figure 4.1 depicts the IMC$_{\text{Reo}}$ models corresponding to the basic stochastic Reo channels. To simplify the picture, transition overlapping is generally avoided by the graphical replication of states suitably annotated with a dashed circle.

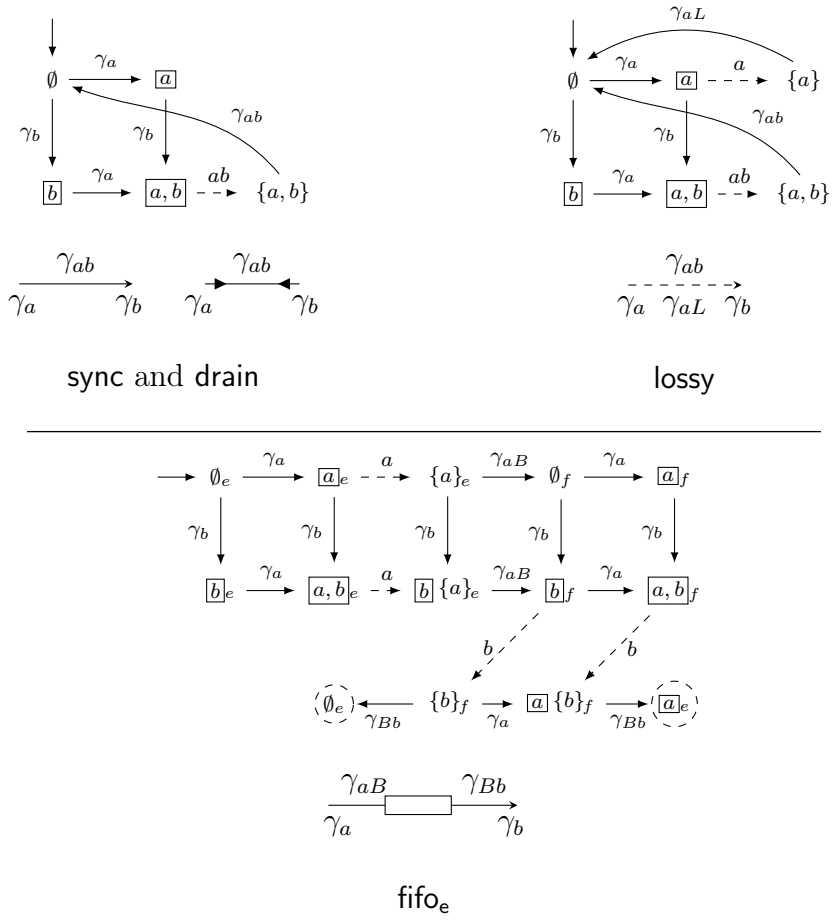


Figure 4.1: IMC for the basic stochastic Reo channels.

The IMC$_{\text{Reo}}$ model of a stochastic sync channel is interpreted as follows: initially, no requests are pending neither in port a nor in port b. Request arrive at port a (respectively, b) at rate γ_a (respectively, γ_b). The channel *blocks* until a request arrives to the other port. When state $\boxed{a,b}$ is reached, representing a configuration in which both ports have pending requests, then both eventually fire. That is, actions a and b are activated simultaneously. At this moment, the channel starts transmitting data between a and b and evolves back to the initial state with a processing delay rate of γ_{ab}. For a drain the interpretation is similar. The IMC$_{\text{Reo}}$

model of a stochastic lossy channel is also similar. However it exhibits two additional transitions to model the possibility of losing data: at state $\boxed{a}$, port a may fire, because there is no pending request at port b. When such is the case, the channel evolves back to the initial state after a delay of discarding data. State $\boxed{a}$ captures the context-dependent behaviour characteristic of this channel. Finally, the $\mathsf{fifo_e}$ stochastic channel differs from the formers by introducing an internal state. Notice how pending requests at port a automatically fire when the *buffer* is empty (states $\boxed{a}_e$ and $\boxed{a,b}_e$), and requests at port b block until it is full (states $\boxed{a,b}_e$ and $\boxed{b}\,\{a\}_e$). Also, notice that, to maintain consistency, the internal state of this channel only changes after Markovian transitions representing processing delays succeed. Actually, this is the rule in $\mathsf{IMC_{Reo}}$ models.

4.2 $\mathsf{IMC_{Reo}}$ composition

Reo connectors are composed through the aggregation of interface nodes. In this section this mechanism is formalised as $\mathsf{IMC_{Reo}}$ combinators, for which composition properties are proved.

4.2.1 Parallel composition

The starting point is an adaptation of the usual definition of IMC parallel composition [143], explicitly dealing with the restructured actions in the labels of interactive transitions.

Definition 4.2 (Parallel Composition)**.** *Let* $I = (S_I, Act_I, \dashrightarrow_I, \longrightarrow_I, s_i)$ *and* $J = (S_J, Act_J, \dashrightarrow_J, \longrightarrow_J, s_j)$ *be two* $\mathsf{IMC_{Reo}}$ *models. The parallel composition of* I *and* J *with respect to a set* $M \subseteq \mathcal{N}$ *is defined as*

$$I \;\|_M\; J = (S, Act, \dashrightarrow, \longrightarrow, (s_i, s_j))$$

where $S = S_I \times S_J$*,* $Act = Act_I \cup Act_J$*, and* $\dashrightarrow$ *and* $\longrightarrow$ *are the smallest relations satisfying*

1. $$\frac{i_1 \overset{A_I}{\dashrightarrow}_I i_2 \quad A_I \cap M = \emptyset}{(i_1, j) \overset{A_I}{\dashrightarrow} (i_2, j),\ \text{for } j \in S_J}$$

2. $$\frac{j_1 \overset{A_J}{\dashrightarrow}_J j_2 \quad A_J \cap M = \emptyset}{(i, j_1) \overset{A_J}{\dashrightarrow} (i, j_2),\ \text{for } i \in S_I}$$

3. $$\frac{i_1 \overset{A_I}{\dashrightarrow}_I i_2 \quad j_1 \overset{A_J}{\dashrightarrow}_J j_2 \quad (A_I \cap A_J) \subseteq M \quad A_I, A_J \neq \emptyset}{(i_1, j_1) \overset{A_I \cup A_J}{\dashrightarrow} (i_2, j_2)}$$

$$4.\quad \frac{i_1 \xrightarrow{\gamma}_I i_2}{(i_1, j) \xrightarrow{\gamma} (i_2, j),\ \text{for } j \in S_J} \qquad 5.\quad \frac{j_1 \xrightarrow{\gamma}_J j_2}{(i, j_1) \xrightarrow{\gamma} (i, j_2),\ \text{for } i \in S_I}$$

The first three clauses in Definition 4.2 deal with interactive transitions: the first two tackle the independent evolution of each connector; the third one addresses their (synchronous) joint evolution. Clauses 4 and 5 deal with Markovian transitions which are always interleaved.

Notation

$i|j$ is used henceforth to graphically represent a pair of states (i, j).

Example 4.1 Figure 4.2 depicts a fragment of the IMC resulting from the parallel composition of a lossy and a fifo_e channel with respect to set $M = \{b\}$ of shared nodes. The complete IMC has 72 states and 182 transitions.

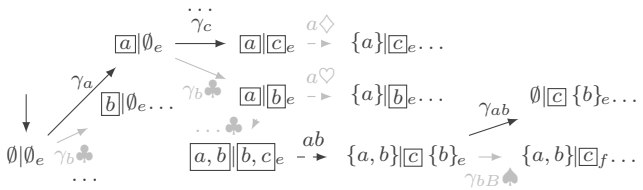


Figure 4.2: Fragment of the parallel composition of a lossy and a fifo_e channel. ✠

Note that clause 3 of Definition 4.2 does not capture the joint evolution of τ-transitions (which is precluded by condition $A_I, A_J \neq \emptyset$). Actually, this is just a design decision in accordance to the homologous definition in [143]. Should this condition be ignored and the model will allow for the joint evolution of two or more τ-transitions. In practice, this would result in an extra internal transition. The resulting $\mathsf{IMC}_{\mathsf{Reo}}$ chains would be equivalent (at least in a weaker way) due to the maximal progress assumption.

To illustrate both alternatives, consider the composition of two $\mathsf{2fifo}_\mathsf{e}$ connectors. A fragment of the $\mathsf{IMC}_{\mathsf{Reo}}$ model for each $\mathsf{2fifo}_\mathsf{e}$ is depicted in Figure 4.9. Focus on the joint state $\emptyset_f|\emptyset_e$, assuming, for simplicity, their piecewise union. Figure 4.3 shows the configuration of the two channels, at that state, alongside with a fragment of the corresponding $\mathsf{IMC}_{\mathsf{Reo}}$ models, which are to be composed in parallel for the set of nodes $\{a, c\}$.

Applying the parallel composition, as in Definition 4.2, will create an $\mathsf{IMC}_{\mathsf{Reo}}$, a fragment of which is depicted in solid black in Figure 4.4. State $\emptyset_{fe}|\emptyset_{fe}$ has four Markovian transitions (from the application of clauses 4 and 5) and two τ-transitions (from clauses 1 and 2). If, otherwise, the alternative version of clause 3 is assumed,

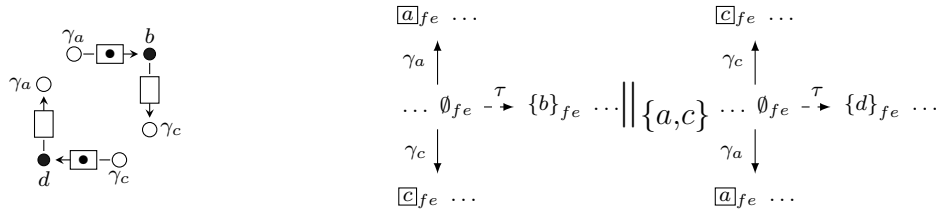


Figure 4.3: Composing two 2fifo$_e$ connectors.

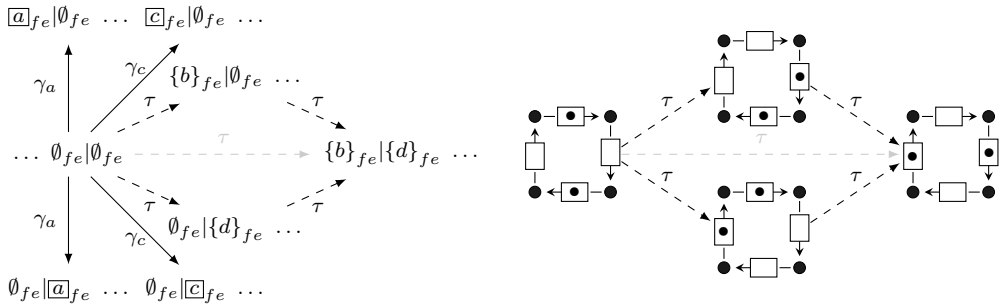


Figure 4.4: Fragment of the parallel composition of two 2fifo$_e$ (left) and the corresponding circuit evolution (right).

i.e., without the $A_I, A_J \neq \emptyset$ condition, the extra transition $\emptyset_{fe}|\emptyset_{fe} \xrightarrow{\tau} \{b\}_{fe}|\{d\}_{fe}$, depicted in light grey in Figure 4.4 would be possible.

4.2.2 Synchronisation

The definition of parallel composition has to be adjusted to correctly capture the intended semantics for channel composition in Reo. The mismatch concerns Reo mixed nodes which are not supposed to actively block behaviour, but rather acting like *self-contained pumping stations*, in Arbab's metaphor [16]. Failing to take this into account generates unwanted behaviour, making the semantics unsound.

Consider again Example 4.1. The presence of transition $\boxed{a}|\boxed{b}_e \xrightarrow{a} \{a\}|\boxed{b}_e$ shows that in the composed connector a data item arriving to port a could be lost, even when the buffer is empty. This possibility violates the mixed node assumption in the Reo *rationale*, in the sense that transitions cannot occur from a state which *actively blocks* a mixed node. The following definition captures such a notion of *active blocking*.

Definition 4.3 (Active Block)**.** *Let* $(S_1 \times S_2, Act, \dashrightarrow, \longrightarrow, s)$ *be an* $\mathsf{IMC}_{\mathsf{Reo}}$ *model over a composite state space, and* $M \subseteq \mathcal{N}$*. A state* (i, j) actively blocks M *(denoted* $(i, j) \rhd M$*) if there exists a transition* $(i, j) \xrightarrow{X} (_, _)$ *with* $X \cap M = \emptyset$*,* $R_i \cap M = \emptyset$ *and* $R_j \cap M \neq \emptyset$*, or vice-versa.*

Notation

$i \not\triangleright M$ is used hereafter to denote that state i does not actively block the set of nodes M.

Clearly, in Example 4.1, state $\boxed{a}|\boxed{b}_e$ actively blocks the (singleton) set of nodes $M = \{b\}$ in the composition of the two connectors.

Defined that is such a notion, it is now possible to introduce a synchronisation operation to prune this sort of unwanted transitions.

Notation

$i\restriction_M$ identifies a restricted state $i = (R, T, Q)$ in which all ports in a set M are removed from the set of active requests in i (*i.e.,* $i\restriction_M = (R \setminus M, T, Q)$). This extends to composite states (i, j) in the obvious way: $(i, j)\restriction_M = (i\restriction_M, j\restriction_M)$.

Definition 4.4 (Synchronisation)**.** *Let $I = (S_1 \times S_2, Act, \dashrightarrow, \longrightarrow, s)$ be an $\mathsf{IMC}_{\mathsf{Reo}}$ model over a composite state space, and $M \subseteq \mathcal{N}$. The synchronisation of I with respect to M is given by*

$$\partial_M I = (S_M, Act \setminus M, \dashrightarrow_M, \longrightarrow_M, s)$$

where $S_M = \{(i, j)\restriction_M |\ (i, j) \in S_1 \times S_2\}$ and $\dashrightarrow_M$ and $\longrightarrow_M$ are the smallest relations satisfying, respectively, conditions 1 and 2 below:

$$1.\quad \frac{(i,j) \dashrightarrow^{X} (i',j') \quad (i,j) \not\triangleright M}{(i,j)\restriction_M \dashrightarrow^{X \setminus M}_M (i',j')\restriction_M} \qquad 2.\quad \frac{(i,j) \xrightarrow{\gamma} (i',j') \quad (R_{i'} \cup R_{j'}) \cap M = \emptyset}{(i,j)\restriction_M \xrightarrow{\gamma}_M (i',j')\restriction_M}$$

In Figure 4.2, interactive transitions that need to be deleted because their source states actively block a set of mixed nodes, are labeled with ♡. One such transition is $\boxed{a}|\boxed{b}_e \dashrightarrow^{a} \{a\}|\boxed{b}_e$, where, as explained above, for $M = \{b\}$, $X = \{a\}$, clearly $X \cap M = \emptyset$, $R_{\boxed{a}} \cap M = \{a\} \cap M = \emptyset$, and $R_{\boxed{b}_e} \cap M$ holds $\{b\} \cap M \neq \emptyset$. That is, port b, which was expected to automatically fire, is blocked and only port a fires. Markovian transitions going to states with requests in the mixed nodes are labeled with ♣. An example is transition $\emptyset|\emptyset_e \xrightarrow{\gamma_b} \boxed{b}|\emptyset_e$, where for $M = \{b\}$, $(R_{\boxed{b}} \cup R_{\emptyset_e}) \cap M \neq \emptyset$.

Composition in $\mathsf{IMC}_{\mathsf{Reo}}$ can finally be defined resorting to both parallel composition and synchronisation combinators,

Definition 4.5. *The composition of $\mathsf{IMC}_{\mathsf{Reo}}$ models I and J, with respect to $M \subseteq \mathcal{N}$, is given by*

$$\partial_M(I_1 \ \|_M \ I_2)$$

This two-step composition approach is not a novelty in the definition of composition operations in the context of Reo. The one introduced above is, in fact, very much in the same spirit of the one defined for Reo automata [54].

4.2.3 Properties

This section discusses a number of properties of the composition operators with respect to (strong) bisimilarity. The latter is introduced in Definition 4.6 as a slight extension of standard IMC bisimilarity [143] to cope with labels as sets.

Definition 4.6 (Strong bisimulation)**.** *Let $I = (S, Act, \dashrightarrow, \longrightarrow, s)$ be an $\mathsf{IMC}_{\mathsf{Reo}}$ model. An equivalence relation $\mathcal{R} \subseteq S \times S$ is a strong bisimulation if for any $(i, j) \in \mathcal{R}$ and equivalence class $C \in S/\mathcal{R}$ the following holds:*

1. *for each $A \subseteq Act$ and some $i' \in C$, if $i \overset{A}{\dashrightarrow} i'$ then there exists a state $j' \in C$ such that $j \overset{A}{\dashrightarrow} j'$, and $(i', j') \in \mathcal{R}$, and vice-versa;*

2. *if i is a* stable *state then $\mathbf{E}(i, C) = \mathbf{E}(j, C)$;*

Recall, from Chapter 2, that $\mathbf{E}(i, Q) = \Sigma_{i' \in Q}[\gamma \mid i \xrightarrow{\gamma} i']$.

Two $\mathsf{IMC}_{\mathsf{Reo}}$ models I and J with disjoint state spaces S_I and S_J and initial states i and j, respectively, are *strong bisimilar* (standardly denoted by $I \sim J$) if there exists a strong bisimulation $\mathcal{R}$ on $S_I \cup S_J$ such that $(i, j) \in \mathcal{R}$. Strong bisimilarity is the largest strong bisimulation, which is, as expected, an equivalence relation.

Theorem 4.1. *Let I, I_1, I_2 and I_3 be $\mathsf{IMC}_{\mathsf{Reo}}$ models, where Act_1 is the alphabet of I_1 and $M, N \subseteq \mathcal{N}$. The following holds:*

1. $\partial_M(I_1 \ \|_N I_2) \sim I_1 \ \|_N \partial_M I_2$, *if* $Act_1 \cap M = \emptyset$

2. $\partial_N(\partial_M I) = \partial_M(\partial_N I) = \partial_{M \cup N} I$

3. $(I_1 \|_M I_2) \|_M I_3 \sim I_1 \|_M (I_2 \|_M I_3)$

4. $I_1 \|_M I_2 \sim I_2 \|_M I_1$

Proof. For 1, note that interactive transitions in $\partial_M(I_1 \ \|_N I_2)$ can be of one of the following forms:

- a M restriction of a transition $(i, j) \overset{X}{\dashrightarrow} (i', j)$, such that $i \overset{X}{\dashrightarrow} i'$ exists in I_1, $X \cap N = \emptyset$ and $(i, j) \not\triangleright M$.
- a M restriction of a transition $(i, j) \overset{X}{\dashrightarrow} (i, j)$, such that $j \overset{X}{\dashrightarrow} j'$ exists in I_2, $X \cap N = \emptyset$ and $(i, j) \not\triangleright M$.

- a M restriction of a transition $(i,j) \stackrel{X_1 \cup X_2}{-\!\!\rightarrow} (i',j')$, such that $i \stackrel{X_1}{-\!\!\rightarrow} i'$ and $j \stackrel{X_2}{-\!\!\rightarrow} j'$ exists in I_1 and I_2, respectively, $X_1 \cap X_2 \subseteq N$ and $(i,j) \not\triangleright M$.

All such transitions are possible for $I_1 \ \|_N \ \partial_M I_2$, but it remains to show that in neither of them can the origin state block M. Consider the first type of transitions (the argument for the others are similar). Note that there are only two ways of state (i,j) being able to block M. The first possibility is $X \cap M = \emptyset$, which is the case: $R_i \cap M = \emptyset$ because $Act_1 \cap M = \emptyset$, and $R_j \cap M \neq \emptyset$, which is false because all labels in any transitions in $\partial_M I_2$ do not contain elements of M. The second alternative requires $X \cap M = \emptyset$ again, and $R_j \cap M = \emptyset$, which holds as discussed above, but also $R_i \cap M \neq \emptyset$ which is false by hypothesis. Therefore, allowed transitions in $I_1 \ \|_N \ \partial_M I_2$ do not block M. In principle, it could exhibit more interactive transitions than $\partial_M(I_1 \ \|_N \ I_2)$, namely transitions in I_1 with elements in M. This cannot be the case, however, because of the assumption $Act_1 \cap M = \emptyset$.

In turn, Markovian transitions in $\partial_M(I_1 \ \|_N \ I_2)$ can be of one of the following forms:

- a M restriction of a transition $(i,j) \xrightarrow{\gamma} (i',j)$, such that $i \xrightarrow{\gamma} i'$ exists in I_1 and $(R_{i'} \cup R_j) \cap M = \emptyset$.
- a M restriction of a transition $(i,j) \xrightarrow{\gamma} (i,j')$, such that $j \xrightarrow{\gamma} j'$ exists in I_2 and $(R_i \cup R_{j'}) \cap M = \emptyset$.

Both cases also occur in $I_1 \ \|_N \ \partial_M I_2$; it remains to show that the request sets of the target states of these transitions do not have elements in M. In fact, assumption $Act_1 \cap M = \emptyset$ implies that Markovian transitions in I_1 cannot have target states i' with $R_{i'} \cap M \neq \emptyset$; and, by definition, for all the target states j' of Markovian transitions in $\partial_M I_2$, $R_{j'} \cap M = \emptyset$ holds. Hence, Markovian transitions in $I_1 \ \|_N \ \partial_M I_2$ are the same as in $\partial_M(I_1 \ \|_N \ I_2)$, which preserves the cumulation of rates.

For 2, it is enough to note that, in both sides of the bisimilarity equation, interactive and Markovian transitions are restricted to $M \cup N$, and also that interactive transitions in I whose origin state does not actively block neither M nor N do the same for $M \cup N$.

Associativity and commutativity of $\|_M$ are easy to prove: for Markovian transitions strict interleaving applies and therefore the contribution of each chain is always preserved; for interactive transitions, when in presence of sharing, associativity and commutativity of $\cup$ entail 3 and 4, respectively. □

Theorem 4.2 (Substitutability for $\|_M$)**.** *Let I_1, I_2 and I_3 be* $\mathsf{IMC}_{\mathsf{Reo}}$ *models, and $M \subseteq \mathcal{N}$, such that $I_1 \sim I_3$. Then, $I_1 ||_M I_2 \sim I_3 ||_M I_2$.*

Proof. Every interactive transition of I_1 is exactly matched by an equally labelled transition in I_3, because $I_1 \sim I_3$. Therefore, the contribution of both I_1 and I_3 for interactive transitions of their parallel composition with I_2 is the same. For Markovian transitions $I_1 \sim I_3$ guarantees that for bisimilar states the cumulative rate in both chains is the same. In the parallel composition Markovian transitions of I_1 are interleaved with those of I_2. Once a state of I_1 is replaced by a bisimilar one, the cumulative rates are not affected, even if the number of Markovian transitions may differ. □

This result establishes substitutability for parallel composition with respect to a set of nodes M. A similar result for ∂_M holds only for a stricter notion of equivalence. Actually, synchronisation prunes a number of transitions which depend on the data requests active in each state. This entails the need to incorporate this sort of information in a suitable definition of behavioural equivalence.

Definition 4.7 (Behavioural equivalence)**.** *Two* $\mathsf{IMC}_{\mathsf{Reo}}$ *models I and J are* behaviourally equivalent*, denoted by $I \equiv J$, if they are strong bisimilar and for each pair (i, j) of related states, their sets of requests are equal: $R_i = R_j$.*

Clearly $\equiv \subseteq \sim$. With this coarser equivalence, a substitutability result for ∂_M can be established.

Theorem 4.3 (Substitutability for ∂_M)**.** *Let I_1 and I_2 be two* $\mathsf{IMC}_{\mathsf{Reo}}$ *models , and $M \subseteq \mathcal{N}$, such that $I_1 \equiv I_2$. Then, $\partial_M I_1 \equiv \partial_M I_2$.*

Proof. An interactive transition in $\partial_M I_1$ is the M restriction of a I_1 interactive transition, say $(i, j) \overset{X}{\dashrightarrow} (i', j')$ such that $(i, j) \not\triangleright M$. As $I_1 \sim I_2$ there exists a transition $(k, l) \overset{X}{\dashrightarrow} (k', l')$ in I_2 for bisimilar states (i, j) and (k, l). The extra condition in the definition of $\equiv$ further imposes that $R_{(i,j)} = R_{(k,l)}$, which, both being composed states, has to be stated piecewise, i.e., $R_i = R_k$ and $R_j = R_l$. But this means that if the original transition in $\partial_M I_1$ does not block M, this one in $\partial_M I_2$ has the same property. A similar argument applies to the analysis of Markovian transitions. Actually, for stable, bisimilar states (i, j) and (k, l) in I_1 and I_2, respectively, and each bisimilarity equivalence class C, $\Gamma((i, j), C) = \Gamma((k, l), C)$. This equality does not necessarily holds when comparing $\partial_M I_1$ and $\partial_M I_2$ because the definition of synchronisation considers only (M restrictions of) Markovian transitions $(i, j) \xrightarrow{\gamma} (i', j')$ such that $(R_{i'} \cup R_{j'}) \cap M = \emptyset$ and so, unless $R_{i'} = R_{k'}$ and $R_{j'} = R_{l'}$, one can not conclude that the corresponding cumulative rates $\Gamma((i', j'), C), \Gamma((k', l'), C)$ are equal, because different markovian transitions could have been pruned in $\partial_M I_1$ and $\partial_M I_2$. But such is the case, however, if $I_1 \equiv I_2$. □

4.2.4 Cleaning up unintended transitions

In general, when Reo channels are set in parallel within a connector, they evolve independently. However, when connected through their ends, data flows from one to the other, in sequence, and there is a clear intended flow *direction*. Up until now, $\mathsf{IMC}_{\mathsf{Reo}}$ has been refrained from explicitly modelling the difference between input and output ports, which are responsible for setting the intended flow direction. This may generate unintended transitions.

Consider Example 4.1 (Figure 4.2) once again. Node $\{a,b\}|\boxed{c}\,\{b\}_e$ evolves interleaved via γ_{ab} or γ_{bB} to the same state. This leads to an artificial configuration in which the buffer becomes full before data is transmitted through the lossy channel.

Another sort of unintended transitions arise when a channel is transmitting data from one port to another and requests arrive to those ports. This is undesirable, as ports are busy. A last sort of unintended transitions is related to the fact that nondeterminism may arise where it shall not exist.

These unintended transitions must be cleaned from the model so that it correctly captures the desired operational behaviour of Reo. The following definition formally presents such an operation.

Definition 4.8 (Cleaning)**.** *Let $M \subseteq \mathcal{N}$ and $I = (S, Act, \dashrightarrow, \longrightarrow, s)$ be an $\mathsf{IMC}_{\mathsf{Reo}}$ model. Suppose the existence of a relation $<$ on $\mathcal{N}$ such that $a < b$ when data flows from a to b, with $a, b \in \mathcal{N}$. The cleaning of I with respect to M, denoted $\mathcal{C}_M I$, corresponds to restricting $\partial_M I$ so that for all its Markovian transitions $i \xrightarrow{\gamma} f$*

1. *$R_f \cap AN(i) = \emptyset$, where $AN(i) = T_i \cup \{j \in \mathcal{N} \mid \exists_{k \in T_i}. j < k \vee k < j\}$;*
2. *$T_f \subset T_i$;*

hold, and all its interactive transitions $j \overset{X}{\dashrightarrow} k$ respect

3. $\neg\exists_{j \overset{Y}{\dashrightarrow} l \in \dashrightarrow} \cdot\ X = Y \wedge T_k \cap M = \emptyset \wedge T_l \cap M \neq \emptyset$.

Informally, an $\mathsf{IMC}_{\mathsf{Reo}}$ model failing to respect condition 1 allows requests on nodes that are actively transmitting data. Whenever condition 2 does not hold, the model fails to preserve transmission sequencing. Finally, if condition 3 fails, nondeterminism is being enforced where it should not exist. Condition 3 is a special case of active blocking resulting from the comparison of transitions; thereof it cannot be identified by Definition 4.3.

Illustration of conditions 2 and 3 can be found in Figure 4.2. Note that the transition marked with ♠ does not respect transmission sequencing: $\{a,b\}|\boxed{c}\{b\}_e \xrightarrow{\gamma_{bB}} \{a,b\}|\boxed{c}_f$. Actually $T_i = T_{\{a,b\}|\boxed{c}\{b\}_e} = \{a,b\}$ and $T_f = T_{\{a,b\}|\boxed{c}_f} = \{a,b\}$, so $T_f \not\subset T_i$.

In turn, the transition marked with $\diamondsuit$ enforces nondeterminism. Note that state $\boxed{a}\vert\boxed{c}$ is equal to $\boxed{a,b}\vert\boxed{b,c}$, modulo $\restriction_M$, for $M = \{b\}$. Therefore, after synchronisation the following transitions coexist: $\boxed{a}\vert\boxed{c} \xrightarrow{a} \{a\}\vert\boxed{c}$ and $\boxed{a}\vert\boxed{c} \xrightarrow{a} \{a,b\}\vert\boxed{c}\{b\}$. Nondeterminism is enforced because both transitions are equally labelled and $T_{\{a\}\vert\boxed{c}} \cap M = \emptyset$ and $T_{\{a,b\}\vert\boxed{c}\{b\}} \cap M \neq \emptyset$.

The composition illustrated in Figure 4.2 is revisited in Figure 4.5 after synchronisation and cleaning.

$$\longrightarrow \begin{array}{c}\emptyset\vert\emptyset_e \\ \vdots\end{array} \xrightarrow{\gamma_a} \begin{array}{c}\boxed{a}\vert\emptyset_e \\ \vdots\end{array} \xrightarrow{\gamma_c} \boxed{a}\vert\boxed{c}_e \xrightarrow{a} \{a,b\}\vert\boxed{c}\{b\}_e \xrightarrow{\gamma_{ab}} \emptyset\vert\boxed{c}\{b\}_e \ldots$$

Figure 4.5: Parallel composition of a lossy and $\mathsf{fifo_e}$ after synchronisation and cleaning.

A substitutability result for $\mathcal{C}_M$ can only be formulated in terms of a slightly coarser version of strong bisimilarity as follows.

Definition 4.9 (Time-independent strong bisimulation)**.** *Let $I = (S, Act, \dashrightarrow, \longrightarrow, s)$ be an $\mathsf{IMC_{Reo}}$ model. An equivalence relation $\mathcal{R} \subseteq S \times S$ is a time-independent strong bisimulation if for any $(i,j) \in \mathcal{R}$ and equivalence class $C \in S/\mathcal{R}$ the following holds:*

1. *for each $A \subseteq Act$ and some $i' \in C$, if $i \xrightarrow{*} \dashrightarrow^{A} \xrightarrow{*} i'$, then there exists a state $j' \in C$ such that $j \xrightarrow{*} \dashrightarrow^{A} \xrightarrow{*} j'$, and $(i',j') \in \mathcal{R}$, and vice-versa,*

where $\xrightarrow{*}$ represents a possibly empty sequence of Markovian transitions.

Two $\mathsf{IMC_{Reo}}$ models I and J with disjoint state spaces S_I and S_J and initial states i and j, respectively, are *time-independent strong bisimilar* (denoted $I \sim_{ti} J$) if there exists a time-independent strong bisimulation $\mathcal{R}$ on $S_I \cup S_J$ such that $(i,j) \in \mathcal{R}$.

Theorem 4.4 (Time-independent Equivalence)**.** *Let I and J be two $\mathsf{IMC_{Reo}}$ models, such that $I \sim J$. Then $I \sim_{ti} J$.*

Proof. From the structure of a $\mathsf{IMC_{Reo}}$ model, it is obvious that each interactive transition is of the form $i \xrightarrow{*} \dashrightarrow^{A} \xrightarrow{*} i'$, where i (respectively, i') is the target (respectively, the source) state of some interactive transition. The same goes to J: each interactive transition is of the form $j \xrightarrow{*} \dashrightarrow^{A} \xrightarrow{*} j'$, with j (respectively, j') being the target (respectively, the source) state of some interactive transition in J. Since $I \sim J$ (by assumption) then, i and j are bisimilar and so are i' and j'. This is exactly the case when $I \sim_{ti} J$. □

Theorem 4.5 (Cleaning Equivalence)**.** *Let $M \subseteq \mathcal{N}$ and I and J be two $\mathsf{IMC_{Reo}}$ models, such that $I \sim J$. Then $\mathcal{C}_M I \sim_{ti} \mathcal{C}_M J$*

Proof. From Theorem 4.4, $I \sim_{ti} J$. Therefore, removing Markovian transitions from I and J in the context of the cleaning operation does not affect the $\sim_{ti}$ relation. It remains to show that removing interactive transitions (that enforce nondeterminism) does not affect the relation as well.

In case that no interactive transition enforces nondeterminism in I (consequently there is also no such transitions in J, because $I \sim J$), then $I \sim_{ti} J$ holds trivially. In case that there is a transition $i \xrightarrow{A} i'$ enforcing nondeterminism in I, then there is also such a transition $j \xrightarrow{A} j'$ in J. They are both removed, in I and J, on cleaning. Possibly, there remain Markovian transitions that have also to be removed as their source states become unreachable after the mentioned interactive transitions are eliminated; this will not affect the relation (because $I \sim_{ti} J$). Possible interactive transitions removed in I for the same reason are also removed in J. This ensures $\mathcal{C}_M I \sim_{ti} \mathcal{C}_M J$. □

Note that $\mathcal{C}_M I \sim \mathcal{C}_M J$ when $I \sim J$, or even when $I \equiv J$, does not hold in general, as illustrated in the following example.

Example 4.2 Consider the two IMC$_{\text{Reo}}$ models, I and J, in Figure 4.6 that appear after their synchronisation over a set $M = \{b\}$. In this case, $I \sim J$.

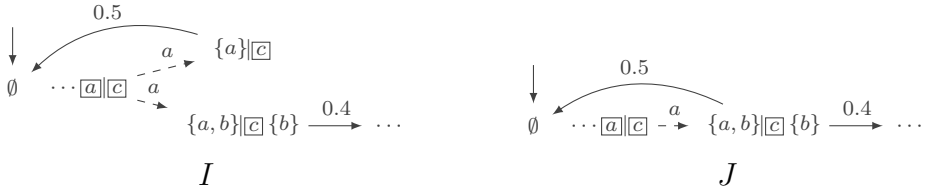


Figure 4.6: Fragments of two equivalent synchronised IMC$_{\text{Reo}}$ models

After cleaning, transitions $a|c \xrightarrow{a} \{a\}|c \xrightarrow{0.5} \emptyset$ in I will disappear. Clearly, states $\{a,b\}|c\,\{b\}$ (in both chains) will not remain bisimilar since their commutative rates become 0.4 in I and 0.9 in J. ✠

4.2.5 Composition idiosyncrasies

This section discusses some particularities of the IMC$_{\text{Reo}}$ composition operation, through a set of suitable examples.

Identity element

The IMC$_{\text{Reo}}$ composition has an *identity element*. The IMC$_{\text{Reo}}$ model of the sync channel plays that role. Figure 4.7 depicts the composition of a lossy and a sync

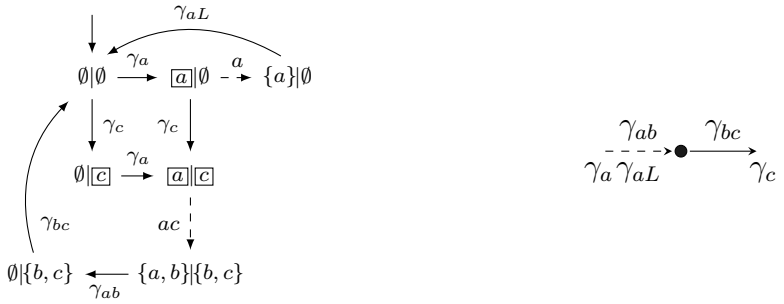


Figure 4.7: Composing a lossy and a sync channel.

channel. The result is an $\mathsf{IMC}_{\mathsf{Reo}}$ corresponding to a lossy channel with ports a and c and a processing delay rate $\phi(\gamma_{ab}, \gamma_{bc})$, which combines (via convolution) the exponential distributions modelled by γ_{ab} and γ_{bc} as an approximated exponential distribution.

Up to this rate, the sync channel behaves as the identity element for $\mathsf{IMC}_{\mathsf{Reo}}$ composition. This is an expected result since the sync plays the same role in Reo. This is confirmed when comparing lossy and lossysync disregarding time, via time-independent strong bisimilarity.

Context-awareness

$\mathsf{IMC}_{\mathsf{Reo}}$ is context-aware, and its composition operation preserves this desired property. The structure of the $\mathsf{IMC}_{\mathsf{Reo}}$ states provides the necessary information to make it possible.

Figure 4.8 depicts the composition of a lossy and a $\mathsf{fifo}_{\mathsf{e}}$ channel, *i.e.,* the lossyfifo connector first shown in Figure 2.4. Observe that data is not lost when the buffer is empty, as it may be the case in other semantic models for Reo unable to capture context-awareness (see discussion in [54]). Contrariwise, data is lost only when the buffer is full and there is a request at the input port.

Atomic dataflow behaviour

The IMC *maximal progression assumption* (inherited by $\mathsf{IMC}_{\mathsf{Reo}}$) makes possible the desired behaviour of atomic data flow. This means that when data is able to flow from a channel to another, the model immediately allows it. $\mathsf{IMC}_{\mathsf{Reo}}$ composition enables this feature via the synchronisation definition (*c.f.,* Definition 4.4).

Figure 4.9 shows a fragment of the composition of two $\mathsf{fifo}_{\mathsf{e}}$ channels (the complete model has 39 states and 75 transitions). Note that, when the first buffer is full and the second one is empty, data may flow immediately to the latter, freeing the former.

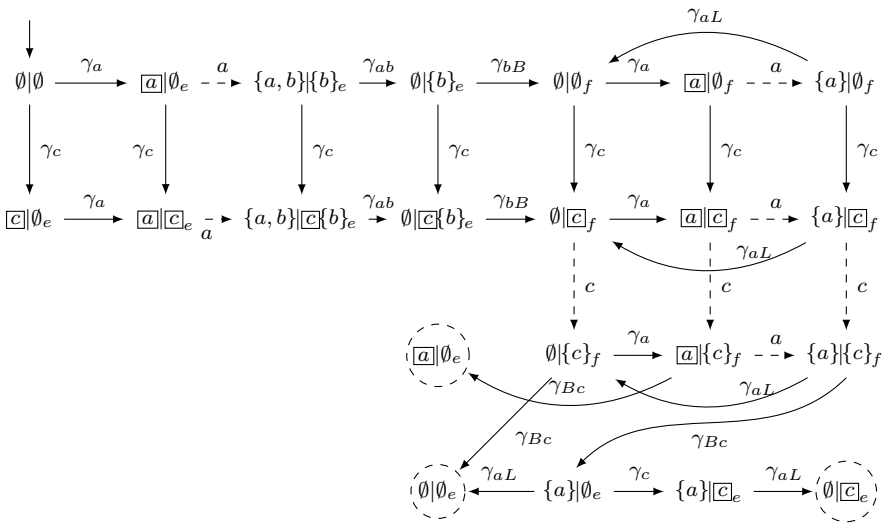


Figure 4.8: Composing a lossy and a fifo$_{\text{e}}$ channel.

The τ-transitions, which result from synchronisation operation, explicitly model this behaviour.

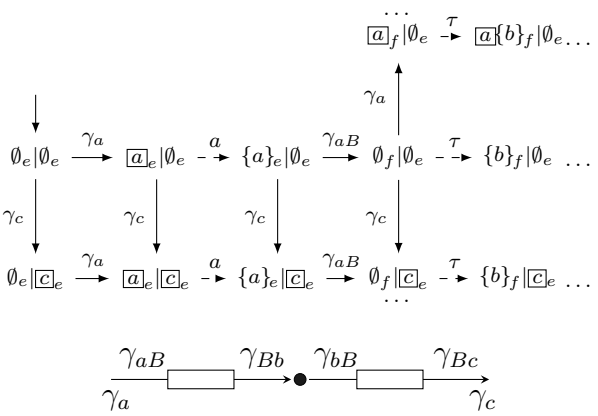


Figure 4.9: Composition of two fifo$_{\text{e}}$ channels (fragment).

Small primitive models

IMC$_{\text{Reo}}$ assumes small models for each Reo primitive. Consequently, complex models arise from their composition. This follows the Reo rationale: obtain complex connectors from the composition of simpler ones [14]. Despite of being a typical approach, it is recognised that the use of higher-level models could improve composition efficiency.

Consider the case of fifo channels. A two- or more-memory positions fifo channel is obtained from the composition of the same number of one-memory position fifo

channels. This introduces inefficiency to composition as the state space grows with the growth of memory positions.

An alternative solution would be to assume the family of *n-memory* positions fifo channels, and regard each of its element as traditional queues [61] with a bounded capacity. This would require, for instance, that states in $\mathsf{IMC}_{\mathsf{Reo}}$ were modelled as triples (R, T, n), where n stands for the number of free memory positions.

Without practical evidence it is hard to draw conclusions about the final state space size generated by both approaches. Intuition says that the difference would not be considerable. Differences could be relevant, though, during the composition process, since the $\mathsf{IMC}_{\mathsf{Reo}}$ model of each n-memory position fifo channel would be regarded as a primitive model.

Binary combination

$\mathsf{IMC}_{\mathsf{Reo}}$ composition is assumed to be performed on nodes that share, at most, two channel ends. When three or more channel ends have to be joined, the connector is refactored to comply to this assumption through the use of mergers, replicators and routers delivered as stochastic primitive *three-port* channels.

Figure 4.10 (first row) presents the (abstracted) stochastic version of the replicator, the merger and the router connectors. The corresponding $\mathsf{IMC}_{\mathsf{Reo}}$ models are in the second and third rows. These abstractions assume that the annotated processing delays correspond to the relevant composition of all delays involved in each relevant flow path. In a replicator there is only one such path which involves all the nodes of the connector. In a merger there are two: one involving nodes a, m and c[1], and another involving nodes b, m and c. Finally, in a router there are also two paths: one involving nodes a, j, l, x and b and the other a, j, m, x and c.

Figure 4.11 illustrates how a connector is transformed to comply to the assumption that each node connects, at most, two channel ends.

4.3 Distilled $\mathsf{IMC}_{\mathsf{Reo}}$

$\mathsf{IMC}_{\mathsf{Reo}}$, as introduced, suffers from some scalability problems. Its composition operation (which involves parallel composition, synchronisation and cleaning) generates a state space that remains considerably big even after minimisation via bisimulation. This has efficiency repercussions, not only on the final model analysis, but also (and perhaps more significantly) on the composition operation. Two main problems contribute to such a lack of efficiency. One is internal to the $\mathsf{IMC}_{\mathsf{Reo}}$ approach. It resides

[1]These node names refer to the connectors depicted in Figure 2.1.

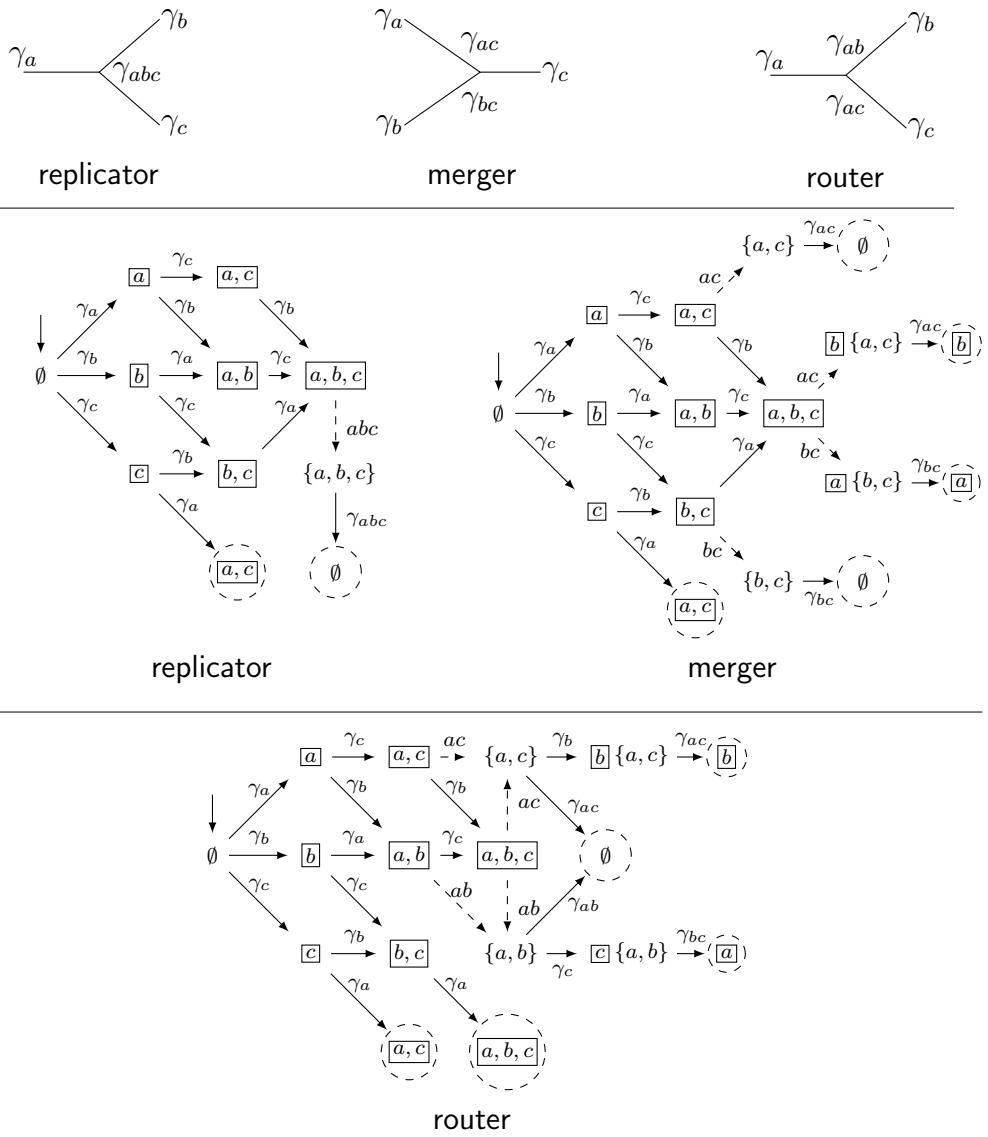


Figure 4.10: Stochastic three-port basic connectors and corresponding IMC$_{\text{Reo}}$ models.

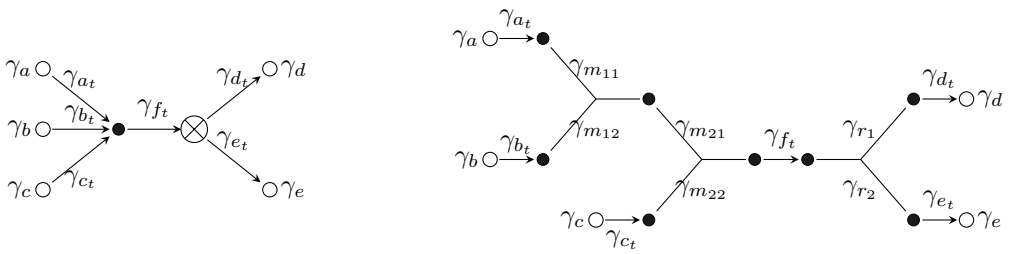


Figure 4.11: Connector refactoring for composition via two channel end nodes.

in the assumption that each Reo node connects at most two channel ends. Since this is not always the case in Reo specifications, a refactoring strategy is adopted as presented before. Three-port channels are assumed, whose associated $\mathsf{IMC}_{\mathsf{Reo}}$ model models the behaviour of replicator, merger and router nodes. These models subsume a considerable state space size. The other one is due to some design decisions on the stochastic Reo model itself.

The latter is discussed in detail in the sequel. In particular, improvements to stochastic Reo are proposed that will influence to look into $\mathsf{IMC}_{\mathsf{Reo}}$ in a more modular and scalable way.

4.3.1 The writer, the reader, the channel and the node

As an exogenous coordination model, Reo disregards services or components when it comes to specifying a coordination schema. It only assumes that such computation *loci* are bound to the ports of the connector, which receive IO impulses whenever communication is requested. Consequently, stochastic Reo inherits the same philosophy. But, does it? Not quite! In fact, stochastic Reo circuits are not completely exogenous in the sense of being concerned only with aspects related to coordination. Actually, they embody, in their request arrival rates: information that is inherently associated to the induced stochastic behaviour of the interacting services, whose communication stochastic Reo circuits coordinate. As expected, this hampers the reutilisation of stochastic Reo models, and introduces unnatural simplifications to make it compositional.

With effect, nodes in Reo and stochastic Reo are built on the *self-contained pumping station* assumption. As any other assumption, it makes sense to simplify the model. But, as any other simplification, it does not reflect the complete reality in the model. As a consequence, undesired analysis errors resulting from such simplification may occur, leading, ultimately, to defective systems.

To avoid such simplifications and mismatches to exogenous coordination philosophy, one proposes to consider the stochastic version of Reo as a two-phase component-based coordination model. It is two-phase because it shall be regarded before (referred to as the design phase) and after deployment (referred to as the deployment phase); it is component-based because it is constructed from four specific components: the writer, the reader, the channel and the node, as graphically presented in Figure 4.12.

The writer and the reader components, represented as black and white squares, respectively, are synchronous stochastic abstractions of the real-world services that are to be bound to the ports of the connector. They are annotated with a delay rate

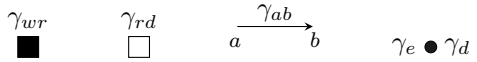


Figure 4.12: The essential components of stochastic Reo. From left to write: the writer, the reader, the channel and the node.

(γ_{wr} and γ_{rd}, respectively), that models the time between consecutive IO requests issued by such components.

The channel component, represented as the standard Reo channel notation (*c.f.,* Figure 2.1), inherits the usual behaviour of Reo channels. As expected, it also inherits the processing delay rate of stochastic Reo, which models the duration of point-to-point data transportation. Note that the request arrival rates are no more part of a channel model.

The node component, abstractly represented as a black dot, is a synchronous component that behaves like the replicator, the merger or the router (for last one, the usual notation $\otimes$ is used). Differently from the original version of stochastic Reo, in this approach, nodes are assumed to take time to enqueue and dequeue data. This behaviour is modelled by the delay rates γ_e and γ_d.

- Enqueueing data takes into account not only the time to process incoming data but also the time needed to select from which channel data will be read (in case of being a merger);

- Dequeuing data takes into account the time to write data in the channels; it further comprises the time to generate copies of the data to write (in case of being a replicator), and the time to decide to which channels it will write (in case of being a router).

This concretises a more realistic stochastic behaviour of nodes, as opposed to the *self-contained pumping station* behavioural assumption.

The design-phase stochastic Reo models originate from the composition of channel and node components. As desired, these models become reusable, compositional and respectful of the exogenous coordination philosophy.

In turn, the deployment-phase models are fixed for a given installation of compounding services. The writer and the reader components are bound to the interface ports of the connector. Such fixation erases the possibility of reusing the coordination model, but it provides an holistic vision of the coordinated system, and, consequently, it allows for analysing the system as a whole. This is, in fact, very close to the original stochastic Reo model, adding to it, however, a more realistic separation of concerns.

Figure 4.13 depicts a simple example of a lossyfifo connector in both the design- and the deployment-phase.

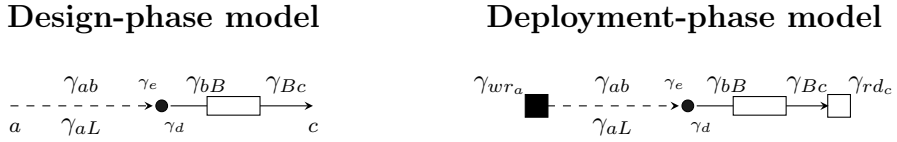


Figure 4.13: The two-pase component-based stochastic Reo model of a lossyfifo.

4.3.2 $\mathcal{D}$IMC$_{\text{Reo}}$: the distilled IMC$_{\text{Reo}}$

This component-based vision of the stochastic Reo model contributes to looking into IMC$_{\text{Reo}}$ in a distilled way, where each channel, node, writer and reader is regarded as an independent stochastic process that may (or may not) synchronise with the other elements. This distilled version of IMC$_{\text{Reo}}$ will be referred to, henceforth, as $\mathcal{D}$IMC$_{\text{Reo}}$. Essentially, $\mathcal{D}$IMC$_{\text{Reo}}$ is not semantically disruptive with IMC$_{\text{Reo}}$, as it will be remarked in the sequel.

The introduction of delays in nodes raise the need for two new sorts of states with specific semantics: the state where the node is enqueueing and the state where it is dequeueing data. A state in $\mathcal{D}$IMC$_{\text{Reo}}$ is fully characterised as (R, T, E, D, Q) with $E, D \in 2^{\mathcal{N}}$, where states of the form $(\emptyset, \emptyset, E, \emptyset, Q)$ are *enqueueing* states, meaning that the node is reading from the channel ends in set E; these states are represented as $\textcircled{E}_Q$. Likewise, states of the form $(\emptyset, \emptyset, \emptyset, D, Q)$ are *dequeueing* states, meaning that the node is writing to the channel ends in set D; these states are represented as $\textcircled{\scriptsize D}_Q$. All the other state forms are represented as previously.

The formal definition of $\mathcal{D}$IMC$_{\text{Reo}}$ is precisely the same as the one presented in Definition 4.1. Consequently, definitions for bisimulation (Definition 4.6) and time-independent bisimulation (Definition 4.9) considered for IMC$_{\text{Reo}}$ also hold for $\mathcal{D}$IMC$_{\text{Reo}}$.

The following paragraphs informally present the $\mathcal{D}$IMC$_{\text{Reo}}$ model of each considered component.

Channels

Consider the $\mathcal{D}$IMC$_{\text{Reo}}$ models for the basic Reo channels, as depicted in Figure 4.14. These models are simply obtained from their counterpart IMC$_{\text{Reo}}$ models by disregarding the environment information.

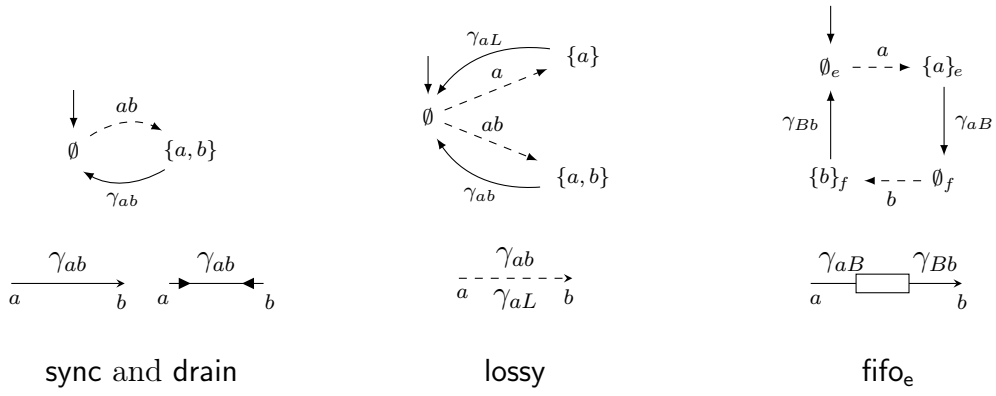


Figure 4.14: The $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ for the basic stochastic Reo channels

Note how simple the models become. When compared to those presented in Figure 4.1, a significant reduction is visible in their state space: $\frac{2}{5}$ for the sync channel, $\frac{1}{2}$ for the lossy channel and $\frac{1}{4}$ for the $\mathsf{fifo}_{\mathsf{e}}$ channel.

The attentive reader will notice that $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ does not entirely capture the semantics of the stochastic Reo channels. It fails to fully capture, for instance, the context-awareness property (*c.f.,* lossy $\mathsf{IMC}_{\mathsf{Reo}}$ model in Figure 4.14). This shall not come as a surprise, however, since the contextual (*i.e.,* environment) information was relegated to another component. Clearly, these are design-phase models, and as such, one shall not require that they comprise such holistic behaviour. Consequently, only the deployment-phase models will capture the semantics of stochastic Reo channels as previously introduced.

Readers and writers

To obtain deployment-phase models it is necessary to compose design-phase models with the environment information, *i.e.,* the reader and the writer components. Observationally, these readers and writers would behave similarly: they issue IO requests by publishing the intention to write (respectively, read) data; then they will block until synchronising with the connector ports. Thus, one single $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ model is enough to model such behaviour. Figure 4.15 depicts it.

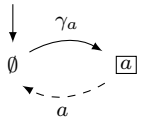


Figure 4.15: The $\mathsf{IMC}_{\mathsf{Reo}}$ for the reader and writer components

A reader is bound to an output port while a writer is bound to an input port. This is how readers and writers are distinguished. The composition of these components with one channel will result in a $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ modelling the semantics of stochastic Reo channels (and consequently, connectors). This will become clear further in Section 4.3.3.

Nodes

Reo defines three distinct types of nodes: replicator, merger and router. Consider Figure 4.16, where six different Reo node configurations are presented based on that three basic node types.

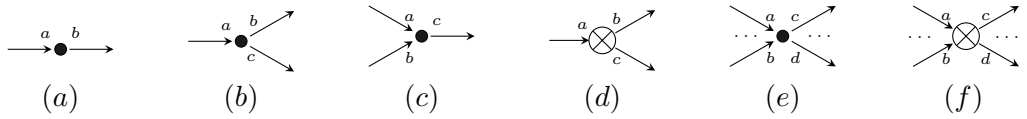


Figure 4.16: Different Reo node configurations: (a) simple node; (b) replicator node; (c) merger node; (d) router node (e) merger-replicator node and (f) merger-router node.

Note that node configurations (a) to (c) are special cases of (e): these nodes select one incoming channel to read data from, and then copy and write the data into all the outgoing channels. In turn, node configuration (d) is a special case of (f): these nodes select one incoming channel to read from, and then route the data to one of the outgoing channels. Nodes (e) and (f) define, in fact, two families of nodes, referred henceforth as merger–replicator and merger–router, respectively. They are parametric on the number of incoming and outgoing channels and also on the delays for reading (enqueueing) and writing (dequeueing) data, if such delays are considered.

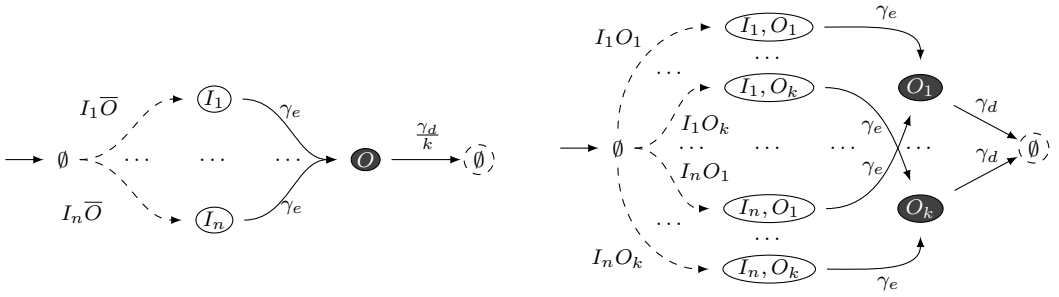
Consistently, the $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ for nodes would be generated from these two families, taking into account their parameters as follows:

$$\begin{aligned} \mathsf{merger–replicator} &: 2^{\mathcal{N}} \times 2^{\mathcal{N}} \times \mathbb{R}^{+} \times \mathbb{R}^{+} \\ \mathsf{merger–router} &: 2^{\mathcal{N}} \times 2^{\mathcal{N}} \times \mathbb{R}^{+} \times \mathbb{R}^{+}, \end{aligned}$$

where the first parameter is a set of output channel ends (which are the inputs of the node); the second is a set of input channel ends (which are the outputs of the node); the third models the time to select and read from one channel end, and finally, the fourth parameter models the time to copy, route and write data into one channel end. For consistency purposes, it is assumed that the first two parameters do not

include the empty set.

The parametric $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ model for both merger–replicator and merger–router family of nodes can now be presented. Figure 4.17 depicts such generic models. For simplicity, notation I_i represents the i^{th} element in set I and $\overline{O}$ represents the concatenation of all elements in set O. Moreover, it is assumed that the cardinality of sets I and O are, respectively, n and k.



merger–replicator $(I, O, \gamma_e, \gamma_d)$ merger–router $(I, O, \gamma_e, \gamma_d)$

Figure 4.17: $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ models for merger–replicator and merger–router nodes.

The merger–replicator node blocks until synchronising with one of the input channel ends and all the output channel ends. Once such synchronisation is achieved, it starts enqueueing data from the input channel end (delayed for some exponentially distributed time modelled by γ_e). Then, it dequeues data to all the output channel ends and returns to the initial blocked state. The delay time of a single dequeue operation is exponentially distributed with rate γ_d; since it performs k such operations, then the average delaying time is exponentially distributed with rate $\frac{\gamma_d}{k}$. The merger–router, in turn, blocks until synchronising with one of the input and one of the output channels ends. Once synchronisation is achieved, it enters in an enqueueing state and remains there for an exponentially distributed time modelled by rate γ_e. Then, it dequeues data to the selected output channel end at a rate γ_d, returning to the initial blocked state.

Example 4.3 Consider the node configurations in Figure 4.16. The corresponding $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ models are presented in Figure 4.18.

Each $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ model is generated by instantiation of the node families with specific parameters as follows:

(a) merger–replicator $(\{a\}, \{b\}, \gamma_e, \gamma_d)$ (b) merger–replicator $(\{a\}, \{b, c\}, \gamma_e, \gamma_d)$

(c) merger–replicator $(\{a, b\}, \{c\}, \gamma_e, \gamma_d)$ (d) merger–router $(\{a\}, \{b, c\}, \gamma_e, \gamma_d)$

(e) merger–replicator $(\{a, b\}, \{c, d\}, \gamma_e, \gamma_d)$ (f) merger–router $(\{a, b\}, \{c, d\}, \gamma_e, \gamma_d)$

✠

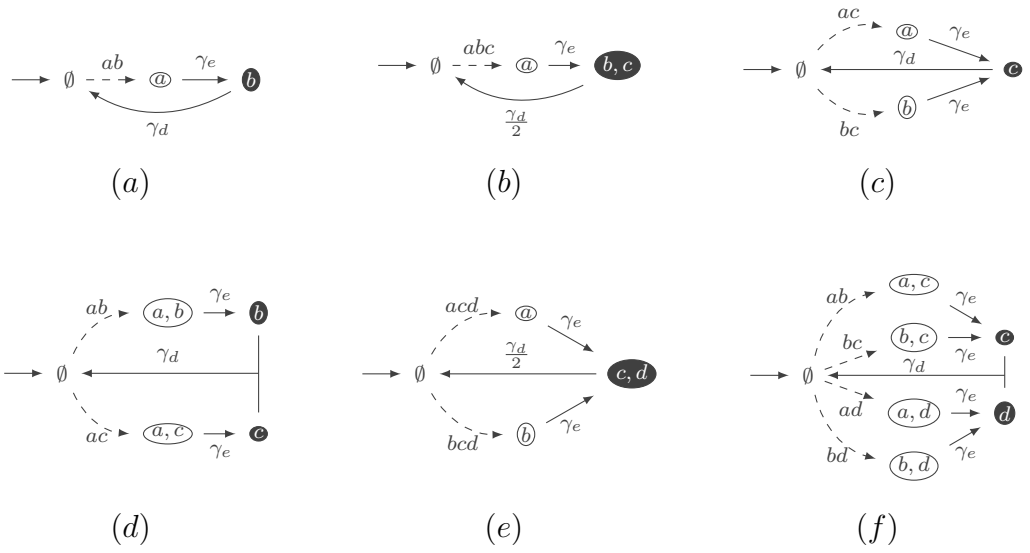


Figure 4.18: The $\mathsf{IMC}_{\mathsf{Reo}}$ models for the node configurations in Figure 4.16.

Note that the enqueueing states of the merger–router $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ are composed of both the selected input and output channel ends. The semantics of these states entails that data is being read from input and output ends, what is inaccurate. This imprecision is required to ensure the correct semantics of a whole connector (as will be discussed later on). If these states were only composed of the synchronised input end, each enqueueing state would have k outgoing Markovian transitions, one to each dequeueing state. This would enable to dequeue data into an end o_x when the selected output end was o_k, with $o_x \neq o_k$. Notably, this increases the state-space of such model; however, not as much as in the $\mathsf{IMC}_{\mathsf{Reo}}$'s approach for node modelling.

By disregarding enqueueing and dequeueing delays, these families of nodes are simplified into a single $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ model with transition space size of n and $n.k$ for merger–replicator and merger–router, respectively, corresponding only to interactive transitions. In fact, this is the result of minimising the $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ models of these families of nodes via the time-independent strong bisimulation relation.

4.3.3 Composition in $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$

Composition in $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ follows the same principles as in $\mathsf{IMC}_{\mathsf{Reo}}$. The operation builds on the parallel composition (Definition 4.2), synchronisation (Definition 4.4) and cleaning (Definition 4.8). This is, however, only the case for the design-phase $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ models, when nodes do not delay the composite. When nodes present delays, the cleaning part of the composition operations needs a revision.

Concretely, the need for a revision to the cleaning operation dues to the fact

that enqueueing/dequeueing data into/from a node shall respect a specific order. Concretely, (*i*) data is always enqueued into the node only after being transmitted to that node; (*ii*) data is always transmitted to any further node only after being dequeued from the current one and (*iii*) data is always enqueued before being dequeued (from the same node). In rigour, $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ requires that enqueueing and dequeueing transitions appear immediately one after the other, except in cases where other operations may occur in parallel; when such are the cases, transitions will appear interleaved.

Formally, the revised cleaning operation remains the same as shown in Definition 4.8, but with the second clause is further extended as follows:

Definition 4.10 ($\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ Clean up)**.** *Let $M \subseteq \mathcal{N}$ and $I = (S, Act, \dashrightarrow, \longrightarrow, s)$ be a $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$. Suppose further the existence of a relation $<$ on $\mathcal{N}$ such that $a < b$ when data flows from a to b, with $a, b \in \mathcal{N}$, with the lift to sets defined as $A < B$ iff $\exists_{a \in A} \cdot \forall_{b \in B} \cdot a < b$.*

The cleaning of I with respect to M, denoted $\mathcal{C}_M I$, corresponds to restricting $\partial_M I$ so that all its Markovian transitions $i \xrightarrow{\gamma} f$ respect:

(i) $R_f \cap AN(i) = \emptyset$, *where* $AN(i) = T_i \cup \{j \in \mathcal{N} \mid \exists_{k \in T_i} . j < k \vee k < j\}$;

(ii)
$$\begin{cases} T_i \setminus T_f < T_f & \text{if} \quad T_i \cap E_i \neq \emptyset \\ T_i = T_f & \text{if} \quad E_i < T_i \\ T_i = T_f & \text{if} \quad T_i \cap D_i \neq \emptyset \\ T_i \setminus T_f < T_f & \text{otherwise} \end{cases}$$

and all its interactive transitions $j \overset{X}{\dashrightarrow} k$ respect:

(iii) $\neg \exists_{j \overset{Y}{\dashrightarrow} l \in \dashrightarrow} \cdot X = Y \wedge T_k \cap M = \emptyset \wedge T_l \cap M \neq \emptyset$.

Example 4.4 This example shows the (design-phase) composition of a lossy channel with a sync channel, as in Figure 4.7, but further considering that enqueueing and dequeueing data in the mixed node is delayed with a rate γ_{enq} and γ_{deq}, respectively. Figure 4.19 depicts the composition of the two channels and the synchronising node.

The greyed-out transitions are those eliminated due to the cleaning operation. In particular, these transitions do not respect the expected sequencing feature. ✠

In order to obtain the deployment-phase model, an extra step is required that compose the design-phase model with the environment model. Such composition step is as before, but it bypasses the synchronisation operation. Formally,

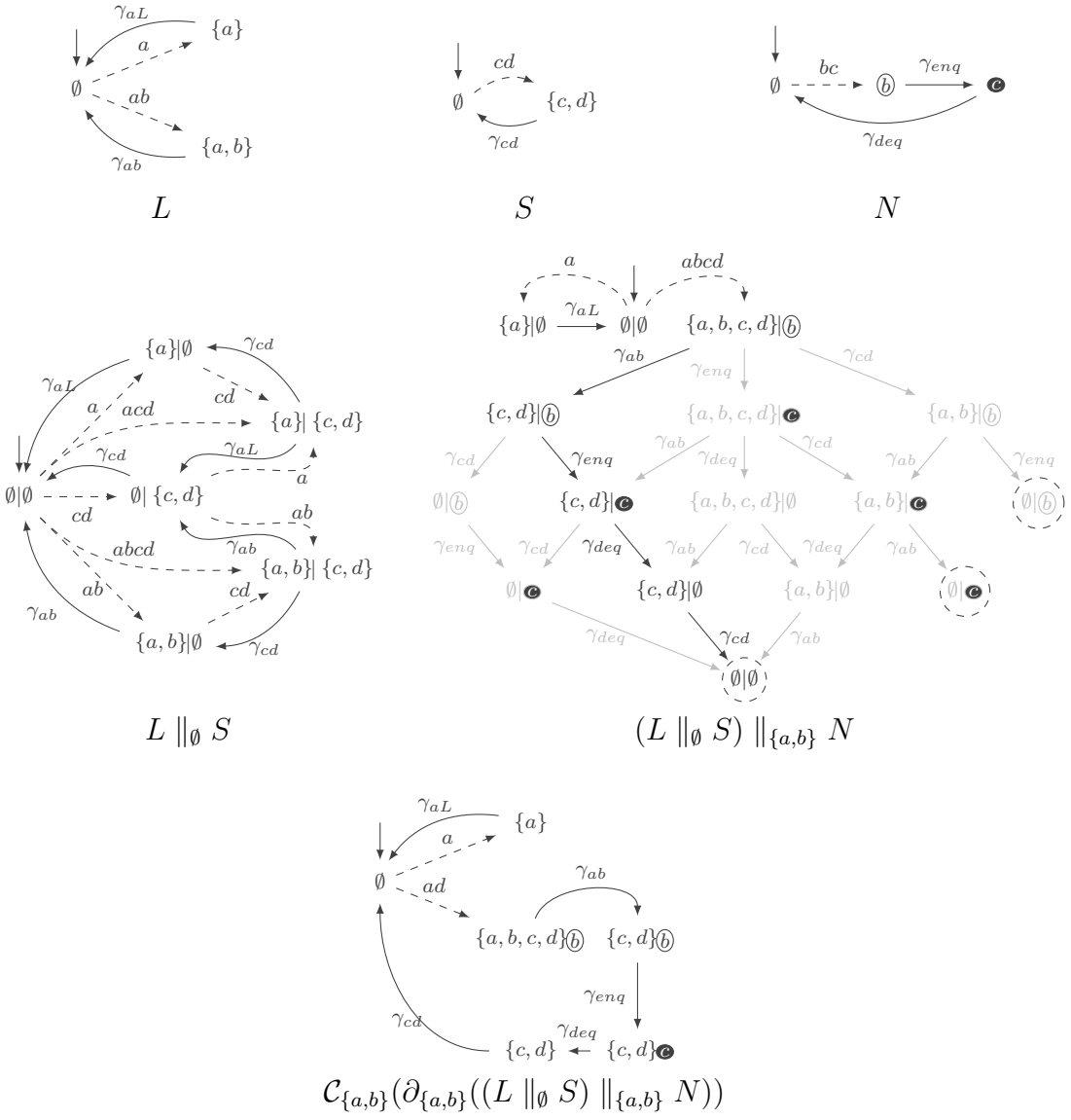


Figure 4.19: Construction of the design-phase $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ model for the lossysync connector with a delayed node.

Definition 4.11 (Deployment)**.** *Let I be a $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ model of a design-phase stochastic Reo connector, E be a set of $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ models representing all the relevant reader and writer components defining the environment for I, and finally $M \subseteq \mathcal{N}$. The deployment-phase model of I in environment E with respect to ports M, denoted $I_{\Delta_M^E}$ is computed as follows:*

$$\mathcal{C}_M(I \parallel_M E_{\parallel}),$$

where $E_{\parallel}$ is the $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ resulting from parallel composing all elements of E, referred to as the global environment model.

Example 4.5 Consider LS^{nd} and LS^{d} to be design-phase $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ models of two lossysync connectors. In the first, nodes are assumed without delays; in the second, nodes present delays. Consider also W and R to be $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ models of a writer and a reader, respectively. The deployment-phase model of both lossysync connectors with respect to $\{W, R\}_{\parallel} = W \parallel_{\emptyset} R$ is depicted in Figure 4.20.

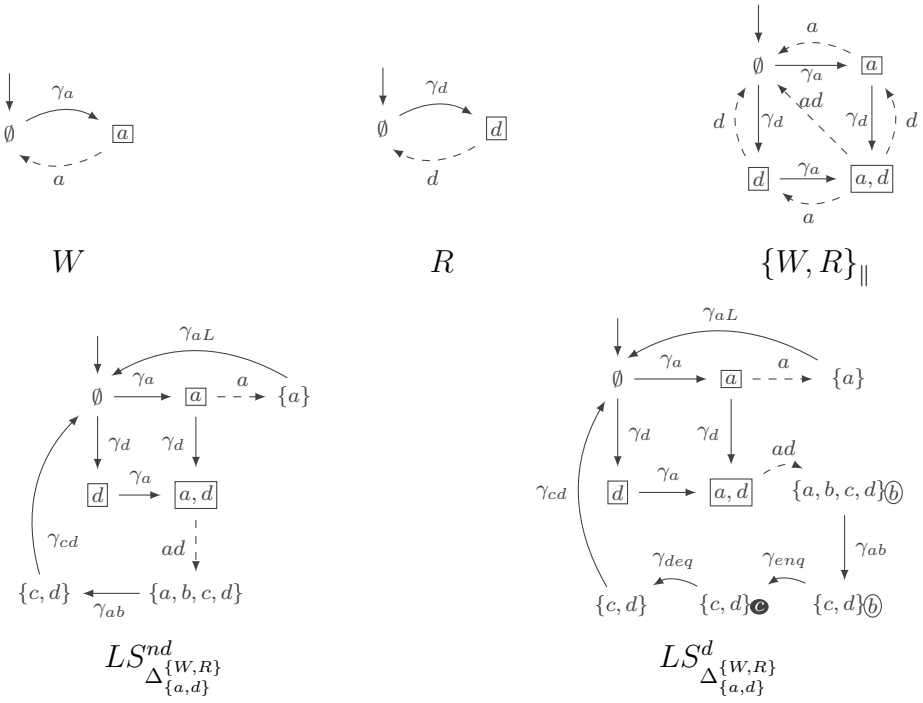


Figure 4.20: Construction of the deployment-phase $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ model for the lossy channel, with and without delaying nodes.

Nota that the resulting deployment-phase $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ without delaying nodes is exactly the same as obtained with $\mathsf{IMC}_{\mathsf{Reo}}$ (*c.f.,* Figure 4.7). The same does not happen when nodes present delays. ✠

Example 4.5 reinforces what was mentioned beforehand: when nodes do not

delay the system, composition of $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ models are not semantically disruptive with $\mathsf{IMC}_{\mathsf{Reo}}$ and the output is the same. This creates, as expected, a relation between the two approaches, captured in the following theorem.

Theorem 4.6. *Let I be an $\mathsf{IMC}_{\mathsf{Reo}}$ and J a deployed $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$. Consider that both I and J model the same stochastic Reo connector (*i.e., *with same stochastic information for channels and environment). Then, $I \sim J$ iff J has no enqueuing and dequeueing states.*

Proof. Let's prove the left to right implication. Suppose that $I \sim J$, and, by absurd, that J has enqueueing and dequeuing delays. Because $I \sim J$, then for each stable state i in I there exists a stable state j in J such that their cumulative rates are equal, and vice versa. But J has at least a state j representing enqueuing or dequeueing data. I does not have one such state to make the cumulative rate equal to the cumulative rate of j. Therefore $I \not\sim J$, what is contradictory with the assumption. Thus it is necessary that J has no enqueuing and dequeueing states.

It remains to prove the other direction. Suppose now that J has no enqueuing and dequeueing states. Since I and J model the same connector and because they are obtained via the same operations, then it is possible to trace paths from the initial states of each model such that every interactive transition evolves via the same set of actions and the Markovian transitions present the same rate. This imposes a relation in the states involved in such a path, which can easily be seen to be a strong bisimulation as in Definition 4.6. Therefore, $I \sim J$. □

When there are no delays on nodes, $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ and $\mathsf{IMC}_{\mathsf{Reo}}$ models may be used interchangeably. This is a result that springs from Theorems 4.1, 4.2 and 4.6.

4.4 Summary

This chapter introduced $\mathsf{IMC}_{\mathsf{Reo}}$, a stochastic model based on IMCs that captures the desired semantics of the stochastic Reo formalism. It differs from the classical IMC model by introducing a concrete structure for the states, and by assuming labels of interactive transitions as sets of actions, while preserving the same important features of that model.

$\mathsf{IMC}_{\mathsf{Reo}}$ was shown to be compositional and to correctly capture intended properties of stochastic Reo: for instance, it embodies context-awareness, it has a suitable identity element and it ensures atomicity of data flow between channels. It presents some drawbacks that do not favour its practical usage, though. The most relevant one is the fact that it assumes composition via nodes made of, at most, two channel

ends. Such number is the exception in practice. When three or more channel ends have to be composed, $\mathsf{IMC}_{\mathsf{Reo}}$ refactors the node so that it complies to that assumption. Such a refactoring introduces complexity in the connector, and consequently to the $\mathsf{IMC}_{\mathsf{Reo}}$ model. In order to avoid this problem, a distilled version of $\mathsf{IMC}_{\mathsf{Reo}}$, the $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$, was proposed, that is more natural and close to the engineering practice.

$\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ regards each channel as a group of four components: writer, reader, channel and node. Each such component has a simplified (low state space) $\mathsf{IMC}_{\mathsf{Reo}}$ model; the node model is parametrised on the number of channels it is made of. It further considers two distinct phases of composition: a design and a deployment phase. The former composes only channel and node components. The obtained models do not consider environment information (*i.e.,* request arrivals) and, therefore, do not completely capture the stochastic Reo semantics. The latter is essentially an extra composition step that couples readers and writers to the design-phase model. This brings environment information into the picture, making it compliant with the stochastic Reo semantics.

Part II

Reconfiguration of interacting services

Chapter 5

State of the Art: Software Reconfiguration

Study history, study history. In history lies all the secrets of statecraft.

– *Winston Churchill*

In this chapter. The state-of-the-art works on software reconfiguration is reviewed and discussed. It starts with an overview of several approaches to architectural reconfiguration. Afterwards, a number of languages used for the description of reconfigurations are introduced. Finally, techniques for self-adaptive systems construction are surveyed.

5.1 Architectural reconfigurations

The topic of software reconfiguration is neither new nor even recent. It can be dated back to the work of J. Kramer and J. Magee [169], in 1985, concerning the evolutionary needs of large scale distributed systems. But only after D. Garlan and M. Shaw [130], in 1993, introduced the concept of software architecture as an high-level abstraction for software development based on components and connectors, was that J. Kramer and J. Magee [180, 181] ported their reconfiguration knowledge to software architectures, coining the term "dynamic architectures". This topic soon became attractive among researchers from theoretical computer science to practical software engineering. Literature reveals that several approaches are addressed when it comes to deriving frameworks to deal with such dynamic changes of software architectures. Such approaches can be set on two perspectives: one is formal

and considers category theory, graph theory, or process algebra for specifying architectures and reasoning about their reconfigurations; the other is more practical and focuses on the manipulation of components, connectors and programming languages to achieve adaptation and update of running systems.

In this section, a number of approaches to architectural reconfiguration based on these perspectives are presented. A focus is set on reconfigurations within the Reo framework as it is central to this thesis.

5.1.1 Algebraic approaches

Much of the research on architectural reconfigurations has been based on the work of H. Ehrig [113], P. Degano [109] and U. Montanari [196] on graph grammars and their application to the reconfiguration of distributed systems. Categorial accounts of graph grammars as formal frameworks for reasoning about software architectures and their dynamic changes are largely explored in the literature. Process algebra also plays an important role when it comes to describe dynamic architectures, often, however, in an implicit way underlying some *architecture description languages (ADLs)*.

Reconfiguration by graph grammars

D. Hirsch, *et al* [147] model distributed systems as graphs, where edges are processes and nodes are shared data. The evolution of these systems is specified in a graph grammar, where context-free rewriting rules are defined for each process, determining how it evolves. The authors further consider synchronous processes of the distributed system that are supposed to evolve together on agreement of conditions imposed on the shared data. Rewriting rules are, for this reason, augmented with constraints that better define how they shall match.

M. Wermelinger, *et al* [252, 251, 253] propose the use of category theory as a base to a framework to model architectures, their reconfigurations and to relate the computational and the architectural levels. They also use labelled graphs of components and connectors to represent the architecture. Categories provide the link between the high-level architecture and the language programs. Reconfigurations are expressed through the graph rewriting formalism, in terms of the double pushout approach [113, 98]. A reconfiguration is a rewriting rule that requires the obtained result to be an architecture. In this work, the authors go further and take into account the state of the system to perform dynamic reconfigurations. In this setting, a dynamic reconfiguration is a rewriting rule extended with a condition over variables

of the initial architecture components. This guarantees the transformation of the architecture without tampering the system's state.

L. Baresi, *et al* [33] propose graphs of *unified modelling language (UML)* class diagrams to model software architectures and to specify reconfigurations through graph rewriting rules. In this work, a rewriting rule is a pair of UML diagrams which define the pre and post conditions of the architectural reconfigurations. Negative application conditions are also assumed to prevent reconfigurations when the required properties are not ensured. The dynamic behaviour of reconfigurations is analysed by validation and verification. Validation is carried out through either interactive simulation, which allows for testing for several scenarios and concentrate on the important aspects of the architecture, or critical pair analysis, which allows for checking whether the application of two rules may generate conflicts. For verification, the authors use techniques like reachability analysis, allowing for checking whether some configuration is reached from an initial one; and model checking of properties expressed in temporal logics.

R. Bruni, *et al* [66] account for the characterisation of four forms of dynamism in software architectures, using typed graph grammars as a formal model. In particular, they investigate whether such forms of dynamism (programmed, repairing, ad-hoc and constructible) are suitable for reasoning about the completeness and correctness of an architectural specification. Such properties are verified by taking into account notions of reachable configurations (the configurations allowed by successive application of reconfigurations, *i.e.,* the graph production rules) and acceptable configurations (the configurations that conform to a generic graph defining an architectural style).

In a different approach, R. Bruni, *et al* [65, 68, 67] adopt architectural design rewrite [69] (a rule-based technique for the design of dynamic software architectures) and exploits the notion of hierarchical graphs in the context of this technique. This allows for nesting graphs to represent partial views of the whole architecture. Architecture elements are represented uniformly and interfaces are added to the graph, enabling reutilisation. Reconfigurations are encoded as rewrite rules over design terms, with a simple sufficient condition that enforces the preservation of the architectural style. Moreover, reconfigurations may be labelled and inductively defined. The Maude framework is used to specify the architecture and to perform analysis on its structural and behavioural properties. Structural properties are analysed by applying spatial logics to both design terms and the architecture; behavioural properties, in turn, are checked using temporal logics. This is claimed to allow for reasoning about infinite sequences of reconfigurations.

A. Bucchiaronne [70] proposes, in his Ph.D. thesis, the traffic light process to design (the structure and behaviour) of dynamic software architectures, analyse them and generate associated code. This process resorts to typed graph grammars to represent architectures and their dynamism forms. Graph rewriting rules applied on the typed graphs represent reconfigurations, *i.e.,* the ways in which new configurations can be generated. Alloy [155] is used to specify the graph of the architecture and to verify both structural properties and its consistency with respect to the defined architectural styles. Moreover, model-checking techniques are employed for behavioural analysis of the designed architecture. For this, the author models the behaviour of each component of the architecture as an UML state-chart, and then model checks them for usual behavioural properties like deadlocks, liveness or safety. Once the specification is structurally and behaviourally correct, Java code is generated using the features of the ADL ArchJava [5, 4]. This process is amenable only for programmed reconfigurations, *i.e.,* reconfigurations established at design time. It was successfully applied in a case study for the automotive domain [39].

M. Tichy and B. Klöpper [246] propose the use of graph transformation as a natural way of modelling architectural reconfiguration for self-adaptive systems. Specifically, they use UML class diagrams to specify the structure of the architecture and employ story patterns to specify graph transformations, which are then applied based on the single pushout formalism [230]. They translate the UML graph specification into the PDDL language so that planning software can be used to simulate the self-adaptiveness of the architecture.

Reconfiguration by process calculi

Differently from the previous approaches, M. Bravetti, *et al* [59] propose the concept of adaptable processes as a way of overcoming the shortage on describing evolution in process calculi. The authors devise a calculus for such processes, considering two variants: structural and behavioural. The former allows for the description of both static and dynamic processes; the latter allows for the definition of behaviour of processes after dynamic updates. The adaptable processes calculus, is intended to be a tool for verification of key properties of dynamic processes. In this work, the calculus is used to investigate bounded and eventual adaptation properties. Different characterisations of the calculus affect the expressiveness and decidability/reachability results. In a follow-up work [60], the same authors extend their verification framework for this sort of processes by introducing a specific temporal logic.

Other contributions using process algebra for architectural description and reconfiguration are found in the context of ADLs. Relevant representatives include

Wright [9, 7, 8], Darwin [181], PADL [43], $\mathcal{P}i\mathcal{L}ar$ [102, 103] or LEDA [78, 79]. Later in this chapter, some of these ADLs will be briefly reviewed. For an interesting survey on these languages and how they cope with process algebra, the interested reader is referred to C. Cuesta *et al* [103].

5.1.2 Pattern-based approaches

This section reviews works that take advantage of high-level models of software architectures and develop reconfigurations by adopting the notion of patterns, .

P. Oreizy, *et al* [211, 212] propose an architecture-based approach to express the evolution of software at runtime. Four different types of software evolution are considered: addition, removal and replacement of components, and reconfiguration. The three first types concern the high-level manipulation of components; the reconfiguration concerns the change of component communication policies in order to modify the system overall behaviour. The authors consider connectors as discrete architectural entities playing a central role on runtime evolution. They can be manipulated as components and shall provide functionalities for binding management. The approach is implemented in a prototype framework named ArchStudio and is specifically targeted to C2-based applications. In this framework, the architecture model is stored in a textual format expressing the interconnection between components and connectors and their realisation in Java. Modifications to the architecture are inserted by the user either textually or visually.

H. Gomaa and M. Hussein [133] introduce an approach for designing reconfiguration patterns in software architectures, with a special focus on software product line architectures. A reconfiguration pattern is regarded as a solution to recurring problems on running (component-based) systems that need to change their configuration. The authors model high-level reconfiguration patterns with UML collaboration and state-chart diagrams, as well as reconfiguration scenarios described in terms of messages between two or more components. The authors propose four such reconfiguration patterns: master-slave pattern, where a master component distributes load by its (identical) slave components and merges the result when slaves finish their work; centralised-control pattern, where a central component coordinates the work of other components (as in an orchestrated approach to service composition); decentralised-control pattern, where no central component coordinates the work of other components and they communicate with each other (as in a choreographed approach to service composition); and client-server pattern, where client components communicate directly with service components. Moreover, a framework is implemented in the RPULSEE prototype, using the Rational Rose Real Time tool. The

framework interacts with the running system to control the reconfiguration process. It inherits basic validation for the application of reconfiguration patterns, under the form of execution control, visual component instance monitoring and analysis of message trace outputs.

P. Hnětynka and F. Plášil [148] propose reconfiguration patterns in the hierarchical component model SOFA [222]. The authors define dynamic reconfigurations as any modification performed to the system architecture while this is running; and consider six basic operations upon which a dynamic reconfiguration mew remove and add a component, remove and add a connection or remove and add a component's interface. However, it is claimed that *ad-hoc* application of the basic operations can lead to uncontrolled architectural modifications (the so called evolution gap problem). Thus, reconfiguration patterns are introduced to avoid such problem. Three patterns are considered: the nested-factory pattern, the component-removal pattern and the utility interface pattern.

F. Moo-Mena and K. Drira [197] present architectural reconfiguration patterns as sequences of basic actions to repair the system from faults. The strategy proposed is model-based and targets faulty web-service applications. In this context, a suitable extension to UML deployment diagrams is used as a high-level notation for specifying the structural aspects of architectures. Reconfigurations follow the graph rewriting approach. Actions are defined as textual specification for pattern matching upon a graph, where nodes are web-services and edges are the connections between them. The basic operators are addition and deletion of web-services and their connections, where restriction rules may be applied to make connections unique. Reconfigurations are ordered sequences of such actions that implement patterns like the duplication and the substitution of web-services.

M. Malohlava and T. Bureš [182] propose a simple language to specify the infrastructure of component-based systems, which enables the procedural description of changes. The authors claim that such a language supports both the separation of coordination patterns from the execution environment and their representation at a more abstract level. The language constructs are minimal and allow for the instantiation of reconfiguration patterns like the factory pattern, which involves adding new components and connectors; the removal pattern, which is dual to the factory pattern; and the dynamic-update pattern, which involves suspending a component, saving its runtime state, changing its business code, restoring the state and resuming the updated component.

A. Ramirez and B. Cheng [225] propose the development of dynamically adaptive systems through a model-based approach, where adaptation design patterns play a

central role. In this work, the authors focus on providing a catalogue of adaptation design patterns harvested from literature. From a dozen of such patterns, four of them concern reconfiguration of the adaptable system: component-insertion, component-removal, server reconfiguration and the decentralised pattern. They further characterise these patterns as structural (the first two), because they actually change the structure of architecture; and behavioural, as they modify the behaviour of the system.

C. Canal *et al* [80, 77] present a formalised framework for reconfiguring architectures for cases in which components were designed without reconfiguration capabilities. *Labelled transition systems (LTSs)* are used to model the components' behaviour, where labels represent their features/actions; and mappings of these labels establish the connections between components. The framework supports the application of reconfigurations without the need for halting the system. For this, they describe variants of reconfigurations as patterns of their application with a notion of contract-awareness. Those patterns include history-aware reconfigurations, which takes into account the history of actions performed until the reconfiguration was required; future-aware reconfigurations, which requires that the arrived configuration allows for the same operations enabled in the initial allowed; property-aware reconfigurations, where the strong imposition of future operations are relaxed by the requirement that only operational-based properties, and not all, are kept after reconfiguration; one-way reconfigurations, which blend both history- and future-aware reconfigurations; and finally, full-reconfigurability, which allows to move back and forth between configurations, restricting, however, some of their behavioural aspects. Moreover, the authors prove several properties of these reconfiguration patterns and provide algorithms for checking if reconfigurations satisfy the particular contract-aware properties in the patterns.

5.1.3 Coordination-targeting approaches

The following list presents some of the state-of-the-art works, which explore architectural reconfigurations from the coordination level.

G. Papadopoulos and F. Arbab [215, 216] were among the first researchers to introduce reconfiguration in the context of coordination models and languages. In particular, they show how control- and event-driven coordination languages can be used to achieve dynamic configuration of software architectures. For this, they use Manifold, a control- and event-driven coordination language, which provides a number of features including composition and separation of (computation from configuration) concerns. Manifold promotes, thereof, the topological definition of a

system by channeling components via their external behaviour (*i.e.,* their published interface). The approach resorts to the notion of events raised by components to enact (among other activities) reconfigurations. Reconfigurations are defined by the explicit rearrangement of connections between components. This is comparable to some reconfigurations practiced in the context of dynamic architectures as seen previously (*e.g.,* the removal/update of components), but with a focus particularly set on the coordination layer. The authors' claim is, in fact, corroborated. However, notwithstanding the possibility of defining reconfiguration criteria based on the observable behaviour of components, which is highlighted as a major advantage to other models, it comes in its disfavour the entanglement of reconfiguration and coordination logic.

D. Clarke [91, 92] proposes a basic logic for reasoning about connector reconfigurations in the context of the Reo coordination language. The author motivates his work by claiming that the reconfiguration of an architecture can be as simple as reconfiguring its underlaying coordination layer, but that there are no means available for reasoning about such reconfigurations. To overcame this hiatus, he first introduces an axiomatisation of connector construction on the Reo coordination model [90], enabling the discussion of connector equivalence and structural reconfiguration; then he presents a semantics for Reo, that considers reconfigurations, and introduce a logic, together with a model checking algorithm, for inspecting properties of the system before and after reconfigurations. In particular, for the axiomatisation of constructors, D. Clarke assumes a formalised Reo connector as a set of channels and a set of node constructions (based on channel ends), and defines the semantics of the usual Reo operations (join, split, hide and forget) upon this structure. A connector reconfiguration is the sequential application of these operations to the connector. The logic introduced in this work expands *computation tree logic (CTL)* by adding new modalities to express reconfiguration changes in a connector. Such a modal logic is interpreted over a graph where, essentially, nodes are configurations of a connector with their respective semantics given as *constraint automata (CA)*; and edges represent reconfigurations Formulas of this logic are expressed in terms of paths and states of CA for the current connector; the special reconfiguration transition will enforce the evaluation of the formulas over the CA of the reconfigured connector. This enables the possibility of both checking interesting (behavioural) properties of the connectors before and after reconfigurations and, of course, inspecting reconfiguration invariants.

C. Krause *et al* [166, 171] develop the topic of connector reconfigurations via high-level replacement systems, also in the context of the Reo coordination model.

These systems are a powerful generalisation of algebraic graph-rewriting techniques to cope with high-level structures from, *e.g.,* labelled graphs to Petri nets. This work, mainly reported in C. Krause's Ph.D. thesis [170], explores graph grammars to model and analyse coordination-based reconfigurations. In this approach, Reo connectors (and architectures, in general) are described by means of typed hypergraphs, where vertices are nodes of the connectors and (typed hyper-) edges are communication channels and components. Expectedly, reconfigurations are specified as graph productions (*i.e.,* rewriting rules), adhering to the double pushout approach [113, 98]. The authors claim three main advantages by following this path on graph rewriting: (*i*) the rewriting rules describe complex reconfigurations that are applied in an atomic step, rather than in a sequence of low-level operations; (*ii*) these rules are applied globally, wherever the left-hand side of each rule matches the source connector; and (*iii*) by using positive or negative conditions associated to rewriting rules, it is possible to define in which specific situations the architecture should be changed. For the analysis of reconfigurations, the authors take advantage of critical pair analysis, resorting to the AGG tool [241], which requires a model translation into an AGG compatible model. This analysis statically checks whether the application of one rule impairs the application of another. Additionally, they use the Henshin tool [21] to generate, from the defined reconfiguration rules, a labelled graph whose vertices are the possible connector configurations and edges are labelled with the applied reconfiguration rule. This enables model checking for state invariants expressed in *object constraint language (OCL)*, qualitative properties expressed in modal μ-calculus formulas, and probabilistic properties taking advantage of the PRISM tool. For the last one, the authors consider to add positive real numbers to the reconfiguration rules, describing the rate at which the rules are applied per unit of time. In an endeavour to tackle dynamic reconfigurations, the authors also propose a methodology to this end, deriving an engine of reconfigurations that is able to apply reconfigurations while a system is running. It does so by interrupting the acceptance of *input/output (IO)* requests in the coordinator ports, creating pending requests; then, after safely applying the necessary reconfiguration rules, the engine reactivates the coordinator ports and the pending requests may be served. The proposed approach for defining reconfiguration rules and their application over connectors is integrated in the *extensible coordination tools (ECT)* plugin for Eclipse [17] as a proof-of-concept. The dynamic reconfiguration approach, in turn, is developed as the ReoLive web-service.

Based on the previous work, C. Krause *et al* [165] devised an approach for triggering connector reconfigurations based on context-dependency and data-flow. For

this, the authors take advantage of the Reo colouring semantics [93] to observe the flow of data within connectors and to enrich the graph-rewriting rules by annotating them with suitable patterns of the colours (from the colouring semantics). This extension to the way reconfigurations are designed enable reconfigurations to be applied non when the rewriting rules match the structure of the connector, but when it matches simultaneously the structure and the current state of execution of a connector. This kind of pattern matching is more restrictive than the previous one, but equips the system with dynamic self-adaptation capabilities: there is no need for an external entity to enact reconfigurations, since these are automatically executed whenever the rewriting rules match the semantics and structure of the connector.

Still in the context of C. Krause's Ph.D., C. Krause *et al* [164] explore reconfigurations in the context of distributed networks of Reo connectors. In this work, the premise is that besides components, also the coordination layer is distributed, *i.e.,* the underlying sub-connectors are not in the same logical or physical location. Moreover, such connectors are regarded as black-boxes that can only communicate and connect to each other through their published interfaces, with the particularity of knowing in which ways they can be reconfigured. Because of such distribution and the inherently absence of a global knowledge about the network, the authors propose that reconfigurations are triggered locally in the context of one sub-connector, creating, then, an asynchronous cascade of reconfigurations through the whole network. In particular, the connector where the reconfiguration is triggered, publishes a reconfiguration request on its published interfaces, to its neighbours, which will pass the message. Reconfigurations in each sub-connector are enacted bottom-up, taking into account that they know their reconfiguration rules. The authors further include a roll-back mechanism to undo a complete reconfiguration when one sub-connector fails to complete its evolution. This maintains the atomicity of reconfigurations as in the non-distributed approach.

M. Bozga *et al* [55] introduce Dy-BIP as an extension to the BIP [35, 34] coordination model by adding features for specification and analysis of dynamic architectures. For a brief reference to BIP, please see Section 3.4. Concretely, Dy-BIP introduces notions of interaction constraints and history variables, and associates them to each transition of the automaton of each component. Interaction constraints are defined as boolean operations on (global) port names, and history variables are used to track communication between components at each state. The latter is claimed to reduce the state space of each component. For the former, the authors provide a set of high-level (causal, acceptance and filter) constraint constructs that hide logic related details. The reconfiguration strategy associated to this model is based on

the *on-the-fly* composition of components by the centralised Dy-BIP engine. In practice, a global behavioural model is defined as usually by the product of the atomic components. However, it is only during execution that transitions are defined. At each state, the engine picks a single transition, based on a global constraint built from the individual interaction constraints available on the current state of each atomic component. Then, each component atomically executes the transition that validates the global constraint, while other transitions are disregarded. This reconfiguration strategy clearly addresses the coordination layer of component-based systems by facing its evolution when some actions/ports of the atomic components are not available. In addition, it is compositional and equipped with a clear semantics. The on-the-fly approach is claimed to be very efficient when systems grow bigger. For small systems, though, the time required for the *on-the-fly* computation of interactions is outperformed by its static counterpart.

5.2 Languages for reconfiguration

In the previous section, several works were reviewed addressing reconfiguration of software architectures. However, in order to analyse and apply reconfigurations in runtime, it is required that they are expressed somehow. This is usually made resorting to textual approaches, following either structured standards, like the *extensible markup language (XML)*, or tailor-made specifications, most notably, in the context of ADLs. In this section, the most prominent works in these categories are reviewed.

5.2.1 Languages for architecture description

ADLs were initially proposed as formalisms to describe the static structure of software architectures. Not surprisingly, the main concepts involved, *e.g.,* component, connector or configuration, are regarded as first-class citizens in these languages [188]. Different ADLs come equipped with different features to tackle different aspects of computation within the area of component-based or, more recently, service-oriented systems. One of the main handicaps of several of these languages is their inability to provide means to describe and analyse runtime changes, whether structural or behavioural.

Wright and dynamic Wright

Wright [9, 7], takes the notions of component, connector and configuration as the main concepts in its specification. A component is a computational entity equipped

with an interface (*i.e.,* a set of input and output ports) and a computation (*i.e.,* a behaviour specification). A connector defines the communication/interaction between components; its specification is divided into a set of roles (*i.e.,* the interface of the connector that defines the behaviour expected from the components connected to it) and the glue (*i.e.,* the behaviour of the connector). A configuration defines the whole structure of a system architecture. It embodies instances of components and connectors and specifies, via attachments, how component ports attach to connector roles. Furthermore, Wright uses *communicating sequential processes (CSP)* [149] for specifying the behaviour of the roles and the glue of connectors, and computation of basic components. However, since Wright is hierarchical, configurations may be used instead of CSP to define the computation part of a (more complex) component.

By itself, Wright is not able to support architectural reconfiguration. However, dynamic Wright [8] fills that hole by introducing a notion of control events as part of the component's behaviour, defining states where the system can be changed and triggering reconfigurations. Reconfigurations are described in a variant of CSP extended with the actions *new, del, attach* and *detach* for manipulating the topology of the architecture. The use of CSP to specify these reconfigurations limits, though, the architectures to a specific number of possible configurations, which shall be known at design time. Moreover, this approach requires that the reconfiguration controller knows the components in the architecture, since the triggering events is component-specific. In this context, changing a component to another could result in deprecating some reconfigurations, if the new component defines different events.

Darwin

Darwin [180, 181] is a π-calculus [191] based ADL. Darwin assumes component and binding as the main concepts in the architectures it describes. A component is a computational structure with an interface defined by the services it requires (input ports) and the services it provides (output ports). Components can be either basic or composite. The latter are defined by interconnecting instances of basic components and other composite components. The interconnection is made through bindings, which associate the names of provided services of one component with the names of required services of another. The system architecture is, in itself, specified as a composite component.

Darwin enables the description of dynamic architectures by providing two mechanisms: lazy instantiation and direct dynamic instantiation. Lazy instantiation is concerned with the enforced delay in instantiating a component until some user attempts to access its provided services. This, in practice, fixes the (possible) re-

configurations at design time, and therefore it does not allow for incrementing the architecture with new components and bindings. In turn, direct dynamic instantiation allows for specification of architectures that may evolve in arbitrary ways, depending, for instance, on the input data at runtime. Such evolution considers the creation of new instances of components that were not predicted at runtime and also of their bindings.

ACME and Plastik

ACME [126, 128], is a format for describing architectures that is easy to read, write and integrate with tools for architectural exploitation. ACME supports four aspects of architecture:(*i*) structure, built on top of a core ontology of usual architectural concepts (*e.g.,* components, connectors, systems, ports, roles, among others); (*ii*) semantics, defined as properties of the several architectural elements; (*iii*) constraints, expressed in Aramani [195], a language based on first-order predicative logic and finally (*iv*) styles, defined through the concept of family.

However, ACME does not provide constructs to express reconfigurations. To overcome this, Plastik [36] was introduced as a reflective framework to express and enable reconfigurations in runtime. It takes ACME as the basic language to describe the architecture and maps that description to a reflective component model that allows for dynamic change. It supports reconfigurations by introducing language constructs for conditional triggering of reconfigurations; deconstruction (detach and remove) of architectural elements; expressing runtime dependencies; and finally dynamic attachment of component ports to connector roles.

xADL 2.0

xADL 2.0 [106] is proposed as a highly-extensible ADL based on XML. It supports not only design time and runtime modelling of software architectures, but also the management of configurations. The latter is responsible for the dynamic evolution of the architecture by supporting the definition of options, variants and versions (of architectural elements).

Options define optional elements of the architecture. Each optional architectural element is equipped with a guard condition that is evaluated on architectural instantiation. If the condition holds, then the associated elements are instantiated with the architecture. Variants enable the specification of architectural elements that can vary on architectural instantiation. This introduces a new type of element that can have multiple (structural or behavioural) facets, among which one is picked. Again, (mutually-exclusive) conditional guards are associated to each facet; its evaluation

to true determines which facet is instantiated with the architecture. Finally, versions allow for the definition of a graph of versions of architectural elements, which enables their runtime replacement/upgrade.

5.2.2 Languages for reflective adaptations

Recent, approaches to architectural reconfiguration suggest the use of reflective models of running systems, instead of ADL-based static scripts. This makes a new category of languages to emerge, that allow for modelling system reflections. These languages provide specific constructs for architecture evolution via reconfiguration.

The K-Component model

J. Dowling and V. Cahill [112] present the K-Component meta-model. It is a reflective model for realising dynamic software architectures. Such realisation is made through a reification of the base architecture into a configuration graph. In this graph representation, nodes are interfaces labelled with component instances and edges are connectors labelled with connector properties, notably their reconfigurable properties. In this approach, the authors free the architects from using ADLs to specify the architectural meta-model, and generate it automatically from the (C++ source code of the) components and connectors implementation. All the concepts usually present in ADLs are scattered in several C++ idioms. Dynamic reconfigurations are realised by adaptation contracts. The latter are series of conditional graph transformation rules (based on the double pushout approach) that represent, in fact, system architectural constraints. Adaptation contracts are specified in the adaptation contract description language, which includes conditional rules that trigger the associated reconfigurations when specific adaptation events occur.

PiLar

$\mathcal{P}i\mathcal{L}ar$ [102, 103] was born as the counterpart language of a reflective framework for dynamic reconfiguration of software architectures. The language provides constructs for specifying both structural aspects of the architecture as well as its dynamic evolution. From the structural point of view, the component is the only architectural notion with relevance. A component is essentially described by its interface (so called avatar) or, additionally, by its internal configuration, *i.e.,* a set of component instances interconnected via attachments (forming a composite component). Connectors are not part of the $\mathcal{P}i\mathcal{L}ar$ lexicon; rather connections between components are seen either as simple direct links, or yet another component with interface and

behaviour. On the dynamic perspective, the language allows for the definition of interaction constraints, reconfigurations and reflection. Interaction constraints are expressed in the π-calculus process algebra. To enable reconfigurations, a set of constructs is provided for addition, deletion, hiding and aliasing of not only components, but also ports and their connections. The support to reflection is guaranteed by the specification of reifications. A reification defines causal links between base- and meta-components, where meta-components control, by reflection, the base ones. Reflective and dynamic constructs may be used together, enabling the envisaged evolution of a system.

FPath and FScript

FPath and FScript [107] were introduced as languages to specifically support dynamic reconfiguration of Fractal architectures [64, 175]. Fractal is a hierarchical component model that allows for implementing, deploying, and managing complex component-based software systems. It provides powerful, yet low-level features for dynamic reconfiguration by supporting reflection. Thereof, introspective and interceptive features allow for both discovering the structure of the application and modifying it, respectively. FPath and FScript are proposed as high-level substitutes of the Fractal intrinsic dynamic management capabilities. The languages, being specific to the Fractal component model, are focused on the direct manipulation of the relevant concepts. This ensures a great degree of reliability when it comes to dynamic manipulation of a system's internal structure.

In particular, FPath is a *domain-specific language (DSL)* for querying Fractal architectures represented as FPath graphs. In this graph representation, nodes are components, their interfaces and attributes; and (directed) edges connect components with other components or their interfaces or attributes, and are labelled with ontological relationships, named axis. FPath expressions are sequences of axis (following a notation similar to XPath [238]) that allow the navigation in the structure of the application. FScript allows for the definition of dynamic reconfigurations of Fractal architectures. It is focused only on the manipulation of architectural elements by means of sequences of functions or actions. It embodies the FPath language to access the relevant architectural elements (from within functions) and uses actions to describe complex structural modifications. These changes are performed by the application of primitive actions for adding or removing components, binding and unbinding interfaces of client and servers, starting and stoping components, changing names, setting new values for component attributes, among others. FScript ensures reliable dynamic reconfigurations by enclosing actions and functions with *atomicity,*

consistency, isolation and durability (ACID) transactional support.

5.3 Self-adaptation

Self-adaptive software systems are known to respond at run time to changes detected either internally or externally, in an attempt to keep meeting their own functional requirements and agreed levels of *quality of service (QoS)* [210, 129]. This entails the need for some degree of introspection. Self-adaptive systems should be able to know their internal compositions and attributes, execution environment, requirements and reference performance levels. But above all, they should be able to observe changes in these elements. Such observations, suitably processed (*e.g.,* by comparison to reference levels assigned to measurable variables) are responsible for triggering adaptations.

The process from acquiring information about changes and consequently enacting adaptations, is known as the *control* or *feedback loop*. This is a concept with origin in control theory, and adopted in several other areas like autonomic computing, robotics or artificial intelligence [202, 131]. The implementation of such a process, typically involves four well defined components: monitor, analyser, planner and executer. The MAPE-K model [160, 153] is the reference model for such process. Self-adaptive software systems realise this model by (*i*) monitoring the environment and probing the system's attributes; (*ii*) analysing the data collected to infer situations in need for adaptation; (*iii*) deciding the adaptation strategy; and finally, (*iv*) enacting reconfigurations to enforce the system's adaptation into acceptable (non disruptive) configurations [111, 62, 249].

5.3.1 Approaches

Literature presents different approaches to implement feedback loops. In general, the approaches agree on external, reusable and component-based feedback loop implementations, rather than on internal, monolithic, and intertwined implementations [232, 152, 87, 254]. Commonly, the feedback loop is implemented as a system controlling a base one, and thus is usually referred to as the *managing system*. The base system is the *managed* system.

The existing approaches for self-adaptation can be regarded over a centralised or a decentralised perspective. In the sequel, some representatives of these approaches are reviewed.

Centralised approaches

Centralised approaches for self-adaptation are characterised by the adoption of a single feedback loop to manage all the system concerns. These approaches differ from one another, though, in the way the systems are modelled or adaptations are realised.

A. Agrawal *et al* [3] and T. Fischer *et al* [121] use UML along with graph transformation techniques to define the adaptation of systems. Performance analysis is not natural, but checking behavioural and structural properties becomes facilitated from the use of constraint languages like OCL.

D. Garlan *et al* [127] present Rainbow one of the most acclaimed frameworks for self-adaptation. It models the system architecture as an abstract graph of computational elements. That model is available at runtime as an active asset of the feedback loop. It endows the loop with accessible knowledge about the system behaviour and enables the verification of properties. Operators, invariants and constraints are used for the definition of adaptation strategies.

M. Litoiu *et al* [176] propose an adaptation strategy to guarantee web services quality. In particular, they propose a control loop implementation that is based on a model of the web services. Moreover, a robust estimator is used to keep the QoS values in accordance to the system goals.

Parra *et al* [217], implement a strategy for adapting *service-oriented architecture (SOA)* systems based on the dynamic software product lines paradigm. Features of the product family, defined in a suitable set of assets, represent the overall architecture of the system. Dynamic adaptation relies on context-aware assets. These define alternative architectures and the conditions for their existence. A straightforward decision algorithm uses data obtained by a context manager to check conditions within the assets before choosing and applying the adaptation.

R. Calinescu *et al* [76, 75, 74] present a framework for the adaptation of software systems, where system components are modelled as Markov chains. The framework performs component analysis via quantitative model checking in PRISM. The results obtained from the analysis are employed for dynamically adjusting the system to its objectives and the changes in the environment. Moreover, a notion of policies is devised to define either constraints to which the system should agree or measures of success that it must optimise. Adaptations are made on the configurable parameters of the system that realise such policies.

A. van Hoorn [150] present SLAstic, a framework for architectural adaptation of distributed systems. The main objective of the framework is to make system running with efficient resource allocation, while maintaining the service quality initially

agreed. The framework counts on a middleware and a controller part. SLAstic middleware is incorporated in the software system. It performs convenient adaptations to minimise system resource utilisation. Concretely, SLAstic middleware is able to perform changes by allocating and deallocating execution nodes; migrate software components between environments; and (un)balance load at the level of the components. The SLAstic controller is an implementation of MAPE-K feedback loop. It is responsible for observing, analysing and enacting adaptations.

Based on SLAstic, R. von Massow *et al* [185] put the Palladio component model [38] to the service of reconfigurations. The authors introduce SLAstic.SIM as a performance simulator to study the impact of reconfiguration in the performance of a system. It is used as well to evaluate adaptation strategies and tactics based on realistic workloads, in the context of the SLAstic control loop.

M. Becker *et al* [37] formalise an approach for the design of self-adaptive systems based on simulation of a specific model of a system. Simulations are performed to gradually find a suitable point to trigger adaptations and consequently to fulfil system requirements. This is done for a range of possible (static) contexts and through multiple design iterations.

Recently, the MUSIC project [140, 122] was presented as a framework for model driven development of (component- and service-based) adaptable mobile applications, in the context of ubiquitous computing. It relies on models for both the context and the application architecture. Annotations of adaptation capabilities and contextual dependencies map both models. The framework instantiates the MAPE-K architecture. It uses a reasoner to search the set of possible configurations for the optimal solution in the current context. When an adaptation is required, a reconfiguration script is derived and executed.

Decentralised approaches

Decentralised approaches are characterised by the use of several feedback loops (or several of its components) to control a system (typically complex and distributed) [255]. Decentralising control-components imply costs of coordination, but it realises self-adaptive systems in a more natural, robust and stable way, yielding better adaptation solutions [250]. Approaches in this perspective typically rely on artificial intelligence mechanisms and similar ones that soak aspects from nature and real-life dynamics.

J. Kephart and W. Walsh [161] proposes a unified framework for autonomic computing based on policies brought from artificial intelligence, namely rational agents. Rational agents reify a sort of MAPE-K model, by perceiving and acting upon its

changing environment. The proposed policies are used to evolve a state-based system, by taking it from a state into another. Three different sorts of policies are proposed: actions, goals and utility functions. Actions define a concrete change in the system, based on if-then expressions. Goals define sets of desired states to which the system shall evolve autonomously when certain conditions are met. Utility functions define objectives and rank states by their utility at each moment. These policies express strategies of how the system decides and enacts adaptations. They can be used interchangeably in different components of a system, in an approximation to a decentralised decision making system. The authors highlight, however, that a relationship between the several policies have to be consistent with the coordination between the components they make autonomous.

B. Caprarescu and D. Petcu [81] take inspiration from natural adaptive systems to propose a robust feedback loop for computational systems. In particular, they resort to multi-agent technology and swarm intelligence to define decentralised feedback loops that mimic ant colonies. The use of multiple feedback loop agents enables robustness. When one agent fails, the others may continue by organising themselves. Adaptation decisions are based on a shared repository of ranked policies (as addressed above [161]). At a time, only one agent is allowed to apply changes, though; an election algorithm based on policy priorities and agent communication is used for this.

F. André *et al* [10] propose SAFDIS, a framework for adaptation of distributed service-based applications, that is fully decentralised. SAFDIS defines feedback loops as independent applications, external to the system which they control. SAFDIS works as a composition of multiple autonomous instances realising the MAPE-K model. Each such instance is responsible for adapting the associated component of the distributed managed application. Although actuating individually, the analysers of each loop cooperate via coordination and negotiation. This becomes necessary in order to obtain decisions based on the overall system information, rather than only on the local part of the system that each loop controls.

V. Cardellini *et al* [82] propose MOSES as a methodology for supporting QoS-driven adaptation of service-oriented systems. In essence, it is intended to act as a service broker providing the best selection and binding of services and coordination patterns (via adaptation). The optimal solutions are found per user request, for a suitable description of the system architecture and its non-functional requirements. A set of distributed monitors collect data about QoS of regionally distributed pools of services, that are candidates for the managed system. The remaining tasks of the MAPE-K reference model that it follows, analyses, decisions and execution, are

made on a centralised approach.

V. Nallur and R. Bahsoon [201] present an approach for decentralised self-adaptation by blending principles of economics with computing. Concretely, the authors focus on a market-oriented programming strategy to define adaptation strategies. This involves the existence of several market places where seller services offer their QoS attributes for some cost, and buyer applications bid for services with a desired QoS and the price they are willing to pay for such service. The existence of several markets makes the approach decentralised. Several decider agents have to work in a distributed way for a suitable solution. When more than one seller fulfils the requirements of the buyer, the decider agents associated to the buyer rank the sellers by computing an aggregation of QoS values offered by each one.

5.3.2 Adaptation specification

With the crescent interest in self-adaptive systems, and the several proposals for the implementation of feedback loops, it became necessary to systematise the definition of adaptation logic. Languages to that end are only a few, as they started to emerge only in the past couple of years.

B. Cheng *et al* [88] propose Stitch as a language to specify adaptation strategies in the context of Rainbow [127], the well-known framework for architecture-based self-adaptation. Stitch directly springs from tasks usually performed by architects and system administrators in order to repair architectures. The language is based on three main concepts: operators, tactics and strategies. Operators are primitive commands for managing configurations. Tactics embed operators in reconfiguration actions. They define conditions for their applicability, and effects to check whether the reconfiguration yielded the expected results. Strategies define adaptation processes as decision trees (with default case) constructed from conditions and tactics. Tactics are triggered upon the successful evaluation of such conditions. The latter are formulas, built from functional and non-functional properties of the system. Delays are assumed as a time-window for reconfigurations to take effect as they occur asynchronously. Upon these delays, the reconfiguration may either be successful or not. On failure, exception treatment is provided by Stitch that includes, as last resort, the explicit means for halting the adaptation strategy.

N. Huber *et al* [151] propose the S/T/A domain-specific modelling language to describe runtime adaptation processes on top of QoS models of component-based system architectures [52, 38, 137, 134]. S/T/A is based on three concepts: actions, tactics and strategies. Actions are the atomic elements of the language that confer change to the configuration of the underlying architecture. These are just refer-

ences to adaptation points specified and realised in the concrete system QoS model. Tactics describe how actions are applied to take the system into a desired configuration. Concretely, each tactic defines a control flow that drives the adaptation plan. Conditions are used to influence the tactic application or the path to follow in the described flow. Finally, strategies combine tactics in a weighted fashion. Strategies are used to trigger adaptations based on events raised when, *e.g.,* objectives (associated to the strategies) are violated. Objectives are predicates over QoS properties of the system that realise the overall goals of the system.

Chapter 6

Modelling Reconfigurations

To improve is to change; to be perfect is to change often.

– *Winston Churchill*

In this chapter. A model for coordination-based architectural reconfigurations is formalised. In particular, notions of coordination and reconfiguration patterns are introduced. These notions are the foundations of a framework for architectural reconfiguration of interacting services — henceforth referred to as ARIS (extending to the title of this thesis), for easier reference — detailed and exploited in this second part of this dissertation. The patterns are regarded from both a static and a dynamic perspective. Moreover, their quantitative extension is studied to meet QoS-related concerns.

Parts of this chapter's content was previously published, by the author, in:

- Nuno Oliveira and Luís S. Barbosa. "On the reconfiguration of software connectors". In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. Vol. 2. SAC '13. Coimbra, Portugal: ACM, Mar. 2013, pp. 1885–1892.
- Nuno Oliveira and Luís S. Barbosa. "Reconfiguration Mechanisms for Service Coordination". In: *Web Services and Formal Methods*. Ed. by Maurice H. ter Beek and Niels Lohmann. Vol. 7843. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 134–149.

6.1 Coordination patterns

The coordination layer of an architecture is the focal point of the ARIS framework proposed in this dissertation. The coordination layer constrains service interaction in an (exogenously) orchestrated software system, by imposing communication policies. Policies are captured in coordination structures, defining how data and control is

shared between services. Altogether, coordination structures and policies encode architectural solutions for coordination problems.

Coordination problems are recurrent in software composition. For instance, sequential or parallel activity, joining and merging activities, communication with acknowledgements, priority execution, among others, are some of these problems. Concrete solutions for them may be varied, but they usually follow an abstract pattern with basic semantics and properties associated.

In the context of this work, these are referred to as *coordination patterns*. Concrete (exogenous-) coordination formalisms, like Reo or BIP, could be used to define such structures. However, the intention of a coordination pattern is to be (as much as possible) independent of such a concrete model. Not in the perspective of proposing yet another coordination formalism, but rather to regard it as an abstract recipient for a concrete model, upon which reconfigurations (and properties) can be studied in an independent framework.

A coordination pattern is formally defined as an abstract graph of communication primitives. Communication primitives are the indissociable part of a coordination pattern, defining a basic coordination policy. In the sequel, they are referred to as *channels*. Although borrowing that name from Reo, the notion is taken here in a looser way. For instance, one assumes that a channel may have two or more ends.

Edges in the coordination pattern graph are labelled with both a unique identifier and a type. The type associates a semantics to the channel, concretising its operational behaviour. Formally,

Definition 6.1 (Channel)**.** *Let $\mathcal{E}$ be a set of channel ends, $\mathcal{I}$ a set of unique identifiers and $\mathcal{T}$ a set of channel types. A channel is a tuple*

$$c = \langle \mathcal{S}, i, t, \mathcal{K} \rangle$$

where $i \in \mathcal{I}$, $t \in \mathcal{T}$, and $\mathcal{S}, \mathcal{K} \subseteq \mathcal{E}$, such that $\mathcal{S} \cap \mathcal{K} = \emptyset$, are the sets of source and sink ends, respectively.

For illustration purposes, the set of types considered in the sequel is borrowed from Reo, $\mathcal{T} = \{\mathsf{sync}, \mathsf{lossy}, \mathsf{drain}, \mathsf{fifo_e}, \mathsf{fifo_f}, \ldots\}$ (*c.f.,* Figure 2.1), unless stated otherwise.

Example 6.1 An instance sc of a sync channel is written as $\langle \{a\}, sc, \mathsf{sync}, \{b\} \rangle$. Similarly, an instance sd of a drain channel is given by $\langle \{\{c\}, \{d\}\}, sd, \mathsf{drain}, \emptyset \rangle$. ✠

Notation

$\pi_i(c)$ selects the i^{th} component of a tuple c.

Nodes in a coordination pattern represent interaction points. The latter are locations where channels synchronise their interfaces (ends) for interaction with other patterns or external components. When these locations are composed of more than one end, the latter are said to be *co-located*.

Definition 6.2 (Coordination pattern)**.** *A coordination pattern ρ is a pair*

$$\rho = \langle \mathcal{C}_\rho, \mathcal{N}_\rho \rangle$$

where $\mathcal{C}_\rho$ is a set of channels, and $\mathcal{N}_\rho$ is a partition on the union of all ends of all channels in $\mathcal{C}_\rho$, such that,

1. *channel identifiers are unique:*

$$\forall_{c_1,c_2 \in \mathcal{C}_\rho} \,.\, \pi_2(c_1) = \pi_2(c_2) \;\rightarrow\; c_1 = c_2$$

2. *the number of channels sharing a node never exceeds the number of co-located ends in it:*

$$\forall_{n \in \mathcal{N}_\rho} \,.\, |n| \geq |\{c \in \mathcal{C}_\rho |\; n \cap (\pi_1(c) \cup \pi_4(c)) \neq \emptyset\}|$$

One uses $\mathbf{0}_\rho = \langle \emptyset, \emptyset \rangle$ to deonote the empty coordination pattern. In addition, sets $\mathcal{N}$ and $\mathcal{P}$ refer to the domain of nodes and coordination patterns, respectively.

A number of operations are considered to access specific parts of a coordination pattern. Thus,

$$I(\rho) = \{i \in \mathcal{N}_\rho |\; \exists_{c \in \mathcal{C}_\rho} \,.\, i \cap \pi_1(c) \neq \emptyset \wedge in(i, \rho)\},$$

where $in(x, \rho) = \forall_{c \in \mathcal{C}_\rho} \,.\, x \cap \pi_4(c) = \emptyset$ and

$$O(\rho) = \{o \in \mathcal{N}_\rho |\; \exists_{c \in \mathcal{C}_\rho} \,.\, o \cap \pi_4(c) \neq \emptyset \wedge out(o, \rho)\},$$

where $out(x, \rho) = \forall_{c \in \mathcal{C}_\rho} \,.\, x \cap \pi_1(c) = \emptyset$, are used to retrieve, respectively, the set of source and sink nodes (*i.e.,* the IO interface) of a coordination pattern ρ. Operation $IO(\rho) = I(\rho) \cup O(\rho)$ retrieves the whole interface.

Notation

– $2^f(C) = \{f(e) |\; e \in C\}$ denotes the application of a function f to the elements of a set C.
– $\mathsf{the}S$ denotes the unique element of a singleton set S.
– $(\phi \;\rightarrow\; e_1,\, e_2)$ is conditional expression (read as *return e_1 if ϕ holds, e_2 otherwise*).

$$\mathfrak{I}_\rho = 2^{\pi_2}(\mathcal{C}_\rho)$$

retrieves the set of channel identifiers in a coordination pattern ρ.

$$\mathfrak{E}^{ch}_\rho = \mathsf{let}\,(c = \mathsf{the}\{c' \in \mathcal{C}_\rho |\ \pi_2(c') = ch\})\ \mathsf{in}\ \pi_1(c) \cup \pi_4(c)$$

computes the set of ends of a channel (uniquely) identified by ch in the context of a coordination pattern ρ. Finally,

$$\mathfrak{N}^{ch}_\rho = \{n \in \mathcal{N}_\rho |\ \mathfrak{E}^{ch}_\rho \cap n \neq \emptyset\}$$

retrieves the nodes of the coordination pattern ρ in which the ends of channel ch participate.

In addition,

$$\iota\langle \mathcal{S}, i, t, \mathcal{K}\rangle = \langle\{\langle \mathcal{S}, i, t, \mathcal{K}\rangle\}, \{\mathcal{S}, \mathcal{K}\}\rangle$$

is used to convert a single channel $\langle \mathcal{S}, i, t, \mathcal{K}\rangle$ into a coordination pattern.

Notation

$\underline{ab}$ will refer to a node $\{a, b\}$. Accordingly a set of nodes $\{\{c\}, \{d\}\}$ will be denoted by $\{\underline{c}, \underline{d}\}$.

Example 6.2 Consider a slightly modified version of the Sequencer connector in Figure 2.1. Its corresponding coordination pattern is as shown in Figure 6.1.

$$\rho_s = \left\langle \begin{Bmatrix} \langle\{a\}, s_1, \mathsf{sync}, \{c\}\rangle & \langle\{d\}, s_2, \mathsf{sync}, \{o_1\}\rangle, \\ \langle\{e\}, x, \mathsf{fifo_e}, \{f\}\rangle, & \langle\{g\}, s_3, \mathsf{sync}, \{o_2\}\rangle \\ \langle\{h\}, s_4, \mathsf{sync}, \{b\}\rangle & \end{Bmatrix}, \right\rangle$$
$$\{\underline{a}, \underline{o_1}, \underline{o_2}, \underline{b}, \underline{cde}, \underline{fgh}, \}$$

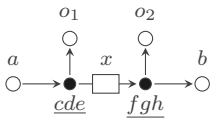


Figure 6.1: The Sequencer coordination pattern.

One can access the parts of ρ_s using its specific operations. For instance, the set of nodes $\mathcal{N}_\rho = \{\underline{a}, \underline{o_1}, \underline{o_2}, \underline{b}, \underline{cde}, \underline{fg}\}$. The input and output ports are, respectively, $I(\rho) = \{\underline{a}\}$ and $O(\rho) = \{\underline{o_1}, \underline{o_2}, \underline{b}\}$. In addition $\mathfrak{I}_\rho = \{s_1, s_2, s_3, s_4, x\}$, $\mathfrak{E}^{s_1}_\rho = \{a, c\}$ and $\mathfrak{N}^{s_1}_\rho = \{\underline{a}, \underline{cde}\}$. ✠

In the sequel, a visual Reo-like representation of coordination patterns (as in Figure 2.1) will be adopted. To increase readability channel identifiers are only added to the visual representation when needed.

6.2 Reconfigurations

In an exogenous coordination framework, the focus of reconfigurations is the set of coordination patterns themselves, instead of the plugged-in components, which are considered external services. Therefore, any change to the original structure of a coordination pattern qualifies as a reconfiguration. These changes are driven by the sequential application of a number of parametric primitive reconfiguration operations which manipulate the basic elements of a coordination pattern. The set of all reconfigurations is denoted, henceforth, as $\mathcal{R}$.

6.2.1 Primitive reconfiguration operations

Primitive reconfiguration operations change *atomically* the basic structure of a coordination pattern, namely, its nodes and channels. The following five primitive operations are considered:

$$Prim = \{\mathsf{const}_\rho, \mathsf{par}_\rho, \mathsf{join}_P, \mathsf{split}_p, \mathsf{remove}_{ch} |\ \rho \in \mathcal{P}, P \subseteq \mathcal{N}, p \in \mathcal{N}, ch \in \mathcal{I}\}$$

Additionally, it is assumed an identity operation, denoted $\mathbf{1}_r$. The subscripts associated to each primitive operation are their expected arguments. Clearly, $Prim \subseteq \mathcal{R}$. The application of a primitive reconfiguration r to a coordination pattern ρ yields a new coordination pattern suitably modified.

Notation

$\rho \circlearrowleft r$ denotes the application of reconfiguration r to coordination pattern ρ.

The semantics of $\circlearrowleft$ is defined below for each primitive reconfiguration.

The most trivial reconfiguration is the constant one, along which a new coordination pattern replaces the original one. Formally:

Definition 6.3 (const)**.** *Let $\rho_1, \rho_2 \in \mathcal{P}$. Then,*

$$\rho_1 \circlearrowleft \mathsf{const}_{\rho_2} = \rho_2$$

The par operation sets the original coordination pattern in parallel with the one given as a parameter. This operation does not create any connection between the two coordination patterns. It assumes, without loss of generality, that ends (consequently, nodes) and channel identifiers in both patterns are disjoint.

Definition 6.4 (par)**.** *Let $\rho_1, \rho_2 \in \mathcal{P}$. Then,*

$$\rho_1 \circlearrowleft \mathsf{par}_{\rho_2} = \langle \mathcal{C}_{\rho_1} \cup \mathcal{C}_{\rho_2}, \mathcal{N}_{\rho_1} \cup \mathcal{N}_{\rho_2} \rangle$$

The join operation performs connections in the coordination pattern by merging a set of nodes into a single one.

Definition 6.5 (join). *Let $\rho \in \mathcal{P}$, and $N \subseteq \mathcal{N}$. Then*

$$\rho \circlearrowright \ \mathsf{join}_N = \langle \mathcal{C}_\rho, (N \subseteq \mathcal{N}_\rho \ \rightarrow \ \{\bigcup N\} \cup (\mathcal{N}_\rho \setminus N), \mathcal{N}_\rho)\rangle$$

Example 6.3 Consider $\rho = \langle\{\langle\{a\}, sc, \mathsf{sync}, \{b\}\rangle, \langle\{c, d\}, sd, \mathsf{drain}, \emptyset\rangle\}, \{\underline{a}, \underline{b}, \underline{c}, \underline{d}\}\rangle$ and use the join primitive to connect nodes $\underline{b}$ and $\underline{c}$. The relevant operation is $\rho \circlearrowright \mathsf{join}_{\{\underline{b},\underline{c}\}}$. Since $\{\underline{b}, \underline{c}\} \subseteq \{\underline{a}, \underline{b}, \underline{c}, \underline{d}\} = \emptyset$, this entails computing

1. $\bigcup\{\underline{b}, \underline{c}\} = \underline{bc}$ and
2. $\mathcal{N}_\rho \setminus \{\underline{b}, \underline{c}\} = \{\underline{a}, \underline{d}\}$.

Performing the union of points 1 and 2, it is obtained $\{\underline{a}, \underline{d}, \underline{bc}\}$. Therefore,

$$\rho \circlearrowright \mathsf{join}_{\{\underline{b},\underline{c}\}} = \langle\{\langle\{a\}, sc, \mathsf{sync}, \{b\}\rangle, \langle\{c, d\}, sd, \mathsf{drain}, \emptyset\rangle\}, \{\underline{a}, \underline{d}, \underline{bc}\}\rangle. \qquad ✠$$

The split primitive reconfiguration is dual to join. It breaks connections within a coordination pattern by separating all channel ends co-located on a given node.

Definition 6.6 (split). *Let $\rho \in \mathcal{P}$ and $n \in \mathcal{N}$. Then,*

$$\rho \circlearrowright \ \mathsf{split}_n = \langle \mathcal{C}_\rho, (n \in \mathcal{N}_\rho \ \rightarrow \ 2^{\mathsf{sing}}\, n \cup (\mathcal{N}_\rho \setminus \{n\}), \mathcal{N}_\rho)\rangle$$

where $\mathsf{sing}(x) = \{x\}$.

Example 6.4 Let ρ stand for the coordination pattern obtained in Example 6.3. Applying the $\rho \circlearrowright \mathsf{split}_{\underline{bc}}$ primitive makes it possible to retrieve the initial pattern of that example. Since $\underline{bc} \in \{\underline{a}, \underline{d}, \underline{bc}\}$, compute

1. $2^{\mathsf{sing}}\, \underline{bc} = \{\underline{b}, \underline{c}\}$ and
2. $\mathcal{N}_\rho \setminus \{\underline{bc}\} = \{\underline{a}, \underline{d}\}$.

Therefore,

$$\rho \circlearrowright \mathsf{split}_{\underline{bc}} \ = \ \langle\{\langle\{a\}, sc, \mathsf{sync}, \{b\}\rangle, \langle\{c, d\}, sd, \mathsf{drain}, \emptyset\rangle\}, \{\underline{a}, \underline{b}, \underline{c}, \underline{d}\}\rangle. \qquad ✠$$

Finally, the remove operation removes a channel from a coordination pattern and updates the nodes in which the channel synchronises its ends.

Definition 6.7 (remove)**.** *Let $\rho \in \mathcal{P}$ and $ch \in \mathfrak{I}_\rho$. Then,*

$$\begin{array}{rcll} \rho \circlearrowleft \mathsf{remove}_{ch} = & \mathsf{let} & c = \mathsf{the}\{e \in \mathcal{C}_\rho |\ \pi_2(e) = ch\}) \\ & & N = (2^{\mathsf{minus}_{\mathfrak{E}^{ch}_\rho}}\ \mathcal{N}_\rho) \setminus \{\emptyset\} \\ & \mathsf{in} & \langle \mathcal{C}_\rho \setminus \{c\}, N\rangle \end{array}$$

where $\mathsf{minus}_E(X) = X \setminus E$.

Example 6.5 Consider again the coordination pattern ρ obtained in Example 6.3, but extended with a $\mathsf{fifo_e}$ linking channels sc and sd:

$$\rho = \langle\{\langle\{a\}, sc, \mathsf{sync}, \{b\}\rangle, \langle\{e\}, q, \mathsf{fifo_e}, \{f\}\rangle, \langle\{c, d\}, sd, \mathsf{drain}, \emptyset\rangle\}\{\underline{a}, \underline{be}, \underline{fc}, \underline{d}\}\rangle$$

The pattern obtained in Example 6.4 can now be achieved by applying $\rho \circlearrowleft \mathsf{remove}_q$. Compute

1. $c = \langle\{e\}, q, \mathsf{fifo_e}, \{f\}\rangle$
2. $N = (2^{\mathsf{minus}_{\{e,f\}}}\ \{\underline{a}, \underline{be}, \underline{fc}, \underline{d}\}) \setminus \{\emptyset\} = \{\underline{a}, \underline{b}, \underline{c}, \underline{d}\}$.

Putting it all together,

$$\rho\ =\ \langle\{\langle\{a\}, sc, \mathsf{sync}, \{b\}\rangle, \langle\{c, d\}, sd, \mathsf{drain}, \emptyset\rangle\}, \{\underline{a}, \underline{b}, \underline{c}, \underline{d}\}\rangle. \qquad ✠$$

The following results characterise the effect of primitive reconfiguration operations. Lemma 6.1 shows that the set $\mathcal{P}$ of coordination patterns is closed under the application of these primitives. Lemma 6.2 identifies contexts which are unaffected by reconfigurations.

Lemma 6.1. *The set $\mathcal{P}$ of coordination patterns is closed under the application of reconfigurations in Prim.*

Proof. Let $\rho, \rho' \in \mathcal{P}$, $N \subseteq \mathcal{N}$; $n \in \mathcal{N}$, $ch \in \mathcal{I}$ and $r \in Prim$. For each primitive reconfiguration let us check the properties in Definition 6.2. Thus,

- for $\rho \circlearrowleft \mathsf{const}_{\rho'}$, all properties hold since ρ and ρ' are assumed to be well-formed.
- For $\rho \circlearrowleft \mathsf{par}_{\rho'}$, the resulting pattern is the component-wise union of ρ and ρ', which are assumed to be disjoint. Therefore, since ρ and ρ' are well-formed and union does not change their specification, well-formedness is preserved.
- For $\rho \circlearrowleft \mathsf{join}_N$, all the nodes in N are merged together. Property 1 is preserved because ρ is well formed and no channels are added to it. Property 2 also holds

because each node in N preserves the inequality. By merging these nodes into one, the inequality still remains, because the nodes are disjoint partitions of channel ends.

- For $\rho \circlearrowleft \mathsf{split}_n$, all the channel ends co-located in n are separated into simple nodes. Property 1 is preserved because ρ is well formed and no channels are added to it. Property 2 is also preserved because, ends being unique, the separation of node n results in $|n|$ nodes, each of which is formed by a single channel end naturally associated to a single channel.

- As a result of $\rho \circlearrowleft \mathsf{remove}_{ch}$ the channel identified by ch is removed and its ends are removed from the nodes to which it was previously connected. Properties 1 and 2 are preserved because no channel is added to the well-formed pattern ρ. Property 2 is also preserved. Because ρ is well-formed, then each node n to which ch is connected preserves the inequality. By removing ends of ch from n it is obtained either $|n| = 0$, in this case the node is removed from ρ; or $|n| > 0$, meaning that a number k of channels share ends in n. Then, either each of these channels contribute with one end to n, yielding $n = k$; or they contribute with more than one end to n, yielding $n \geq k$.

□

Lemma 6.2. *Let $\rho, \rho' \in \mathcal{P}$; $P \subseteq \mathcal{N}$; $p \in \mathcal{N}$ and $ch \in \mathcal{I}$. The following properties, stated as strict equalities between coordination patterns, hold:*

$$\rho \circlearrowleft \mathsf{const}_{\rho'} = \rho \text{ if } \rho = \rho' \tag{6.1}$$

$$\rho \circlearrowleft \mathsf{join}_P = \rho \text{ if } P \setminus \mathcal{N}_\rho \neq \emptyset \tag{6.2}$$

$$\rho \circlearrowleft \mathsf{split}_p = \rho \text{ if } p \notin \mathcal{N}_\rho \tag{6.3}$$

$$\rho \circlearrowleft \mathsf{remove}_{ch} = \rho \text{ if } \forall_{c \in \mathfrak{I}_\rho} c \neq ch \tag{6.4}$$

Proof. The lemma guarantees that reconfiguration operations that do not affect elements of the coordination pattern have no effect. All of them come easily from the definitions. For (6.4), note that if $\forall_{c \in \mathfrak{I}_\rho} c \neq ch$ then $\{e \in \mathcal{C}_\rho |\ \pi_2(e) \neq ch\} = \emptyset$, $\mathfrak{E}_\rho^{ch} = \emptyset$ and $2^{\mathsf{minus}_\emptyset}(\mathcal{N}_\rho) \setminus \{\emptyset\} = \mathcal{N}_\rho$ Therefore, $\rho \circlearrowleft \mathsf{remove}_{ch} = \langle \mathcal{C}_\rho, \mathcal{N}_\rho \rangle = \rho$. □

6.2.2 Composing reconfigurations

In most cases, the application of a single primitive reconfiguration is not enough. Single steps, however, can be combined to yield broader transformations of the

coordination pattern. This sub-section discusses the *sequential* composition of reconfigurations.

Definition 6.8 (Sequential Composition)**.** *Let $\rho \in \mathcal{P}$ and $r_1, r_2 \in \mathcal{R}$. The application of r_1 followed by r_2 is given by*

$$\rho \circlearrowleft \{r_1 \ ; \ r_2\} = (\rho \circlearrowleft r_1) \circlearrowleft r_2$$

Lemma 6.3. *The set $\mathcal{P}$ of coordination patterns is closed for sequential composition.*

Proof. The proof is by induction on the structure of reconfigurations. The base case, of primitive reconfigurations, is already proved in lemma 6.1. Consider now $\rho \circlearrowleft \{r_1; r_2\}$, and assume, without loss of generality that r_2 is a primitive reconfiguration. If not, r_2 can always be rewritten as sequence of reconfigurations $r_1', r_2', \cdots$, such that its last element is a primitive reconfiguration. By induction hypothesis $\rho \circlearrowleft r_1$ is in $\mathcal{P}$. Then conclude by lemma 6.1, for r_2 primitive. □

Lemma 6.4. *Let $\rho, \rho_1, \rho_2 \in \mathcal{P}$, $\{n\}, N, N_1, N_2 \in \mathcal{N}$. Then,*

$$\rho \circlearrowleft \mathsf{const}_{\rho_1} = \rho_1 \tag{6.5}$$

$$\rho \circlearrowleft \mathsf{par}_{\rho_1} = \rho_1 \circlearrowleft \mathsf{par}_{\rho} \tag{6.6}$$

$$\rho \circlearrowleft \{\mathsf{par}_{\rho_1}; \mathsf{par}_{\rho_2}\} = \rho \circlearrowleft \mathsf{par}_{\rho_1 \circlearrowleft \mathsf{par}_{\rho_2}} \tag{6.7}$$

$$\rho \circlearrowleft \{\mathsf{join}_{N_1}; \mathsf{join}_{N_2}\} = \rho \circlearrowleft \mathsf{join}_{N_1 \cup N_2} \tag{6.8}$$

$$\rho \circlearrowleft \{\mathsf{join}_{N}; \mathsf{split}_{n}\} = \rho \quad \textit{if } \bigcup N = n \wedge \forall_{p \in N} \cdot |p| = 1 \tag{6.9}$$

$$\rho \circlearrowleft \{\mathsf{split}_{n}; \mathsf{join}_{N}\} = \rho \quad \textit{if } \bigcup N = n \tag{6.10}$$

$$\rho \circlearrowleft \{\mathsf{par}_{\iota\langle \mathcal{S}, i, t, \mathcal{K}\rangle}; \mathsf{remove}_i\} = \rho \tag{6.11}$$

Proof. Property (6.5) is an immediate consequence of the definition of const. Laws (6.6) and (6.7) are proved in a similar way resorting to the commutativity and associativity of set union, respectively. Thus,

$$\begin{aligned}
& \rho \circlearrowleft \mathsf{par}_{\rho_1} \\
= & \quad \{ \text{ definition of } \mathsf{par} \ \} \\
& \langle \mathcal{C}_\rho \cup \mathcal{C}_{\rho_1}, \mathcal{N}_\rho \cup \mathcal{N}_{\rho_1} \rangle \\
= & \quad \{ \cup \text{ commutative } \} \\
& \langle \mathcal{C}_{\rho_1} \cup \mathcal{C}_\rho, \mathcal{N}_{\rho_1} \cup \mathcal{N}_\rho \rangle \\
= & \quad \{ \text{ definition of } \mathsf{par} \ \} \\
& \langle \mathcal{C}_{\rho_1}, \mathcal{N}_{\rho_1} \rangle \circlearrowleft \mathsf{par}_{\langle \mathcal{C}_\rho, \mathcal{N}_\rho \rangle}
\end{aligned}$$

$$
\begin{array}{ll}
= & \{ \text{ projections } \} \\
 & \rho_1 \circlearrowleft \mathsf{par}_\rho
\end{array}
$$

For (6.7),

$$
\begin{array}{ll}
 & \rho \circlearrowleft \{\mathsf{par}_{\rho_1}; \mathsf{par}_{\rho_2}\} \\
= & \quad \{ \text{ definition of sequential composition } \} \\
 & (\rho \circlearrowleft \mathsf{par}_{\rho_1}) \circlearrowleft \mathsf{par}_{\rho_2} \\
= & \quad \{ \text{ definition of } \mathsf{par} \ \} \\
 & \langle \mathcal{C}_\rho \cup \mathcal{C}_{\rho_1} \cup \mathcal{C}_{\rho_2}, \mathcal{N}_\rho \cup \mathcal{N}_{\rho_1} \cup \mathcal{N}_{\rho_2} \rangle \\
= & \quad \{ \cup \text{ associative, definition of } \mathsf{par} \ \} \\
 & \langle \mathcal{C}_\rho, \mathcal{N}_\rho \rangle \circlearrowleft \mathsf{par}_{\langle \mathcal{C}_{\rho_1} \cup \mathcal{C}_{\rho_2}, \mathcal{N}_{\rho_1} \cup \mathcal{N}_{\rho_2} \rangle} \\
= & \quad \{ \text{ definition of } \mathsf{par} \ \} \\
 & \langle \mathcal{C}_\rho, \mathcal{N}_\rho \rangle \circlearrowleft \mathsf{par}_{\langle \mathcal{C}_{\rho_1}, \mathcal{N}_{\rho_1} \rangle \circlearrowleft \mathsf{par}_{\langle \mathcal{C}_{\rho_2}, \mathcal{N}_{\rho_2} \rangle}} \\
= & \quad \{ \text{ projections } \} \\
 & \rho \circlearrowleft \mathsf{par}_{\rho_1 \circlearrowleft \mathsf{par}_{\rho_2}}
\end{array}
$$

For the laws, involving join and split, attention is restricted to the cases in which the reconfiguration actually changes the pattern (*i.e.,* in the *then*-cases of the conditional definition in both operators). The other cases are trivial. Thus, for (6.8),

$$
\begin{array}{ll}
 & \rho \circlearrowleft \{\mathsf{join}_{N_1}; \mathsf{join}_{N_2}\} \\
= & \quad \{ \text{ definition of sequential composition } \} \\
 & (\rho \circlearrowleft \mathsf{join}_{N_1}) \circlearrowleft \mathsf{join}_{N_2} \\
= & \quad \{ \text{ definition of } \mathsf{join} \ (\textit{then}\text{-case}) \text{ applied twice } \} \\
 & \langle \mathcal{C}_\rho, \{\bigcup N_2\} \cup (\{\bigcup N_1\} \cup (\mathcal{N}_\rho \setminus N_1)) \setminus N_2 \rangle \\
= & \quad \{ \text{ property of } \cup \ \} \\
 & \langle \mathcal{C}_\rho, \{\bigcup (N_1 \cup N_2)\} \cup ((\mathcal{N}_\rho \setminus N_1) \setminus N_2) \rangle \\
= & \quad \{ \text{ property of } \setminus \ \} \\
 & \langle \mathcal{C}_\rho, \{\bigcup (N_1 \cup N_2)\} \cup (\mathcal{N}_\rho \setminus N_1 \cup N_2) \rangle \\
= & \quad \{ \text{ definition of } \mathsf{join} \ (\textit{then}\text{-case}) \ \} \\
 & \rho \circlearrowleft \mathsf{join}_{N_1 \cup N_2}
\end{array}
$$

For (6.9)

$$
\begin{aligned}
& \rho \circlearrowleft \{\mathsf{join}_N; \mathsf{split}_n\} \\
= & \quad \{ \text{ definition of sequential composition } \} \\
& (\rho \circlearrowleft \mathsf{join}_N) \circlearrowleft \mathsf{split}_n \\
= & \quad \{ \text{ definition of } \mathsf{join} \text{ (\textit{then}-case) } \} \\
& \langle \mathcal{C}_\rho, \{\bigcup N\} \cup (\mathcal{N}_\rho \setminus N)\rangle \circlearrowleft \mathsf{split}_n \\
= & \quad \{ \text{ definition of } \mathsf{split} \text{ (\textit{then}-case) } \} \\
& \langle \mathcal{C}_\rho, 2^{\mathsf{sing}} n \cup (\{\bigcup N\} \cup (\mathcal{N}_\rho \setminus N)) \setminus \{n\}\rangle \\
= & \quad \{ \text{ assumption: } \bigcup N = n \} \\
& \langle \mathcal{C}_\rho, 2^{\mathsf{sing}} n \cup (\mathcal{N}_\rho \setminus N)\rangle \\
= & \quad \{ \text{ assumption of } \forall_{p \in N} \cdot |p| = 1 \ \} \\
& \langle \mathcal{C}_\rho, N \cup (\mathcal{N}_\rho \setminus N)\rangle \\
= & \quad \{ \text{ projections } \} \\
& \rho
\end{aligned}
$$

For (6.10),

$$
\begin{aligned}
& \rho \circlearrowleft \{\mathsf{split}_n; \mathsf{join}_N\} \\
= & \quad \{ \text{ definition of sequential composition } \} \\
& (\rho \circlearrowleft \mathsf{split}_n) \circlearrowleft \mathsf{join}_N \\
= & \quad \{ \text{ definition of } \mathsf{split} \text{ (\textit{then}-case) } \} \\
& \langle \mathcal{C}_\rho, 2^{\mathsf{sing}} n \cup (\mathcal{N}_\rho \setminus \{n\})\rangle \circlearrowleft \mathsf{join}_N \\
= & \quad \{ \text{ assumption: } \bigcup N = n \text{ (which implies } 2^{\mathsf{sing}} n = N) \} \\
& \langle \mathcal{C}_\rho, N \cup (\mathcal{N}_\rho \setminus \{n\})\rangle \circlearrowleft \mathsf{join}_N \\
= & \quad \{ \text{ definition of } \mathsf{join} \text{ (\textit{then}-case) } \} \\
& \langle \mathcal{C}_\rho, \{\bigcup N\} \cup (N \cup (\mathcal{N}_\rho \setminus \{n\})) \setminus N\rangle \\
= & \quad \{ \bigcup N = n \} \\
& \langle \mathcal{C}_\rho, \mathcal{N}_\rho\rangle \\
= & \quad \{ \text{ projections } \} \\
& \rho
\end{aligned}
$$

For (6.11),

$$
\begin{aligned}
& \rho \circlearrowleft \{\mathsf{par}_{\iota\langle\mathcal{S},i,t,\mathcal{K}\rangle}; \mathsf{remove}_i\} \\
= & \quad \{ \text{ definition of sequential composition } \} \\
& (\rho \circlearrowleft \mathsf{par}_{\iota\langle\mathcal{S},i,t,\mathcal{K}\rangle}) \circlearrowleft \mathsf{remove}_i \\
= & \quad \{ \text{ definition of } \iota \ \} \\
& (\rho \circlearrowleft \mathsf{par}_{\langle\{\langle\mathcal{S},i,t,\mathcal{K}\rangle\},\{\mathcal{S},\mathcal{K}\}\rangle}) \circlearrowleft \mathsf{remove}_i \\
= & \quad \{ \text{ definition of } \mathsf{par} \ \} \\
& \langle \mathcal{C}_\rho \cup \{\langle \mathcal{S}, i, t, \mathcal{K}\rangle\}, \mathcal{N}_\rho \cup \{\mathcal{S}, \mathcal{K}\}\rangle \circlearrowleft \mathsf{remove}_i \\
= & \quad \{ \text{ definition of } \mathsf{remove} \ \} \\
& \langle \mathcal{C}_\rho \setminus \{\langle \mathcal{S}, i, t, \mathcal{K}\rangle\}, 2^{\mathsf{minus}_{\mathcal{S}\cup\mathcal{K}}} \mathcal{N}_\rho \cup \{\mathcal{S}, \mathcal{K}\}\rangle \\
= & \quad \{ \text{ sets } \mathcal{S} \text{ and } \mathcal{K} \text{ are new nodes, } \textit{i.e.,} \text{ not present in } \mathcal{N}_\rho \ \} \\
& \langle \mathcal{C}_\rho, \mathcal{N}_\rho \rangle \\
= & \quad \{ \text{ projections } \} \\
& \rho
\end{aligned}
$$

□

6.2.3 Reconfiguration patterns

Via composition, one may express 'big step' reconfigurations that are able to affect significant parts of a connector (rather than just a node or a channel). These constitute what one will be referred to as *reconfiguration patterns*. The word *pattern* is employed here to emphasise their generic nature and reusability.

The following paragraphs present a set of six reconfiguration patterns, first introduced in [204]. This set is not restricted to the six patterns, though. More can be found and tailored to cover the specific needs of systems. The six patterns defined next had shown to be useful in practice, making possible a plethora of reconfigurations of several distinct coordination patterns, only by changing their parameters.

To the left of each pattern definition, it is presented an abstract visual representation of the pattern behaviour. Each such picture shall be read from left (*the original configuration*) to right (*what results after applying the reconfiguration*). Black dots represent nodes, arrows and lines stand for channels, cloud-shapes stand for any internal structure of a coordination pattern and dashed lines represent node bindings.

Let $\rho \in \mathcal{P}$ be the coordination pattern to which each reconfiguration pattern is

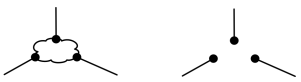
applied.

Remove

Pattern $\mathsf{removeP}(Cs)$ removes a set of channel identifiers $Cs \subseteq \mathfrak{I}_\rho$ from ρ, by successively applying the primitive remove operation. Formally,

$$\rho \circlearrowleft \mathsf{removeP}(Cs) = rS(\rho, Cs)$$

where



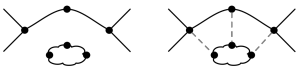
$$rS(\rho, \emptyset) = \rho$$
$$rS(\rho, \{c\} \cup C) = rS(\rho \circlearrowleft \mathsf{remove}_c, C)$$

Overlap

Pattern $\mathsf{overlapP}(\rho_r, X)$ connects a new coordination pattern $\rho_r \in \mathcal{P}$ to ρ, by joining specific in $X \subseteq \mathcal{N}_\rho \times \mathcal{N}_{\rho_r}$. Each pair in X indicates which nodes from ρ and ρ_r are to be joined. Formally,

$$\rho \circlearrowleft \mathsf{overlapP}(\rho_r, X) = rO(\rho \circlearrowleft \mathsf{par}_{\rho_r}, X)$$

where



$$rO(\rho, \emptyset) = \rho$$
$$rO(\rho, \{y\} \cup Y) = rO(\rho \circlearrowleft \mathsf{join}_{\{\pi_1(y), \pi_2(y)\}}, Y)$$

Example 6.6 Consider the Sequencer connector as depicted in Figure 6.1. Suppose that a company uses this protocol to coordinate the sequential execution of two services coupled to ports o_1 and o_2. For some reason there was a need to restrict the second service to execute only on completion of the first. A possible solution is to let the first service to acknowledge its finishing status, and automatically allow the second service to execute. If ρ_s is the Sequencer coordination pattern one may propose the following reconfiguration: $r_{waiting} = \rho_s \circlearrowleft \mathsf{overlapP}(\ i_1 \circ \rightarrow\!\!-\!\!\leftarrow \circ\ i_2\ , \{(\underline{fgh}, \underline{i_2})\})$.

This yields the WaitingSequencer coordination pattern presented in Figure 6.2 (a). Naturally, the first service should now be connected to port i_1 for status acknowledgement.

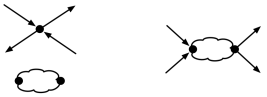
✠

Insert

Pattern $\mathsf{insertP}(\rho_r, n, m_i, m_o)$ places ρ side by side with a given $\rho_r \in \mathcal{P}$, and splits $n \in \mathcal{N}_\rho$ to make room for ρ_r to be inserted. Connections are then re-built as follows: all the output ports produced by the split operation are joined with $m_i \in I(\rho_r)$.

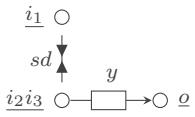
Dually, the input ports produced by the split operation are joined with $m_o \in O(\rho_r)$. Formally,

$$\begin{aligned}
&\rho \circlearrowleft \mathsf{insertP}(\rho_r, n, m_i, m_o) = \\
&\qquad \text{let} \quad \rho_1 = \rho \circlearrowleft \mathsf{par}_{\rho_r} \\
&\qquad\qquad \rho_2 = \rho_1 \circlearrowleft \mathsf{split}_n \\
&\qquad\qquad I_{sp} = I(\rho_2) \setminus I(\rho_1) \\
&\qquad\qquad O_{sp} = O(\rho_2) \setminus O(\rho_1) \\
&\qquad \text{in} \quad (\rho_2 \circlearrowleft \mathsf{join}_{O_{sp} \cup \{m_i\}}) \circlearrowleft \mathsf{join}_{I_{sp} \cup \{m_o\}}
\end{aligned}$$



Example 6.7 Consider the same situation as in Example 6.6. A different solution is to let the first service to acknowledge its termination and the protocol to memorise it. Reconfiguration

$$r_{proactive} = \rho_s \circlearrowleft \mathsf{insertP}(\quad , \underline{fgh},\ \underline{i_2 i_3},\ \underline{o})$$

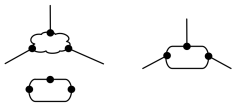


does the job, yielding the ProActiveWaitingSequencer coordination pattern depicted in Figure 6.2 (*b*). ✠

Replace

Pattern $\mathsf{replaceP}(\rho_r, X, Cs)$ replaces a sub-structure of ρ by removing the old structure composed of the channels in set $Cs \subseteq \mathfrak{I}_\rho$ and overlapping ρ_r via information in set $X \subseteq \mathcal{N}_\rho \times \mathcal{N}_{\rho_r}$. Formally,

$$\begin{aligned}
&\rho \circlearrowleft \mathsf{replaceP}(\rho_r, X, Cs) = \\
&\qquad (\rho \circlearrowleft \mathsf{removeP}(Cs)) \circlearrowleft \mathsf{overlapP}(\rho_r, X))
\end{aligned}$$



Example 6.8 Consider again the Sequencer coordination pattern. Suppose that the services connected to ports o_1 and o_2 can fail for long periods of time, possibly leading to deadlocks. A possible solution to prevent deadlocks is to avoid enforcing services to answer. Replacing the sync channels that provide ports o_1 and o_2 by lossy, as encoded in reconfiguration:

$$r_{weak} = \rho_s \circlearrowleft \mathsf{replaceP}(\begin{matrix} \underline{i_1} \circ\!\!- \overset{s_5}{- -}\!\!\to \circ\, \underline{o_1} \\ \underline{i_2} \circ\!\!- \overset{s_6}{- -}\!\!\to \circ\, \underline{o_2} \end{matrix} , \{(\underline{cde}, \underline{i_1}), (\underline{fgh}, \underline{i_2})\}, \{s_2, s_3\})$$

channels solves the problem. This produces the WeakSequencer depicted in Figure 6.2 (*c*). ✠

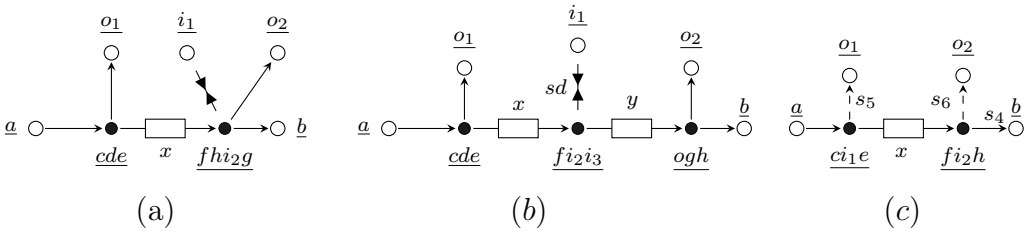
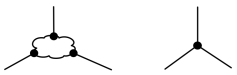


Figure 6.2: Reconfigurations of the Sequencer coordination pattern: (*a*) the WaitingSequencer, (*b*) the ProActiveWaitingSequencer and (*c*) the WeakSequencer coordination patterns.

Implode

Pattern $\mathsf{implodeP}(Cs)$ collapses a sub-structure of ρ composed of the channels in $Cs \subseteq \mathfrak{I}_\rho$. The resulting ports are joined together into an updated node. Formally,

$$\begin{aligned} \rho \circlearrowleft \mathsf{implodeP}(Cs) = \quad & \text{let} \quad \rho_1 = \rho \circlearrowleft \mathsf{removeP}(Cs) \\ & \text{in} \quad \rho_1 \circlearrowleft \mathsf{join}_{\mathcal{N}_{\rho_1} \setminus \mathcal{N}_\rho} \end{aligned}$$

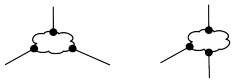


Example 6.9 Consider again the Sequencer coordination pattern. Suppose now that the company desires that the services connected to ports o_1 and o_2 execute simultaneously. An immediate solution is to parallelise their execution. Reconfiguration: $r_{parallel} = \rho_s \circlearrowleft \mathsf{implodeP}(\{x\})$ solves the problem. This produces the counterpart coordination pattern of the Replicator connector depicted in Figure 2.1 (with the obvious node names). ✠

Move

Pattern $\mathsf{moveP}(ch, e, n)$ moves the end of a channel $ch \in \mathfrak{I}_\rho$ from node e to node $n \in \mathcal{N}_\rho$. Formally,

$$\begin{aligned} & \rho \circlearrowleft \mathsf{moveP}(ch, e, n) = \\ & \qquad \text{let} \quad e' = \mathfrak{N}_\rho^{ch} \setminus \{e\} \\ & \qquad\qquad \rho_1 = \rho \circlearrowleft \mathsf{split}_e \\ & \qquad\qquad E = \mathfrak{N}_{\rho_1}^{ch} \\ & \qquad\qquad I_{sp} = I(\rho_1) \setminus I(\rho) \\ & \qquad\qquad O_{sp} = O(\rho_1) \setminus O(\rho) \\ & \qquad \text{in} \quad (\rho_1 \circlearrowleft \mathsf{join}_{(I_{sp} \cup O_{sp}) \setminus E}) \circlearrowleft \mathsf{join}_{(E \setminus \{e'\}) \cup \{n\}} \end{aligned}$$

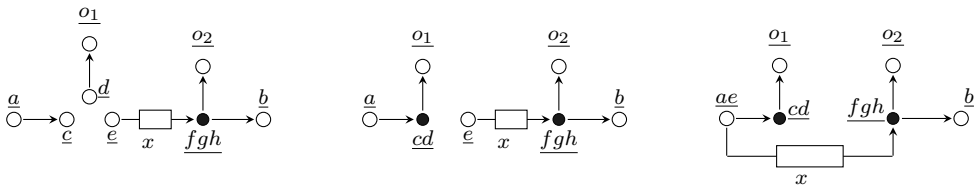


Example 6.10 This example uses the Sequencer coordination pattern in order to explain, step by step, the moveP reconfiguration pattern. Consider the reconfiguration $r_{s_{kept}} =$

$\rho_s \circlearrowleft \mathsf{moveP}(x, \underline{cde}, \underline{a})$, that takes the channel identified by x with an end in node $\underline{cde}$ to be moved to node $\underline{a}$. The first step is to obtain the node where the other end of channel x is: $e' = \mathfrak{N}^x_{\rho_s} \setminus \{\underline{cde}\} = \underline{fgh}$. Then reconfiguration $\mathsf{split}_{\underline{cde}}$ is applied to the Sequencer pattern, yielding ρ_1 as depicted in Figure 6.3 (1). Now, the three sets of nodes E, I_{sp} and O_{sp} are computed:

$$\begin{aligned} E &= \mathfrak{N}^x_{\rho_1} = \{\underline{e}, \underline{fgh}\}; \\ I_{sp} &= I(\rho_1) \setminus I(\rho) = \{\underline{a}, \underline{e}, \underline{d}\} \setminus \{\underline{a}\} = \{\underline{e}, \underline{d}\}; \\ O_{sp} &= O(\rho_1) \setminus O(\rho) = \{\underline{c}, \underline{o_1}, \underline{o_2}\} \setminus \{\underline{o_1}, \underline{o_2}\} = \{\underline{c}\}. \end{aligned}$$

Finally, two join reconfigurations are applied with arguments $(I_{sp} \cup O_{sp}) \setminus E = \{\underline{c}, \underline{d}\}$ and $(E \setminus \{\underline{fgh}\}) \cup \{\underline{a}\} = \{\underline{a}, \underline{e}\}$, respectively. Their effects are depicted in Figure 6.3 (2) and (3). The final coordination pattern is $\rho_{s_{kept}}$. ✠



$\rho_1 = \rho_s \circlearrowleft \mathsf{split}_{\underline{cde}}$ (1) $\quad \rho_2 = \rho_1 \circlearrowleft \mathsf{join}_{\{\underline{c},\underline{d}\}}$ (2) $\quad \rho_{s_{kept}} = \rho_2 \circlearrowleft \mathsf{join}_{\{\underline{a},\underline{e}\}}$ (3)

Figure 6.3: Step-by-step example of moveP reconfiguration.

In spite of the apparently complex definition of some of these reconfiguration patterns, they all arise as sequential applications of primitive reconfigurations. Lemma 6.5 makes this observation precise.

Lemma 6.5. *Each reconfiguration pattern arises as product of a sequence of primitive reconfigurations.*

Proof.

- An inductive argument establishes the result for $\mathsf{removeP}(C)$:
  - Base case: $C = \emptyset$. By definition, $\rho \circlearrowleft \mathsf{removeP}(\emptyset) = rS(\rho, \emptyset) = \rho$.
  - Inductive case: $C \neq \emptyset$. Let $C = \{c\} \cup C'$. Assume that $rS(\rho, C') = \rho \circlearrowleft r$, where r is a sequence of remove reconfiguration primitives. By definition, $\rho \circlearrowleft \mathsf{removeP}(C) = rS(\rho, C) = rS(\rho \circlearrowleft \mathsf{remove}_c, \{c\} \cup C')$. Using the hypothesis we obtain $(\rho \circlearrowleft \mathsf{remove}_c) \circlearrowleft r$, which is equal to $\rho \circlearrowleft \{\mathsf{remove}_c \ ; \ r\}$. Therefore, the pattern may be expressed as a sequence of remove primitives.

- For pattern $\mathsf{overlapP}(\rho_r, X)$ the proof is also by induction (over set X):

  - Base case: $X = \emptyset$. By definition, $\mathsf{overlapP}(\rho_r, \emptyset) = sO(\rho \circlearrowleft \mathsf{par}_{\rho_r}, \emptyset) = \rho \circlearrowleft \mathsf{par}_{\rho_r}$.
  - Inductive case: $X \neq \emptyset$. Let $X = \{x\} \cup X'$. Assume, as induction hypothesis, that $rO(\rho, X') = \rho \circlearrowleft r$, where r is a sequence of reconfiguration primitives. By definition, $\rho \circlearrowleft \mathsf{overlapP}(\rho_r, X) = rO(\rho \circlearrowleft \mathsf{par}_{\rho_r}, \{x\} \cup X')$, and yet, it is equivalent to $rO(((\rho \circlearrowleft \mathsf{par}_{\rho_r}) \circlearrowleft \mathsf{join}_{\{\pi_1(x),\pi_2(x)\}}), X')$. Using the hypothesis we get $((\rho \circlearrowleft \mathsf{par}_{\rho_r}) \circlearrowleft \mathsf{join}_{\{\pi_1(x),\pi_2(x)\}}) \circlearrowleft r$.This is equivalent to

    $$\rho \circlearrowleft \{\mathsf{par}_{\rho_r} \ ; \ \mathsf{join}_{\{\pi_1(x),\pi_2(x)\}} \ ; \ r\}$$

    which encodes the $\mathsf{overlapP}$ pattern as a sequence of reconfiguration primitives.

- For pattern $\mathsf{insertP}(\rho_r, n, m_i, m_o)$ the result comes directly from the definition. Let $I_{sp} = I((\rho \circlearrowleft \mathsf{par}_{\rho_r}) \circlearrowleft \mathsf{split}_n) \setminus I(\rho \circlearrowleft \mathsf{par}_{\rho_r})$ and $O_{sp} = O((\rho \circlearrowleft \mathsf{par}_{\rho_r}) \circlearrowleft \mathsf{split}_n) \setminus O(\rho \circlearrowleft \mathsf{par}_{\rho_r})$. Then,

$$\begin{aligned}
&\rho \circlearrowleft \mathsf{insertP}(\rho_r, n, m_i, m_o) \\
&\quad = \ (((\rho \circlearrowleft \mathsf{par}_{\rho_r}) \circlearrowleft \mathsf{split}_n) \circlearrowleft \mathsf{join}_{O_{sp} \cup \{m_i\}}) \circlearrowleft \mathsf{join}_{I_{sp} \cup \{m_o\}}) \\
&\quad = \ \rho \circlearrowleft \{\mathsf{par}_{\rho_r} \ ; \ \mathsf{split}_n \ ; \ \mathsf{join}_{O_{sp} \cup \{m_i\}} \ ; \ \mathsf{join}_{I_{sp} \cup \{m_o\}}\}
\end{aligned}$$

- The proof for $\mathsf{replaceP}(\rho_r, X, C)$ is also a consequence of its definition and the fact that this Lemma holds for both $\mathsf{removeP}$ and $\mathsf{overlapP}$. Therefore, assuming $r_1 = \mathsf{removeP}(C)$ and $r_2 = \mathsf{overlapP}(\rho_r, X)$, with r_1 and r_2 sequences of primitive reconfigurations, one gets

$$\begin{aligned}
&\rho \circlearrowleft \mathsf{replaceP}(\rho_r, X, C) \\
&\quad = \ (\rho \circlearrowleft \mathsf{removeP}(C)) \circlearrowleft \mathsf{overlapP}(\rho, X) \\
&\quad = \ (\rho \circlearrowleft r_1) \circlearrowleft r_2 \\
&\quad = \ \rho \circlearrowleft \{r_1 \ ; \ r_2\}
\end{aligned}$$

- For $\mathsf{implodeP}(C)$ the proof is similar to the previous one. Assume that $r = \mathsf{removeP}(C)$, where r is a sequence of reconfiguration primitives, because this lemma holds for $\mathsf{removeP}$. Let $\rho_1 = \rho \circlearrowleft \mathsf{removeP}(C)$ Therefore,

$$\begin{aligned}
&\rho \circlearrowleft \mathsf{implodeP}(C) \\
&\quad = \ (\rho \circlearrowleft \mathsf{removeP}(C)) \circlearrowleft \mathsf{join}_{\mathcal{N}_{\rho_1} \setminus \mathcal{N}_\rho}
\end{aligned}$$

$$
\begin{aligned}
&= (\rho \circlearrowleft r) \circlearrowleft \mathsf{join}_{\mathcal{N}_{\rho_1} \setminus \mathcal{N}_\rho} \\
&= \rho \circlearrowleft \{r \ ; \ \mathsf{join}_{\mathcal{N}_{\rho_1} \setminus \mathcal{N}_\rho}\}
\end{aligned}
$$

- Finally, for $\mathsf{moveP}(ch, e, n)$ let $e' = \mathfrak{N}_\rho^{ch} \setminus \{e\}$, $\rho_1 = \rho \circlearrowleft \mathsf{split}_e$; $I_{sp} = I(\rho_1) \setminus I(\rho))$; $O_{sp} = O(\rho_1) \setminus O(\rho))$ and $E = \mathfrak{N}_{\rho_1}^{ch}$. Then,

$$
\begin{aligned}
&\rho \circlearrowleft \mathsf{moveP}(ch, e, n) \\
&\qquad = (\rho_1 \circlearrowleft \mathsf{join}_{(I_{sp} \cup O_{sp}) \setminus E}) \circlearrowleft \mathsf{join}_{(E \setminus \{e'\}) \cup \{n\}} \\
&\qquad = \rho \circlearrowleft \{\mathsf{split}_e \ ; \ \mathsf{join}_{(I_{sp} \cup O_{sp}) \setminus E} \ ; \ \mathsf{join}_{(E \setminus \{e'\}) \cup \{n\}}\}.
\end{aligned}
$$

□

6.3 Supporting dynamic reconfigurations via consistent state transfer

The application of reconfigurations to the architecture of a software system at runtime is a non-trivial problem, because a reconfiguration has to be transparently applied, while the exact system execution state in which such a reconfiguration is required (henceforth referred to as the *interrupted state*), is hardly known *a priori*.

Concretely, difficulties arise due to the fact that the system has to change its internal configuration without disruption during and after reconfigurations. This entails (*i*) the *fast* application of reconfigurations; (*ii*) the atomic application of reconfigurations with roll-back mechanisms triggered when such application fails; (*iii*) resuming the execution of the system in a state that is consistent (as much as possible) with the interrupted state; and (*iv*) keeping the system in line with its functional and non-functional requirements.

In the static perspective, the interrupted state is, usually, either ignored or always assumed to be the initial one. After reconfigurations, the system is again in its initial state. Ensuring system consistency on the static perspective of reconfigurations is not a challenge. It must be taken seriously, though, when dynamism enters the equation. Consistently transferring system state is the *de facto* question addressed in this section.

6.3.1 Symbolic states

Coordination patterns essentially provide a structural (static) dimension of the modelled system. In order to have a behavioural and dynamic perspective, one has to

consider a concrete behaviour associated to the channels of the coordination pattern. In concrete, this is achieved by assuming a concrete semantic model that, in turn, underpins a coordination model. As stated before, Reo is chosen in this thesis to type the edges of coordination patterns. For this section, a slightly modified version of CA (actually *port automata (PA)* [171], which is as CA, but with all guards assumed to be true) is the semantic model used to endow coordination pattern with semantics.

The approach for consistent state transfer proposed here requires semantic models based on automata. Moreover, it takes a notion of symbolic state that must enrich the traditional semantic model considered. Symbolic state annotations are generated by the following grammar $\mathcal{S}$:

$$s \ni \varsigma \mid \neg s \mid s \wedge s,$$

where ς is an atomic symbolic state. An atomic symbolic state refers to the name of a coordination pattern channel to which data is assigned. In the concrete case of Reo, these identifiers refer to the edges typed with a fifo channel (either empty or full), as this is the only stateful channel in the subset of Reo channels considered in $\mathcal{T}$.

Note that, although the notation is borrowed from logic, the meaning is entirely different. Actually, $\neg\varsigma$ means that the channel identified by ς has no data assigned (and therefore can be omitted from the formula) and $\varsigma_1 \wedge \varsigma_2$ means that both channels have data in the context of the pattern. Moreover, it is asserted that

- $\neg\varsigma_1 \wedge \varsigma_1 = \neg\varsigma_1$
- $\neg(\varsigma_1 \wedge \varsigma_2) = \neg\varsigma_1 \wedge \neg\varsigma_2$.

Additionally, consider $\bot_\rho$ to express that there is no data in any internal state of a coordination pattern ρ and $\top_\rho$ for its dual. Subscripts may be omitted when clear from the context.

Consider now the modification of PA to cope with symbolic states, denoted henceforth as PA_ς.

Notation

$[\![\rho]\!]_{\mathcal{M}}$ is used, henceforth, to refer to the behaviour of a coordination pattern ρ under the semantic model $\mathcal{M}$.

Definition 6.9 (Symbolic Port Automata)**.** *A symbolic port automaton $\mathcal{A}_\varsigma$ is an automaton $(Q, P, \rightarrow, q_0)$, where $Q \subseteq \mathcal{S}$ is a set of symbolic states, P is a set of ports, $q_0 \in Q$ is the initial state, and $\rightarrow \subseteq Q \times 2^P \times Q$ is a transition relation.*

Auxiliary operation $\mathsf{IS}(\llbracket\rho\rrbracket_{\mathsf{PA}_\varsigma})$ is used to obtain the initial state of the symbolic port automaton associated to $\rho \in \mathcal{P}$, with the particular case $\mathsf{IS}(\llbracket\mathbf{0}_\rho\rrbracket_{\mathsf{PA}_\varsigma}) = \bot_\rho$.

6.3.2 State transfer

The state transfer operation is defined as follows.

Definition 6.10 (State Transfer)**.** *Let ρ be a coordination pattern, $S_r \in \mathcal{S}$ the symbolic interrupted state in reconfiguration $r = \{r_0; r_1; \ldots; r_n\}$ (for each $r_i \in Prim$). The state transfer operation for $\rho \circlearrowleft r$ at state S_r, denoted by $\circledcirc^\rho_{S_r}$, is inductively defined as $\circledcirc^\rho_{S_{r_0}} \wedge \circledcirc^\rho_{S_{\{r_1,\ldots,r_n\}}}$, where for each r_i:*

$$\circledcirc^\rho_{S_{r_i}} = \begin{cases} \mathsf{IS}(\llbracket\rho'\rrbracket_{\mathsf{PA}_\varsigma}) & \text{if } r_i = \mathsf{const}_{\rho'} \\ S_{\mathsf{par}_{\rho'}} \wedge \mathsf{IS}(\llbracket\rho'\rrbracket_{\mathsf{PA}_\varsigma}) & \\ S_{\mathsf{remove}_c} \wedge \neg c & \text{if } c \in \mathfrak{I}_\rho \wedge \pi_3(c) = \mathsf{fifo} \\ S_{r_i} & \text{otherwise} \end{cases}$$

This can be generalised as follows. Assume a reconfiguration r; a (possibly empty) coordination pattern ρ_{in} formed either by (i) all patterns introduced by par primitives in r or (ii) the pattern introduced by the last const primitive and all patterns introduced by the sequent par primitives in r; a coordination pattern ρ_{out} as the result of applying r to ρ; and finally $R(\rho, \rho_{out})$ as the set of *stateful* channel names removed during the reconfiguration. Then,

$$S_r \wedge \mathsf{IS}(\llbracket\rho_{in}\rrbracket_{\mathsf{PA}_\varsigma}) \wedge \neg \bigwedge R(\rho, \rho_{out})$$

is the generalisation of $\circledcirc^\rho_{S_r}$. The obtained state from this operation is referred to as the *resuming state*.

There are situations, though, in which it is not possible to find a suitable resuming state on the new configuration. When such is the case, the usual approach is to start the execution of the reconfigured system from its initial state. This symbolic-based approach is more comprehensive on this aspect: it automatically delivers the state that best approximates the desired one in an attempt of minimising data loss.

Example 6.11 Let $\rho_{proactive}$ stand for the ProActiveWaitingSequencer coordination pattern shown in Figure 6.2. Consider that it is the model of a coordination activity in some software system. The PA_ς underlying $\rho_{proactive}$ is shown (replicated) in the first row of Figure 6.4.

For some reason the system is required to evolve into a normal Sequencer coordination pattern. Let ρ_s refer to it. The $\llbracket\rho_s\rrbracket_{\mathsf{PA}_\varsigma}$ is represented (replicated) in the third row of

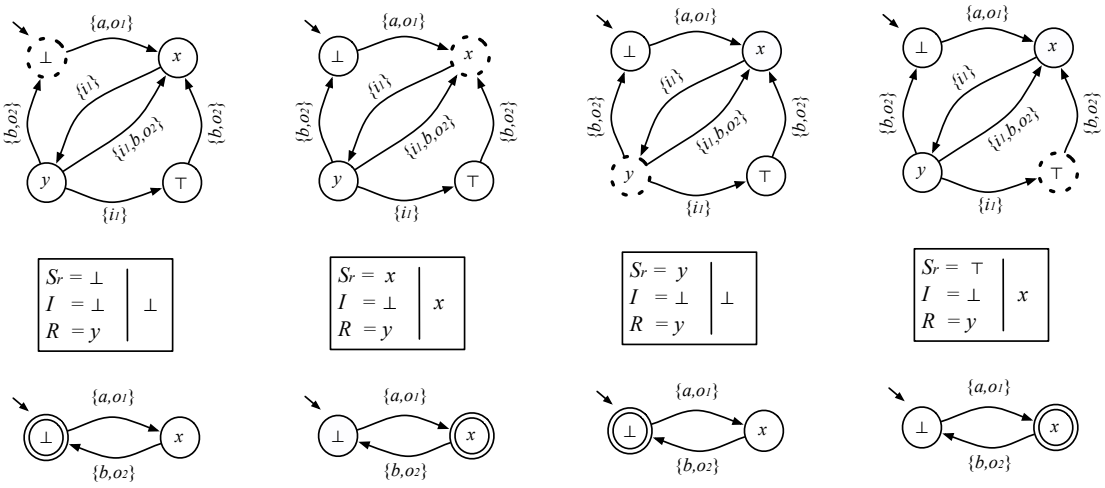


Figure 6.4: Example of dynamic reconfigurations and state transfer.

Figure 6.4. Such evolution is achieved by applying reconfiguration $r = \mathsf{implodeP}(\{sd, y\})$ to the running system.

On a dynamic setting, reconfiguration can occur at any point in time. Consequently, the system execution can be interrupted in any of the four possible dashed states in Figure 6.4, first row.

The tables in between the automata present values for S_r, the state interrupted for application of the reconfiguration r; I, the initial state of the structure added to the pattern; and R, the conjunction of the identifiers of stateful channels (fifo channels in this case) removed from the original pattern. These are the necessary ingredients to apply the general state transfer operation in order to obtain the desired *resuming states* (double-circled states in the third row of Figure 6.4).

Recall that $r = \{\mathsf{removeP}(\{sd, y\}); \mathsf{join}_{\{f,g\}}\}$. Therefore, the only stateful channel removed is exactly identified by y. From here, $R = \bigwedge R(\rho_{proactive}, \rho_s) = y$, and no patterns are added to the original pattern: $I = \mathsf{IS}(\llbracket \mathbf{0}_\rho \rrbracket_{\mathsf{PA}_\varsigma}) = \bot_{\mathbf{0}_\rho}$.

Let's discuss the four situations depicted in Figure 6.4, from left to right, in more detail. In the first situation the reconfiguration is applied when the pattern is in its initial state. In this case it is $\bot$, meaning that no data is assigned to stateful channels of the pattern. The resuming state is trivially $\bot$ in the new configuration. In the second situation the reconfiguration is applied when the system is in symbolic state x. Hence, the resuming state is $x \wedge \bot_{\mathbf{0}_\rho} \wedge \neg y = x$.

In the third situation the (desired) resuming state would be y. But, since y is removed during reconfiguration, the best approximated resuming state is the initial $\bot = y \wedge \bot_{\mathbf{0}_\rho} \wedge \neg y$. In this case, data is hopelessly lost. For the same reason, in the fourth case, the interrupted state can not be resumed as is. However, in this case, the best approximated state is $x = \top \wedge \bot_{\mathbf{0}_\rho} \wedge \neg y$, where $\top$ in this case is $x \wedge y$. ✠

6.4 Reconfigurations on the stochastic setting

This section considers coordination patterns with stochastic behaviour, deployed in environments (independent of the coordination patterns) that also present stochastic information. Reconfigurations are revisited under these settings.

6.4.1 Stochastic coordination patterns

Recall that a coordination pattern is a graph of channels, where channels are entities for data transmission and nodes are locations for data synchronisation and interaction with other patterns or external services. Each channel is further defined by its ends, an identifier and a type conceding it a specific behaviour.

The incorporation of stochastic information into coordination patterns necessarily requires a revision of the channel model. In particular, delays for both reading and writing data to the ends of a channel and processing data between these ends should be considered. This leads to a notion of stochastic coordination pattern, formalising an abstract model of the component-based one informally developed for stochastic Reo in Section 4.3.1. In this way, stochastic Reo can type the edges of a stochastic coordination pattern as much as Reo does for plain coordination patterns.

For keeping consistency with the exposed in Section 4.3.1, the delays are considered to be non-negative real values. They define stochastic values describing the probability of a certain stochastic variable. Concretely, a delay is a rate associated to an exponential distribution that models the time mediating the occurrence of two consecutive events.

Definition 6.11 (Stochastic channel)**.** *A stochastic channel is a tuple*

$$c_{sto} = \langle \mathcal{S}_{sto}, i, t, \mathcal{K}_{sto}, \mathcal{F} \rangle,$$

where $i \in \mathcal{I}$, $t \in \mathcal{T}$, $\mathcal{S}_{sto}, \mathcal{K}_{sto} \subseteq \mathcal{E} \times \mathbb{R}^+$ *such that* $2^{\pi_1}(\mathcal{S}_{sto}) \cap 2^{\pi_1}(\mathcal{K}_{sto}) = \emptyset$, *and* $\mathcal{F} \subseteq 2^{\mathcal{E}} \times \mathbb{R}^+$.

Informally, each element in $\mathcal{S}_{sto}$ defines a source end of the channel with a delay for writing data into the channel. Dually, elements of $\mathcal{K}_{sto}$ define the sink ends of the channel with their delay for reading data from the channel. $\mathcal{F}$ defines a set of flows, where its first component describes which ends are involved in the data transmission and the second component describes the time required to transmit data between those ends.

Example 6.12 Consider a lossy channel as depicted in Figure 2.1. One possible stochastic

instance of this channel may be

$$l_{sto} = \langle\{\langle a, 1000\rangle\}, l_1, \mathsf{lossy}, \{\langle b, 500\rangle\}, \{\langle\{a, b\}, 450\rangle, \langle\{a\}, 600\rangle\}\rangle.$$

This means that channel l_{sto} is able to write 1000 requests into end a, and to read 500 requests into end b. Moreover, it is able to transmit 450 request between ends a and b and it may lose data at a rate of 600[1]. ✠

As expected, infinite many instances of a stochastic channel are possible. For clarity, though, it will be assumed that each instance has different end names.

The definition for a stochastic coordination pattern does not deviate much from the plain one. It remains a pair with a set of channels and a set of nodes. Formally,

Definition 6.12 (Stochastic coordination pattern)**.** *A stochastic coordination pattern is a pair*

$$\rho = \langle \mathcal{C}_\rho^{sto}, \mathcal{N}_\rho^{sto} \rangle$$

where $\mathcal{C}_\rho^{sto}$ is a set of stochastic channels and $\mathcal{N}_\rho^{sto}$ is a partition on the union of all ends of all channels in $\mathcal{C}_\rho^{sto}$, this is, generically, $\mathcal{N}^{sto} = 2^{\mathcal{E}\times\mathbb{R}^+}$.

The invariant properties of a coordination pattern are still required in its stochastic version. The operations associated to channels and coordination patterns for accessing their components suffer the obvious lift to the stochastic version. Denote by $\mathcal{P}^{sto}$ the set of all stochastic coordination patterns.

Each node will now delay the flow of data. This is so because it must enqueue and dequeue data from and to the channels it synchronises. These delays are, of course, related to the write and read delays associated to each end composing the node.

Definition 6.13 (Node delays)**.** *Let $\rho \in \mathcal{P}^{sto}$ and $n \in \mathcal{N}_\rho^{sto}$. The enqueue and dequeue delays associated to n, denoted n_{enq} and n_{deq} are given, respectively, by the following functions:*

$$\eth_n^{\mathsf{enq}} = \frac{\sum_{e\in n} \begin{cases} \pi_2(e) & \text{if} \quad out(\pi_1(e), \rho) \\ 0 & \text{otherwise} \end{cases}}{|\{e \in n |\ out(\pi_1(e), \rho)\}|}$$

and

$$\eth_n^{\mathsf{deq}} = \mathsf{min}(\{\pi_2(e) |\ e \in n \wedge in(\{\pi_1(e)\}, \rho)\})$$

Function $\eth_n^{\mathsf{enq}}$ computes the average rate from the rates of all sink ends co-located in n; $\eth_n^{\mathsf{deq}}$ computes the minimum rate from the rates of all source ends co-located in

[1] All these values are read as *the possible maximum per unit of time*, whatever the time unit is.

n. The choice of these functions are in respect to the usual behaviour of nodes in Reo. Merger nodes take just one of its inputs, therefore, the average rate associated to these inputs provides the right tool to define the average delay that the node should take to enqueue data. Replicator nodes replicate data to all channels, therefore the average delay that the node should take to dequeue data is given by the highest average delay of the end co-located in the node, which is given by the smallest rate in the set of rates.

Clearly, these rates define just an approximation to the real stochastic process that arises from the composition of the exponential distributions. In any case, other functions may be used to obtain similar rates. These functions become a parameter in the definition of a stochastic coordination pattern.

Example 6.13 A stochastic instance ρ_s^{sto} of a Sequencer coordination pattern (with $\eth_n^{\mathsf{enq}}$ and $\eth_n^{\mathsf{deq}}$ functions associated to each $n \in \mathcal{N}_{\rho_s^{sto}}^{sto}$) may be defined as follows:

$$\rho_s^{sto} = \left\langle \left\{ \begin{array}{l} \left\langle \begin{array}{c} \{\langle a, 1000\rangle\}, s_1, \mathsf{sync}, \{\langle c, 955\rangle\}, \\ \{\langle \{a, c\}, 100\rangle\} \end{array} \right\rangle, \\ \left\langle \begin{array}{c} \{\langle e, 900\rangle\}, x, \mathsf{fifo_e}, \{\langle f, 800\rangle\}, \\ \{\langle \{e\}, 90\rangle, \langle \{f\}, 80\rangle\} \end{array} \right\rangle, \\ \left\langle \begin{array}{c} \{\langle h, 1000\rangle\}, s_4, \mathsf{sync}, \{\langle b, 955\rangle\}, \\ \{\langle \{h, b\}, 100\rangle\} \end{array} \right\rangle \end{array} \right\}, \left\{ \begin{array}{l} \left\langle \begin{array}{c} \{\langle d, 1000\rangle\}, s_2, \mathsf{sync}, \{\langle o_1, 1000\rangle\}, \\ \{\langle \{d, o_1\}, 150\rangle\} \end{array} \right\rangle, \\ \left\langle \begin{array}{c} \{\langle g, 1000\rangle\}, s_3, \mathsf{sync}, \{\langle o_2, 1000\rangle\}, \\ \{\langle \{g, o_2\}, 150\rangle\} \end{array} \right\rangle \end{array} \right\}, \left\langle \right\rangle \right\rangle$$
$$\{\underline{a}, \underline{o_1}, \underline{o_2}, \underline{b}, \underline{cde}, \underline{fgh}\}^2$$

Now, one can compute the delays associated to the nodes. For instance, $\underline{cde}_{enq} = 955$ and $\underline{cde}_{deq} = 900$; in turn, $\underline{fgh}_{enq} = 800$ and $\underline{fgh}_{deq} = 1000$. ✠

In order to completely define the stochastic model for coordination patterns it is still missing the stochastic information about the environment. This information is considered, however, independent of the coordination pattern. In this context, a notion of stochastic environment is introduced as a map $\mathcal{E}nv : \mathcal{N} \mapsto \mathbb{R}^+$.

Each entry of $\mathcal{E}nv$ is then a key-value pair, where the key is a node corresponding to a port of a stochastic coordination pattern to which the environment may be connected; and the value is a positive real value defining the rate at which the environment issues requests to the associated port. In practice, each element of $\mathcal{E}nv$ may correspond, for instance, to a service.

The connection of a stochastic coordination pattern ρ to an environment $\mathcal{E}nv$ is regarded as the *deployment* of that pattern and the composite is denoted $\rho_{\Delta^{\mathcal{E}nv}}$. *Deployment* appears emphasised to denote that it does not necessarily mean the dynamic installation and activation of a coordination pattern in an execution envi-

[2]In order not to burden notation, the stochastic version of nodes is denoted as before.

ronment. Rather it is just a conceptual notion that enables both static and dynamic analysis of a system performance.

When deployed, all ports of the stochastic pattern should have a corresponding port on the environment. The environment may, however, define more ports besides those of the stochastic coordination pattern.

Example 6.14 Consider the stochastic Sequencer coordination pattern presented in Example 6.13. A possible environment for this pattern is, for instance,

$$\mathcal{E}nv = \begin{Bmatrix} \underline{a} \mapsto 50, \\ \underline{b} \mapsto 50, \\ \underline{o_1} \mapsto 30 \\ \underline{o_2} \mapsto 25 \end{Bmatrix}$$

✠

6.4.2 Reconfigurations revisited

Since the essential structure of a stochastic coordination pattern does not change from its plain version, the definition of each primitive reconfiguration operation also remains the same. This is true for the coordination pattern alone. However, in the stochastic setting, reconfigurations are also applied upon the deployed coordination pattern, *i.e.,* the environment also enters the equation. In this section, one will investigate how each primitive reconfiguration deal with the environment.

Environment reconfiguration

Specific environment operations are assumed to exist that manipulate such an entity. These operations follow generic patterns as advocated in the literature [211, 133, 148]. In this case, three such operations are assumed.

Definition 6.14 (Environment manipulation)**.** *Let $\mathcal{E}nv : \mathcal{N} \mapsto \mathbb{R}^+$ be an environment, $n \in \mathcal{N}$ and $\gamma \in \mathbb{R}^+$. The following operations are associated to the environment:*

- $\mathsf{add}_{n,\gamma}^{\mathcal{E}nv}$ *defines* $\mathcal{E}nv(n) = \gamma$, *if* $n \notin \mathsf{dom}(\mathcal{E}nv)$;
- $\mathsf{del}_{n}^{\mathcal{E}nv}$ *removes* n *from* $\mathsf{dom}(\mathcal{E}nv)$, *if* $n \in \mathsf{dom}(\mathcal{E}nv)$;
- $\mathsf{upd}_{n,\gamma}^{\mathcal{E}nv}$ *updates* $\mathcal{E}nv(n)$ *to* γ, *if* $n \in \mathsf{dom}(\mathcal{E}nv)$.

If the environment is regarded as a set of associated services with their stochastic values, then these operations represent the addition, removal and replacement of

services to an architecture. When the environment is associated to some coordination pattern, the add and upd operations perform attachments between the ports of the services and the coordination pattern (if the pattern defines such ports) while the del operation removes attachments between these ports.

Coordination patterns reconfiguration

In the stochastic setting, reconfigurations are applied over patterns *deployed* in an environment. So, reconfigurations in $Prim$ are revisited as follows.

Reconfigurations const and par are trivial. The former replaces the original coordination pattern and the environment by the given arguments; the latter performs the component-wise union of both coordination patterns and environments. Recall that each stochastic instance of a coordination pattern assumes different names for channel identifiers and ends.

const. Let $\rho, \rho' \in \mathcal{P}^{sto}$ and $\mathcal{E}nv, \mathcal{E}nv'$ be two environment definitions. Then,

$$\rho_{\Delta^{\mathcal{E}nv}} \circlearrowleft \mathsf{const}_{\rho'_{\Delta^{\mathcal{E}nv'}}} = \rho'_{\Delta^{\mathcal{E}nv'}}$$

par. Let $\rho, \rho' \in \mathcal{P}^{sto}$ and $\mathcal{E}nv, \mathcal{E}nv'$ be two environment definitions. Then,

$$\rho_{\Delta^{\mathcal{E}nv}} \circlearrowleft \mathsf{par}_{\rho'_{\Delta^{\mathcal{E}nv'}}} = (\rho \circlearrowleft \mathsf{par}_{\rho'})_{\Delta^{\mathcal{E}nv \cup \mathcal{E}nv'}}$$

The join primitive operation performs connections in the coordination pattern via a given set of nodes into a single one, as before, and changes the environment by removing from it all the ports that were part of the original coordination pattern interface.

join. Let $\rho \in \mathcal{P}^{sto}$, $\mathcal{E}nv$ be an environment definition and $N \subseteq \mathcal{N}^{sto}$. Then,

$$\rho_{\Delta^{\mathcal{E}nv}} \circlearrowleft \mathsf{join}_N = (\rho \circlearrowleft \mathsf{join}_N)_{\Delta^{\mathcal{E}nv'}}$$

where $\mathcal{E}nv' = \mathsf{del}^{\mathcal{E}nv}_{\pi_1(n)}$, for all $n \in N$.

The split reconfiguration is also as before with respect to the coordination pattern structure. It changes the environment, however, by adding new connections to dummy services that issue infinite many requests per unit of time. The rate of these services may be then updated via upd environment operation.

split. Let $\rho \in \mathcal{P}^{sto}$, $\mathcal{E}nv$ be an environment definitions and $n \in \mathcal{N}^{sto}$. Then,

$$\rho_{\Delta^{\mathcal{E}nv}} \circlearrowleft \mathsf{split}_n = (\rho \circlearrowleft \mathsf{split}_n)_{\Delta^{\mathcal{E}nv'}}$$

where $\mathcal{E}nv' = \mathsf{add}^{\mathcal{E}nv}_{\{e\},\infty}$, for all $e \in 2^{\pi_1} n$.

Finally, the remove primitive operation also performs as before in regards to the coordination pattern. It changes the environment in two ways: first it removes all connections to elements of the original interface that are not part of the coordination pattern interface; and second, it adds connections to the new ports as much as split does.

remove. Let $\rho \in \mathcal{P}^{sto}$, $\mathcal{E}nv$ be an environment definitions and $c \in \mathcal{I}_\rho$. Then,

$$\rho' = \rho_{\Delta^{\mathcal{E}nv}} \circlearrowleft \mathsf{remove}_c = (\rho \circlearrowleft \mathsf{remove}_c)_{\Delta^{\mathcal{E}nv'}}$$

where $\mathcal{E}nv' = \mathsf{add}^{\mathcal{E}nv''}_{\{e\},\infty}$, for all $e \in 2^{\pi_1}(IO(\rho') \setminus IO(\rho))$ and $\mathcal{E}nv'' = \mathsf{del}^{\mathcal{E}nv}_{\{e\}}$, for all $e \in 2^{\pi_1}(IO(\rho) \setminus (IO(\rho) \cap IO(\rho')))$.

6.5 Summary

This chapter introduced ARIS, a framework for architectural reconfiguration of interacting services. ARIS targets reconfigurations at the coordination level rather than at the level of the individual components. It is based on two foundational notions: coordination and reconfiguration patterns.

Coordination patterns define the coordination layer of a system. A coordination pattern is formally defined as an abstract graph of communication primitives, referred to as channels. Edges of the graph are, then, channels labelled with a unique identifier and a type. The type incorporates a coordination policy (*i.e.,* the behaviour) into a channel. Each channel has two or more ends which originate the nodes of the graph. The nodes of the coordination pattern graph represent, thus, locations for interaction of channels, other coordination patterns, or external components.

A reconfiguration is taken as any change made to the structure of a coordination pattern. Five primitive reconfiguration operations were considered to guide such changes. Relevant properties about these operations were discussed.

Reconfigurations were also studied in the dynamic setting, where the concern of preservation of system consistency on state transfer was addressed. To deal with this, a state transfer approach based on a symbolic view of states was presented.

Finally, coordination patterns were extended to embody stochastic information. To this end, the structure of channels suffer a small change. Channel ends were enriched with a stochastic value that describes delays in reading/writing data from/to the channel. In turn, channels were enriched with stochastic values describing the delays in processing data between ends. The stochastic values in the channel ends were used to define delays for enqueueing and dequeueing data in the coordination pattern nodes. However, the overall structure of a coordination pattern remains the same. This is desirable since the basic reconfiguration operational behaviour does not need to change. In order to complete the whole stochastic model, a notion of environment (as a map from coordination pattern ports to service request delays) was introduced. This led to a conceptual notion of coordination pattern *deployment*, which joins both pattern and environment in a unique reconfigurable structure. The primitive reconfiguration operations were revisited in this new setting.

Chapter 7

Reasoning about reconfigurations

Comparison is the death of joy.

– *Mark Twain*

In this chapter. Three criteria for reasoning about reconfigurations are investigated. Their objective is to provide means to rule out configurations and reconfigurations that, for instance, fail to preserve some system invariant property. One criterium focuses on a behaviour perspective; the other is complementary and addresses structural concerns; the third considers performance notions. These criteria aim at enabling the working software architect to compare reconfigurations and choose among them. To help on this, a taxonomy is introduced for classifying reconfigurations.

Part of this chapter's content was previously published, by the author, in:

- Nuno Oliveira and Luís S. Barbosa. "On the reconfiguration of software connectors". In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. Vol. 2. SAC '13. Coimbra, Portugal: ACM, Mar. 2013, pp. 1885–1892.

7.1 Behavioural reasoning

The objective of this chapter is to discuss how to compare and classify reconfigurations. This comparison can be done in terms of the behaviour enforced on the coordination patterns to which the reconfigurations are applied to. So, it is intended to compare the behaviour of coordination patterns before and after a reconfiguration.

As reported before (in Section 6.3), in order to obtain a behavioural semantics for coordination patterns, one has to consider a concrete semantic model associated to the types of the channels, which must be compositional. Finally, suitable notions

of similarity and bisimilarity are needed. These must be studied in the context of the semantic models considered.

For illustration purposes, automata-based semantic models will be considered here, concretely, CA and *Reo automaton (RA)* (*c.f.,* Chapter 2). The choice of these two models is due to the fact that they are popular in Reo, and provide the required characteristics just mentioned. Clearly, the $\mathsf{IMC}_{\mathsf{Reo}}$ model presented in the first part of this dissertation could also be chosen: it is compositional and comparison notions are equally studied. The reason to take two models, instead of just one, is to show that the reasoning approach is independent of the fixed semantic model.

7.1.1 Comparing reconfigurations

Recall that $[\![\rho]\!]_{\mathcal{M}}$ stands for the semantics of the coordination pattern $\rho \in \mathcal{P}$ in the semantic model $\mathcal{M}$. Moreover, assume $\sim$ and $\preceq$ to represent the bisimilarity and similarity relations in model $\mathcal{M}$, respectively.

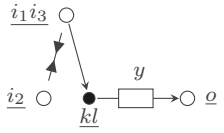
Reconfigurations can be compared for their effect when applied to any coordination pattern. But, $\mathcal{P}$ is uncountable. Therefore, it is hard to find any two interesting reconfigurations that produce the same effect when applied to any $\rho \in \mathcal{P}$. Only the identity reconfiguration $\mathbf{1}_r$ or the const primitive (with a suitable argument) would cater for such a behaviour.

More interesting is to compare reconfigurations with respect to their application to a specific coordination pattern.

Definition 7.1. *Let $\rho \in \mathcal{P}$, r_1, r_2 be reconfigurations and $\mathcal{M}$ a semantic model. Then,*

$$(r_1 \mathrel{\mathring{=}}_{\mathcal{M}} r_2)_\rho \quad \text{iff} \quad [\![\rho \circlearrowleft r_1]\!]_{\mathcal{M}} \sim [\![\rho \circlearrowleft r_2]\!]_{\mathcal{M}}$$
$$(r_1 \mathrel{\mathring{\preccurlyeq}}_{\mathcal{M}} r_2)_\rho \quad \text{iff} \quad [\![\rho \circlearrowleft r_1]\!]_{\mathcal{M}} \preceq [\![\rho \circlearrowleft r_2]\!]_{\mathcal{M}}$$

Example 7.1 Consider again the Sequencer coordination pattern in Figure 6.1. Suppose that a new requirement enforces a strict dependence between services. In particular, suppose that the second service (connected to port o_2) is launched with the output of the first service, and that such an output is memorised whenever the second service is not ready to consume it. Reconfiguration

$r_{dependent} = \mathsf{insertP}($  $, \underline{fgh}, \underline{i_2}, \underline{o})$

which is akin to $r_{proactive}$ (see Example 6.7), meets the envisaged requirement. Figure 7.1 presents the resulting pattern, which will be referred to as the ProActiveDependentSequencer. The corresponding RA semantic model is exactly the same of the coordination pattern ob-

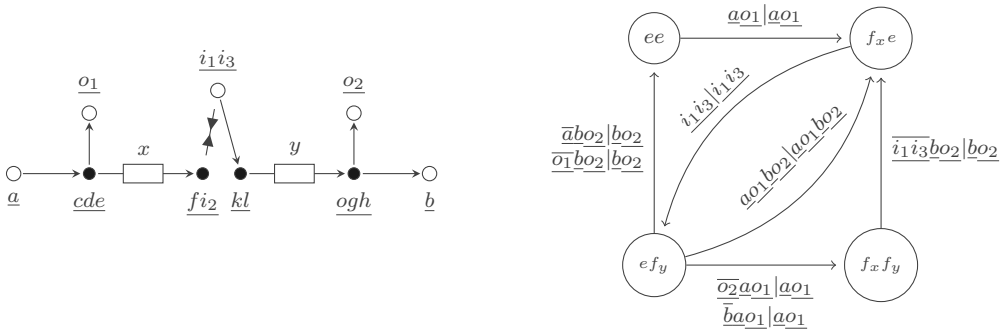


Figure 7.1: The ProActiveDependentSequencer pattern and corresponding Reo automaton.

tained through application of $r_{proactive}$, *i.e.,* the ProActiveWaitingSequencer. So, although both patterns are slightly different, they exhibit a bisimilar behaviour (up to port names), expressed in the Reo automata model. Therefore,

$$(r_{proactive} \stackrel{\circ}{=}_{RA} r_{dependent})_{\rho_s}$$

Using constraint automata as a semantic model, however, bisimilarity fails. Figure 7.2 shows the constraint automata for each case. The difference (highlighted in the dashed transition) is that in the ProActiveWaitingSequencer, data on the buffer y comes from buffer x; while in the ProActiveDependentSequencer data on buffer y comes from node $i_1 i_3$. ✠

This example highlights that the same reconfiguration may yield different comparison results when assessed over two different semantic models. It is the job of the architect to choose the semantics that better fulfils the objectives of the comparison.

At this moment, the attentive reader will conclude that the proposed comparisons do not deliver any insight about the relation between the compared reconfigurations and the original coordination pattern. Questions like *do these reconfigurations preserve the original behaviour?* or *do they change that behaviour?* cannot be answered this way. In order to obtain an answer for these questions, fine grained variants of both $\stackrel{\circ}{=}$ and $\preccurlyeq$ are needed. Such granularity is achieved by introducing an extra comparison dimension as follows; for $\stackrel{\circ}{=}$ one may consider whether

$$(r_1 \stackrel{\circ}{\Updownarrow}_{\mathcal{M}} r_2)_\rho \quad \text{iff} \quad (\mathbf{1}_r \stackrel{\circ}{=}_{\mathcal{M}} r_1 \stackrel{\circ}{=}_{\mathcal{M}} r_2)_\rho \tag{7.1}$$

$$(r_1 \stackrel{\circ}{\Uparrow}_{\mathcal{M}} r_2)_\rho \quad \text{iff} \quad (\mathbf{1}_r \preccurlyeq_{\mathcal{M}} r_1 \stackrel{\circ}{=}_{\mathcal{M}} r_2)_\rho \tag{7.2}$$

$$(r_1 \stackrel{\circ}{\Downarrow}_{\mathcal{M}} r_2)_\rho \quad \text{iff} \quad (r_1 \stackrel{\circ}{=}_{\mathcal{M}} r_2 \preccurlyeq_{\mathcal{M}} \mathbf{1}_r)_\rho; \tag{7.3}$$

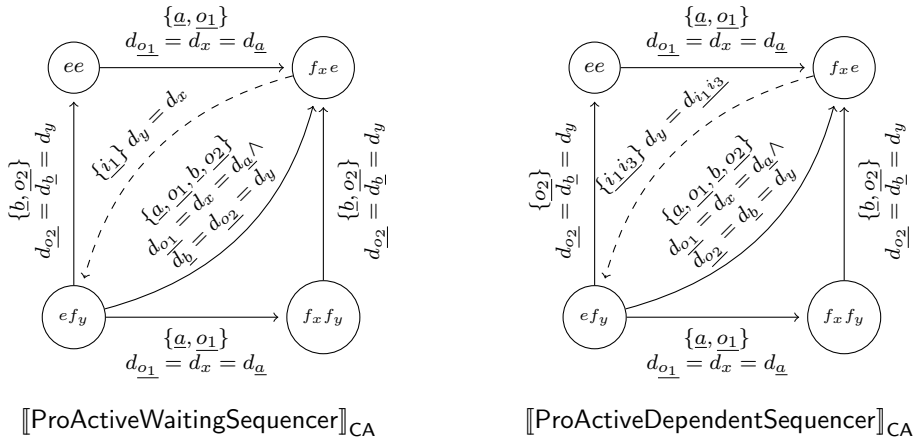


Figure 7.2: CA semantics for the ProActiveWaitingSequencer (left) and the ProActiveDependentSequencer (right).

Similarly, for $\preccurlyeq$,

$$(r_1 \not\succcurlyeq_{\mathcal{M}} r_2)_\rho \quad \text{iff} \quad (\mathbf{1}_r \not\preccurlyeq_{\mathcal{M}} r_1 \preccurlyeq_{\mathcal{M}} r_2)_\rho \tag{7.4}$$

$$(r_1 \not\preccurlyeq_{\mathcal{M}} r_2)_\rho \quad \text{iff} \quad (r_1 \not\preccurlyeq_{\mathcal{M}} \mathbf{1}_r \preccurlyeq_{\mathcal{M}} r_2)_\rho \tag{7.5}$$

$$(r_1 \not\succcurlyeq_{\mathcal{M}} r_2)_\rho \quad \text{iff} \quad (r_1 \not\preccurlyeq_{\mathcal{M}} r_2 \not\preccurlyeq_{\mathcal{M}} \mathbf{1}_r)_\rho. \tag{7.6}$$

This more fine-grained comparison allows to link the compared reconfigurations and their effect to the original coordination pattern. For instance, the three comparisons of $\stackrel{\circ}{=}$ highlight that the reconfigurations preserve the original behaviour (Equation 7.1); the reconfigurations change the original behaviour by adding new behaviour (Equation 7.2) and the reconfigurations change the original behaviour by removing part of it (Equation 7.3). The reading is similar for $\preccurlyeq$.

Example 7.2 Consider once again the Sequencer coordination pattern in Figure 6.1. Suppose that the results delivered by the second service are just a complement to those offered by the first. In this situation, whenever the second service fails, the system may proceed normally, disregarding port o_2. This requirement is met by applying the following reconfiguration:

$$r_{quasiweak} = \rho_s \circlearrowleft \mathsf{replaceP}(\ \underline{i} \circ\!\!- - - \!\!\to\!\circ\ \underline{o_2}\ , \{(\underline{fgh}, \underline{i})\}, \{s_3\})$$

which is actually part of the reconfiguration r_{weak} discussed in Example 6.8. The resulting coordination pattern (referred to as QuasiWeakSequencer) is a variant of the WeakSequencer. Their structure and semantic models are depicted in Figure 7.3.

In spite of their similarity, it is possible to see that the upper transition of the automata for the WeakSequencer has an extra guard expressing that data may be lost on the first lossy

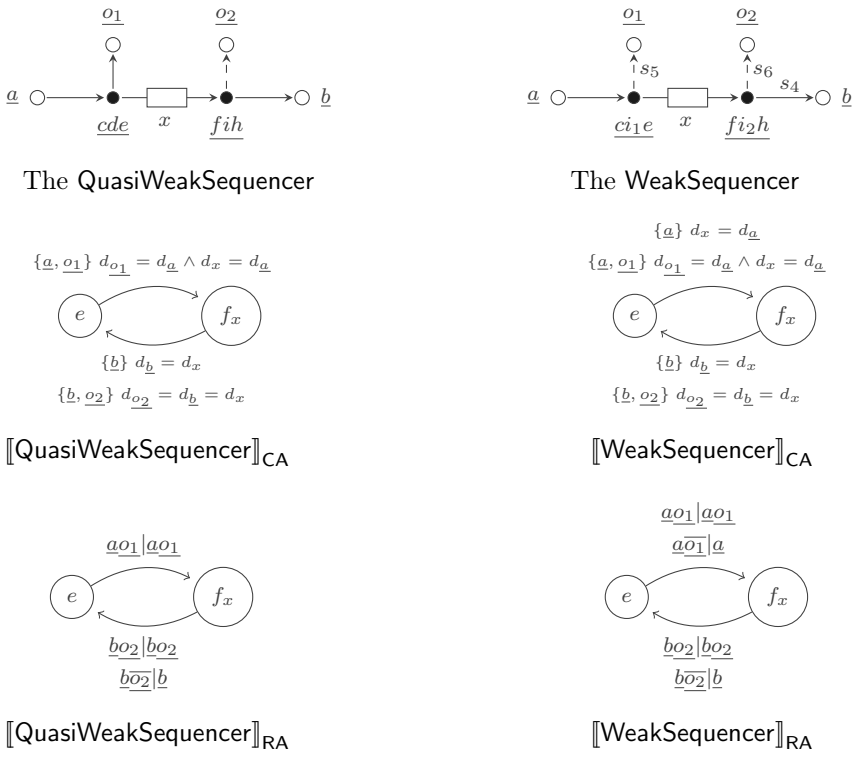


Figure 7.3: The QuasiWeakSequencer and the WeakSequencer semantics.

channel. Therefore, for $\mathcal{M} \in \{\mathsf{CA}, \mathsf{RA}\}$, $[\![\mathsf{QuasiWeakSequencer}]\!]_{\mathcal{M}} \preceq [\![\mathsf{WeakSequencer}]\!]_{\mathcal{M}}$. Thus, $(r_{quasiweak} \preccurlyeq r_{weak})_{\rho_s}$. But one may be more pretentious here: both patterns simulate the Sequencer, leading to $(\mathbf{1}_r \preccurlyeq_{\mathcal{M}} r_{quasiweak} \preccurlyeq_{\mathcal{M}} r_{weak})_{\rho_s}$, this is, in a fine grained comparison $(r_{quasiweak} \npreccurlyeq r_{weak})_{\rho_s}$. ✠

7.1.2 A behavioural classification of reconfigurations

It is possible to propose a classification of reconfigurations with respect to a coordination pattern and a semantic model. Such a classification categorises the behavioural effects of reconfigurations when applied to specific coordination patterns. Figure 7.4 presents a possible taxonomy.

Consider $\rho \in \mathcal{P}$, r a reconfiguration and $\mathcal{M}$ any semantic model.

- r is *beh-unobtrusive* when it preserves the original behaviour of ρ. Formally, $(\mathbf{1}_r \mathrel{\mathring{=}}_{\mathcal{M}} r)_\rho$.

- r is *beh-obtrusive* when it changes the original behaviour of ρ, while preserving part of it. Formally, $(\mathbf{1}_r \preccurlyeq_{\mathcal{M}} r)_\rho \vee (r \preccurlyeq_{\mathcal{M}} \mathbf{1}_r)_\rho$.

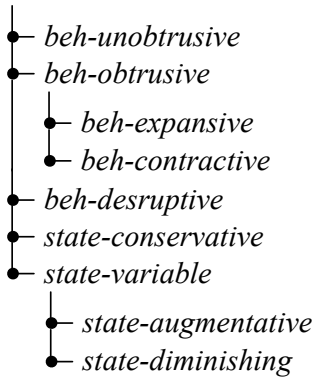


Figure 7.4: A taxonomy for classifying the (behavioural) effects produced by the application of reconfigurations.

- r is *beh-expansive* when it adds new behaviour to the original one. Formally, $(\mathbf{1}_r \preccurlyeq_{\mathcal{M}} r)_\rho$.
- r is *beh-contractive* when the original behaviour is partially removed. Formally, $(r \preccurlyeq_{\mathcal{M}} \mathbf{1}_r)_\rho$.
- r is *beh-disruptive* when it is neither *unobtrusive* nor *obtrusive*. The application of r does not preserve any of the original behaviour.

The classes above are disjoint, in the sense that no reconfiguration is, for instance, simultaneously *beh-expansive* and *beh-unobtrusive*, or *beh-contractive* and *beh-disruptive*.

The remaining classes are also disjoint among themselves, but may overlap with the ones above. In fact, the remaining classes provide a different perspective of behavioural classification. They classify reconfigurations based on a comparison of the number of states before and after reconfigurations. Informally,

- r is *state-conservative* when it conserves the number of states from the original coordination pattern;
- r is *state-variable* when it changes the number of states in comparison to the original coordination pattern;
- r is *state-augmentative* when it augments the number of states;
- r is *state-diminishing* when it diminishes the number of states.

Example 7.3 It is now possible to classify all the reconfigurations, considered in the previous examples, for the Sequencer pattern with respect to either the CA or the RA semantic

models. The $r_{proactive}$ and the $r_{dependent}$ reconfigurations are *disruptive*, while the r_{weak} and the $r_{quasiweak}$ are *beh-expansive*. In turn, $r_{s_{kept}}$ is an *beh-unobtrusive* reconfiguration.

Additionally, $r_{proactive}$ and $r_{dependent}$ are also *state-augmentative*. In turn, r_{weak}, $r_{s_{kept}}$ and $r_{quasiweak}$ are *state-conservative*. ✠

The classification of reconfigurations may play an important role in practice. For instance, two simple observations are that *expansive* reconfigurations are not *state-diminuitive* and *contractive* reconfigurations are not *state-augmentative*. More interesting is the possibility of reasoning about reconfigurations as a class, instead of individually. In this context, properties may be defined that remain invariant for all the class elements.

7.2 Structural reasoning

In the previous section it was discussed that the effect of a reconfiguration can be *measured* through behavioural changes entailed by its application.

There is, however, another perspective whose focus is placed on the interconnection structure, with no reference to the emerging behaviour. In this case, the effects of a reconfiguration can be *measured* by the structural changes induced from its application. Structural changes are observed from the analysis of structural or syntactic properties. Examples of such properties are:

i) every $\mathsf{fifo_e}$ *channel from a node n is connected to at least a* lossy *channel* or

ii) node i is a connector's output node.

One may then require that a reconfiguration preserves such properties. This will lead to a different family of relations to compare reconfigurations. What should be remarked is the fact that such relations are independent of the underlying semantic model and, in a broader sense, not committed to the use of a specific coordination modelling language. This section introduces a hybrid logic to express and reason about these properties.

7.2.1 A hybrid logic

Modal logic provides the standard way of expressing properties over the graph-like structure underlying coordination patterns. Each coordination pattern ρ gives rise to a transition system $\mathcal{G}_\rho$ over nodes in $\mathcal{N}$ and labelled by connector types in $\mathcal{T}$, such that $m \xrightarrow{t} n$ if *data may flow* from node m to node n through a connector of type t in ρ. Formally,

Definition 7.2. *Given a coordination pattern $\rho = \langle \mathcal{C}_\rho, \mathcal{N}_\rho \rangle$, its* may-flow graph $\mathcal{G}_\rho$ *is a labelled transition structure over $\mathcal{N}_\rho$, labelled by channel types, and given by*

$$\bigcup_{\langle \mathcal{S},i,t,\mathcal{K}\rangle \in \mathcal{C}_\rho,\, \mathcal{S},\mathcal{K}\neq\emptyset} \{\langle m,t,n\rangle |\ m,n \in \mathcal{N}_\rho \ \wedge\ m \cap \mathcal{S} \neq \emptyset \ \wedge\ n \cap \mathcal{K} \neq \emptyset\} \qquad (7.7)$$

The set of nodes in $\mathcal{G}_\rho$ is denoted by $\mathcal{G}_\rho^{\mathsf{nds}}$.

Clearly, $\mathcal{G}_\rho^{\mathsf{nds}} \subseteq \mathcal{N}_\rho$. A similar transition could be defined labelled by connector identifiers in $\mathcal{I}$ or even by pairs in $\mathcal{T} \times \mathcal{I}$, both cases giving rise to deterministic transition systems. The logic below is independent of whatever labels are chosen for representing specific views of the connector structure.

Often, however, structural properties are to be formulated relatively to a particular node in the pattern. An example is given by property *ii*) stated in the beginning of this section. In general, one may require, for instance, that all the channels incident in a specific node and their interconnections remain unchanged under a reconfiguration. This justifies the choice of *hybrid* logic [47] to express such properties.

Recall (from Chapter 2) that hybrid logic adds to a modal language the ability to name, or to explicitly refer to, specific states of the underlying Kripke structure. This is done through the introduction of propositional symbols of a new sort, called nominals, each of which is true at exactly one possible state. The sentences are then enriched in two directions. On the one hand, nominals are used as simple sentences holding exclusively in the state they name. On the other hand, explicit reference to states is provided by a *satisfaction* operator @ such that $@_i\phi$ asserts the validity of ϕ at the state named i.

Structural properties of coordination patterns will be expressed in a variant of hybrid propositional logic, called $Hp\mathcal{E}$, where modalities are indexed by channel *types* in $\mathcal{T}$, or, more generally, by *subsets* of $\mathcal{T}$. The logic is interpreted over $\mathcal{G}_\rho$, for each coordination pattern ρ. Each node n in $\mathcal{G}_\rho^{\mathsf{nds}}$ is endowed with a model of a propositional logic $p\mathcal{E}$ defined over channel ends in $\mathcal{E}$ and the usual Boolean connectives:

$$p, p' \ \ni\ e \ \mid\ \neg p \mid\ p \wedge p'$$

where $e \in \mathcal{E}$. The satisfaction relation is as follows

$$\begin{array}{ll} n \models^{p\mathcal{E}} e & \mathsf{iff}\ \ e \in n \\ n \models^{p\mathcal{E}} \neg p & \mathsf{iff}\ \ n \not\models^{p\mathcal{E}} p \\ n \models^{p\mathcal{E}} p \wedge p' & \mathsf{iff}\ \ n \models^{p\mathcal{E}} p \ \wedge\ n \models^{p\mathcal{E}} p' \end{array}$$

On top of $p\mathcal{E}$ a hybrid language is defined as follows:

$$\phi, \phi' \ni p \mid i \mid \neg\phi \mid \phi \wedge \phi' \mid [K]\phi \mid [\![K]\!]\phi \mid @_i\phi$$

where p is a sentence of $p\mathcal{E}$, $K \subseteq \mathcal{T}$, and $i \in \mathsf{Nom}$ for a set Nom of nominals. Constants true and false, as well as the usual Boolean connectives, are defined by abbreviation. For simplicity, in modalities '$-$' stands for the whole set $\mathcal{T}$, and $-t$, with $t \in \mathcal{T}$, stands for $\mathcal{T} \setminus \{t\}$.

Modality $[K]$ quantifies universally over the edges of $\mathcal{G}_\rho$ labelled by channel types in K; its dual $\langle K\rangle = \neg[K]\neg$ provides an existential quantification. Modalities $\langle K\rangle$ and $[K]$ express properties of *outgoing* connections from the node in which they are evaluated, in a coordination pattern. Dually, modalities $\langle\langle K\rangle\rangle$ and $[\![K]\!]$ express properties of *incoming* connections. Finally, the satisfaction operator @ *redirects* the formula evaluation to the context of a specific node. Nominals make possible to express proprieties *local* to a specific node.

A semantic model, $\mathfrak{M}$, for this language is a pair $\langle \mathcal{G}_\rho, \sigma : \mathsf{Nom} \to \mathcal{N}\rangle$, where ρ is a coordination pattern, and σ assigns to each nominal a node in $\mathcal{G}_\rho$. The satisfaction relation, given a model $\mathfrak{M} = \langle \mathcal{G}_\rho, \sigma\rangle$ and a node n, is defined as follows:

$$\begin{array}{lll}
\mathfrak{M}, n \models p & \mathsf{iff} & n \models^{p\mathcal{E}} p \\
\mathfrak{M}, n \models \neg\phi & \mathsf{iff} & \mathfrak{M}, n \not\models \phi \\
\mathfrak{M}, n \models \phi \wedge \phi' & \mathsf{iff} & \mathfrak{M}, n \models \phi \text{ and } \mathfrak{M}, n \models \phi' \\
\mathfrak{M}, n \models i & \mathsf{iff} & \sigma(i) = n \\
\mathfrak{M}, n \models @_i\phi & \mathsf{iff} & \mathfrak{M}, \sigma(i) \models \phi \\
\mathfrak{M}, n \models [K]\phi & \mathsf{iff} & \forall_{m \in \{p \mid \langle n,t,p\rangle \in \mathcal{G}_\rho \wedge t \in K\}} \,.\, \mathfrak{M}, m \models \phi \\
\mathfrak{M}, n \models [\![K]\!]\phi & \mathsf{iff} & \forall_{m \in \{p \mid \langle p,t,n\rangle \in \mathcal{G}_\rho \wedge t \in K\}} \,.\, \mathfrak{M}, m \models \phi
\end{array}$$

The satisfaction relation $\models$ lifts, as usual, to (global) satisfiability by quantifying over all the nodes in the model. this is, ϕ is *globally satisfied* in $\mathfrak{M}$ ($\mathfrak{M} \models \phi$) if it is satisfied at all nodes in $\mathfrak{M}$.

Example 7.4 Consider the following properties:

- Property i) stated above is expressed as

$$@_n[\mathsf{fifo_e}]\langle\mathsf{lossy}\rangle\mathsf{true}.$$

- Property *ii*), expressed as

$$@_i[-]\mathsf{false},$$

uses $[-]\mathsf{false}$ to state the absence of outgoing channels from the node referred by nominal i.

- All outgoing channels from i are lossy:

$$@_i(\langle - \rangle \mathsf{true} \wedge [-\mathsf{lossy}]\mathsf{false}).$$

- Absence of a loop formed by a sync followed by a lossy channel at i:

$$i \rightarrow \neg\langle \mathsf{sync} \rangle \langle \mathsf{lossy} \rangle i.$$

Notice that the absence of loops, and in general, irreflexivity of a binary relation is not expressible in classical modal logic [56].

- All output nodes are accessible through a sync channel but never through a $\mathsf{fifo_e}$ channel:

$$[-]\mathsf{false} \rightarrow (\langle\!\langle \mathsf{sync} \rangle\!\rangle \, \mathsf{true} \wedge [\![\mathsf{fifo_e}]\!]\mathsf{false}) \qquad \text{✠}$$

Properties expressed in $Hp\mathcal{E}$ can be verified, after a translation to classical propositional hybrid logic in the HyloRes [20] system. The translation involves a restriction of propositional symbols to nominals and an encoding of the backward modalities $\langle\!\langle K \rangle\!\rangle$ and $[\![K]\!]$.

7.2.2 Bisimulation for $Hp\mathcal{E}$

The notion of bisimulation for $Hp\mathcal{E}$-models provides the essential tool to compare coordination patterns from a *structural* point of view (in a way similar to what was done in the behavioural perspective). Formally, let $\mathfrak{M} = \langle \mathcal{G}_\rho, \sigma \rangle$ and $\mathfrak{M}' = \langle \mathcal{G}_{\rho'}, \sigma' \rangle$, defined over the same set Nom of nominals and the same set of channel ends $\mathcal{E}$. Then,

Definition 7.3. *A* bisimulation *for $Hp\mathcal{E}$-models is a binary relation $R \subseteq \mathcal{G}_\rho^{\mathsf{nds}} \times \mathcal{G}_{\rho'}^{\mathsf{nds}}$ such that*

i) *for any $i \in \mathsf{Nom}$ and nRn', $\sigma(i) = n$ iff $\sigma'(i) = n'$*

ii) *for any $i \in \mathsf{Nom}$, $\sigma(i) R \sigma'(i)$*

iii) *if nRn', nodes n and n' are elementary equivalent,* i.e.,

$$\forall_{p \in p\mathcal{E}} \; n \models^{p\mathcal{E}} p \, \mathsf{iff} \, n' \models^{p\mathcal{E}} p$$

iv) (zig) *for any* $t \in \mathcal{T}$*, if* nRn' *and* $\langle n, t, m\rangle \in \mathcal{G}_\rho$*, then there exists a node* m' *such that* $\langle n', t, m'\rangle \in \mathcal{G}_{\rho'}$ *and* mRm'

v) (zag) *for any* $t \in \mathcal{T}$*, if* nRn' *and* $\langle n', t, m'\rangle \in \mathcal{G}_\rho$*, then there exists a node* m *such that* $\langle n, t, m\rangle \in \mathcal{G}_\rho$ *and* mRm'

Lemma 7.1. *The union and the relational composition of two bisimulations are still bisimulations.*

Proof.
Union. Let $R = R_1 \cup R_2$, where both R_1, R_2 are bisimulations. Clearly, any two nodes n, n' related by R also satisfy nR_1n' or nR_2n'. Therefore, clauses i), ii) and iii) of Definition 7.3 hold. For clause iv) consider a connection $\langle n, c, m\rangle$. If nR_1n' (respectively, nR_2n') there is a node m' such that $\langle n', c, m'\rangle$ and mR_1m' (respectively, mR_2m'), which, in either case, entails mRm'. Clause v) is proved similarly.
Composition. Consider two (composable) bisimulations R_1, R_2 and suppose $nR_2 \cdot R_1n'$, for nodes n, n'. Clauses i), ii) and iii) of Definion 7.3 hold for $R_2 \cdot R_1$ because they also hold individually for R_1 and R_2; for clause iii) note that the elementary equivalence used in its statement is an equivalence relation. For clause iv), suppose $nR_2 \cdot R_1n'$, for nodes n, n'. By definition of relational composition, there is a node p such that nR_1p and pR_2n'. Suppose there is a connection $\langle n, c, m\rangle$; then, R_1 being a bisimulation, there exists a node p' and a connection $\langle p, c, p'\rangle$, with mR_1p'. Connection $\langle p, c, p'\rangle$ and the fact that pR_2n' for R_2 a bisimulation, on the other hand, entails the existence of a node m' such that there is a connection $\langle n', c, m'\rangle$ and $p'R_2m'$. Thus $mR_2 \cdot R_1m'$, as wanted. Clause v) is proved similarly. □

Definition 7.4. *Given two models,* $\mathfrak{M} = \langle \mathcal{G}_\rho, \sigma\rangle$ *and* $\mathfrak{M}' = \langle \mathcal{G}_{\rho'}, \sigma'\rangle$*, and two nodes* $n \in \mathcal{G}_\rho^{\mathsf{nds}}$ *and* $n' \in \mathcal{G}_{\rho'}^{\mathsf{nds}}$*. Nodes* n, n' *are* bisimilar*, denoted by* $n \rightleftharpoons n'$*, iff there is a relation* R *which is a bisimulation over* $\mathfrak{M}, \mathfrak{M}'$ *and also over* $\mathfrak{M}^\circ, \mathfrak{M}'^\circ$*, such that* nRn'*, where* $\mathfrak{M}^\circ$ *is a model identical to* $\mathfrak{M}$ *but defined over the relational converse of* $\mathcal{G}_\rho$*.*

This extra requirement for R in the definition above comes from the presence of a backwards modality $[\![K]\!]$ to reason about incoming connections. Now, clearly,

Lemma 7.2. *Let* $\mathcal{U} \subseteq \mathcal{P}$*. Relation* $\rightleftharpoons$ *is an equivalence relation over the set of nodes of all patterns in* $\mathcal{U}$*.*

Proof. It is immediate to show that the identity relation is a bisimulation, and so is the relational converse of a bisimulation (if $n \rightleftharpoons n'$ is witnessed by a bisimulation

$R \subseteq \mathcal{G}_{\rho}^{\mathsf{nds}} \times \mathcal{G}_{\rho'}^{\mathsf{nds}}$, for $\rho, \rho' \in \mathcal{U}$, then $n' \rightleftharpoons n$ is witnessed by $R^{\circ} \subseteq \mathcal{G}_{\rho'}^{\mathsf{nds}} \times \mathcal{G}_{\rho}^{\mathsf{nds}}$). Transitivity, on the other hand, comes from the fact that the relational composition of bisimulations is also a bisimulation, as proved in the second part of Lemma 7.1. □

The usual connection between bisimilarity and modal satisfaction also holds as shown in the following.

Lemma 7.3. *Given two models, $\mathfrak{M} = \langle \mathcal{G}_{\rho}, \sigma \rangle$ and $\mathfrak{M}' = \langle \mathcal{G}_{\rho'}, \sigma' \rangle$, and two nodes $n \in \mathcal{G}_{\rho}^{\mathsf{nds}}$ and $n' \in \mathcal{G}_{\rho'}^{\mathsf{nds}}$, such that $n \rightleftharpoons n'$, then*

$$\mathfrak{M}, n \models \Phi \iff \mathfrak{M}', n' \models \Phi$$

for every formula $\Phi \in Hp\mathcal{E}$.

Proof. The proof proceeds by induction on the formulas' structure.

i) $\Phi = p$**.** Let $\mathfrak{M}, n \models p$ which, by definition of $\models$, is equivalent to $n \models^{p\mathcal{E}} p$. Nodes n and n' are elementary equivalent because $n \rightleftharpoons n'$, which entails $n' \models^{p\mathcal{E}} p$. Therefore, $\mathfrak{M}', n' \models p$.

ii) $\Phi = \phi \wedge \psi$**.** Let $\mathfrak{M}, n \models \phi \wedge \psi$ which, by definition of $\models$, is equivalent to $\mathfrak{M}, n \models \phi$ and $\mathfrak{M}, n \models \psi$. By induction hypothesis $\mathfrak{M}', n' \models \phi$ and $\mathfrak{M}', n' \models \psi$, which entails $\mathfrak{M}', n' \models \phi \wedge \psi$. The argument is similar for $\Phi = \neg\phi$ (and, in general, for the derived Boolean connectives).

iii) $\Phi = i$**.** Let $\mathfrak{M}, n \models i$ which, by definition of $\models$, is equivalent to $\sigma(i) = n$. Then $\sigma'(i) = n'$ because $n \rightleftharpoons n'$, which entails $\mathfrak{M}', n' \models i$.

iv) $\Phi = @_i\phi$**.** Let $\mathfrak{M}, n \models @_i\phi$ which, by definition of $\models$, is equivalent to $\mathfrak{M}, \sigma(i) \models \phi$. By definition of bisimulation $\sigma(j) \rightleftharpoons \sigma'(j)$, for any nominal j. This combined with the induction hypothesis yields $\mathfrak{M}', \sigma'(i) \models \phi$. Therefore, $\mathfrak{M}', n' \models @_i\phi$.

v) $\Phi = [K]\phi$**.** Let $\mathfrak{M}, n \models [K]\phi$ which, by definition of $\models$, is equivalent to

$$\forall_{m \in \{p \mid \langle n,c,p \rangle \in \mathcal{G}_{\rho} \wedge c \in K\}} \,.\, \mathfrak{M}, m \models \phi.$$

Assume that $n \rightleftharpoons n'$ is witnessed by a relation R, and for each m above, consider the connection $\langle n, c, m \rangle$. As R is a bisimulation for $\mathfrak{M}, \mathfrak{M}'$, by clause iv) in Definition 7.3, there is a node m' in a connection $\langle n', c, m' \rangle$ and mRm'. By hypothesis $M, m \models \phi$ and then, by induction hypothesis, $M', m' \models \phi$. Therefore, $\forall_{m' \in \{p' \mid \langle n',c,p' \rangle \in \mathcal{G}_{\rho} \wedge c \in K\}} \,.\, \mathfrak{M}', m' \models \phi$ which entails $\mathfrak{M}', n' \models [K]\phi$. Clause v) in Definition 7.3 gives the converse implication.

v) $\Phi = [\![K]\!]\phi$**.** The argument is similar to the one used in iv) with R being a bisimulation for $\mathfrak{M}^{\circ}, \mathfrak{M}'^{\circ}$

□

The converse of this result also holds whenever $\mathcal{G}_\rho$ is image finite. Being image finite means that the number of incident connections in a node, for all nodes in $\mathcal{G}_\rho$, is finite. Notice that this is always the case, for any $\rho \in \mathcal{P}$, as coordination patterns are typically composed of a finite number of connectors.

Lemma 7.4. *Consider two models,* $\mathfrak{M} = \langle \mathcal{G}_\rho, \sigma \rangle$ *and* $\mathfrak{M}' = \langle \mathcal{G}_{\rho'}, \sigma' \rangle$*, and two nodes* $n \in \mathcal{G}_\rho^{\mathsf{nds}}$ *and* $n' \in \mathcal{G}_{\rho'}^{\mathsf{nds}}$*, such that* $\mathfrak{M}, n \models \Phi \iff \mathfrak{M}', n' \models \Phi$*, for every formula* $\Phi \in Hp\mathcal{E}$*. Then* $n \rightleftharpoons n'$*.*

Proof. Let us show that

$$Z = \{(n, n') \in \mathcal{G}_\rho^{\mathsf{nds}} \times \mathcal{G}_{\rho'}^{\mathsf{nds}} |\ \forall_{\phi \in Hp\mathcal{E}} \cdot\ \mathfrak{M}, n \models \phi \iff \mathfrak{M}', n' \models \phi\}$$

is a bisimulation over $\mathfrak{M}, \mathfrak{M}'$ and $\mathfrak{M}^{\circ}, \mathfrak{M}'^{\circ}$. The atomic conditions, in clauses i), ii) and iii) of Definition 7.3 trivially hold. Consider, now clause iv) (the *(zig)* condition). Let nZn' and assume there is a connection $\langle n, c, m \rangle$.

Both $\mathcal{G}_\rho$ and $\mathcal{G}_{\rho'}$ are, by construction, image finite, which makes $S = \{p' |\ \langle n', c, p' \rangle\}$ finite. It cannot be empty, however, because in such a case $M', n' \models \neg\langle c \rangle \mathsf{true}$ and, by definition of Z, $M, n \models \neg\langle c \rangle \mathsf{true}$ which is inconsistent with the existence of connection $\langle n, c, m \rangle$ assumed above.

To obtain a contradiction, suppose that there is no node $m' \in \mathcal{G}_{\rho'}^{\mathsf{nds}}$ such that mZm' and is part of a connection $\langle n', c, m' \rangle$. Therefore, for every $v \in S$, there is a formula ψ_v such that $\mathfrak{M}, m \models \psi_v$ and $\mathfrak{M}', v \not\models \psi_v$. Consider now the conjunction

$$\psi = \bigwedge_{v \in S'} \psi_v$$

of all of these formulas. Then, on the one hand, $\mathfrak{M}, n \models \langle c \rangle \psi$, but, on the other, $\mathfrak{M}', n' \not\models \langle c \rangle \psi$. This, however, contradicts nZn'. The *(zag)* condition, clause v) in Definition 7.3 is shown in a similar way.

This proves that relation Z is a bisimulation over $\mathcal{G}_\rho$ and $\mathcal{G}_{\rho'}$. A similar argument shows that it is also a bisimulation for their relational converses, which are also image finite. □

Combining both lemmas entails a Hennessy-Milner like result: modal equivalence in $Hp\mathcal{E}$ and bisimilarity coincide for models constructed from coordination patterns

according to Definition 7.2. Note that the result is valid in general for image finite models.

Theorem 7.1. *Consider two models* $\mathfrak{M} = \langle \mathcal{G}_\rho, \sigma \rangle$ *and* $\mathfrak{M}' = \langle \mathcal{G}_{\rho'}, \sigma' \rangle$*, and two nodes* n, n' *in* $\mathcal{G}_\rho^{\mathsf{nds}}$ *and* $\mathcal{G}_{\rho'}^{\mathsf{nds}}$*, respectively. Then,*

$$n \rightleftharpoons n' \;\; \textit{iff} \;\; \forall_{\Phi \, in \, Hp\mathcal{E}} \, \mathfrak{M}, n \models \Phi \iff \mathfrak{M}', n' \models \Phi$$

Proof. Corollary of lemmas 7.3 and 7.4 above. □

7.2.3 Expressing 'long scope' properties

$Hp\mathcal{E}$ can be extended to allow for modalities which take the form of regular expressions over the labels of $\mathcal{G}_\rho$, for any $\rho \in \mathcal{P}$. Technically this is a particular case of the extension of a modal logic with fixed points as in the μ-calculus [167]. This is particularly useful to express *long scope* properties, such as

i) A channel of type t *is accessible from a node referred to by* i

ii) All input ports lead to an output port via, at least, one $\mathsf{fifo_e}$ *channel*

Consider, thus, the following syntax for modalities:

$$\nu \;\ni\; \epsilon \mid t \mid \nu.\nu \mid \nu + \nu \mid \nu^* \mid \nu^+$$

where ϵ is the *empty word*, $t \in \mathcal{T}$, '.' denotes concatenation, '+' choice and ν^* and ν^+ the Kleene and transitive closures. The intuitive semantics is given below (a precise rendering requires the introduction of fixed points). Note that the modalities referring to *incoming* connections are defined dually.

Definition 7.5. *Let* ν *,*ν_1 *and* ν_2 *be modalities and* ϕ *a formula. The following define the semantics of the modalities extension:*

- $\langle \epsilon \rangle \phi = \phi = [\epsilon]\phi$
- $[\nu_1.\nu_2]\phi = [\nu_1][\nu_2]\phi$
- $[\nu_1 + \nu_2]\phi = [\nu_1]\phi \vee [\nu_2]\phi$
- $[\nu^*]\phi = \phi \vee [\nu]\phi \vee [\nu][\nu]\phi \vee \ldots$
- $[\nu^+]\phi = [\nu]\phi \vee [\nu][\nu]\phi \vee \ldots$
- $\langle \nu_1.\nu_2 \rangle \phi = \langle \nu_1 \rangle \langle \nu_2 \rangle \phi$
- $\langle \nu_1 + \nu_2 \rangle \phi = \langle \nu_1 \rangle \phi \vee \langle \nu_2 \rangle \phi$
- $\langle \nu^* \rangle \phi = \phi \vee \langle \nu \rangle \phi \vee \langle \nu \rangle \langle \nu \rangle \phi \vee \ldots$
- $\langle \nu^+ \rangle \phi = \langle \nu \rangle \phi \vee \langle \nu \rangle \langle \nu \rangle \phi \vee \ldots$

Example 7.5 With this extension it is now possible to express the properties above as follows:

i) $@_i \langle -^*.t \rangle \mathsf{true}$

ii) $[\![-]\!]\mathsf{false} \rightarrow \langle -^*.\mathsf{fifo_e}.-^*\rangle[-]\mathsf{false}$ ✠

7.2.4 Comparing reconfigurations

As mentioned above, with a structural bisimulation notion, one is able to compare reconfigurations in a way similar to what have been done from the behavioural point of view. However, equipped with a language to express structural properties of coordination patterns, one can define a different criterium for comparing reconfigurations. This entails a notion of invariant properties that shall be preserved by reconfigurations. The ability to preserve (or not) these invariants will guide the comparison of reconfigurations.

Notation

id_X defines the identity function on set X, *i.e.,* for all $x \in X$ $id_X(x) = x$.

In the remaining of this section, the set of nominals in each model $\mathfrak{M} = \langle \mathcal{G}_\rho, \sigma \rangle$ will be identified with the set $\mathcal{N}_\rho$ of nodes in pattern ρ. This makes $\sigma = id_{\mathcal{N}_\rho}$, without loss of generality.

Definition 7.6 (Structural Invariant)**.** *A structural property ϕ is* invariant *for a reconfiguration r in a model $\mathfrak{M} = \langle \mathcal{G}_\rho, id_{\mathcal{N}_\rho} \rangle$ iff it is preserved by r. Formally,*

$$\langle \mathcal{G}_\rho, id_{\mathcal{N}_\rho} \rangle \models \phi \rightarrow \langle \mathcal{G}_{\rho \circlearrowright r}, id_{\mathcal{N}_{\rho \circlearrowright r}} \rangle \models \phi$$

In practice, however, reconfigurations entail a sort of *displacement* of the structural relationships in the coordination pattern. This means they often remain valid but at a different node in the pattern. A typical situation is illustrated in Example 7.6.

Example 7.6 Consider the coordination pattern in Figure 7.5.a). Clearly, at node $\underline{cde}$ it is true that, after a connection made through a sync channel, all the others are established by lossy channels, *i.e.,* $@_{\underline{cde}} \langle \mathsf{sync} \rangle (\langle - \rangle \mathsf{true} \wedge [-\mathsf{lossy}]\mathsf{false})$.

Consider now that an insertP reconfiguration is applied at node $\underline{cde}$. In a first step, node $\underline{cde}$ is split because of the application of the split elementary reconfiguration (Figure 7.5.b). Then a new structure is linked to the nodes resulting from this operation (Figure 7.5.c). In both steps, however, the property is still valid for nodes $\underline{e}$ and $\underline{m_o e}$, respectively. With respect to the example in Figure 7.5.c, this is expressed as $@_{\underline{m_o e}} \langle \mathsf{sync} \rangle (\langle - \rangle \mathsf{true} \wedge [-\mathsf{lossy}]\mathsf{false})$.
✠

This example motivates a generalisation of Definition 7.6.

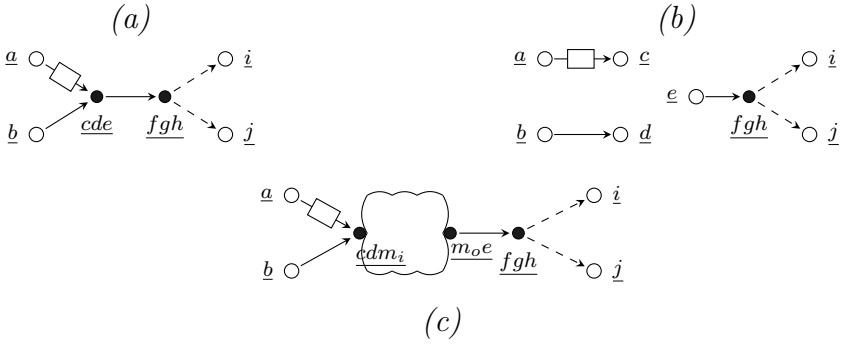


Figure 7.5: Example of a *displaced* invariant.

Definition 7.7. *Let τ be a surjection on nominals. A structural property ϕ is* invariant *for a reconfiguration r, up to a name translation τ iff*

$$\langle \mathcal{G}_\rho, id_{\mathcal{N}_\rho} \rangle \models \phi \Rightarrow \langle \mathcal{G}_{\rho \circlearrowleft r}, id_{\mathcal{N}_{\rho \circlearrowleft r}} \rangle \models \phi[\tau]$$

where $\phi[\tau]$ stands for ϕ with all occurrences of a nominal i replaced by $\tau(i)$.

Notation

$[x_1 \mapsto y_1, \ldots, x_n \mapsto y_n]$ denotes a function mapping nodes x_i to y_i; it behaves as the identity for all other cases.

Back to Example 7.6, if $\langle \mathcal{G}_\rho, \sigma \rangle \models \psi$, then $\langle \mathcal{G}_{\rho \circlearrowleft r}, \sigma \rangle \models \psi[\underline{cde} \mapsto \underline{m_o e}]$ for r the relevant reconfiguration.

From these notions of structural invariant, one is able to define what does it mean to two reconfigurations be structural equivalent.

Definition 7.8 (Structural Equivalence)**.** *Given a model $\mathfrak{M} = \langle \mathcal{G}_\rho, id_{\mathcal{N}_\rho} \rangle$, reconfigurations r_1 and r_2, a set of formulas Φ, and two surjections τ and τ' it is said that r_1 and r_2 are* structurally equivalent *with respect to Φ, written $r_1 \equiv^{\mathfrak{M}}_{\Phi} r_2$, iff*

$$\langle \mathcal{G}_{\rho \circlearrowleft r_1}, id_{\mathcal{N}_{\rho \circlearrowleft r_1}} \rangle \models \phi[\tau] \iff \langle \mathcal{G}_{\rho \circlearrowleft r_2}, id_{\mathcal{N}_{\rho \circlearrowleft r_2}} \rangle \models \phi[\tau']$$

for every $\phi \in \Phi$.

Informally, if two reconfigurations preserve the same set of invariant properties, they are said structurally equivalent. Consequently, one can derive a weak version of this equivalence notion that allows for finer comparisons of reconfigurations.

Definition 7.9 (Structural Weak Equivalence)**.** *Given a model $\mathfrak{M} = \langle \mathcal{G}_\rho, id_{\mathcal{N}_\rho} \rangle$, reconfigurations r_1 and r_2, a set of formulas Φ, and two surjections τ and τ' it is said that r_1 is structurally weaker than r_2 with respect to Φ, written $r_1 \leq^{\mathfrak{M}}_{\Phi} r_2$, iff*

$$\langle \mathcal{G}_{\rho \circlearrowleft r_1}, id_{\mathcal{N}_{\rho \circlearrowleft r_1}} \rangle \models \phi[\tau] \Rightarrow \langle \mathcal{G}_{\rho \circlearrowleft r_2}, id_{\mathcal{N}_{\rho \circlearrowleft r_2}} \rangle \models \phi[\tau']$$

for some $\phi \in \Phi$.

This implies that r_1 preserves formulas in a set Φ_1, and r_2 preserves formulas in a set Φ_2, such that $\Phi_1 \subseteq \Phi_2 \subseteq \Phi$. In this situation, r_1 preserves less structure than r_2, therefore it can be said less conservative with respect to structure. This inequality may even be quantified by taking the difference between the number of formulas preserved by each reconfiguration. Regarding it over this perspective, a natural order of reconfigurations would arise.

Example 7.7 Let $\mathfrak{M} = \langle \mathcal{G}_{\rho_s}, id_{\mathcal{N}_{\rho_s}} \rangle$. Consider Φ as a set composed of the following three invariant formulas:

ϕ_1 $@_a[-^+]b$ – node $\underline{a}$ connects to node $\underline{b}$, via one or more channels;

ϕ_2 $@_a[-^*.\mathsf{fifo}]\mathsf{true}$ – data transmission is made asynchronously;

ϕ_3 $@_{o_1}[-]\mathsf{false} \wedge @_{o_2}[-]\mathsf{false}$ – nodes $\underline{o_1}$ and $\underline{o_2}$ are output ports.

and recall reconfigurations $r_{proactive}$, r_{weak} and $r_{dependent}$ presented in previous sections as well as their resulting patterns when applied to the Sequencer (Figures 6.2, and 7.1, respectively) .

Clearly, the three reconfigurations preserve ϕ_2 and ϕ_3; and reconfiguration $r_{dependent}$ does not preserve ϕ_1, but the others do. From this simple reasoning, one conclude that

- $r_{proactive} \equiv^{\mathfrak{M}}_{\Phi} r_{weak}$
- $r_{dependent} \leq^{\mathfrak{M}}_{\Phi} r_{proactive}$
- $r_{dependent} \leq^{\mathfrak{M}}_{\Phi} r_{weak}$ ✠

Recall Example 7.1. In that example it was concluded that $r_{dependent}$ and $r_{proactive}$ were semantically equivalent. The same observation does not hold in this structural point of view, at least for the set of formulas considered in Example 7.7. These two reasoning perspectives are complementary to one another, providing different insights about reconfigurations.

7.2.5 A structural classification of reconfigurations

Similarly to the behavioural perspective, one can derive a classification of reconfigurations with respect to this structural vision. Such a classification takes into account a specific model $\mathfrak{M} = \langle \mathcal{G}_{\rho}, id_{\mathcal{N}_{\rho}} \rangle$, and a set of formulas Φ. Figure 7.6 presents the corresponding taxonomy.

Informally, a reconfiguration r is said to be:

- *str-compatible* when it preserves all formulas in Φ.
  Formally, $\forall_{\phi \in \Phi} \cdot \langle \mathcal{G}_{\rho \circlearrowleft r}, id_{\mathcal{N}_{\rho \circlearrowleft r}} \rangle \models \phi$.

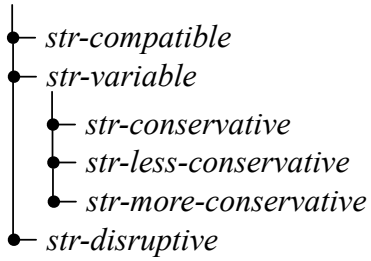


Figure 7.6: A taxonomy for classifying the (structural) effects produced by the application of reconfigurations.

- *str-variable* when it preserves formulas in Φ that also hold in the original coordination pattern.
- *str-conservative* when it preserves all formulas in Φ that also hold in the original coordination pattern.
  Formally, $\mathbf{1}_r \equiv_{\Phi}^{\mathfrak{M}} r$.
- *str-less-conservative* when it preserves less formulas in Φ than those holding on the original coordination pattern.
  Formally, $r \leq_{\Phi}^{\mathfrak{M}} \mathbf{1}_r$.
- *str-more-conservative* when it preserves more formulas in Φ than those holding on the original coordination pattern.
  Formally, $\mathbf{1}_r \leq_{\Phi}^{\mathfrak{M}} r$.
- *str-disruptive* when it does not preserve any formula in Φ.
  Formally, $\forall_{\phi \in \Phi} \cdot \langle \mathcal{G}_{\rho^{\smile} r}, id_{\mathcal{N}_{\rho^{\smile} r}} \rangle \not\models \phi$.

Using the hybrid logic to define formulas that specifically target the structure of coordination pattern, one can go further and add new classification dimensions to this taxonomy. For instance, one can define a set of properties that *talk* about the interface of a coordination pattern. Then, one is able to classify reconfigurations that, for instance, lead coordination patterns with different interface when compared to its original version. These may provide a deeper insight to reconfigurations, allowing architects to directly choosing among a set of them.

7.3 The stochastic case

In this section the focus is set on the stochastic version of coordination patterns and their reconfigurations. Reasoning about the structure of these patterns remains

as in the non stochastic case. In some sense, the same applies to the behavioural reasoning perspective. However, for this dimension, a different sort of semantics has to be chosen in order to take full advantage of the richer set of information present in stochastic coordination patterns.

Earlier, in Chapter 4, it was introduced a stochastic semantic model, $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$, that can now be adopted. $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ was initially thought for (a component-based view of) stochastic Reo. Therefore, in order to use $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ as a semantic model for stochastic coordination patterns, the channels of the latter must be typed with Reo channel names. The models underlying $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ and stochastic coordination patterns match, because of the separation enforced upon channels, nodes and environment. Notwithstanding this match, a translation between the two entities is required. In Appendix A, Algorithm A.2 shows how this conversion is obtained. Note that $\mathsf{IMC}_{\mathsf{Reo}}$ could, as well, be used for this matter. However, the models underlying $\mathsf{IMC}_{\mathsf{Reo}}$ (which is stochastic Reo) and stochastic coordination patterns have several mismatches. Anyway, Algorithm A.1 shows how similar translation could be obtained.

Having such a semantic model associated with stochastic coordination patterns, it is now possible to reason from a behavioural perspective about their reconfigurations. $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ provides the necessary comparison mechanisms to that end. But in possession of stochastic information, it makes sense to reason about reconfigurations up to an even different perspective: the quantitative one. In that setting one can compare reconfigurations by the quantitative effects they have upon the coordination patterns.

7.3.1 Quantitative reasoning

In a stochastic setting, the effects of reconfigurations can be *measured* by the quantitative implications of their application. To this end, one has to concretely define what are the relevant dimensions that characterise the quantitative behaviour of the subject system. Then, one may require that reconfigurations preserve the performance for those dimensions or, for instance, that such a performance does not deviate (above or below) from some fixed quantifiable threshold. These dimensions are, as expected, related to QoS [187].

Constraint semirings (c-semirings for short) [46, 108] provide the natural way of algebraically modelling the structure of QoS dimensions.

Definition 7.10 (C-semiring)**.** *An algebraic structure* $\langle V, \oplus, \otimes, \mathbf{0}, \mathbf{1}\rangle$ *is a* c-semiring *for* V *a set where* $0, 1 \in V$ *and* $\oplus$ *and* $\otimes$ *are binary operations over* V *satisfying:*

- $\oplus$ *is commutative, associative, idempotent,* $\mathbf{0}$ *is its unit element and* $\mathbf{1}$ *is the absorbing element.*
- $\otimes$ *is commutative, associative, distributes over* $\oplus$, $\mathbf{1}$ *is its unit element and* $\mathbf{0}$ *is its absorbing element.*

Intuitively, $\oplus$ and $\otimes$ define a choice and a combination operations, respectively. A c-semiring further defines a partial order over V as $v_1 \leq_V v_2 \iff v_1 \oplus v_2 = v_2$, for $v_1, v_2 \in V$, meaning (informally) that v_2 is *better than* v_1.

This partial order is essential for comparing QoS dimension values. In fact, for the reasoning approach proposed in this section, one is not interested in defining how the QoS dimension value is computed from the structure of the coordination pattern. Instead, such a value is obtained via pre-established formulas that use the results from the quantitative analysis of the underlying performance model (expressed in $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$, in this case).

With this definition, one is able to express whatever QoS dimensions are relevant for characterising the performance of a system. Usual performance dimensions are:

- $\langle \mathbb{R}^+, \mathsf{max}, \mathsf{min}, 0, +\infty \rangle$, *e.g., throughput* (generating a partial order $\leq_{\mathbb{R}^+}$);
- $\langle \mathbb{R}^+, \mathsf{min}, \mathsf{max}, +\infty, 0 \rangle$, *e.g., latency*, *response-time* or *mean-time-to-fail* (generating a partial order $\geq_{\mathbb{R}^+}$);
- $\langle [0,1], \mathsf{max}, \cdot, 0, 1 \rangle$, *e.g., throughput-rate* or *availability* (generating a partial order $\leq_{[0,1]}$);
- $\langle [0,1], \mathsf{min}, \mathsf{max}, 1, 0 \rangle$, *e.g., loss-rate* or *blocking-rate* (generating a partial order $\geq_{[0,1]}$).

In the sequel, Ω will denote a set of QoS dimensions, and the carrier of the c-semiring associated to dimension $\omega \in \Omega$ will be denoted by V_ω.

Of course, the quantitative analysis of the QoS dimensions in a set Ω associated to a stochastic coordination pattern is made taking into account an environment. In turn, the comparison of reconfigurations will have to consider the original coordination pattern, a fixed environment (or at least, an environment that remains compatible with the original coordination pattern) and the dimensions. Consequently, a model for quantitative reasoning $\mathfrak{S}$ is a pair $\langle \rho_{\Delta^{\mathcal{E}nv}}, \Omega \rangle$, where the first component is a stochastic coordination pattern deployed in an environment $\mathcal{E}nv$, and the second is the set of relevant dimensions for ρ.

Now, one defines how two reconfigurations are quantitatively compared with respect to a suitable quantitive reasoning model.

Definition 7.11 (Quantitative Equivalence)**.** *Let $\mathfrak{S} = \langle \rho_{\Delta^{\mathcal{E}nv}}, \Omega \rangle$, be a quantitive reasoning model and r_1, r_2 be two reconfigurations. Then*

$$r_1 \equiv_{\mathfrak{S}} r_2 \iff v_{r_1} =_{V_\omega} v_{r_2},$$

for all $\omega \in \Omega$ and $v_{r_1}, v_{r_2} \in V_\omega$ corresponding to the values of dimension ω evaluated after reconfigurations r_1 and r_2, respectively.

Similarly,

Definition 7.12 (Quantitative Weak Equivalence)**.** *Let $\mathfrak{S} = \langle \rho_{\Delta^{\mathcal{E}nv}}, \Omega \rangle$, be a quantitive reasoning model and r_1, r_2 be two reconfigurations. Then,*

$$r_1 \leq_{\mathfrak{S}} r_2 \iff v_{r_1} \leq_{V_\omega} v_{r_2},$$

for all $\omega \in \Omega$ and $v_{r_1}, v_{r_2} \in V_\omega$ corresponding to the values of dimension ω evaluated after reconfigurations r_1 and r_2, respectively.

In this case, reconfiguration r_1 is quantitatively weaker than r_2 as it consistently presents worse performance for each QoS dimension considered for the coordination pattern.

Example 7.8 Consider the stochastic Sequencer pattern ρ_s^{sto} in Example 6.13 with its stochastic values associated, and the environment $\mathcal{E}nv$ defined as in Example 6.14.

Now consider reconfigurations r_{weak} and $r_{quasiweak}$ in Examples 6.8 and 7.2, respectively. The stochastic coordination patterns introduced by these reconfigurations have the following stochastic information:

$$\rho_w^{sto} = \left\langle \left\{ \left\langle \begin{array}{c} \{\langle i_1, 1000\rangle\}, s_5, \mathsf{sync}, \{\langle o_1, 1000\rangle\}, \\ \{\langle \{i_1, o_1\}, 150\rangle, \langle \{i_1\}, 175\rangle\} \end{array} \right\rangle, \left\langle \begin{array}{c} \{\langle i_2, 1000\rangle\}, s_6, \mathsf{sync}, \{\langle 0_2, 1000\rangle\}, \\ \{\langle \{i_2, o_2\}, 150\rangle, \langle \{i_2\}, 175\rangle\} \end{array} \right\rangle \right\}, \left\langle \begin{array}{c} \underline{i_1}, \underline{i_2}, \underline{i_3}, \underline{o_2} \end{array} \right\rangle \right\rangle$$

$$\rho_{qw}^{sto} = \left\langle \left\{ \left\langle \begin{array}{c} \{\langle i, 1000\rangle\}, s_6, \mathsf{lossy}, \{\langle o_2, 1000\rangle\}, \\ \{\langle \{i, o\}, 150\rangle, \langle \{i\}, 175\rangle\} \end{array} \right\rangle, \right\}, \left\langle \begin{array}{c} \underline{i}, \underline{o} \end{array} \right\rangle \right\rangle$$

Finally, consider that the QoS characterising ρ_s^{sto} is defined as $\Omega_{\rho_s^{sto}} = \{\mathsf{LT}, \mathsf{TP}, \mathsf{Blk}\}$, where LT is latency[1], TP is the throughput[2] and Blk is the blocking percentage[3]. These dimensions present partial orders $\geq_{\mathbb{R}^+}$, $\leq_{\mathbb{R}^+}$ and $\geq_{[0,1]}$, respectively.

The values for these dimensions are obtained after quantitative analysis of each coordination pattern with appropriate tools (*c.f.,* Chapter 9). Table 7.1 shows these values.

[1]Latency is regarded here as the time (in seconds) needed for one request to be processed, *i.e.,* flowing from port a to port b.

[2]Throughput is regarded here as the number of request *accepted* in a second by port b.

[3]Blocking percentage is the percentage of time in which the system is blocked

Table 7.1: Values for Latency, Throughput and Blocking quality dimensions

	$\rho^{sto}_{s\Delta\mathcal{E}nv}$	$\rho^{sto}_{qw\Delta\mathcal{E}nv}$	$\rho^{sto}_{w\Delta\mathcal{E}nv}$
LT	0.083	0.070	0.057
TP	12.048	14.286	17.544
Blk	0.759	0.714	0.649

One can now compare the two reconfigurations in the context of $\mathfrak{S} = \langle \rho^{sto}_{s\Delta\mathcal{E}nv}, \Omega_{\rho^{sto}_{s}} \rangle$: $0.070 \leq_{\mathsf{LT}} 0.057$; $14.286 \leq_{\mathsf{TP}} 17.544$ and $0.714 \leq_{\mathsf{Blk}} 0.649$. Therefore, $r_{quasiweak} \leq_{\mathfrak{S}} r_{weak}$. ✠

In many practical situations, however, one may be interested in a more loose definition of quantitative equivalence for reconfigurations. This is because, often, even though the reconfiguration change the value of some QoS dimensions, it still guarantees the agreed quality of the system. This presupposes the existence of some acceptable threshold that will define an interval of values in which each QoS dimension should be valued.

Definition 7.13 (Quantitatively Bounded Reconfigurations)**.** *Let $\mathfrak{S} = \langle \rho_{\Delta\mathcal{E}nv}, \Omega \rangle$, be a quantitive reasoning model and r_1, r_2 be two reconfigurations. Consider now a threshold $t_\omega \in V_\omega$, for each $\omega \in \Omega$. Then,*

$$r_1 [\equiv]_{\mathfrak{S}} r_2 \iff v_{r_1} =^{t_\omega}_{V_\omega} v_{r_2},$$

for all $\omega \in \Omega$, $v_{r_1}, v_{r_2} \in V_\omega$ corresponding to the values of dimension ω evaluated after reconfigurations r_1 and r_2, respectively; and $=^{t_\omega}_{V_\omega}$ the equality in V_ω up to t_ω, defined as, for $x, y \in V_\omega$

$$x =^{t_\omega}_{V_\omega} y \iff x \geq_{V_\omega} t \wedge y \geq_{V_\omega} t.$$

Moreover, in the stochastic perspective it makes sense that reconfigurations replace parts of some coordination pattern with the objective of improving performance without changing neither its structure nor its behaviour. New versions of channels may be created that are more efficient or performative. In the presence of a catalogue of channels presenting different QoS guarantees, architects may decide whether to apply reconfigurations or not based on the overall objectives for the system.

Example 7.9 Consider Example 7.8 again and assume the objective of improving the Sequencer pattern to present LT below 0.070, TP above 14.000 and Blk below 0.700.

Reconfiguration

$$r_{improve} = \rho^{sto}_{s\Delta\varepsilon_{nv}} \circlearrowleft \mathsf{replaceP}(\ \underline{e_i} \circ\!\!-\!\!\underset{x_i}{\square}\!\!\rightarrow\!\!\circ\ \underline{f_i}\ , \{(\underline{fgh}, \underline{f_i}), (\underline{cde}, \underline{e_i})\}, \{x\})$$

replaces the buffered channel x with a similar one x_i able to write to/read from its queue 100 times more requests per second. It applies an improvement of all QoS dimensions to values similar to those obtained by applying reconfiguration $r_{quasiweak}$. In this case, $r_{quasiweak}[\equiv]_{\mathfrak{S}} r_{improve}$. ✠

7.3.2 A quantitative classification of reconfigurations

As before, one can now derive a classification of reconfigurations with respect to their quantitative effects on the original coordination patterns. In the quantitative setting this classification has to consider a quantitative reasoning model $\mathfrak{S} = \langle \rho_{\Delta\varepsilon_{nv}}, \Omega \rangle$. Figure 7.7 presents the corresponding taxonomy.

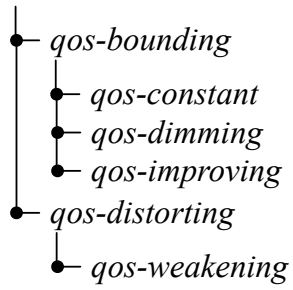


Figure 7.7: A taxonomy for classifying the (quantitative) effects produced by the application of reconfigurations.

The informal definition of each concept in the presented taxonomy is shown in the following list. The prefix *qos* is added to each concept to avoid name clashing. A reconfiguration r is said

- *qos-bounding* when it preserves the QoS performance up to some considered thresholds. Formally, $\mathbf{1}_r[\equiv]_{\mathfrak{S}} r$.
- *qos-constant* when it is equivalent to the identity reconfiguration. Formally, $\mathbf{1}_r \equiv_{\mathfrak{S}} r$.
- *qos-dimming* when it preserves the QoS performance up to considered thresholds, but all values of each QoS dimension are below the original ones. Formally, $\mathbf{1}_r[\equiv]_{\mathfrak{S}} r \wedge r \leq_{\mathfrak{S}} \mathbf{1}_r$.

- *qos-improving* when it augments the performance of each QoS dimension. Formally, $\mathbf{1}_r[\equiv]_{\mathfrak{S}} r \wedge \mathbf{1}_r \leq_{\mathfrak{S}} r$.

- *qos-distorting* when the performance of some QoS dimension is not preserved by r. Formally, $\exists_{\omega \in \Omega} \cdot v_r <_{V_\omega} v$, for $v, v_r \in V_\omega$ the values associated to ω before and after reconfiguration r, respectively.

- *qos-weakening* when r does not preserve the performance of any QoS dimension. Formally, $r \leq_{\mathfrak{S}} \mathbf{1}_r$.

Example 7.10 Reconfigurations in Examples 7.8 and 7.9 are *qos-improving* with respect to the quantitative reasoning model considered therein. ✠

Again, one may relate these concepts with those from the previous reasoning perspectives. New concepts will rise that allow software architects to sharpen their choices when designing reconfigurations. For instance, one may be interested in finding reconfigurations that improve QoS while maintaining all behavioural aspects of the original coordination patterns. For instance, concept "*improvement*" can be crafted for that matter. It may, then, be defined that a reconfiguration is an "*improvement*" when it is simultaneously *qos-improving* and *beh-unobtrusive*.

A complete ontology of reconfigurations can now be defined, assuming these three sets of taxonomic concepts as its foundation.

7.4 Summary

This chapter introduced three dimensions for reasoning about reconfigurations of interacting services with focus on coordination aspects: behavioural, structural and QoS.

For behavioural reasoning it is required that a specific semantic model is associated to the coordination pattern. The semantic model is required to be compositional and to provide suitable comparison mechanisms like the notions of similarity and bisimilarity. Reconfigurations are compared by the effects produced on the behaviour of the coordination patterns. Several granularity levels of behavioural comparison are introduced.

For structural reasoning no fixed behavioural semantic model is required. The coordination pattern itself defines a graph upon which structural properties may be investigated. In this perspective, comparison of reconfigurations are made via the preservation of such structural properties after their application. These properties

are expressed in a specific hybrid logic tailored for that matter. The underlying Kripke structure was defined with a *data-may-flow* flavour, but other approaches may have been adopted without changing the overall reasoning mechanisms.

For stochastic reasoning, the stochastic version of coordination patterns are in order, as well as the definition of a deployment environment. One could reason about reconfiguration applied to stochastic coordination patterns over the two previous perspectives. Comparing reconfigurations in this perspective amounts to comparing the pairs *QoS dimension-value* obtained upon applying reconfigurations.

The three perspectives were introduced with different approaches for comparing and classifying reconfigurations. One is done by simulatoin/bisimulation verification; other is by the preservation of key invariant properties; and the other is by quantitatively comparing performance. The more generic one is the reasoning by preservation of properties. Structural perspective does it, but one could also define properties upon the semantic models on both the behavioural and stochastic perspectives.

Finally, each reasoning perspective allows for the derivation of a taxonomy to classify reconfigurations. This classification is done after comparing the application of any reconfiguration and the identity reconfiguration over the same coordination pattern. The organisation of reconfigurations upon these taxonomies defines starting point for a suitable ontology as depicted in Figure 7.8.

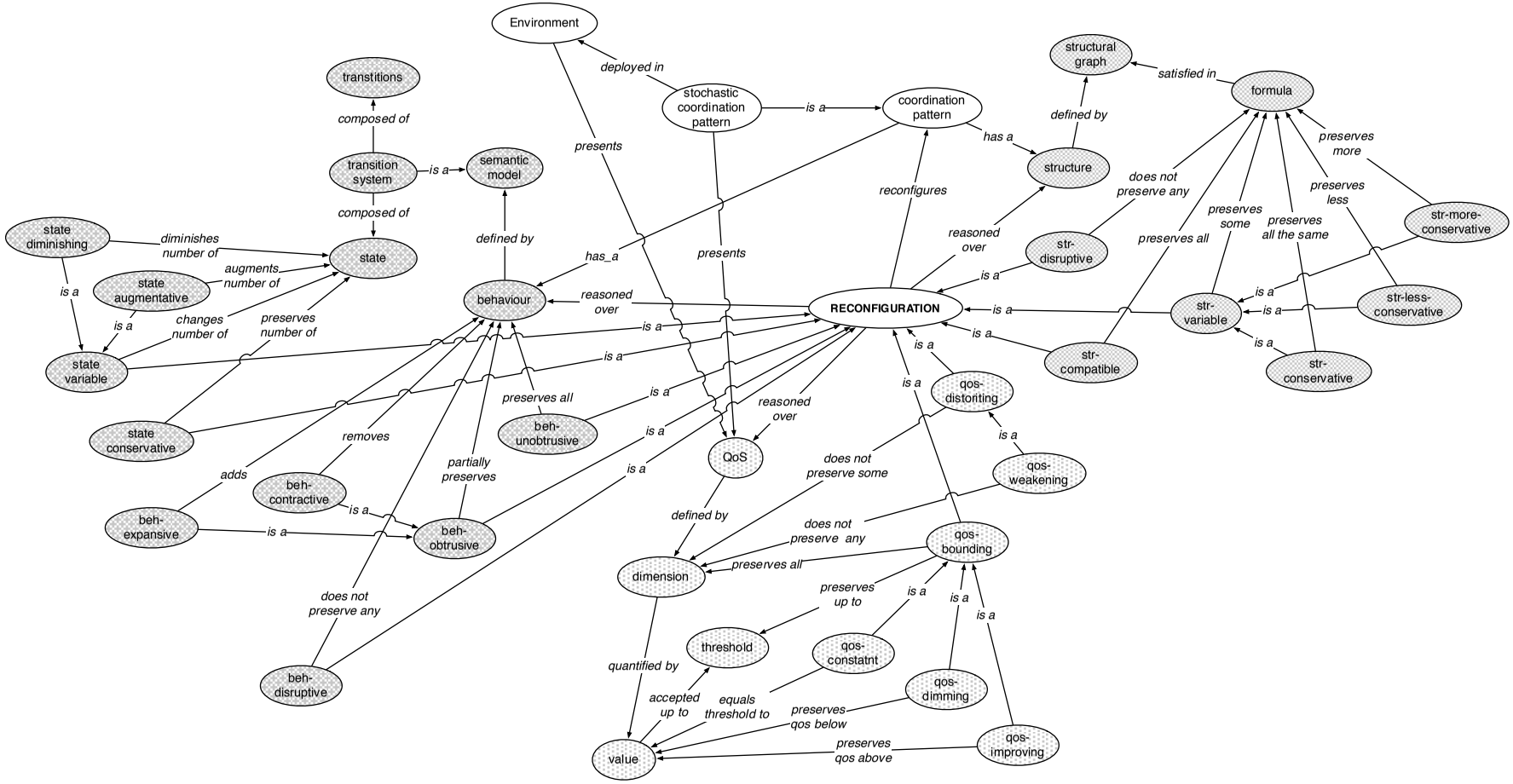


Figure 7.8: An ontology base for reconfigurations. Different colouring patterns are used for visually separating the families of concepts in the three reasoning perspectives.

Chapter 8

Self-adaptation of Architectures

It is neither the strongest nor the most intelligent who will survive, but the one most adaptable to change.

– *Charles Darwin*

In this chapter. An approach for self-adaptation of interacting services is proposed. It combines the basics of autonomic computing, namely the concept of feedback loop, with ARIS, as one possible practical application of the latter. In particular, it is discussed how the elements studied in ARIS define suitable mechanisms for the construction of (*i*) a model that lays down reconfiguration strategies and (*ii*) the main components for decision making within the feedback loop. A simple but effective approach for triggering reconfigurations is introduced. Finally, an attempt of porting the proposed self-adaptation solution into the cloud context is briefly discussed as a step towards delivering adaptation as a service.

Part of this chapter's content was previously published, by the author, in:

- Nuno Oliveira and Luís S. Barbosa. "A self-adaptation strategy for service-based architectures". In: *Proceedings of the 8th Brazilian Symposium on Software Components, Architectures and Reuse*. SBCARS'2014. Distinguished with Best Paper Award. Maceió, Alagoas, Brasil: IEEE, Sept. 2014, pp. 1–10.

8.1 A self-adaptation approach

The dynamic evolution of the context where software systems are deployed usually induces both degradation of QoS levels and faults in meeting expected functional requirements. These quality drops and faults usually lead system performance to unsatisfactory levels. In order to continue delivering their services at a previously

agreed level, these systems have to dynamically adapt to the new context. But to perform such an adaptation, the system has to know itself as well as its surrounding context. Self-adaptive systems present exactly these abilities.

Self-adaptive systems are often distributed, component-based systems with highly demanding requirements. Differently to others, they present a subsystem (usually referred to as the managing system) that monitors both context and internals and uses that information to decide the need for and to enact reconfigurations at runtime. Of course, reconfigurations must be planned in advance, so that the system is able to know which one to choose for a given set of contextual attributes.

In this sense, the self-adaptation strategy proposed here is organised around two main phases, built on top of ARIS. One phase is offline and concerns the planning of possible reconfigurations by the software architect. It is assumed that reconfigurations can be planned in advance provided that a number of relevant contextual attributes are identified and translated into measurable variables. Indeed, suitable ranges of values for those attributes may help to plan (at design time) configurations that will, most likely, drive the system into stable, well-behaved states. The other phase is online and focuses on the autonomous selection (based on triggering mechanisms) of such planed reconfigurations to adapt a running system as part of a monitoring feedback loop.

The following sections propose a semi-formal account of the overall strategy.

8.1.1 The offline phase: planning reconfigurations

This phase is solely devoted to the preparation of the adaptation assets that, in the online phase, will be autonomously used. In the sequel, the set of all assets will be denoted as $\mathcal{A}$.

The first asset to be produced is a faithful model of the system architecture. This model comes under the form of stochastic coordination patterns. It is regarded as an abstract reflection of the system upon which performance analyses, simulations and property verifications are to be realised.

The second asset is concerned with the system (functional and non-functional) requirements. These are encoded into verifiable properties targeting concerns like behaviour, structure, QoS, among others. This set of properties, denoted henceforth as $\mathcal{P}rop$, is actually logically divided into five parts including functional (FUN), non-functional (QoS), system generic (SYS), environment specific (ENV) and ontological (ONT) properties. The last set of properties is related with the ontology of reconfigurations. *Properties* in that set are formed by concepts (regarded as propositions of classic propositional logic), and are to be validated over the reconfigurations that

are possibly applied on the system (not over the system itself).

Upon such properties, the architects shall define the system's adaptation logic by means of constraints. Formally,

Definition 8.1 (Constraint)**.** *Given a property $p \in \mathcal{P}rop$, which takes values in a domain D_p, the constraint associated to p is a predicate $\psi_p \subseteq D_p$. If v_p is the current value of p, then the associated constraint ψ_p holds if $\psi_p(v_p)$.*

The current value, v_p, of a property p (*i.e.,* the value resulting from evaluating p) is denoted by p itself.

Example 8.1 The following items represent valid constraints:

- $\mathsf{hold}_{\mathsf{FUN}.p}$, holds when property $\mathsf{FUN}.p$ is valid.
- $\geq^{0.9}_{\mathsf{QoS}.q}$, holds when the current value of $\mathsf{QoS}.q$ is *greater or equal than* the fixed reference value 0.9, *i.e.,* $\mathsf{QoS}.q \geq 0.9$.
- $\mathsf{min}^{S}_{\mathsf{SYS}.c}$, holds when the current value of $\mathsf{SYS}.c$ is the minimum in a given set of values S.

✠

The set of all possible constraints is denoted by Ξ. Constraints and their utility are further addressed in Section 8.2. A suitable set of constraints constitutes the third adaptation asset.

The final asset from this phase is concerned with preparing (modelling and analysing) reconfigurations. The architects plan them by taking into account both the system requirements and possible ranges of values for the attributes that characterise its environment. From this planning results a set of possible configurations and reconfigurations with a dependency relation between them. Such a dependency relation is captured by a *reconfiguration transition system (RTS)*. Formally,

Definition 8.2 (Reconfiguration Transition System)**.** *A RTS is a tuple $(C, \rightarrow, k_i)$, where $C \subseteq \mathcal{P}^{sto} \times 2^{\Xi} \times 2^{\mathcal{A}}$ is a set of configuration states, $k_i \in C$ is the initial configuration state and $\rightarrow \subseteq C \times \mathcal{R} \times C$ is a symetric transition relation.*

A RTS is, in essence, a labelled transition system. Transitions from each state κ are labelled with the reconfigurations that can be applied to the stochastic coordination pattern associated to κ. States represent valid configurations of the deployed systems. Each state is actually composed of a coordination pattern; a set of state-specific constraints, which enable finer decisions (details further in Section 8.2); and a set of necessary assets for the analysis *e.g.,* PRISM specifications, that are obtained after conversion of the stochastic coordination pattern into $\mathsf{IMC}_{\mathsf{Reo}}$ (*c.f.,*

Algorithms A.1 or A.2). Doing the latter in this phase improves the efficiency of the online phase as conversions into $\mathsf{IMC}_{\mathsf{Reo}}$ and (quantitative analysis) tool-specific formats is costly.

8.1.2 The online phase: the feedback loop

The online phase consists of a monitor feedback loop (which springs from traditional MAPE-K approaches [160, 153]) built on top of ARIS. Figure 8.1 depicts its main elements.

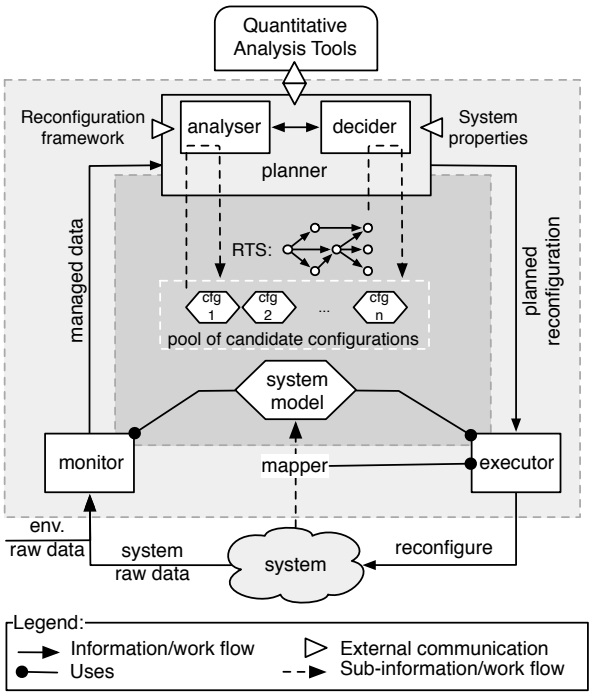


Figure 8.1: Feedback loop based on a reconfiguration transition system.

This is referred to as a feedback loop based on a RTS, because the transition system of reconfigurations is a first-class entity in this approach. Globally, the implementation of this feedback loop approach requires: (*i*) a RTS; (*ii*) a model of the deployed system; (*iii*) a mapper, which maps concrete connections to services to the logical ports of the model; (*iv*) the instant observations (measures) of the system properties; (*v*) a pool of candidate configurations (and their analysable assets); (*vi*) the mechanisms in ARIS for reasoning about the possible reconfigurations; (*vii*) the properties of interest of the system and (*viii*) the tools for quantitative analysis of the configuration analysable assets.

Two invariants assert that (*a*) the current state (*i.e.,* the current configuration) of a RTS always points to the current configuration model of the system architec-

ture and (*b*) the pool of candidate configurations only contains models obtained by applying to the original model a valid reconfiguration script.

In the sequel it is detailed how the three main components (*monitor, planner* and *executor*) work together to achieve adaptability.

Monitoring

The *monitor* component aggregates data from the deployment environment and the system itself. Probes are assumed to collect different sorts of data, depending on the variables that drive the adaptation. Latency, throughput, bandwidth, number of clients, number of servers or type of connection (*e.g.,* wifi, bluetooth, GSM) are typical variables. The *monitor* uses the information from the *mapper* to associate raw data from the system to the model, which is then used as-is by the *planner* component (specifically, the *analyser* sub-component). Figure 8.2 shows a UML sequence diagram which describes the interaction between these elements.

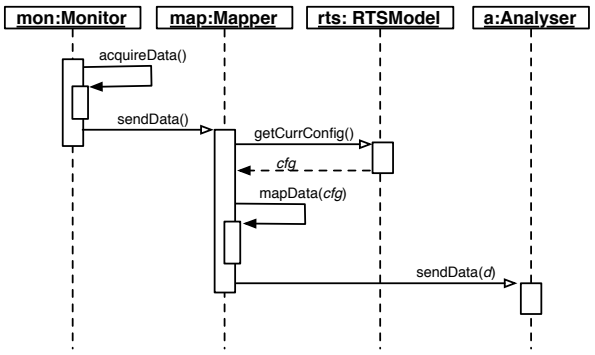


Figure 8.2: Sequence diagram for the *Monitor* component.

Planning

The *planner* is made out of two components: the *analyser* and the *decider*, that work together to plan (if necessary) the most adequate adaptation to the given context. These components rely ARIS to formally verify the functional and non-functional properties of the architecture. Figure 8.3 shows the sequence diagram for such a component. Therein, *FPChecker* and *NFPChecker* entities refer, respectively, to interfaces for the suitable functional and non-functional property analysing tools.

In a first step the *decider* uses the RTS for picking all the configurations reachable from the current state. This action creates a pool of candidate configurations along with their *pre-compiled* analysable assets. In a second step, the *analyser* reduces the pool by discarding configurations that fail to meet the required functional properties.

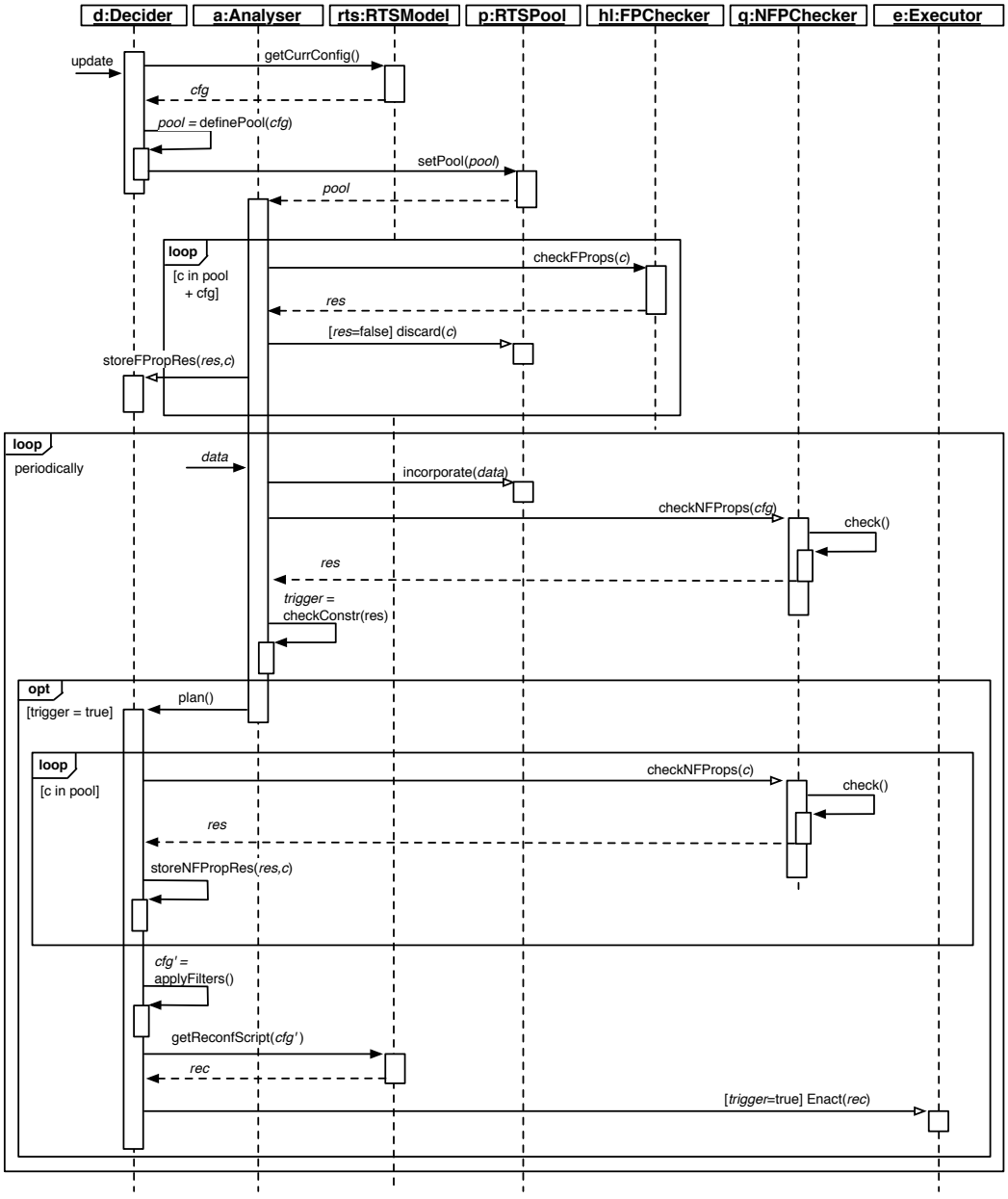


Figure 8.3: Sequence diagram for the *Planner* component.

These two steps are performed only once each time an adaptation occurs, or every time the functional properties of the system change.

Then, on a periodic basis, the *analyser* incorporates this data into the analysable assets of each configuration in the pool. This is used to check for non-functional properties of the current configuration. Whenever non-functional properties fail, a reconfiguration is triggered. At this moment, the *decider* is responsible for choosing a suitable configuration (and associated reconfiguration operation) from the pool to embody the adaptation step. This choice, which is part of what is called the

triggering of a reconfiguration, is made based on the results of the (qualitative and quantitative) analyses performed.

Execution

The *executor* component (whose sequence diagram is depicted in Figure 8.4) receives the reconfiguration selected and applies it to the running system. In particular, it

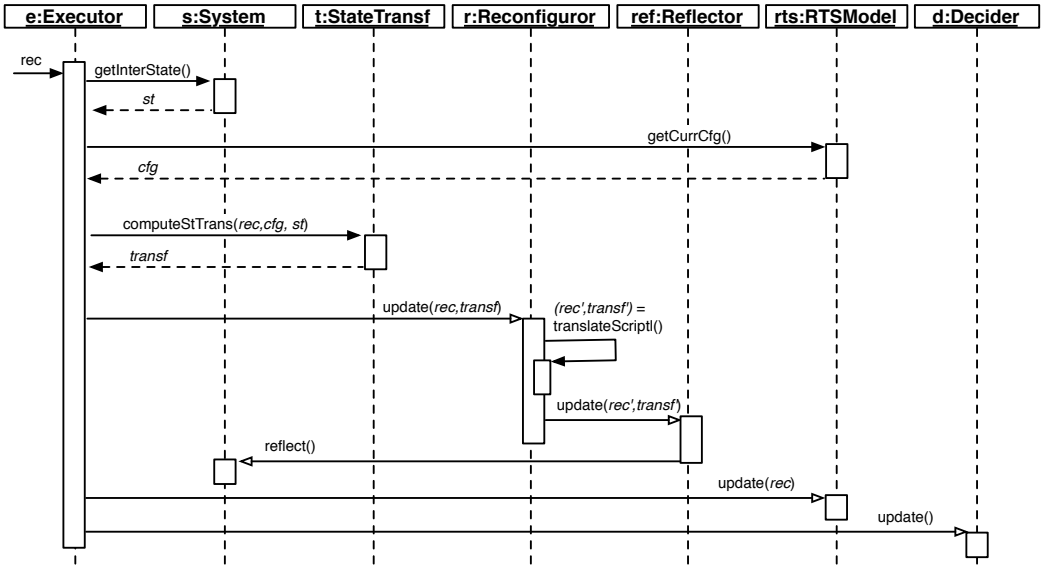


Figure 8.4: Sequence diagram for the *Executor* component.

computes the resuming state (as introduced in Section 6.3) and translates it, along with the selected reconfiguration, into an executable reconfiguration script to be applied over the system. This is done resorting to a *Reconfiguror* entity. A *Reflector* entity, awaits for the system to reach a quiescent state; when such a state is attained, it makes the system reflect the changes by applying the reconfiguration script.

Concurrently, a sequence of updates are made: the system model is substituted by the selected configuration; the state of the RTS is updated accordingly, to meet the first feedback loop invariant; and finally, the candidate configurations in the pool are substituted by new candidates, computed in the new system's state by the decider component (*c.f.,* Figure 8.3).

8.2 Triggering of reconfigurations

Usually, a reconfiguration of a system is enacted whenever a non-functional property fails, violating the *service level agreement (SLA)* contract. However, this vision is not always enough since the company owning the adaptable system may have other

objectives besides providing the performance stated in the SLA. For instance, diminishing the financial costs of the system or agreeing to new functional requirements may constitute part of these objectives.

Actually, in the approach proposed here, the adaptation triggering is lead by a propositional logic formula, where propositions are constraints, validated against the current system configuration. These triggers, referred to as *trigger constraints*, establish both the objectives of the company *w.r.t.* the system and, consequently, the adaptation rationale.

Example 8.2 For instance, $\mathsf{hold}_{\mathsf{FUN}.p} \wedge \geq^{0.9}_{\mathsf{QoS}.q} \wedge \mathsf{min}^{S}_{\mathsf{SYS}.c}$ defines a trigger constraint that enacts an adaptation when functional property p does not hold in the current configuration, the measure for the non-functional property q is not above 0.9 or system specific property c is not the minimum (when compared to the same property of candidate configurations). ✠

The property prefixes may be omitted when the properties' provenance is clear from the context.

Once a trigger constraint is violated, the adaptation is unavoidable. But choosing a suitable new configuration is a complex task. It may even be non-deterministic or lead the system to an (infinite) chain of reconfigurations. To avoid this, it is necessary to define a base strategy to direct the choice of such configurations. A notion of filter is introduced for this.

Definition 8.3 (Filter)**.** *Let* $c_{1_1}, \ldots, c_{1_n}, c_{2_1}, \ldots, c_{2_m}, \ldots, c_{k_1}, \ldots, c_{k_l} \in \Xi$*. A filter is a vector of vectors* $\langle\langle c_{1_1}, \ldots, c_{1_n}\rangle, \langle c_{2_1}, \ldots, c_{2_m}\rangle, \ldots, \langle c_{k_1}, \ldots, c_{k_l}\rangle\rangle$*, where only the first element is mandatory.*

Notation

$c_{1_1}, \ldots, c_{1_n} | \ldots | c_{k_1}, \ldots, c_{k_m}$ is used to separate the several elements of a filter.

A filter is used to discard, in sequence, candidate configurations that do not verify the constraint property.

Example 8.3 Consider the filter

$$\mathsf{hold}_{\mathsf{ONT}.\mathit{beh\text{-}unobtrusive}}, >^{105}_{\mathsf{QoS}.\mathit{throghput}}, \mathsf{max}^{C}_{\mathsf{QoS}.\mathrm{availability}}$$

The application of this filter discards, in a first step, all the configurations reached after application of reconfigurations that are not *beh-unobtrusive*; then, in a second step, it discards from the remaining candidate configurations those that do not deliver non-functional

property *throughput* above value 105 and; finally, in a third step, it takes (from the remaining configurations, represented by set C) the one that delivers the maximum value for property *availability*. ✠

However, in some situations the filter may either discard all configurations or more than one configuration may prevail. In these cases it is possible to add optional filters to be used whenever the previous filters do not find a suitable configuration.

Example 8.4 Consider the following filter

$$>^{105}_{\mathsf{QoS}.throughput}, \mathsf{max}^{C}_{\mathsf{QoS}.availability} \mid >^{95}_{\mathsf{QoS}.throughput}, \mathsf{min}^{C}_{\mathsf{SYS}.cost}$$

In case no configuration is able to deliver a value above 105 for property *throughput* or the second constraint will pick more than a one configuration with a maximum value for property *availability*, then the optional filter (the one after '|') is applied to all the pool of configurations and it will discard those that do not deliver a value above 95 for property *throughput* and it will pick the one presenting the lower *cost* for the system. ✠

More optional filter elements may be added to prevent that none or more than one configuration remain. If even though, still multiple configurations prevail, the default is to select the first in a ranking that contemplates the results for a prioritisation of requirements. However, for an even finer and controlled selection of suitable configuration, it is allowed the specification of constraints in each state of the RTS (*c.f.,* Definition 8.2). These act as specific pre-conditions to the inclusion of the corresponding configuration in the pool of candidates.

8.3 Adaptation as a Service

The self-adaptation strategy approach proposed above can be reused in different systems since only its central pieces (properties, constraints, filters and the RTS) are system-dependent. This assures the expected separation of concerns between managed and managing systems [255]. Such a separation is not a novelty. Most of the similar self-adaptation approaches promote it [127]; and the MAPE-K reference model almost obliges it. However, notwithstanding the separation of concerns, managed and managing systems are usually running in the same physical environment. This makes the adaptable system to decrease its performance, since the feedback loop allocates part of the available resources for its own use.

The essential solution for such a problem is to physically separate both entities. This entails the need for companies to acquire more processing and storage power

as well as to be willing to manage such extra resources with all the costs associated. A smoother solution is to rent virtual machines from a cloud service, and deploy therein the feedback loop system. On the one hand, this eases management, but on the other hand it requires an extra effort in order to set the whole system up.

In order to avoid these problems, a new strategy towards delivering *adaptation as a service (AaaS)* is proposed next.

8.3.1 Architecture and main workflow

The essential components of the feedback loop proposed in Section 8.1 (monitor, analyser, decider and executor) are loosely coupled entities with a specific behaviour. Regarding them as services is therefore natural. With this in mind, one proposes to refactor the self-adaptation strategy so that the essential parts of the feedback loop are deployed in the cloud for immediate usage. The expected result is that the common computational activities for adaptation (*e.g.,* analysing data for perceiving the need for adaptation or deciding which reconfiguration to chose among a set of possible ones) are transparent to (and not developed by) the users. Figure 8.5 presents an overview of the expected global architecture, along with traces of the main workflow for both users and the adaptation service. In the next paragraphs, the *adaptation service* will be referred to as AaaS, and the hosting cloud as *AaaS cloud*.

Into the AaaS cloud

With effect, all the tasks that are known to be time and resource consuming are encapsulated as services. In particular, it is assumed the existence of online versions of well-established analysis tools (*e.g.,* CADP, PRISM, IMCA, HyLoRes, among others) that make available, through public interfaces, services of which AaaS will be client. Moreover, the tools associated to ARIS are also assumed to be available as services in a dedicated cloud environment. These two sets of services are expected to release most of the workload from the feedback loop.

The feedback loop constitutes the core of the AaaS. In this context, it is made more comprehensive by supporting multiple monitoring and decider components, in an attempt of decentralising the feedback loop [255, 250, 10, 201]. This comes with the price of extra coordination and synchronisation efforts. But it is essential. For instance, instead of having a single filter-based strategy to decide reconfigurations, one can have several others, including one that uses artificial intelligence techniques (*e.g.,* case-based reasoning) to make such a decision. The results of all decider

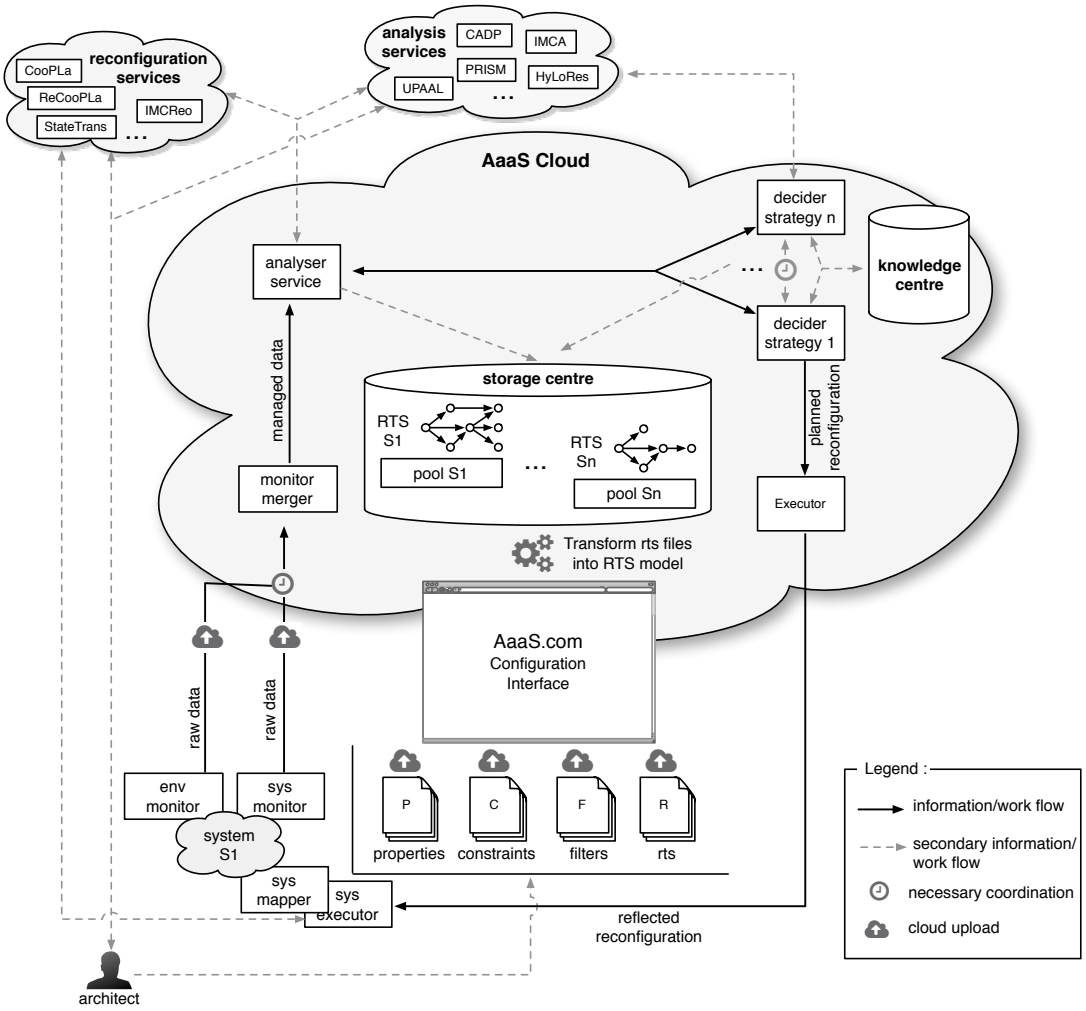


Figure 8.5: Adaptation as a Service architecture overview.

components have to be coordinated, somehow.

AaaS is able to track more than one single system. The cloud support for multi-tenancy and the service-orientation of the approach allow AaaS to deliver the same adaptation service with the same expected quality to several systems. For this, each tracked system is given a space in a storage centre, where, for instance, the RTS and the current pool of configurations are placed.

AaaS remains loyal to the coordination-centred vision for reconfigurations, though. Although there must be a main coordination entity for a distributed system, there can be several sub-coordination entities distributed in several nodes of the same system that are themselves tracked by AaaS. Again, this is based on the theories for feedback loop decentralisation discussed by D. Weyns et al. [255, 250], and consequently, requires a distributed notion of coordination-targeted reconfigurations [164], which is out of the scope of this thesis.

In the remaining of this section, the offline and online phases of this cloud based approach for system adaptation are exploited.

The offline phase

In this phase the architects have to prepare the assets (as suitable files) that make adaptation possible. This includes the system properties, that translate functional and non functional requirements; the constraints, that define the system goals for adaptation; the filters, that define the main strategy for deciding the reconfiguration to lead the system into a desired configuration; and finally the RTS that lays down a consistent path of adaptation.

The production of the RTS is a complex and time-consuming task. To help the architects accomplishing it, the reconfiguration services assumed can be used. The analysis tools to fine tuning thresholds and other measures are also assumed to be used from the available services. In the end, the RTS is expected to be delivered as a comprehensive set of files written in languages suitable for the definition of coordination patterns and reconfiguration scripts (*c.f.,* Sections 9.1 and 9.2). Together with the other assets, all these files have to be uploaded to the AaaS cloud through the configuration interface as depicted in Figure 8.5. Once uploaded, the RTS files are transformed into a RTS model and all the associated assets (*e.g.,* the final PRISM files) of each state are conveniently generated and stored in the storage centre.

The configuration interface is expected to guide the architect through all the configuration of an instance of AaaS. Besides the upload of the required files, the architect is also able to choose, for instance, which analysis tool(s) shall be used to

verify the properties of the system or which strategy(ies) shall be applied to decide the reconfigurations to apply when needed.

In addition, the architect is responsible for coupling monitors to its systems that are able to ship data to the AaaS cloud every time a (relevant) change occur either in the environment or in the internals of the system. Also, the architect has to define a local mapper component that contributes a reflection model of the managed system. A local executor component, actually an AaaS off-the-shelf component, has also to be attached to the system. The usefulness of these extra components is exploited next.

The online phase

When the configuration is over and the architect decides to explicitly enable AaaS to manage the system, the online phase begins.

As expected, monitors send data to the AaaS cloud, which is synchronised and merged therein. A monitor merger service is assumed to merge the monitored data and send it to the analyser service. The latter behaves exactly as before. The particularity is that it now evokes services for the necessary quantitative analysis. It is still responsible for triggering the need for a reconfiguration by analysis of the user-uploaded constraints.

When an adaptation is triggered, the decider (or deciders) starts the necessary analysis to plan a new adaptation. Depending on the user's configuration, one or more decision strategies may be associated to the managed system. Each strategy is different. For instance, the filter-based strategy uses the analysis services to analyse the configurations in the pool, which are send as a unique workload. A strategy adopting case-based reasoning mechanisms would delegate its tasks into services to that end, but will rely on a knowledge centre to define its decision, as depicted in Figure 8.5. The decider service is also responsible for updating the pool of configurations, as explained in Section 8.1.

Upon decision, the chosen reconfiguration is passed to the executor. The executor translates the reconfiguration into a script able to concretely apply the changes to the managed system. This script is passed to the local executor component. The latter uses the reconfiguration services to compute the resuming state, and when the system enters a quiescent state, applies the changes via the mapper component.

The option for having a local executor component dues to the fact that the AaaS is not aware of the internal state of the systems it manages. Thus, interrupted and resuming states have to be computed locally. This is also necessary because such states have to be computed in the instant before the changes are applied to the

system, so that the managed system consistently resumes its production.

8.3.2 Discussion

The AaaS approach has potential to bring several benefits when compared to other approaches. It promotes a clear (physical) separation between the managed system and its feedback loop. It allows architects and developers to focus on the design and development of the system and, consequently, frees them from dealing with the always complex implementation of feedback loop components. It eases the evolution of legacy static systems into self-adaptable ones. It allows for more comprehensive and robust decisions, by enabling the combination of several strategies. It enables the decentralisation of the feedback loop, augmenting the dependability of the system as a whole.

AaaS would be a one-size-fits-all approach for adaptation. This can be seen as a drawback, but in fact the approach is highly configurable in order to support the demands of their tenant systems. In fact, the adaptation logic is mainly delivered by the architects in the uploaded analysable assets. AaaS essentially performs intense computations in order to deliver decisions based on such assets. The adaptation logic is not static. At any time the company may change its goals or the system requirements; the architects may update the RTS to cope with new system configurations. This entails the need for re-uploading new asset files. The AaaS is expected to reconfigure its behaviour to conform to these changes immediately. Moreover, the customisation of AaaS behaviour (*e.g.,* which deciding strategies to use) can be performed at any time, as well.

8.4 Summary

This chapter introduced an approach for self-adaptation of software architectures. It is essentially a demonstration of a practical application of ARIS.

The approach is organised in two phases. The first phase is offline and is concerned with the preparation of assets that are to be used during the second phase. A relevant asset produced in this phase is the so called RTS, a transition system of reconfigurations. Each state is a configuration of the system that copes with a defined set of conditions; transitions are labelled with reconfiguration scripts. The second phase is online (*i.e.,* at runtime) and is concerned with the actual monitoring of the adaptable system and the inspection of the need for adaptation. This phase is in line with the concept of feedback loop, as proposed in autonomic computing or control

theory. It follows the MAPE-K approach closely. However, the main components of that approach (analyser and planner) are built on top of ARIS.

Particular attention was given to reconfiguration triggering. It was defined a simple, yet effective, way of deciding when should the system adapt itself. Furthermore, a notion of filter was devised that allows for the definition of a strategy for choosing the right reconfiguration for the current context.

Finally this approach was refactored in order to be ported into a cloud context. This was proposed as an attempt of delivering adaptation as a service.

Part III

Tool support and case study

Chapter 9

Tool Support

Man is a tool-using animal. Without tools he is nothing, with tools he is all.

– *Thomas Carlyle*

In this chapter. Tool support is introduced to the ARIS framework. In particular, the CooPLa language is introduced for modelling coordination patterns, whether they present stochastic features or not. Its companion language, ReCooPLa, is also introduced, for modelling reconfigurations. These languages and associated tools (*e.g.,* compilers and processors) are bundled in an *integrated development environment (IDE)*, referred to as CooPLa Editor.

Part of this chapter's content was previously published, by the author, in:

- Flávio Rodrigues, Nuno Oliveira, and Luís S. Barbosa. "ReCooPLa: a DSL for Coordination-based Reconfiguration of Software Architectures". In: *3rd Symposium on Languages, Applications and Technologies*. Ed. by Maria J. V. Pereira, José P. Leal, and Alberto Simões. Vol. 38. OpenAccess Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, June 2014, pp. 61–76.

9.1 CooPLa

CooPLa (standing for **Coo**rdination **P**atterns **La**nguage) is a lightweight *domain-specific language (DSL)* for modelling coordination patterns, as formally introduced in Chapter 6. It was thought for software architects, as a tool for designing the coordination layer of a system. The language has roots on the Reo formalism and, as a byproduct, allows for specifying Reo circuits. It is not intended, though, to be another Reo-specific notation like CARML or RSL [27, 162].

CooPLa allows for defining the graph structure underlying coordination patterns, by connecting edges, which are instances of *channels*. In the sequel, the details of

CooPLa are exploited. For a detailed account of its syntax, the grammar is presented in Appendix B.

9.1.1 Channels

In CooPLa, a channel is specified as a structure with input and output ports and a well defined behaviour. Ports are defined on the channel signature (which resembles the signature of a function in traditional programming languages), and are positionally grouped as input and output ports. The behaviour is defined in the body of the channel, by a set of rules defining how data flows within the channel. Such rules are specified taking into account the stimuli received on the channel ports and the internal configuration of the channel.

Internal specificities

Channels may present different flavours. CooPLa enables the definition of channels with (*i*) a clock, to impose delays in a normal data flow; construct `@T` is used for that end, where `T` is the delay; (*ii*) a datatype pattern-based condition, to decide data flow based on the matching result; `cond=<...>` is the corresponding construct; and finally (*iii*) an internal structure (*e.g.,* a buffer), to store data in asynchronous communications; this is expressed with construct `~N`, where `N` is either a variable or a (comma-separated) list of variables, that define the dimension of the structure. Whenever a channel is specified with an internal structure, it is required that both a state and a set of state observers are defined in its body. The state is a name refering to the structure and state observers are predicates over it. The latter are used to check relevant properties of the channel internal structure (and are implemented in an external programming language).

Behaviour

The behaviour of a CooPLa channel is specified as a set of flow rules of the form `R->f`, where `R` is a subset of ports of the channel or internal state observers (possibly negated with `!`); and `f` is a flow as explained below. The meaning of each rule is that flow `f` occurs whenever there are *input/output (IO)* requests at the ports in `R` and the associated state observers hold.

A flow is defined by the construct `flow p1 to p2`, where `p1` and `p2` are ports of the channel or `NULL` (a special port where data is lost or automatically produced). In structured channels, `p1` and `p2` may refer to the name of the channel internal structure. This construct defines that data flows from `p1` to `p2`. Furthermore,

flows may occur atomically-synchronised or datatype-conditioned. Construct `f1|f2` is used to express atomic synchronisation between flows `f1` and `f2`. Construct `cond ? f1 : f2` is used to trigger flow `f1` if there is a match between pattern `cond` and the data, or `f2`, otherwise. Such conditioned flow is employed, *e.g.,* when a channel filters data based on a datatype pattern.

Inheritance

CooPLa supports simple inheritance for channels. Construct `extends b` is appended to the signature of a channel that is to extend a channel `b`. In this context, a new channel may extend a basic one through the addition/redefinition of flows. When internal specificities are inherited, these are, however, not changed. Internal structure reference name and state observers remain the same as defined in the base channel. A requirement for inheritance is that the new channel presents equal interface and similar internal specificities.

Example 9.1 Figure 9.1 shows how well-known channels in Reo can be defined in CooPLa.

```
channel sync(a:b) {
  a,b -> flow a to b ;
}

channel lossy(a:b) extends sync {
  a,!b -> flow a to NULL ;
}

channel fifo~N(a:b){
  state: buffer; observers: E, F;
  a,!F -> flow a to buffer ;
  !E,b -> flow buffer to b ;
}

channel drain(a,b:)  {
  a,b -> flow a to NULL
           |
         flow b to NULL ;
}
```

```
channel shiftFifo~N(a:b)
                  extends fifo {
  a, F -> flow buffer to NULL
                    |
          flow a to buffer;
}


channel filter(a:b:c=<_,_>) {
  a,b -> c ?
       flow a to b
            :
       flow a to NULL;
}


channel timed@T (a:b) {
  a,b -> flow a to@T b;
}
```

Figure 9.1: CooPLa description of some Reo channels.

The `sync` channel is seen as a structure with input port `a` and output port `b`. Whenever there are IO requests pending at both ports simultaneously, data flows from `a` to `b`.

The `lossy` channel extends `sync` with an extra flow: whenever there is a write (input) request at port `a` and no read (output) request at port `b` (notice the use of '!' to convey negation), data is lost (*i.e.,* it flows from `a` to `NULL`).

The `fifo` channel is a structure with input port `a`, output port `b`, a state named `buffer` with dimension `N` and observers `E` and `F` (specified in an external programming language) that check whether `buffer` is (E)empty and (F)ull, respectively. The behaviour of this

channel is defined taking into account the pending requests at the ports as well as the configuration of its internal structure.

The `drain` channel expects simultaneous stimuli at its two input ports; whenever this clause is fulfilled, data flows atomically synchronised (notice the use of construct '|') from each of these ports to `NULL`, being lost.

The `shiftFifo` channel inherits from the `fifo` channel. Notice that interface is equal and it also defines an internal structure, but it does not redefine the reference name and the observers for such structure. Behaviourally, it defines a new flow rule expressing that whenever there is an input stimulus at port `a` and the `buffer` is full, a datum in `buffer` is lost and the datum in `a` flows to `buffer`.

The `filter` channel presents a signature added of a datatype pattern `c` that matches data of sort pair (with components of any sort). This pattern is used in the definition of its flow rule. The latter expresses that whenever there are pending requests at both ports of the channel, if the sort of the data to be written matches pattern `c`, then data flows from `a` to `b`; otherwise it is lost.

```
1 channel router (a: b,c) {
2   a,b,!c -> flow a to b;
3   a,c,!b -> flow a to c;
4   a,b,c  -> flow a to b;
5   a,b,c  -> flow a to c;
6 }
```

Figure 9.2: CooPLa description of the router channel.

The `timed` channel is defined with a clock. Such clock imposes a delay `T` on the flow of data from ports `a` to `b`, whenever there are IO stimuli on the channel interface.

But not all channels in CooPLa have exactly two ports. In order to faithfully cope with the formal definition of channels (Definition 6.1), no upper bound is set to the number of ports that a CooPLa channel may present; the lower bound is two, though.

Figure 9.2 shows the `router` channel. It is presented as a structure with one input port and two output ports. Notice also the existence of two flow rules that are triggered whenever there are pending requests at all the channel ports. This explicitly expresses nondeterministic behaviour of the channel. Consequently, data will flow nondeterministically from port `a` to either port `b` or port `c`. This is an example of where CooPLa deviates from the typical notion of channel as in the Reo formalism. ✠

9.1.2 Patterns

In CooPLa, a `pattern` is the main construct of a coordination pattern. A pattern is specified by its interface and a body of interconnections. Similarly to channels, the

ports are defined in a signature, and positionally grouped as input and output ports. The body of interconnections is where the desired coordination graph is obtained, in two stages: the channel instantiation and the connection configuration.

Channel instantiation

CooPLa takes the reserved word `use` as the delimiter for the beginning of the first stage in coordination patterns construction. In this stage, all the elements to be connected in the pattern are instantiated (*i.e.,* declared). Such elements may be instances of channels or other patterns. The latter enables the reutilisation and creation of complex coordination structures.

An element may be seen as a variable in traditional programming languages. Its instantiation takes the full signature of the channel (or pattern) to define its type. The full signature is required in order to concretely define both logic names to refer to ports of the channels (or patterns) and the dimensions for channel internal specificities like clocks or structures. For channels that define a structure it is mandatory to specify which state observer holds at its initial configuration.

Connection configuration

The reserved word `in` delimits the beginning of the second stage in coordination patterns construction. In this stage, the graph of elements is assembled according to the concrete definition of the pattern interface.

The concrete definition of the pattern interface is expressed as a set of assignments of elements' ports to the names of the pattern ports. The port of an element is accessed via the '.' (`dot`) construct as in `el.p`. Here, `el` is the name of an instance and `p` is the logic name of a port of that instance, both as defined in the instantiation stage.

The assembling of the coordination graph is done through the `join` construct. This construct takes a fresh name and a set of element ports (accessed via the `dot` construct) and performs the connection of the elements in a single graph node which is renamed to the provided fresh name. Although this is the typical connection operation, CooPLa also supports the creation of special nodes with the `xor` construct. This construct takes a fresh name, a set of output ports and a set of input ports and creates a node with the specific semantic overload of an *exclusive or* data router. In practice, this is a shorthand for defining a channel or pattern with such a behaviour (*e.g.,* `router` channel in Figure 9.2) and connect its ports to the ports of other

channels.

Example 9.2 Consider the Sequencer coordination pattern depicted in Figure 6.1. Figure 9.3 shows how that coordination pattern is implemented in CooPLa.

```
1  pattern Sequencer (a : o1, o2, b) {
2    use :
3      sync(i:o) as s1, s2, s3, s4;
4      (E) fifo~1(i:o) as x;
5    in:
6      a  = s1.i;
7      o1 = s2.o;
8      o2 = s3.o;
9      b  = s4.o;
10     join [s1.o, s2.i, x.i] as cde;
11     join [x.o, s3.i, s4.i] as fgh;
12 }
```

Figure 9.3: CooPLa specification of the Sequencer.

As expected, both CooPLa channels and patterns have a specific mapping to channels and coordination patterns as defined in Chapter 6. It is worth to point out the relation between CooPLa channel names and the types in the formal representation of channels. Indeed, as explained in the beginning of this section, CooPLa provides the essential type system associated to the edges of a coordination graph. Another relevant aspect is that the `join` construct of CooPLa defines the desired partition of all channel ends into nodes of the coordination pattern.

9.1.3 Stochastic extension

CooPLa is extended with mechanisms for supporting coordination structures that present stochastic behaviour. A limitation is the assumption that such stochastic behaviour is always modelled by exponential distributions. Since the only parameter of interest for such probability distributions is their rate (*i.e.,* the number of independent events occurring per unit of time), then, this is the only value that CooPLa requires to be specified along with patterns and channels.

In this context, two event sets are considered: request production and data processing. Request production rates are associated to the ports of coordination patterns and essentially model the environment; data processing rates are associated to both the channels (*i.e.,* to their specific flows) and the nodes of coordination patterns.

Since channels and patterns can be used in the context of other patterns, these rates shall not be bounded to their definitions. Rather, CooPLa introduces the notion of stochastic labels and stochastic instances of patterns.

Stochastic labels

Stochastic labels are just a syntactic annotation appended to the end of each flow rule in the body of a channel. Construct `#L`, where each `L` is a unique identifier within the channel, provides the relevant mechanism to later associate processing rates to the flows.

Figure 9.4 shows how the `sync` and the `fifo` CooPLa channels coded in Figure 9.1 are extended with stochastic labels in their flow rules.

```
1 channel sync(a:b) {
2   a,b -> flow a to b #proc;
3 }
```

```
1 channel fifo~N(a:b){
2   state: buffer; observers: E, F;
3   a,!F -> flow a to buffer #inB;
4   !E,b -> flow buffer to b #Bout;
5 }
```

Figure 9.4: Channels added of stochastic labels

Inheritance in CooPLa is still compatible with stochastic labels. Channels that inherit from a channel specified with stochastic labels will also inherit such labels. Unlike what happens to internal structure reference name and state observers, the stochastic labels associated to a specific flow may be renamed. This is valid as long as the new names are unique in the context of the channel.

Ports and internal nodes of CooPLa patterns are overloaded with the same role played by stochastic labels in channels.

Note that no stochastic information is directly associated to the channel ends; rather, it is added to the nodes (representing the approximated rates for enqueueing and dequeueing data).

Stochastic instances

Stochastic instances are instances of coordination patterns that specify concrete values for the rates associated to the environment request production and data processing within channels and nodes.

Request production rates are associated to the interface names of the pattern. The data processing rates are associated to both the channels (stochastic labels) and the pattern internal node names. The information associated to the latter is twofold as it considers the rates of enqueueing and dequeueing data to and from a channel end.

The association of the rates is defined as a typical assignment, where the tradi-

tional symbol = is replaced by @.

Example 9.3 Figure 9.5 shows how the stochastic instance of the Sequencer coordination pattern (*c.f.,* Figure 6.13) is specified in CooPLa. This example, covers all its the relevant syntactic aspects. The construct `stochastic` defines these structures and requires a pattern to be referenced (to specify the type of instance being created) and the name of the instance provided as a unique identifier (line 13). Finally, the list of rates is specified.

```
1  stochastic Sequencer {
2    a        @ 50.0;
3    o1       @ 30.0;
4    o2       @ 25.0;
5    b        @ 50.0;
6    s1#proc  @ 100.0;
7    s2#proc  @ 150.0;
8    s3#proc  @ 150.0;
9    x#inB    @ 90.0;
10   x#Bout   @ 80.0;
11   cde      @ (955.0, 900.0);
12   fgh      @ (800.0, 1000.0);
13 } sseq
```

Figure 9.5: Stochastic instance `sseq` of the Sequencer coordination pattern.

✠

The stochastic labels of channels are accessed using the # construct applied to the channel name (as used in the context of the coordination pattern being instantiated). The rates for internal nodes are provided as pairs of float values, where first and second components refer, respectively, to enqueueing (reading to node) and dequeueing (writing to channel) events.

A stochastic instance can only be defined if all the channels constituting the corresponding pattern provide stochastic labels. Moreover, a stochastic instance is only correctly defined if rates are assigned to all the interface ports of the pattern and to all the stochastic labels of the constituting channels. Defining rates for internal nodes is optional.

9.2 ReCooPLa

ReCooPLa (standing for **Re**configuration of **Coo**rdination **P**atterns **La**nguage) is a DSL for modelling reconfiguration patterns, as formally introduced in Chapter 6. This language is the necessary companion of CooPLa to completely establish the bridge between theory and practice, in the context of this Ph.D. work. ReCooPLa and associated tools were designed and implemented in collaboration with Flávio Rodrigues, in the context of his M.Sc. thesis [227].

The objective of ReCooPLa is the specification of structural changes on the graph underlying the coordination patterns. It does so based on a small set of constructs designed with an imperative programming paradigm flavour. The full grammar of ReCooPLa is delivered in Appendix C. A comprehensive operational semantics for the language constructs was reported in Rodrigues' thesis [227, 228].

9.2.1 Reconfigurations

In ReCooPLa, a reconfiguration is specified similarly to a function in traditional programming languages. It is composed of a signature and a body, and it is always applied to, and always returns, a coordination pattern (henceforth, referred to as the *target* coordination pattern).

The signature is where the relevant arguments for the reconfiguration are expressed, optionally, grouped by datatype. Dtatypes in ReCooPLa are limited to: `Channel`, a reference to a channel structure as defined in CooPLa; `Pattern`, a reference to a pattern structure as defined in CooPLa; `Name`, an identifier referencing either a channel identifier or the ends of a channel; `Node`, an identifier referring to a set of names referencing channel ends; `XOR`, a particular case of `Node` with at least one input channel end and no less than two output channel ends; and finally, `Set<T>`, `Pair<T>` and `Triple<T>`, a set, a pair and a triple of elements of the generic datatype `T`, respectively.

The body of a reconfiguration is a list of instructions. These instructions are limited to variable declaration, assignments, an iterative control structure and the application of reconfigurations. Declarations and assignments are used to prepare arguments for reconfiguration calls and are expressed as in traditional programming languages. The control structure, with construct `forall(T e : s){b}`, allows for iterating over elements `e` in set `s` of datatype `T` in the body `b` of the control structure (which is as the body of a reconfiguration). Finally, the application of a reconfiguration is the only instruction that performs changes to the *target* coordination patterns; construct `p @ r` is used to this end, where `r` is the name of a reconfiguration (either primitive or previously defined) and `p` is a reference to a variable of type `Pattern`. The latter can be omitted. When such is the case, `r` is applied to the *target* coordination pattern.

Primitive reconfigurations

ReCooPLa includes a set of primitive reconfigurations corresponding to those introduced in Chapter 6. They are built-in operations that act uniquely as operands of

the `@` construct. Table 9.1 defines the signatures of each primitive aligned with their corresponding formal names.

Table 9.1: Primitive reconfigurations and their counterpart formal names

ARIS	ReCooPLa
$\mathbf{1}_r$	`id()`
const_ρ	`const(Pattern p)`
par_ρ	`par(Pattern p)`
join_N	`join(Set<Node> N)`
split_n	`split(Node n)`
remove_c	`remove(Name c)`

Operations

Besides the `@` operator, ReCooPLa provides a fixed set of operations specific for each datatype. This is essential for expressions that prepare arguments for the reconfiguration applications.

`Node`, `XOR` and `Name` are regarded as primitive datatypes, and in ReCooPLa they have no specific operations associated. Contrarily, the `Set<T>` datatype counts with the typical set operations: `+` (union), `&` (intersection) and `-` (subtraction). `Pair<T>`, `Triple<T>`, `Pattern` and `Channel` are regarded as structured datatypes. As such, ReCooPLa provides the `.f` operator to access to the elements of their structure. Here, `f` is either a field or an operation provided by these structures. In particular, `f` can be one of the following list:

- `in`, to retrieve the set of input ports from `Pattern` and `Channel` variables. Optionally, `in(i)`, for `i` a 0-based integer index, can be used to retrieve a specific port;
- `out`, to retrieve the set of output ports from `Pattern` and `Channel` variables. The optional index is also supported;
- `name`, to retrieve the name of a `Channel` variable;
- `nodes`, to retrieve all interface and internal nodes of a `Pattern` variable;
- `names`, to retrieve all channel identifiers associated to a `Pattern` variable; and
- `fst`, `snd` and `trd` are, respectively, to access the first, second and third projection of `Pair<T>` and `Triple<T>` variables. With `trd` being only available for the `Triple<T>` datatype.

In addition, constructors for sets, pairs and triples are defined (respectively) as `S(e1,...)`, `P(e1,e2)` and `T(e1,e2,e3)`, where each `ei` is an element of datatype `T`. Finally, operator `p # c` is associated to the `Pattern` datatype, and is used to directly access the `Channel c` from `Pattern p`.

Example 9.4 Figure 9.6 presents the ReCooPLa implementation of the OverlapP reconfiguration.

```
1  reconfiguration OverlapP(Pattern p; Set<Pair<Node>> X) {
2      @ par(p);
3      forall(Pair<Node> n : X) {
4          Node n1, n2;
5          n1 = n.fst;
6          n2 = n.snd;
7          Set<Node> E = S(n1, n2);
8          @ join(E);
9      }
10 }
```

Figure 9.6: ReCooPLa implementation of the OverlapP reconfiguration.

The implementation is self-explanatory. It uses the `@` construct (line 2) to apply the `par(p)` primitive reconfiguration to the *target* coordination pattern, where `p` is passed as an argument. Then it iterates over the elements of the second argument (line 3) and stores the first and second components of each such element into `Node` variables. The set construct is used (line 7) to instantiate a set `E` with these nodes. Finally, the `join(E)` primitive is applied directly to the target coordination pattern. ✠

More examples of reconfiguration patterns are provided in Appendix C. In fact, it lists the ReCooPLa implementation of all reconfiguration patterns introduced in Section 6.2.3.

9.2.2 Application of reconfigurations

In order to specify the actual application of reconfigurations onto coordination patterns, ReCooPLa introduces the special construct `main`.

The `main` can be executed in three different ways. One is by importing stochastic instances of patterns specified in CooPLa files. The other is by freshly instantiating them as argument of the `main`, the third is by creating them in the body of the `main`. The concrete program goes inside the body of the `main` construct. It is, as usually, a list of instructions, which are however, limited to declarations with initialisation (of new instances via application of a reconfiguration to another instance) and *isolated*

application of reconfigurations (which only change the target coordination pattern instance).

Example 9.5 Assume that ReCooPLa code in Figure 9.6 is stored in a file named `overlap.rcpla` and the CooPLa code in Figures 9.3 and 9.5 exist in a file named `seq.cpla` and is, actually, deployed. Consider now the existence of another file named `ext.cpla` containing a CooPLa pattern as in Figure 9.7 (left).

In order to extend the Sequencer coordination pattern to cope with three services in sequence (rather than two), a reconfiguration to its structure has to be performed. Figure 9.7 (right) presents the complete ReCooPLa program that is able to enact such a change.

```
1  pattern Extension (a:b)
2  {
3    use :
4      (E)fifo~1(i:o) as f2;
5      sync(i:o) as s2;
6    in :
7      a = f2.i;
8      b = s2.o;
9      join [f2.o, s2.i] as ij;
10 }
```

```
1  import seq.cpla;
2  import ext.cpla;
3  import overlap.rcpla;
4  main[Extension sext] {
5    Sequencer3 seq3 =
6      sseq @ overlapP(
7               sext,
8               S(P(sseq#f1.out(0),sext.in(0)))
9           );
10 }
```

Figure 9.7: Reconfiguration script (left) for updating the Sequencer pattern with an extension (left).

Lines 1 to 3 import the necessary files. In line 4 the main structure is declared with a fresh instance `sext` of type Extension. Line 5 declares a new coordination pattern that will store the result of the sequent lines. In line 6, the `sseq` instance is applied the `overlapP` reconfiguration. Line 7 refers to the stochastic instance `sext` that is going to extend `sseq`. Finally, line 8 defines a set of pairs of nodes using the relevant operations to obtain the actual nodes that are to be glued together. ✠

9.3 The CooPLa Editor

The CooPLa Editor is a plugin for Eclipse available from `coopla.di.uminho.pt`. It supports prototyping and analyses of software coordination code and associated reconfigurations. It integrates environments for edition of CooPLa and ReCooPLa specifications. Additional tools (added as plugins to the editor) transform sources of these languages into assets accepted by model checkers and other tools used for qualitative and quantitative/probabilistic analysis.

The CooPLa Editor may be regarded as an interface for a toolchain for the analysis

of coordination code and their reconfigurations. Figure 9.8 shows how it integrates with external tools for that matter.

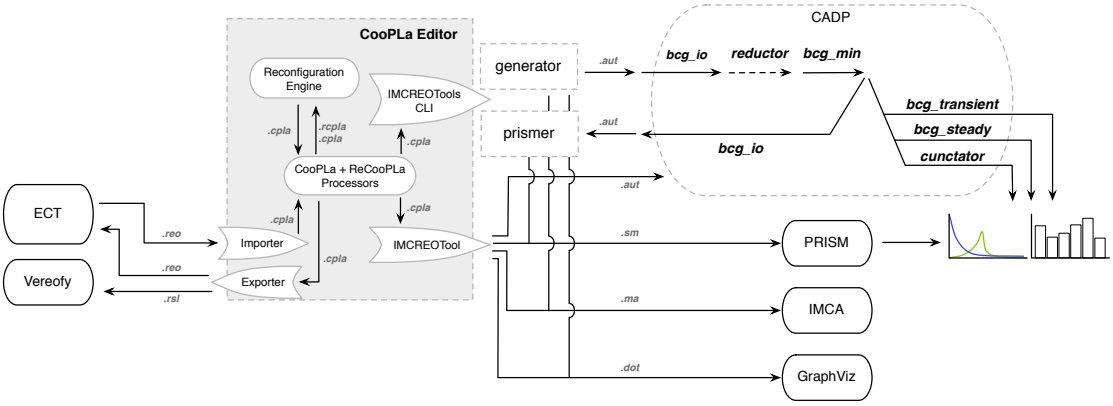


Figure 9.8: A toolchain for the CooPLa Editor.

The CooPLa Editor is presented in a shadowed-box. Each element inside it represents one of its internal plugins. Some of these communicate with the exterior and are responsible for the integration with the third-party tools. Each arrow in the chain identifies the type of the files passing from one tool to another. For instance, interface with the CADP tool is made via `.aut` files, while with *extensible coordination tools (ECT)* it is made via `.reo` *extensible markup language (XML)* files.

In the remainder of this section, the CooPLa Editor, its basic features and the internal plugins are discussed.

9.3.1 Editor Overview

As an Eclipse plugin, the CooPLa Editor presents a well known *graphical user interface (GUI)*. The elements of the interface are organised according to a specific *perspective*. Figure 9.9 identifies the five main parts.

The zone identified with (1) corresponds to the package explorer where the CooPLa projects (its folders and files) are presented in a usual tree-like structure; part (2) is the specialised editor for CooPLa and ReCooPLa languages; part (3) presents a visualisation of the coordination pattern being edited in part (2); part (4) displays the errors occurring in both CooPLa and ReCooPLa sentences edited in part (2); finally, part (5) displays an outline of the CooPLa patterns and channels, identifying their internals.

A number of specific features are associated to this.

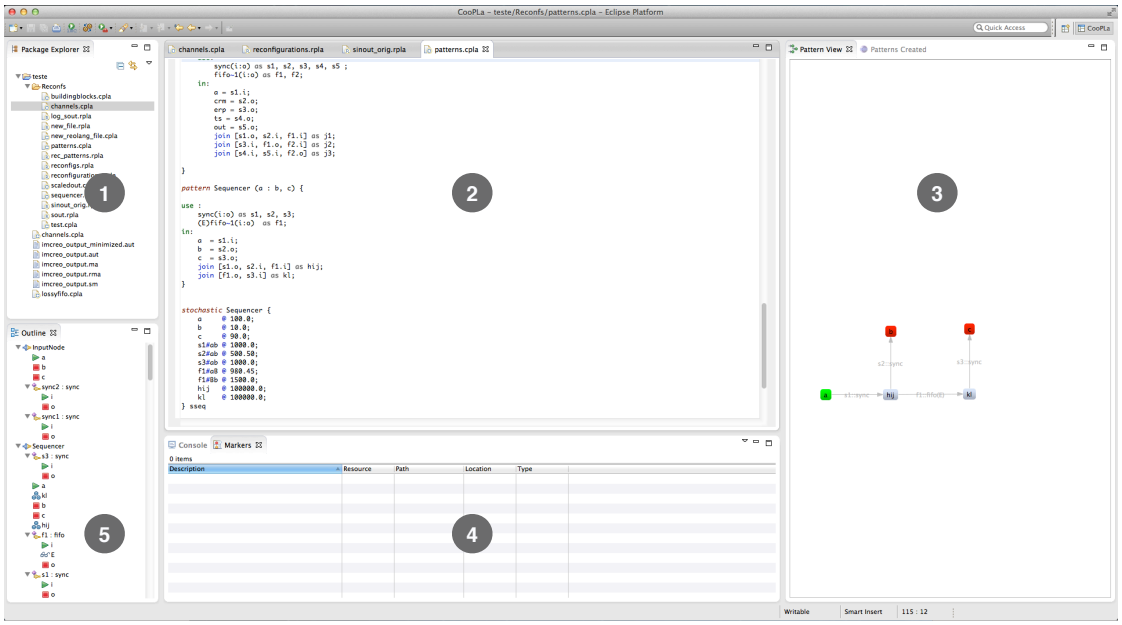


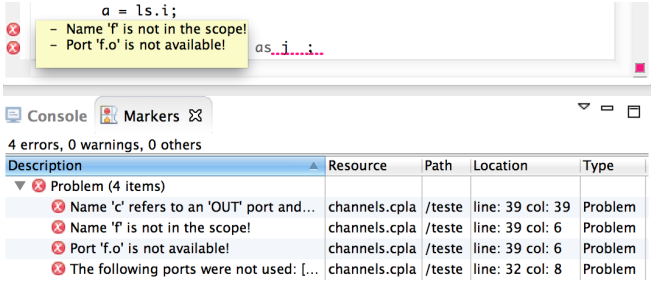
Figure 9.9: The CooPLa Editor interface

Syntax highlighting

```
pattern LossyFifo(a:c){
    use:
        (E)fifo~1(i:o)      as fe ;
        lossy(i:o)          as ls ;
    in:
        a = ls.i;
        c = fe.o;
        join [fe.i, ls.o] as j  ;
}
```

Different colours and text styles highlight specific lexical elements of both CooPLa and ReCooPLa languages. As usual in other code editors, the syntax colouring is automatically made, taking advantage of lexical analysers associated to the language. This is a feature associated to the specialised editors of both languages.

Error marking



Syntactic and semantic errors are clearly pointed out in the editor with annotations and markers. The errors are checked and identified while editing the code. This happens every time the user stops writing for some milliseconds. Robust and fast AnTLR-based parsing and customized semantic analysis underpin this

feature. It is a feature associated to the specialised editors of both CooPLa and ReCooPLa languages.

Intellisense



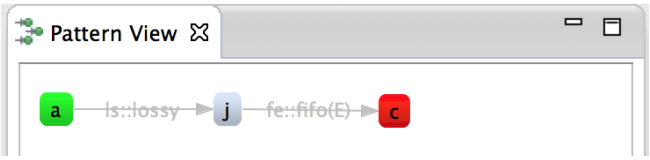
This features delivers syntactic and semantic suggestions for the code being written. Contextual analysis is performed so that suggestions are correct and code completion is enabled for a fast writing of coordination patterns. Such feature is triggered either whenever the user types a '.' or whenever a combination of keys (control + space) is pushed. It is present only in the specialised editor for CooPLa.

Outline



This feature shows an abstract structural representation of channels and coordination patterns written in CooPLa. It also offers a direct access to each syntactic element of these two entities. In particular, to channels, patterns, input and output ports, channel observers or nodes. This feature is brought by a specific CooPLa fuzzy parser that ensures the (possibly partial) creation of the outline model even though the specification is neither complete nor correct.

Visualisation



This feature delivers a graph-based visualisation of coordination patterns. The graph is built during code edition, in a fast and non-intrusive way. Like in the outline feature, a fuzzy parser is employed here for the construction of the (possibly partial) model behind the graph. The default representation is tree-like, but nodes may be moved around to obtain senseless representations. The

visualisation highlights input and output nodes in green and red colours, respectively.

9.3.2 IMCREOtool

IMCREOTool is an internal plugin of the CooPLa editor. It allows for the transformation of stochastic instances of CooPLa coordination patterns into a variety of output formats. These formats are mostly related to quantitative analysis, but other formats, *e.g.,* for visualisation, may be obtained. Figure 9.8 shows some of these possible formats.

Tool workflow

This tool works upon an implementation of the $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ model discussed in Chapter 4. This model is, of course, obtained from a stochastic instance of some coordination pattern specified in CooPLa. Algorithms in Appendix A are applied to realise the model transformation. Figure 9.10 magnifies the button in the CooPLa Editor toolbar that triggers this process.

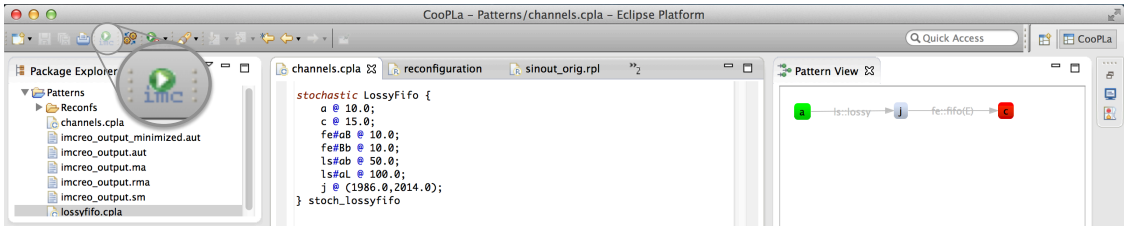


Figure 9.10: IMCREOTool triggering button.

By pushing that button a three-page wizard pops up. Figure 9.11 presents these three pages (in order from left to right).

In the first page, the user is asked to select which stochastic instance (if any exists in the current active CooPLa file) is to be one to converted into a $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ model. The second page presents the stochastic information associated to nodes of the chosen instance and its deployment environment. At this moment, the values may be changed/reviewed. The node information is labelled with `RD` (*i.e.,* reading or enqueuing to the node) and `WR` (*i.e.,* writing to the channel or dequeueing from the node). By setting the node values to zero, makes the tool ignore the node stochastic information. Moreover, the user may chose whether or not the coordination pattern is deployed.

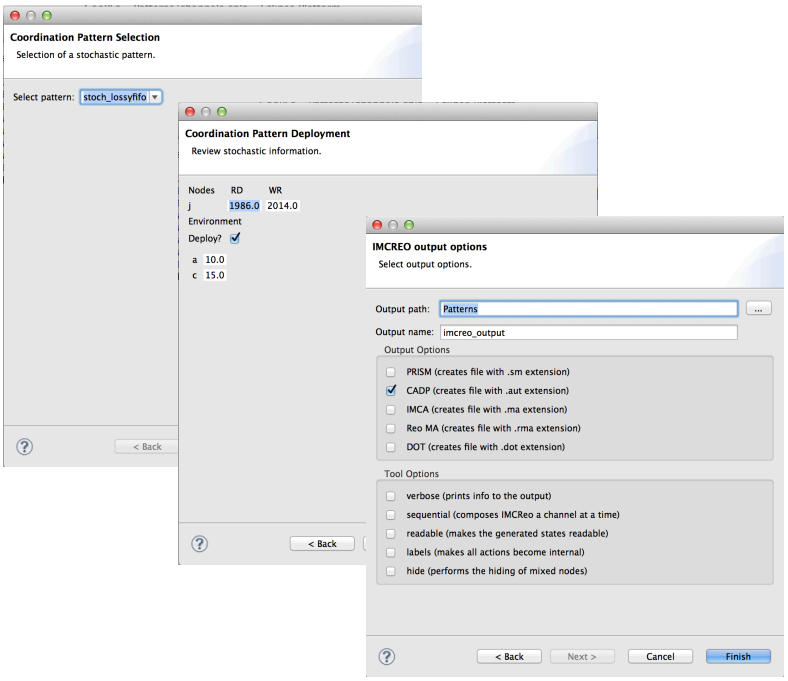


Figure 9.11: IMCREOTool wizard pages.

Finally, in the third page, a wisp of output and tool options is presented. Therein, the user is asked to select an output file name and location, as well as the output format(s) to generate. If the PRISM option is selected, the user must make sure that CADP is installed and that paths to CADP directory and binaries are set in the Eclipse preferences for the CooPLa Editor.

The latter is a must because the PRISM model checker does not accept *interactive Markov chains (IMCs)*. Therefore, it is necessary to convert the $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ model into *continuous-time Markov chains (CTMCs)*. Such a transformation takes a $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ model where all interactive transitions were made internal. This model is then passed to the *reductor* CADP tool for both minimisation via a *weak trace* equivalence relation and *determinisation*. In the end, the resultant CTMC model is converted, within IMCREOTool, into a PRISM file.

All this work is, as expected, hidden from the user. However, a *command line interface (CLI)* version of the IMCREOTool exists[1] that delegates all this chaining to the user.

[1] `http://reo.project.cwi.nl/reo/wiki/ImcReo`

Output working labels

In order to ease the use of the output formats of IMCREOTool within other tools, meaningful labels are added to the transitions of the $\mathcal{D}$IMC$_{\text{Reo}}$ (whenever the format allows). This is the case of output formats for CADP or PRISM, for instance. The use of these labels is due to the transformation of the $\mathcal{D}$IMC$_{\text{Reo}}$ states into abstract numbered states of the output formats. Without these labels, the users would not be able to understand the generated models, since a direct mapping rarely exists between $\mathcal{D}$IMC$_{\text{Reo}}$ and such models. Actually, the labels go further and create a mapping that shorten the gap between the CooPLa design and the final output format.

All the labels agree to the following syntax: `<ENT>_<NAME>_<ACT>_<PORTS>`. Where `<ENT>` is the entity (environment, channel or node) active in the transition with that label. It may expand to one of the fixed names shows in Table 9.2. `<NAME>`

Table 9.2: Working label values and meaning.

	Value	Meaning
ENT	ENV	Environment
	MREP	Merger-replicator node
	XOR	Exclusive router node
	SYNC	sync channel
	LOSSY	lossy channel
	DRAIN	drain channel
	FIFO1e	fifo$_e$ channel
	...	...

	Value	Meaning
ACT	TRs	Transmission success
	TRl	Transmission lost
	ARR	Request arrival/produced
	RD	Node read/enqueue data
	WR	Node write/dequeue data

is the name of the entity. If it is a channel, then it corresponds to the name given to that channel instance in the CooPLa design. If it is a node or environment, then this name corresponds to the concatenation of all channel ends co-located in the node or environment. `<ACT>` is the action performed by the entity. For instance, it may be a data transmission, a data arrival, a data enqueueing or dequeueing among others. Table 9.2 specify the associated names to each considered action. Finally, `<PORTS>` correspond to the set of ports associated to the entity and involved in the action. Depending on the output format, they may appear either as the concatenation of the co-located channel ends, or as a list of such ends.

Furthermore, these labels may be added of the corresponding rate, which is associated to the transition in the $\mathcal{D}$IMC$_{\text{Reo}}$ model.

Figure 9.12 shows two small excerpts of how these labels appear in the context of the `.aut` and `.sm` formats for CADP and PRISM tools, respectively.

```
( 0, "ENV_ls_i_ARR_[ls_i]; rate 1.0", 20)
( 8, "LOSSY_ls_TRI_[ls_i]; rate 1.0", 1)
( 4, "MREP_ls_ofe_i_RD_[ls_o]; rate 1.0", 5)
...
```

```
[ENV_ls_i_ARR__ls_i_] s=0 -> 1.0 :(s'=11);
[LOSSY_ls_TRI__ls_i_] s=2 -> 1.0 :(s'=5);
[MREP_ls_ofe_i_RD__ls_o_] s=4 -> 1.0 :(s'=2);
...
```

Figure 9.12: Working labels for CADP (left) and PRISM (right).

9.3.3 Reconfiguration engine

The reconfiguration engine enables the application of reconfigurations written in ReCooPLa upon CooPLa coordination patterns. The reconfiguration engine was discussed in detail as a M.Sc. project [227]. As such, only an overview of it is provided in the following paragraphs.

In the context of the CooPLa Editor, the reconfiguration engine is activated by clicking the button magnified in Figure 9.13. Its activation only occurs if the content of the active specialised editor consists of a ReCooPLa sentence with a *main* reconfiguration. As discussed before, it is this reconfiguration that will lead the whole process.

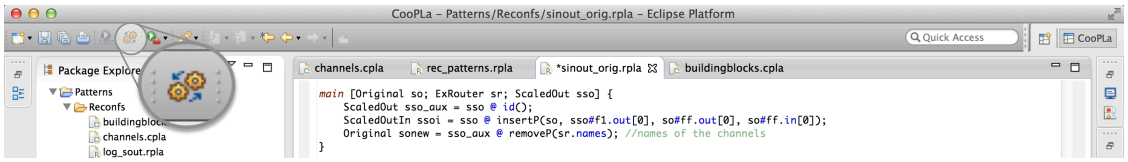


Figure 9.13: Reconfiguration engine triggering button.

The process follows, then, in a sequence of steps. First, the CooPLa processor transforms the imported coordination patterns into an internal model that realises the coordination pattern model introduced in Chapter 6. Afterwards, the the ReCooPLa processor translates each reconfiguration referred in the *main* into a Java source file. The *main* reconfiguration is translated into a main Java program. These translations are supported by a formal translation schema [228, 227]. The generated sources are compiled in runtime and the output class files are added to the running *Java virtual machine (JVM)* via reflection mechanisms. After these steps, it is obtained a Java program, which is ready to be executed. The reconfiguration engine is responsible for calling the *main* method of that program (again via reflection) in order to apply the changes onto the coordination pattern models. In the end, the obtained coordination patterns are converted back into CooPLa specifications that can be saved for later usage.

Figure 9.14 shows the moment right after the application of a reconfiguration along with its result. The generated patterns are shown in a view similar to that of

the visualisation of coordination patterns during their edition in CooPLa. At this moment the user is asked whether the generated patterns shall be saved into CooPLa files.

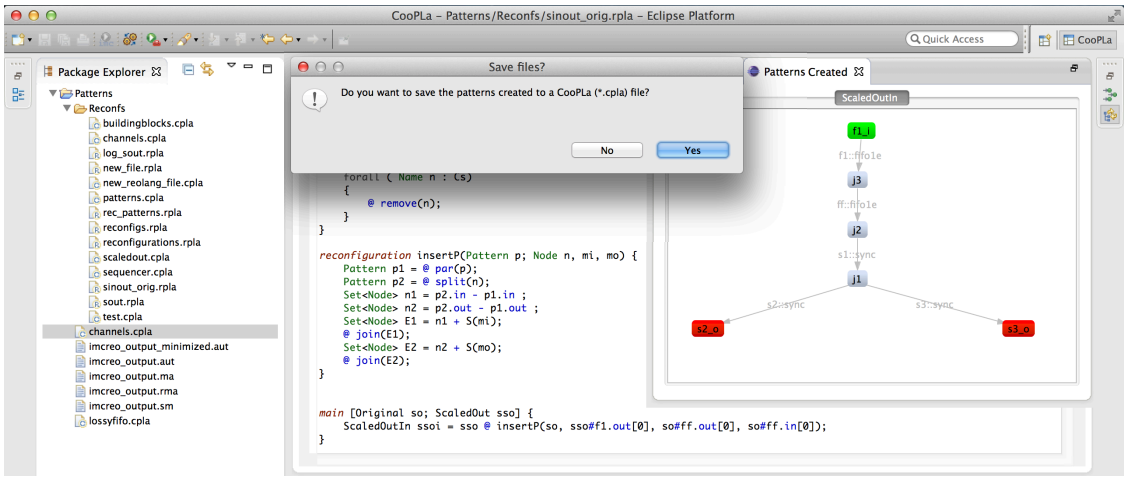


Figure 9.14: The application of a reconfiguration.

The saved coordination patterns may be the subject of yet another reconfiguration, creating a cyclic application of the engine. Also, these patterns can be exported into external tools for further analyses or other applications.

9.3.4 Importer and exporter

The importer and exporter plugins of the CooPLa Editor are limited to importing Reo XML files and exporting both ECT and Vereofy compatible files. More formats can be added in the future. Figure 9.15 shows where these two features are placed in the context of the editor.

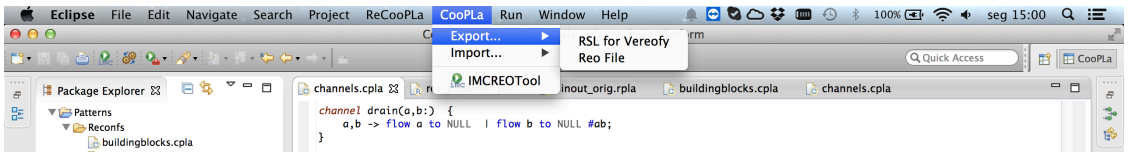


Figure 9.15: The importing and exporting features.

In the following, the process of importing and exporting these formats is briefly presented.

Vereofy RSL

Exporting Vereofy (*i.e.*, RSL) files is made via source-to-source transformation. The input is a CooPLa file. Only the coordination patterns therein are translated. Channels are assumed built-in in the Vereofy processor implementation. Consequently, they are imported to the generated file. Figure 9.16 shows the CooPLa coordination pattern (left) and the generated RSL circuit (right).

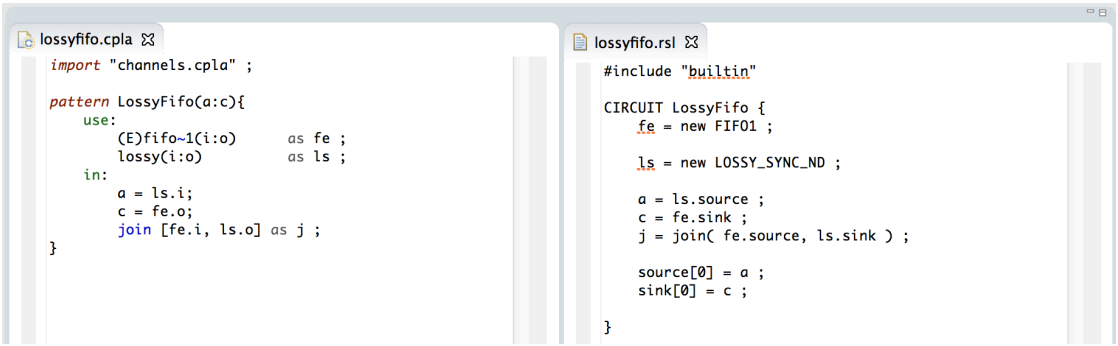


Figure 9.16: Exported coordination pattern to Vereofy notation.

Reo XML

For (importing/exporting) Reo XML files the process is more involving. It is not based on source-to-source transformation. Instead, it has the intermediate step of converting the contents of the files (whether they are Reo XML to import or CooPLa to export) into the concrete model of coordination pattern (as introduced in Chapter 6), which allows for more straightforward conversion algorithms.

Although limited to a specific set of basic Reo channels, this feature enhances the productivity with both ECT and CooPLa Editor. In particular, they complement each other. For instance, ECT is adequate for an easier design of coordination patterns enabling animation for behaviour verification; the CooPLa editor improves the application of reconfiguration and delivers a robust basis for quantitative analysis.

Figure 9.17 shows a chain of steps on how these tools can work together. In this particular case, the CooPLa editor is being used as an external tool for applying a reconfiguration to a Reo circuit.

In the ECT, a Sequencer coordination pattern is modelled. Then, it is imported to the CooPLa Editor. Another coordination pattern, named synchroniser, is designed and imported to the reconfiguration written in ReCooPLa. The applied reconfiguration changes the original Sequencer coordination pattern into what was named ParallelSynch, which activates services in parallel and synchronises their responses.

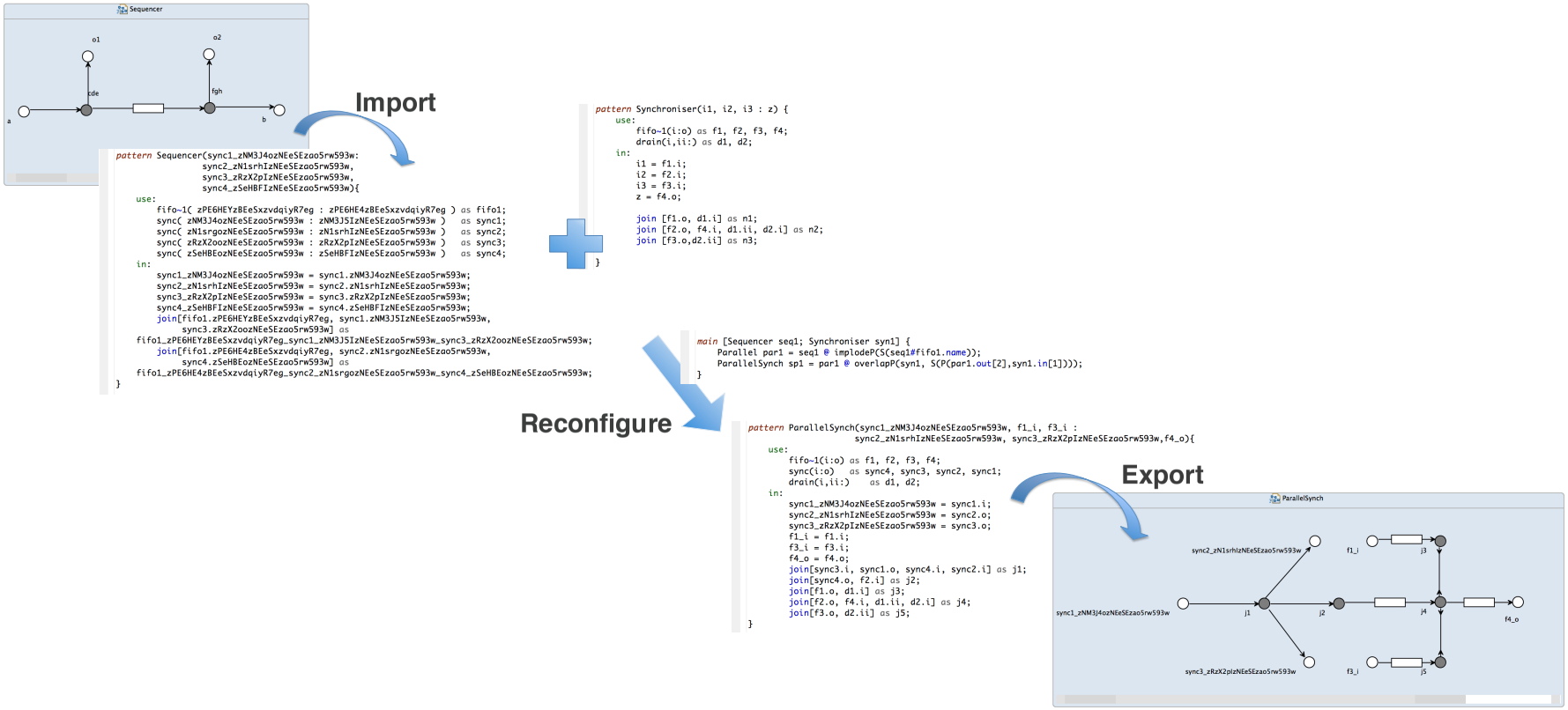


Figure 9.17: Using the ECT and the CooPLa Editor together.

Finally, the obtained pattern is exported into Reo XML, resulting in a reusable Reo circuit in the context of ECT.

9.4 Summary

This chapter introduced tool support for ARIS, based on an Eclipse plugin referred to as the CooPLa Editor.

The CooPLa Editor provides features for (*i*) the design of both coordination patterns and reconfigurations; (*ii*) the application of these reconfigurations upon the coordination patterns; and (*iii*) transformation of these elements into formats acceptable by multi-purpose external tools.

For item (*i*), two DSLs were presented: CooPLa and ReCooPLa. The former enables the design of the coordination layer of a system by coordination patterns. It allows for the definition of channels, patterns, and their stochastic instances. The latter is tailored to the design of reconfigurations of coordination patterns.

For (*ii*), a reconfiguration engine was devised. It converts ReCooPLa reconfigurations into Java files, compiles them and, via reflection, imports the generated class files into the JVM running the CooPLa Editor. Again by reflection, reconfigurations are applied upon coordination patterns that were imported into the original ReCooPLa script. The obtained coordination patterns may be saved for later use within the CooPLa Editor.

Finally, for (*iii*), a set of transformation mechanisms was developed. The IMCREOTool is an internal plugin of the CooPLa Editor that implements the $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ model (*c.f.,* Chapter 4). It takes CooPLa specifications, converts them into the formal model of coordination patterns (*c.f.,* Chapter 6) and, upon such a model, generates a $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ model. From this model, several output formats are possible. This enables the interface of the CooPLa Editor with other tools. In particular with tools for quantitative analysis like PRISM or CADP.

Chapter 10

Case Study: towards an adaptable system

Nothing endures but change.

– *Heraclitus*

In this chapter. A case study is presented as a proof-of-concept of the adaptation approach introduced in Chapter 8. In particular, it deals with the design of an adaptable version of a static system. This entails the application of ARIS, as well as the use of its supporting tools. The runtime behaviour of the system is conducted by a simulator. PRISM is in charge of analysing the performance of the the system possible configurations, modelled in $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$.

This is an hypothetic situation in the context of the real-world *Access Society's Knowledge (ASK)* system. Although hypothetic, the study is made based on real data obtained from the company's logs.

Parts of this chapter's content was previously published, by the author, in:

- Nuno Oliveira and Luís S. Barbosa. "A self-adaptation strategy for service-based architectures". In: *Proceedings of the 8th Brazilian Symposium on Software Components, Architectures and Reuse*. SBCARS'2014. Distinguished with Best Paper Award. Maceió, Alagoas, Brasil: IEEE, Sept. 2014, pp. 1–10.

10.1 The ASK system

The ASK system is an industrial solution from the Dutch company *Almende*. It is a communication software that facilitates mediation between service consumers their providers. For instance, it may mediate a company in need for a temporary

worker and an available person whose profile matches such request. Matching mechanisms are used to mediate the interveners, which take into account the needs of the consumers and the profiles of the providers.

The business goals of the ASK system are to deliver the best consumer-provider match within the lower time possible. On top of this, *Almende* desires to achieve such goals while keeping the entailing costs low.

10.1.1 The architecture

The architecture of the ASK system is modular (*c.f.,* Figure 10.1). It is based on three high-level components: a *web front-end*, a *database*, and a *contact engine*. The *web front-end* serves as the interface between the consumers/providers and the system. The *database* stores typical system data, along with specific data for the improvement of the matching algorithm. The *contact engine* is where the main business is processed: matches are computed and the interveners are set in contact. For this, the *contact engine* consists of the following four components.

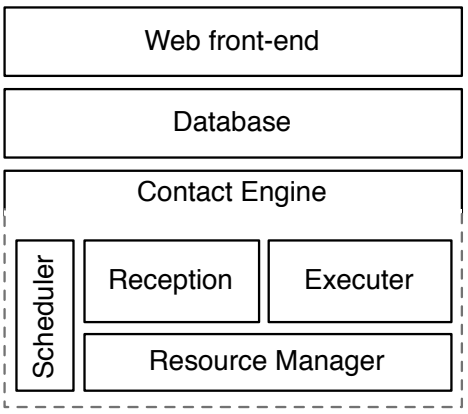


Figure 10.1: The architecture of ASK.

The *Reception* collects the users' requests and converts them into tasks. The *Executer* takes each task and determines the best providers/consumers to serve a request. It is also responsible for defining how the matched entities should communicate. The *Resource Manager* establishes the connection between the matched entities. The *Scheduler* schedules requests stored in the database, taking into account time constraints associated to them.

Both *Reception* and *Executer* are further specialised with components, services and queueing mechanisms. The most typical workflow within ASK reports their interaction. It starts with either the user issuing requests or the triggering of request by the scheduler. These requests are sent to the *Reception* where a *HostessTask* (HT) subcomponent converts them into tasks. Such tasks are enqueued into a *Task-Queue* (TQ). Whenever possible, a *HandleRequestTask* (HRT) subcomponent dequeues these tasks, generating new requests to the *Executer* component. Within the *Executer*, requests are also enqueued into an *Execution-Queue* (EQ). A *HandleRequestExecution* (HRE) service performs the matching between consumer and provider for each request in the EQ. The result (a connection between consumer and provider) is sent to the *Resource Manager*. Fi-

nally, the *Resource Manager* will deal with the physical contact between the matched entities.

10.1.2 The static performance on a dynamic environment

From the logs and previous analyses [198], the *Executer* component was identified as the responsible for undesired bottlenecks in the system performance.

In the current ASK configuration, the server running the HRE service is not dedicated to it, dealing with several other different processes. Its task of finding and establishing the best consumer-provider connection is time and resource (mainly memory) consuming. For this reason, there is a top limit of 20 HRE service instances able to run concurrently in that server. In average, each instance of the HRE service takes $0.703s$ to produce an output (*i.e.,* accepts aprox. 1.422 requests per second); meaning that the server is potentially able to deal with roughly 28.440 requests per second. The EQ queue runs on a different server and is able to enqueue and dequeue at a rate of 10000 jobs per second.

Figure 10.2 shows the simple model of the *Executer* component with associated stochastic information. In this figure it was assumed a mean time interval between requests arriving to this component of 0.063s. This is, of course, just illustrative. The real value will vary since the number of requests also vary with time.

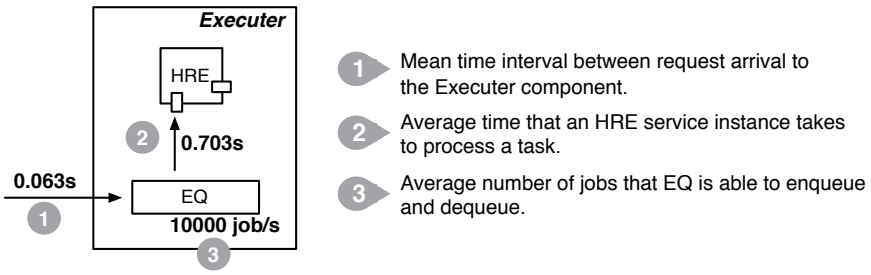


Figure 10.2: The *Executer* component model with stochastic information.

In concrete, from years of experience, logs and monitored data, the ASK team has learnt that during the night period there is, usually, a drop of user requests, and that after lunch time until mid-afternoon, such demand reaches a peak. Table 10.1 depicts that variation for non-uniform time intervals of a day. The request per second in each cell are an average.

Table 10.1: Requests to the ASK system during a day

hours (interval)	0-8	8-12	12-14	14-17	17-24
requests (per second)	0.125	12.420	8.321	30.460	12.260

Moreover, it was found that roughly every six months there is a slight down time on the server where the HRE web-service is hosted. Therefore, the rate of failure is about 6.43×10^{-8} per second[1]. Another important observation was that the mean time to recover from a failure was of about 10s.

In these situations, a fixed architecture and a fixed number of resources may contribute to undesired financial losses for the company. In this sense, the case study here reported centres attentions on the *Executer* component. Dynamic adaptation for this component will be planned and analysed. It is, in particular, an attempt to dynamically maximise the company's objectives. This is achieved by both contracting the right amount of system resources and defining the most appropriate behaviour for the right contextual settings.

Henceforth, Adaptable-ASK will be used to refer to the adaptable version of the ASK system.

10.2 Planning adaptations

In this case study, the approach for architectural self-adaptation introduced in Chapter 8 is followed. In particular, recall what is involved in the offline phase: the system is designed, possible reconfigurations are organised into a *reconfiguration transition system (RTS)* and the main adaptation logic is settled by the definition of constraints and filters. This section reports precisely on the construction of these assets in the context of the Adaptable-ASK.

10.2.1 Adaptable-ASK design

The design of the Adaptable-ASK (which is reduced to the design of the *Executer* component) is reported here in two parts. The first is concerned with requirements modelling. The second dives into the construction of both a model of the system's coordination and a RTS.

Requirements and properties

The properties and requirements for the system have to be set clearly upfront before its design. The ASK team defends that two requirements must remain invariant during all adaptation process. One is that "*there must always be a queueing system in between the entry port of the component and the HRE service*". The other is

[1]The *mean time between failure quality of service (QoS)* attribute of the server is consequently set to $15552000s = (360 * 24 * 60 * 60)/2$.

that "*requests must not be lost during interaction*". These two requirements can be translated into properties of the $Hp\mathcal{E}$ logic. Let Nom_i and Nom_o be sets of nominals referring to input and output ports in the coordination patterns. Then, for each $in \in \mathsf{Nom}_i$ and $out \in \mathsf{Nom}_o$,

$\phi_1 \ : \ @_{in}[-^*.\mathsf{fifo}.-^*]out$

$\phi_2 \ : \ @_{in}(\langle -* \rangle \mathsf{true} \wedge [-^*.\mathsf{lossy}-^*]\mathsf{false})$

are formula schemes that represent the two requirements, respectively. Note that the first conjunct in ϕ_2 captures the fact that the expected interaction involves at least one channel.

As expected, these properties must also remain valid for all the envisaged configurations that the system may present under adaptation. Along with this, the company requires that the main behaviour of each system's configuration is also preserved. This brings to discussion properties of the reconfigurations that will be part of the RTS. In particular, this enforces these reconfigurations to be either *beh-unobtrusive* or *beh-expansive* (with regard to behaviour) and either *str-compatible* or *str-conservative* (concerning the structure).

Besides behavioural and structural properties, it is mandatory the identification of QoS dimensions of interest for the system. In this case study, only the dimension $\langle [0,1], \leq \rangle$, referring to *throughput ratio* (TR), is considered. The throughput ratio measures the ratio between the effective throughput and the maximum throughput possible. From its formal definition it should be understood that a higher ratio is desired for the system's performance. A desired value for this QoS dimension is analysed in the following sections.

The *cost* of the system's configurations is also an important indicator in what concerns adaptation. As such, this is also considered as a global system property. Moreover, a constraint associated to this indicator is that its pecuniary value must always be the minimum possible, formally (SYS.*cost*,min,true).

Table 10.2 summarises the properties of the Adaptable-ASK (and envisaged reconfigurations). Some of them must remain unchanged during the system life-time (unless their stakeholders decide otherwise); others must conform to system's constraints (which are addressed later in this section).

Table 10.2: Adaptable-ASK properties.

FUN	QoS	SYS	ONT	ENV
• ϕ_1 • ϕ_2	TR	*cost*	• *beh-unobtrusive* $\vee$ *beh-expansive* • *str-compatible* $\vee$ *str-conservative*	–

Stochastic model and reconfigurations

The original model for coordination of the components within the *Executer* component is easily extracted from its current implementation (*c.f.,* Figure 10.2). In particular, the EQ may be represented as a fifo channel. The coordination pattern composed of one such channel is enough to represent the whole component. The HRE service will be assumed connected to the output port of that coordination pattern. This is to cope with the vision of exogenous coordination. It will solely make part of the environment; as such, only its processing time is of interest for stochastic modelling purposes.

Considering this original coordination pattern, the properties above and the average fluctuation of user requests in Table 10.1, it was possible to define configurations that would, most likely, overcome the reported changes on the environment. Figure 10.3 shows the configurations obtained (in the context of the CooPLa Editor) for the adaptation strategy of the Adaptable-ASK.

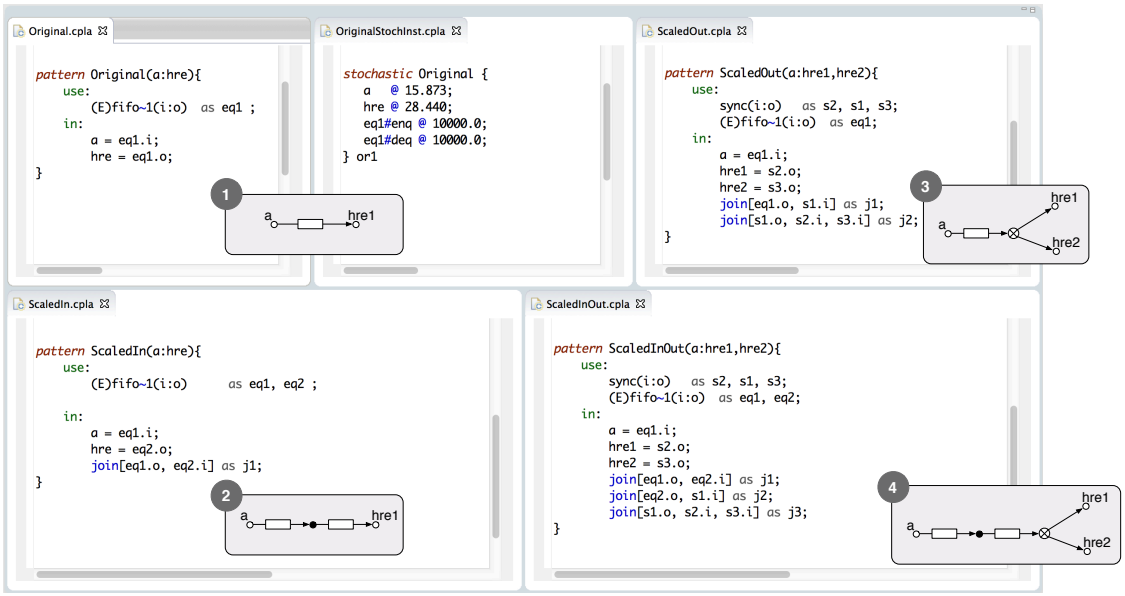


Figure 10.3: Configurations for the Adaptable-ASK system.

Configuration ❶ is the original coordination pattern resorting to one queue; it has a cost per hour of €0.47. Configuration ❷ is a *scaled in* version of ❶, where more memory was added to the original queue; it has a cost per hour of €0.54. Configuration ❸ is a *scaled out* version of ❶, where a second HRE server (with same performance) is added in such a way that both servers, connected to hre_1 and hre_2, process different tasks in parallel; this configuration has a cost per hour of €0.67. Finally, configuration ❹ is a *scaled in and out* version of ❶, where more memory and a second server are added in such a way that both servers, connected to hre_1

and hre_2, execute in parallel; it has a cost per hour of €0.74.

Figure 10.4 complements these configurations with the respective reconfigurations. Altogether, they compose the RTS for Adaptable-ASK. To enhance readability, the backward reconfigurations were omitted.

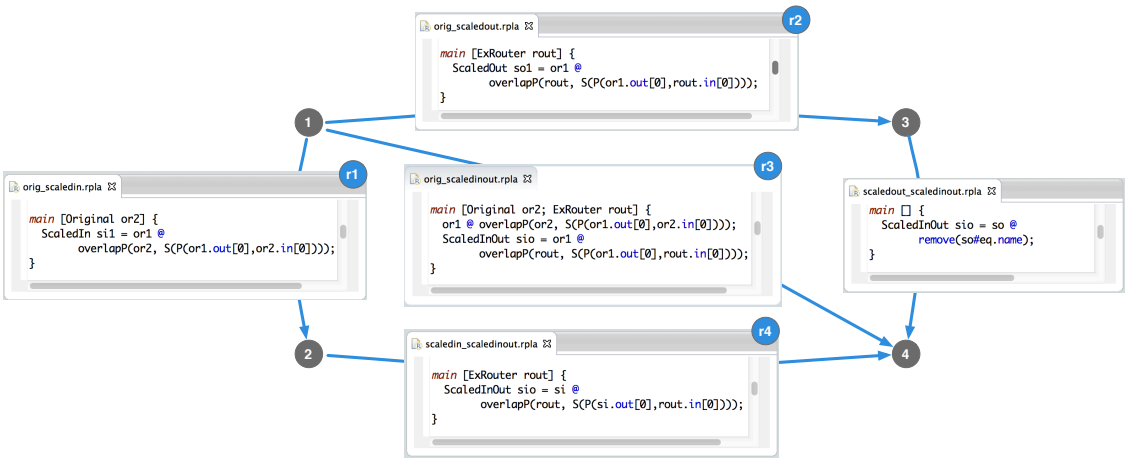


Figure 10.4: RTS for the Adaptable-ASK system.

Note that the four configurations preserve the functional properties when considering $\mathsf{Nom}_i = \{a\}$ and $\mathsf{Nom}_o = \{hre1\}$ (for configurations ❶ and ❷) or $\mathsf{Nom}_o = \{hre1, hre2\}$ (otherwise). The same happens for the reconfigurations. All of them meet the ONT properties. The backward reconfigurations are mandatory and assumed by default; there is, therefore, no need for them cope with this family of properties.

10.2.2 Analysis of RTS configurations

In a simple analysis, it is possible to see how each configuration performs against the variability of the environment. In order to achieve this, the IMCREOTool plugin is used for generating suitable analysable assets from the CooPLa specifications that compose the RTS. In this case study, PRISM is the target quantitative model checker for that analysis. Figure 10.5 shows a fragment of the PRISM module generated for the *scaled out* CooPLa coordination pattern.

The QoS dimension of interest for Adaptable-ASK, the *throughput ratio*, can be expressed in PRISM using the notion of *rewards* as follows: `R{"runs"}=? [S] / T`. Here, `runs` is a reward structure that assigns the value 1 to each transition that transmits data to hre_1 (and hre_2); and `T` is a variable representing the user requests. Table 10.3 summarises the obtained steady-state values for this property at the precise average rate of user requests expected at each hour interval.

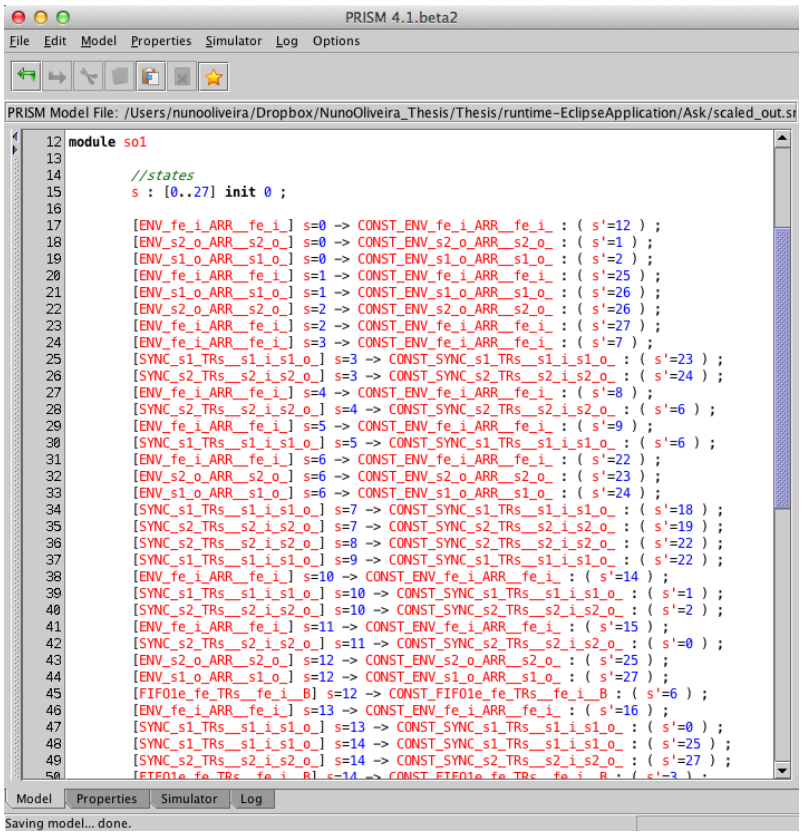


Figure 10.5: The PRISM model for the *scaled out* coordination pattern.

Table 10.3: Steady-state throughput ratio analysis for RTS configurations.

hours (interval)		0-8	8-12	12-14	14-17	17-24
requests (per second)		0.125	12.420	8.321	30.460	12.260
❶ Original	NFS	0.999	0.950	0.981	0.721	0.951
	FS	0.661	0.008	0.012	0.003	0.008
❷ Scaled In	NFS	0.999	0.978	0.994	0.769	0.979
	FS	0.702	0.008	0.012	0.003	0.008
❸ Scaled Out	NFS	0.999	0.996	0.998	0.944	0.996
	FS	0.999	0.951	0.982	0.722	0.952
❹ Scaled In and Out	NFS	0.999	0.998	0.999	0.970	0.998
	FS	0.999	0.978	0.994	0.770	0.979

Non-faulty server (NFS) and faulty-server (FS) marks say respect to experiments where, first, the server connected to port hre_1 is always available and, second, it is constantly failing (accepting one request in each 10s). In both cases, the server on port hre_2 (when present) is always up. The graphs in Figure 10.6 provide a similar view, but this time it is presented an evolution of the TR property depending on

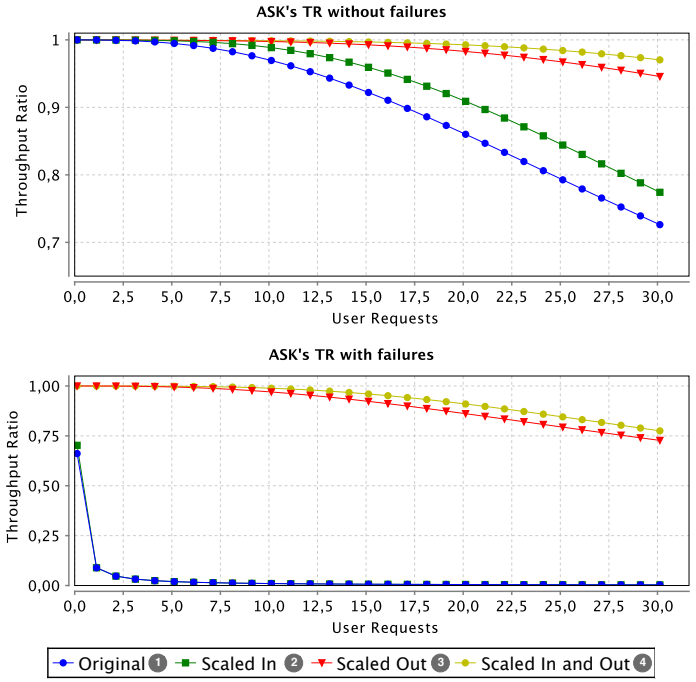


Figure 10.6: Performance analysis of the throughput ratio property for the several configurations. Without faulty server (above) with faulty server (below).

the number of user requests. The graph on the top shows the evolution of TR for the servers without failures; the one in the bottom shows the that evolution for the presence of a faulty server, in the conditions explained before.

As expected, the *scaled out* and *scaled in and out* configurations perform significantly better than the others. In the FS situation, these two perform similarly to the *original* and *scaled in* configurations when servers do not fail. Note that the vertical scale in the two graphs is different in order to visually accommodate the discrepancy of results in the FS situation.

10.2.3 Objectives, constraints and filters

Adding resources like servers and memory to the system is costly as it can be seen by the presented cost per hour of each configuration. Assuming that these resources are *paid-per-use* as in a cloud environment, it is essential to spend only the minimum required time on the proposed configurations. But delivering a service only with

minimum costs in mind is not advantageous, since the obvious slowness of the system will alienate its customers. This brings the need for defining a suitable *service level agreement (SLA)* for the Adaptable-ASK. As such, the Adaptable-ASK team defined that an optimal value for the TR QoS dimension would be somewhere above 0.970. In the remainder of this chapter, 0.970 is referred to as the TR threshold, or t for short). This being fixed, one can now define a suitable trigger constraint for the system in hand.

$$\geq_{TR}^{t} \wedge \mathsf{min}_{cost}^{P}$$

This imposes that adaptations will be raised in one of the following situations:

1. the current system configuration presents a throughput ratio value below the threshold t;
2. there is a configuration (in the pool of possible configurations, represented by P) that is cheaper;
3. both 1. and 2. occur simultaneously.

In order to finish the offline phase, it is still necessary to define filters that will lead the decision making in case of an adaptation being required. Table 10.4 associates the most suitable configuration to each hour interval, considering multiple adaptation objectives, defined by possible filters.

Table 10.4: Predicted configurations for filters.

	0-8	8-12	12-14	14-17	17-24
min_{cost}^{P}	❶	❶	❶	❶	❶
max_{TR}^{P}	❶	❹	❹	❹	❹
NFS – $\geq_{TR}^{t}$,min_{cost}^{P}	❶	❷	❶	❹	❷
FS – $\geq_{TR}^{t}$,min_{cost}^{P}	❸	❹	❸	–	❹
NFS – $\geq_{TR}^{t}$,min_{cost}^{P} | max_{TR}^{P}	❶	❷	❶	❹	❷
FS – $\geq_{TR}^{t}$,min_{cost}^{P} | max_{TR}^{P}	❸	❹	❸	❹	❹

The top two rows are concerned with the selection of candidate configurations filtering, exclusively, by minimum cost and maximum TR value, respectively. As expected, these filters define adaptation strategies that make the system practically static. The top one reduces company costs, but will as well reduce the TR values; the second row augments TR value (augmenting customer satisfaction), however, costs are higher.

The third row presents a filter that selects, in a first step, the configurations delivering a TR value above the threshold, and then selects the one with minimum cost. For the NFS situation, the selected configurations are balanced and, thus,

the adaptation is more in line with the company objectives. For the FS situation, however, there is no configuration able to deliver a TR above the desired threshold in the interval where the user requests reach its peak (*i.e.,* 14-17). In this case, the system would not reconfigure itself. If for some reason the configuration at that moment is ❶ or ❷, then the system would perform low (see Figure 10.6, bottom graph) for a while, increasing the losses for the company.

Contrariwise, the fourth row extends the previous filter by adding an optional filter that selects the configuration delivering the maximum TR value, when the first filter is not able to propose a configuration. In this way, it is now possible to have a suitable configuration for the case where servers are down or when the users demand is higher. Moreover, this filter provides a well-balanced and mainly low cost adaptation strategy. For this reason, this filter is the one that best fits and ensures the Adaptable-ASK objectives.

10.3 Runtime situation

At runtime, however, the environment changes are more unpredictable. Therefore, the previous analysis and the defined adaptation strategy are only a basis of what should be perfected at runtime. In any case, the more accurately the analysis in the offline phase is, the better the results in the online phase will be.

Since the dynamic part of the adaptation methodology proposed in Chapter 8 is not currently implemented, simulation was used instead.Concretely, the simulation consisted on one day in the system execution. It was assumed that servers do not fail in this period. The user requests for the simulation were obtained from traces of the system, such that the average in each part is the one in Table 10.1.

The results of this simulation are shown in Figure 10.7. Performance was evaluated per minute. Each evaluation considered the current request rate and the four configurations: the active one and the three candidates. The obvious exception is when the active configuration is ❷ or ❸, for which the candidates are only configurations ❶ and ❹[2]. Actually, by configuration here one means PRISM modules generated in the offline phase for the four coordination patterns that essentially define the configuration.

From the top graph in Figure 10.7, the first need for adaptation occurred at minute 480. For the first 8 hours of the day, the system has shown a good performance in configuration ❶. Then, in the first minutes of the 8^{th} hour of execution,

[2]Remember that inverse reconfigurations are omitted in the RTS of Figure 10.4 but assumed to exist.

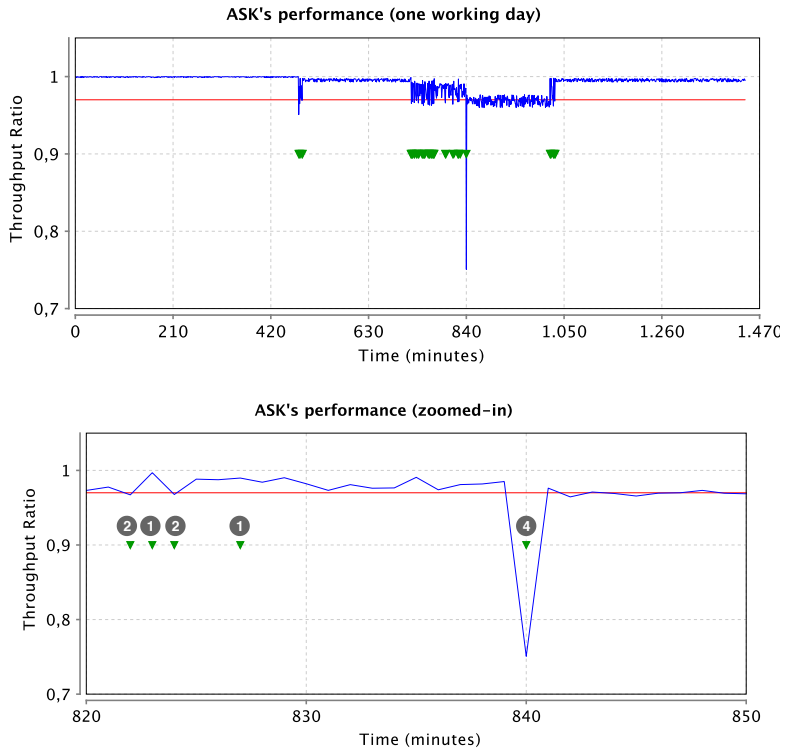


Figure 10.7: Performance of Adaptable-ASK. A twenty-four-hour span (top) and a specific half-hour span (below).

the system adapts until stabilised for the amount of requests. However, from minute 720 until minute 840 the system is constantly adapting itself. Three hours later, at minute 1020, the system adapts again for some times until stabilise for the rest of the day.

The bottom graph of Figure 10.7 was zoomed-in in a zone that spans for 20 minutes before entering the peak of requests (at minute 840) and 10 minutes during it. Before entering the peak zone, the system is able to deal with the requests in its original configuration: ❶. Note that the second adaptation to configuration ❶ is enacted not because the system is performing below the TR threshold, but because there is a cheaper configuration that delivers similar performance. This is the intended behaviour ensured by the constraint previously defined.

When the users' requests augment significantly, the system performs considerably below the TR threshold. As a consequence, the system adapts itself to configuration ❹. In the subsequent minutes there are no adaptations even though the system performs roughly below the threshold for TR. This is because (*i*) there are no selectable configurations after filtering and (*ii*), the alternative filter (TR,max,true) defined for the adaptation strategy keeps deciding (in each evaluation) that configuration ❹ is the best one for the current environment.

In this simulation, during 24 hours the system adapted 48 times, with a *mean time to adapt* of 1800s (*i.e.,* 30 minutes). This seems to be a reasonable value, however it may be misleading. In fact, note that the system only adapts itself in, roughly, three parts of the day. The most critical one being from minute 720 to minute 840, where 75% of adaptations occur. This result in a local mean time between adaptations of 200s, or roughly 3.3 minutes). This augments the time spent in reconfigurations (for simplicity assumed to be instantaneous), which consequently decrease the productivity of the system.

Such a situation can be mitigated by increasing the complexity of the adaptation algorithm. In particular, on the analysis and decision moments. For instance, instead of choosing a configuration based on its performance on the current rate of requests, one could use the history of requests (or at least the last n rates) to predict the next one, and elaborate the decision based on the system performance for such prediction. Also, one could resort to a notion of *hysteresis* to gracefully stabilise the system. For instance, one could delay the next adaptation for some time or until a cheaper configuration does not ensure a TR value above some threshold strictly greater than 0.970. The latter would improve performance and, in the long run, decrease the costs (that may be associated to reconfigurations).

10.4 Summary

In this chapter a case study was developed. It focused on the enhancement of a static software system towards a dynamic, self-adaptive, version. The whole process followed the self-adaptation strategy introduced in Chapter 8 as a concrete application of the ARIS framework.

The subject system was *Almende*'s ASK, a system for mediation facilitation, with particular focus on temporary work. Previous analysis made to this system pointed out performance bottlenecks. A dynamic version of this system would, most likely, improve such performance faults. With effect, this study showed that by making the system dynamic, able to adapt itself in response to the environment changes, it is possible to maintain the performance above a predefined level, while keeping entailed costs low.

The self-adaptive system (Adaptable-ASK) was designed by taking into account the notions of coordination patterns, and coordination-targeting reconfigurations. These notions are central to ARIS, as advocated in this thesis. Moreover, the stochastic modelling of the system by stochastic coordination patterns, led the workflow into the practical application of the $\mathsf{IMC}_{\mathsf{Reo}}$ and $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ quantitative models. In

particular, these models were translated with the IMCREOTool plugin (from the CooPLa Editor) into PRISM-compatible modules.

Consequently, in the online phase of the self-adaptation approach, PRISM was used to obtain the necessary performance analysis of the system. This was incorporated in a simulation for one day of the Adaptable-ASK system execution. Taking into account constraints and filters defined during analysis, it was possible to see that the approach is effective in adapting when it most needs.

During analysis of the adaptation results, the observations were twofold. On the one hand, the system kept its performance roughly always above an acceptable threshold. On the other hand, the system went through several reconfigurations before stabilising when abrupt changes in the environment occurred. Clearly, the algorithm used was simplistic. It used only constraints and filters to report the need for adaptations and decide which reconfigurations shall be applied.

Moreover, some simplifications concerning reconfigurations were assumed that may threaten the validity of the simulation. For instance, reconfigurations were assumed to be applied immediately after a need for adaptation is reported. In real world, one may require that reconfigurations occur only in quiescent states of the system. Also, their application was assumed instantaneous. Again, in a real-world situation, any change to a system would require time to take effect.

Chapter 11

Conclusion

He who is not contented with what he has, would not be contented with what he would like to have.

– *Socrates*

In this chapter. Conclusions about this thesis are drawn. A journey never ends without a reflection on its good and bad aspects; on the lessons learned from it; on the achievements... This is precisely what this chapter is about. Concretely, it starts with a retrospective on the main contributions of the thesis. Then, it looks back to the state of the art and discusses how the underlying research area(s) was (were) pushed forward with the work reported in this document. Finally, research challenges left open, or opened meanwhile, are discussed and presented as future work.

11.1 Retrospective

The need for highly available and dependable software has exponentially increased over the last decade. Distributed systems with flexible architectures became, for that reason, the rule underlying these systems; *service-oriented architectures (SOAs)* became the *de facto* approach for their development. The SOA architectural style promotes the construction of large-scale systems by the composition of reusable loosely-coupled computational entities named services. Such composition, realised by the coordination of services according to their public interfaces, defines, in a broad sense, the overall behaviour of SOA systems.

Deployed in environments where change is the rule rather than the exception,

these systems have to be resilient in order to maintain their behaviour and performance. This entails, of course, the ability to adapt to change. In particular, this may be achieved by reconfiguring the internal architecture; the coordination between the interacting services.

The focus of this thesis was *architectural reconfiguration of interacting services*. Special light was shed upon the modelling and analysis of reconfigurations that focus on a system's coordination layer, targeting three different concerns of systems: behaviour, structure and stochastic performance. A formal perspective was taken, pushing forward the current state of formal methods for the correct design and development of reconfigurable systems.

The quantitative analysis of software systems is a non-trivial task. Despite several stochastic formal models have been proposed for such analysis, they mostly target the modelling of a system as a whole. Few are the approaches that allow for deriving the quantitative model of a system from its compounding parts; those catering for composition usually do not cover coordination-specific features like synchronisation, mutual exclusion, non-determinism or context-dependency.

The above problem was tackled in Part I of this thesis. It was proposed a stochastic model — $\mathsf{IMC}_{\mathsf{Reo}}$— based on *interactive Markov chains (IMCs)* that captures the desired semantics of the stochastic Reo formalism. $\mathsf{IMC}_{\mathsf{Reo}}$ inherits composition from IMCs and extends them by incorporating intended properties of the stochastic Reo coordination model *e.g.,* context-dependency, identity element or atomicity. $\mathsf{IMC}_{\mathsf{Reo}}$ suffers of state-space explosion, though. This is more evident when three or more channel ends compose a node; in this case, $\mathsf{IMC}_{\mathsf{Reo}}$ refactors such a node into several others so that it always composes two ends per node. To avoid this undesired issue, $\mathsf{IMC}_{\mathsf{Reo}}$ was redesigned into a different perspective — the $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$. $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ preserves the properties of $\mathsf{IMC}_{\mathsf{Reo}}$, while regarding each stochastic Reo channel as four interacting components (environment writer, environment reader, channel and node), each one with a lower state-space model. This improves the practical use of the model: composition is achieved modularly, and only considers environment information in the last composition step (referred to as the deployment). Although the deployed model may have the same state-space explosion as in $\mathsf{IMC}_{\mathsf{Reo}}$, the model obtained upon each previous composition step is notably smaller. As a byproduct, this approach defines a more realistic model for *exogenous* stochastic coordination, which stochastic Reo is unable to offer because of the undesired coupling of the environment information.

Moreover, since IMC formalism is behind $\mathsf{IMC}_{\mathsf{Reo}}$ and $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$, typical measures obtained from IMC analysis become available for the study of coordination scenar-

ios expressed in stochastic Reo. In the same perspective, tool support for IMC (*e.g.,* CADP [125], PRISM [173], and IMCA [139]) can be made available to support analysis. With effect, as discussed in Chapter 9, the CooPLa Editor delivers the IMCREOTool plugin that enables a toolchain for qualitative and quantitative analysis of coordination models.

Part II of the thesis introduces a framework for architectural reconfiguration targeting the coordination layer of a system. It was named ARIS (expanding to the title of this thesis, for an easier reference). ARIS is twofold: on the one hand it tackles the modelling of a system by its coordination aspects; on the other hand, it allows for reasoning about reconfigurations upon the three axes: behaviour, structure and stochastic performance.

Concretely, the *modelling part* of ARIS defines notions of coordination and reconfiguration patterns. A coordination pattern was formally defined as an abstract graph of channels. Edges are typed channels, uniquely identified and with two or more ends. The type of the channel confers concrete behaviour to the coordination pattern. It must define a specific coordination policy expressed in terms of a concrete coordination model like Reo or BIP. Reconfigurations were defined based on five primitive operations. These operations were proved to cope with invariant properties of coordination patterns. Moreover, they are compositional in the sense that they can be combined (in sequence) to yield complex and reusable reconfigurations. The dynamic setting was also considered, where consistency on state transfer upon reconfiguration is the *de facto* question. An approach to deal with this was addressed. It considers a symbolic view of states of the automata-based semantic model of the coordination pattern (which is composed from the semantic models underlying the type of each channel).

Both coordination and reconfiguration patterns were additionally discussed from a stochastic perspective. In this view, stochastic delays for data reading/writing and processing were associated to channel ends and to the channels themselves, respectively. Following an approach similar to that considered for $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$, the deployment environment of the (stochastic) coordination patterns was considered as an independent entity issuing requests, with some delay, to the ports of the patterns. Reconfigurations are, as before, applied to these stochastic patterns.

The *reasoning part* of ARIS introduced the necessary mechanisms to analyse, compare and classify reconfigurations in terms of the three dimensions: behaviour, structure and stochastic performance. Reasoning from a behavioural perspective requires the association of a semantic model to the coordination patterns. Reconfigurations are then studied from the comparison of the resulting coordination patterns

or the changes inflicted upon the original coordination pattern. The comparisons of these patterns are made considering the simulation and bisimulation relations defined for the chosen semantic model. Reasoning from a structural perspective do not require the fixation of any semantic model. The graph of a coordination pattern already provides the necessary structure upon which (structural) properties may be investigated. A suitable hybrid logic — $Hp\mathcal{E}$ — was devised for the direct formulation of these properties. In this perspective, reasoning about reconfigurations is made by considering the preservation of (invariant) formulas of $Hp\mathcal{E}$ after application of reconfigurations. Additionally, a notion of bisimulation to compare the structure of the coordination patterns was defined and an Hennessy-Milner-like result proved. Finally, reasoning on a performance perspective entailed the association of a quantitative semantic model to coordination patterns, for which $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ was chosen. Its use demanded for the definition of a notion of *quality of service (QoS)* for coordination patterns. $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ is used to obtain the concrete values for each QoS dimension defined. Then, reasoning about reconfigurations is made upon the comparison of the values of these dimensions for the obtained patterns or the changes produced in these values comparing with an original coordination pattern.

From the three reasoning perspectives, a classification of reconfigurations was created, and a taxonomy defined for each perspective. The concepts in these taxonomies, and the relation with other more generic concepts, allowed for the definition of a (base) ontology of reconfigurations. This ontology may grow by considering either other reasoning perspectives; the relation between concepts of the different perspectives; or the relation of these concepts with other concerns of reconfigurations *e.g.,* state transfer, transparency, among others.

In the end of Part II of this thesis, an application of ARIS was presented. In essence it is the integration of the framework features and mechanisms in the context of a self-adaptation approach. The approach is based, as many others, in the MAPE-K (monitor, analyser, planner, executer and knowledge) feedback loop reference model from autonomic computing and control theory. The particularity of the approach is the use of a transition system of coordination-based reconfigurations — the *reconfiguration transition system (RTS)* — to drive most of the analyses and decisions. Of course, analysis is based on property verification by taking advantage of $Hp\mathcal{E}$ logic or $\mathsf{IMC}_{\mathsf{Reo}}$-based tool support. For decisions, a simple yet effective approach based on tailored notions of trigger constraints and filters are used. The overall approach was further refactored in an attempt to deliver adaptation as a service.

Part III of the thesis was concerned with tool support and illustration of the

ARIS framework. Tool support is provided as an Eclipse plugin editor — the CooPLa Editor — that serves as an interface for several external tools. The CooPLa Editor provides two *domain-specific languages (DSLs)*: CooPLa and ReCooPLa. The former is targeted on modelling coordination patterns with or without stochastic information; the latter allows for the design of reconfigurations to be applied upon the CooPLa patterns. The editor also provides multiple plugins that enable a toolchain for coordination modelling, reconfiguration, quantitative analysis, among others. In particular, the IMCREOTool plugin creates $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ models from the CooPLa stochastic instances and is able to export these models into established quantitative analysis tools *e.g.,* PRISM, CADP or IMCA among others. The reconfiguration engine takes both CooPLa and ReCooPLa specifications and generate simulations of the application of ReCooPLa reconfigurations upon the CooPLa patterns. Moreover, the CooPLa Editor is able to export CooPLa patterns as both Vereofy circuits, enabling their behavioural analysis, and Reo, enabling a more comprehensive set of possibilities ranging from animation to simulation and interfacing with other external tools. One of these tools may be again the CooPLa Editor, which is also able to import Reo specification into CooPLa patterns.

Finally, a case study was conducted to show the applicability of ARIS and how the tool support is able to give a hand to the working software architect. The case study was an hypothetical situation based on real-world data from a real-world system — the *Access Society's Knowledge (ASK)* system. It entailed the transformation of a static version of ASK into a dynamic, self-adaptive one.

The work developed in this thesis fits in the area of architectural reconfiguration of software systems. Only a small part of it was tackled: the reconfiguration of the coordination layer of SOA systems. It was objective of this work to formalise a framework for modelling reconfigurations on that layer, and analyse them over three concerns, considered of uttermost importance. It is believed that such a framework may contribute to the correct construction of SOA systems with a strong emphasis on performance requirements, which, in particular, seek on dynamic reconfigurations the way to keep such requirements above predefined values.

11.2 Related work discussion

In this section, the state of the art presented in Chapters 3 and 5 is revisited. A comparison between its most relevant entries and the work reported in this document is made.

11.2.1 Models for performance evaluation

The contribution of this thesis for research on performance evaluation is the introduction of the $\mathsf{IMC}_{\mathsf{Reo}}$ semantic model. $\mathsf{IMC}_{\mathsf{Reo}}$ is, in fact, a concrete instance of the IMC formalism, with necessary extensions to support coordination-specific features. As such, it deviates from stochastic models, like *stochastic process algebras (SPAs)*, *stochastic Petri nets (SPNs)*, *stochastic automata networks (SANs)*, queueing networks, or even other Markovian models, in the obvious ways.

$\mathsf{IMC}_{\mathsf{Reo}}$ focus on providing a quantitative semantics for stochastic Reo. Not surprisingly, the relevant related works are those accomplished within the Reo community.

The *continuous-time constraint automata (CCA)* [30] model attempts to provide the necessary stochastic semantic model for Reo. It closely follows the IMC formalism, but it is, in fact, a direct descendant of *constraint automata (CA)*. Like $\mathsf{IMC}_{\mathsf{Reo}}$, it is compositional, but it does not capture the context-dependency and it is not consistent in the sense that it does not provide a unique model for each Reo channel. Also, as it is a pre-stochastic Reo model, it obviously does not capture the correct semantics of stochastic Reo.

Another attempt was the *quantitative intensional automata (QIA)* model [16]. It was introduced as an operational semantic model for stochastic Reo. It was one of the first models to capture context-dependency. However, it is a non-compositional model [198, 16]. Also, it has no direct tools for its quantitative analysis; a conversion into *continuous-time Markov chain (CTMC)* is necessary to that end. $\mathsf{IMC}_{\mathsf{Reo}}$ can directly use CADP tools due to its IMC inheritance. For using it with PRISM tool, however, a conversion has to be performed, which is done with techniques already associated to IMC like determinisation. This spares the need for tailor-made algorithms as those introduced for QIA.

Similarly, *stochastic Reo automata (SRA)* [199, 198, 200] also arose as an operational semantics for stochastic Reo. Contrarily to the other models for Reo, including $\mathsf{IMC}_{\mathsf{Reo}}$ itself, SRA is a compact, low state-space model. Like $\mathsf{IMC}_{\mathsf{Reo}}$, it is compositional and captures context-dependency. It falls behind $\mathsf{IMC}_{\mathsf{Reo}}$, however, due to the lack of tool support. As for the QIA model, translations into CTMC were devised, enabling the use of such stochastic model for computing performance of stochastic Reo. But CTMC are not compositional and therefore any change to the stochastic Reo model would entail the computation of the SRA and its derived CTMC models again. A conversion of SRA into IMC was also attempted, but it was concluded that IMCs were not suitable to capture stochastic Reo semantics [200]. $\mathsf{IMC}_{\mathsf{Reo}}$ proved precisely the opposite.

The introduction of $\mathcal{D}$IMC$_{\text{Reo}}$ comes with the byproduct of improving (by clearly separating concerns) stochastic Reo. This approach differs, thus, from those mentioned above, as the model is more in line with real-world scenarios. Moreover, it still preserves the same properties of IMC$_{\text{Reo}}$, but enables faster compositions, due to its two-phase composition operation.

Although on a different coordination formalism, SBIP [40] plays a role similar to that of stochastic Reo. It has a specific stochastic semantics associated that is based on transitions systems. This semantics is compositional and captures both stochastic and non-deterministic behaviours. Specific tool support is made available via algorithms for statistical model checking. An advantage of this model as compared to IMC$_{\text{Reo}}$, is the ability to support various probabilistic distributions, rather than only exponential ones.

IMC$_{\text{Reo}}$, its tools within the CooPLa Editor and its external interfacing tools, define a specific framework for performance evaluation. It is focused on the coordination layer and therefore it substantially differs from other approaches like CB-SPE, ROBOCOP, KLAPPER or Palladio. In particular, these approaches target specific component models and rely on high-level languages for the description of components and their stochastic information. In the CooPLa Editor approach, this information is expressed in the CooPLa DSL, and it is limited to processing delays. The strategy for performance evaluation is achieved by the translation of the system model into some stochastic model, where queueing networks are mostly preferred. In the CooPLa Editor, the system model (as represented by its coordination layer) is translated into $\mathcal{D}$IMC$_{\text{Reo}}$ from which performance is computed.

11.2.2 Software reconfiguration

The contribution of this thesis for research on software reconfiguration was concretised in the development of ARIS, the framework discussed along Part II of this document. ARIS allows for modelling coordination patterns and reconfigurations both in stochastic and non-stochastic settings, and for reasoning about these entities from three different perspectives. The focus of ARIS is the reconfiguration of the coordination structure of a system, which is known to deliver its overall behaviour, in the context of *service-oriented computing (SOC)*. As such, this approach for reconfigurations differs from traditional ones that target components (not the interaction between them) and define high-level patterns for adding, updating or removing such elements [211, 133, 148, 197, 182, 225, 77].

In ARIS, coordination patterns are modelled as graphs, and reconfigurations are composed operations able to change these structures. This deviates, in several as-

pects, from the algebraic and categorical approaches referred in the state of the art. It is by no means a better or a worst approach; only different, with pros and contras. Categorical approaches take full advantage of graph grammars and graph transformation as a mathematically sound framework for dealing with reconfigurations. ARIS is a methodological approach. In graph grammars, reconfigurations are presented as rewriting rules able to change the architectural graph via pattern matching (*i.e.,* in a functional/declarative perspective); in this thesis, reconfiguration are concrete operations with a precise behaviour defined over the structure of the coordination pattern graph (*i.e.,* in an imperative perspective). This reveals low-level operations applied in a single location of the architecture, while rewriting rules are high-level, and able for application wherever they match the architectural structure. But the latter are neither reusable nor compositional operations, while the former are naturally both.

ARIS falls somehow behind in what concerns reconfiguration reasoning techniques. The graph transformation approaches are provided with a comprehensive set of tools and techniques to that end. Critical pair analysis, dynamic reconfiguration verification, state space analysis or invariant properties checking are relevant examples and only a few. From the literature, however, there are no works reporting on how reconfigurations can be compared or classified in that setting.

The work of C. Krause [170] is a relevant state-of-the-art attempt for coordination-based reconfigurations that takes advantage of graph transformations. In general, Krause's approach differs from this thesis's work on the aspects stated above. But, the author goes further and derives strategies to both triggering reconfigurations based on context-dependency and data-flow, and applying reconfigurations when the coordination layer of a system is distributed, which were not addressed inhere. Moreover, C. Krause proposes a methodology for applying reconfigurations in dynamic settings. In his approach, *input/output (IO)* requests in the coordinator ports, are interrupted, creating pending requests. After safely applying the necessary reconfiguration rules, the engine reactivates the coordinator ports and the pending requests are served. For state transfer he assumes the existence of projections of a global semantic model in the semantic models of all the channels in a connector; from there he proposes to restore states of the channels that are not changed and assume the initial states of the created ones. In ARIS, dynamic reconfigurations are assumed to occur when the system is in a quiescent state. The state transfer is computed via a formalised symbolic approach. Moreover, C. Krause's approach does not consider reconfigurations in the stochastic setting, while this thesis does. In fact, it focus on such information to derive triggers for reconfigurations.

Also D. Clarke [92] has reported work in reconfigurations of coordination structures. In his work, the author defines a model for Reo connectors, proposes an axiomatisation for their construction, and a logic to reason about system properties before and after the application of reconfigurations. This work shares similarities with ARIS. First, the models for Reo connectors and coordination patterns are very similar. The latter differ in the fact that channels may have more than two ends, and its independence from the coordination model. Second, reconfigurations are defined as low-level operations for adding channels and joining, splitting, hiding and forgetting nodes. By only targeting nodes, the underlying connector becomes complex, requiring extra garbage collection features to reduce its size. In ARIS, reconfigurations are defined from five primitives that target the direct manipulation of channels, nodes and coordination patterns structures as a whole, while keeping their atomicity and the underlying structure with only the essential pieces. Both approaches define a logic for reasoning about the reconfigurations. They are different in spirit, though. D. Clarke's is a modal logic expanding *computation tree logic (CTL)* with a specific modality for reconfigurations. Formulas of this logic are verified upon a structure where edges represent a reconfiguration operation and nodes are Reo connectors with a semantic model expressed via CA. Only behavioural properties can be verified, though. In ARIS, a hybrid logic is used. Its formulas are verified over the graph structure of a coordination pattern, and express properties about structural aspects of such entities. In ARIS, however, two more perspectives to reason about reconfigurations were presented, allowing for the definition of a comprehensive classification method for such operations. Concerning dynamic reconfigurations, in specific to state transfer, D. Clarke presents a solution based on the inductive computation of *possible* resuming states from the application of each primitive reconfiguration. Oppositely, the symbolic approach proposed in ARIS uniquely identifies the resuming state.

Dy-BIP [55] is another state-of-the-art approach for architectural reconfigurations targeting the coordination layer of a system. The approach for reconfiguration is completely different from the one followed in ARIS, therefore a concrete comparison is difficult. The concept of reconfiguration is not defined in Dy-BIP, actually. It is related with computation of a global semantic model on-the-fly, during execution and based on global system constraints. While this reduces the state space of the underlying semantic model, it precludes the verification of global properties. The dynamic evolution of the system to suitable configurations may be provoked when some actions/ports of the atomic components are not available; however no work was found in the literature that connects Dy-BIP with SBIP (or other stochastic

model) for stochastic based evolution of a system.

11.3 Future work and research directions

Several research issues arose along the development of this thesis. Some are expected to extend the techniques proposed in ARIS; others may lead to new and independent research topics. In the following paragraphs, some of these topics are briefly discussed.

Partitioned stochastic analysis

$\mathsf{IMC}_{\mathsf{Reo}}$ and $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ models still present some drawbacks. One of them is the number of states of the final (deployed) models. For analysis purposes this may become undesirable as the coordination patterns grow and/or become more complex. Although PRISM and CADP [186] are known to accept models with millions of states, there is always a limit imposed by, *e.g.,* memory issues of the machine running these tools. For this reason, a proposed future work is to explore recent work on Reo [157], which introduces a notion of connector partitions for distributing connector schemes over multiple machines. $\mathsf{IMC}_{\mathsf{Reo}}$ and $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ models may benefit from such research in order to be partitioned and analysed separately. The following research questions arise:

- *how can one correctly compose the values obtained from the separated analysis of each partition?*
- *how will the analysis in these partitions cope with features like context-dependency or nondeterminism, if these features depend on a global view of the coordination model?*

Queueing networks for stochastic coordination modelling

Queueing networks are a possible solution to improve both readability and scalability of approaches, based on transition systems, for stochastic analysis. Using queueing networks to model the coordination layer of a system may be a suitable substitute to $\mathsf{IMC}_{\mathsf{Reo}}$ or $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$. However, despite of this formalism being compositional and the amount of related research, *e.g.,* on finite capacity queues [32, 31, 213] or on lossy queues [163], it is not immediate how could they model, compositionally, a coordination structure. Actually,

- *Is it really possible to obtain a queueing network model that enables analysis of stochastic channel-based coordination?*
- *How can context-dependency, synchronisation or atomicity features of coordination formalisms be correctly captured in such a model?*

Failure model for quantitative analysis

Both $\mathsf{IMC}_{\mathsf{Reo}}$ and $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ consider processing delays for channels, environment and nodes. They always assume, however, that these components never fail. For instance, channels always transmit data from an end to another. For sure, lossy channels may lose data, but this is due to its dependency on the context. In this sense, a failure model for $\mathsf{IMC}_{\mathsf{Reo}}$ and $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ is in order for more realistically representing system coordination. A simple approach would be to add a failure state to each entity. At least, two Markovian transitions would also be necessary: one with a delay for failure and another with a delay to recover. This would raise, of course, some interesting questions:

- *How can such a failure model be added while keeping the state-space dimension low?*
- *How would such model influence coordination features like context-dependency?*
- *Is it possible to define a failure model that is able to conveniently and consistently restore the execution state?*

Reconfigurations with costs

In this work, reconfigurations were assumed to be applied both immediately and without any cost or failure. These may be acceptable simplifications to the problem, but it does not completely meet the reality. An extension to the model of reconfiguration proposed in this thesis would be in order. A suitable notion of costs for reconfiguration could consider, for instance, delays in their application, probabilities to fail, energy costs, degradation or improvement on system QoS values, among others. This raises, for instance, the following research topics:

- *How exactly a quantitative relation between the system and its reconfigurations may be established?*
- *In which way could these costs be used to improve decisions within self-adaptation strategies?*
- *How could these costs be used to compare and classify reconfigurations?*

Families of coordination patterns

In software product lines, software systems are built from a set of common features. These features are selected with based on target markets and their specific objectives. Constraints exist that determine how features relate to each other, and, ultimately, define which features may pertain to a system. Reconfigurations are closely related to this research topic. For instance, at any time, a company that acquired a basic system may require to evolve into a premium version of it that offers extra features. Such a change may definitely entail a reconfiguration of the company system.

With this in mind, one may think of porting these concepts into coordination patterns and their reconfigurations. A notion of family of coordination patterns would, for example, bind the possible reconfigurations associated with respect to some criterium. Behavioural, structural and quantitative properties or ontology-based classifications could play a determinant role for this matter. Typical questions are:

- *What would characterise a family of coordination patterns?*
- *How would families of coordination patterns be compared and related by the reconfigurations that may be applied to more than one family?*

Relating behaviour and structural perspectives

An interesting question that was not addressed in this work was *how can the behavioural and structural perspectives, considered in this thesis, be related*? One perspective resorts to a semantic model associated to the coordination pattern graph; the other uses the graph itself. One compares coordination patterns and reconfigurations using bisimulation and simulation relations; the other uses an hybrid logic for the same objective. Hidden between these perspectives may lie an Hennessy-Milner-like result relating modal equivalence in $Hp\mathcal{E}$ and bisimilarity in a specific (could it be whichever?) semantic model chosen for coordination patterns. This would be an interesting result to connect these apparently unrelated perspectives.

Self-adaptation approach development

Last, but certainly not least, an obvious work for further research is the development of a platform for self-adaptation of SOA systems, following the approach proposed in this thesis. This would entail the development of the MAPE-K modules and, additionally, deliver a specification language for the definition of triggers and filters. One solution for the latter would be to extend ReCooPLa DSL with such capabilities.

Moreover, the step into the cloud environment for delivering adaptation as a service would be a promising work, full of interesting challenges.

* * *

Certainly, many more questions were raised and left open for further research. This is, for sure, what leverages Science. The contribution of this thesis is, no doubt, a tiny little bump in the Knowldege boundary line. Answering to the opened research questions would, over time, reestablish that round, smooth surface.

Appendix A

Deriving $\mathsf{IMC}_{\mathsf{Reo}}$ from Stochatic Coordination Patterns

This appendix presents algorithms for the conversion of stochastic coordination pattern into $\mathsf{IMC}_{\mathsf{Reo}}$ (Algorithm A.1) and into $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ (Algorithm A.2). Therein, functions to access the obvious components of tuple-defined datatypes are assumed.

Algorithm A.1: Converting stochastic coordination pattern into $\mathsf{IMC}_{\mathsf{Reo}}$.

```
1  datatype State : R × T × Q
2  datatype Trans : Mark ℝ+ × State | Inter {ℰ} × State
3  datatype IMC_Reo : State ↦ Trans
4  datatype End : ℰ × ℝ+
5  datatype Flow : {{ℰ} × ℝ+}
6  datatype Channel : {End} × ℐ × 𝒯 × {End} × {Flow}
7  datatype CoordPat : {Channel} × {{End}}
8  datatype Env : {End} ↦ ℝ+
9
10 input: CoordPat ρ, Env env
11 output: IMC_Reo
12 begin
13   imcs ← {};
14   foreach c in channels(ρ)
15     imc ← IMC_Reo(type(c))
16     foreach st in keys(imc)
17       foreach t in imc[ st ]
18         if construct(t) = Mark
19           if R(state(t)) ≠ R(st) ∧ R(state(t))\ R(st)∈ keys(env)
20             rate(t) ← env[ R(state(t))\R(st) ]
21           else
22             if T(state(t)) ≠ T(st) ∧ T(st)\T(state(t))∈ map(π1,flows(c))
23               rate(t) = the{γ ∈ f | f ∈ flow(c) ∧ π1(f)= T(st)\T(state(t))}
24             end
25           end
26         end
27       end
28     end
29     imcs ← imcs ∪ {imc}
30   end
31   return compose(imcs)
32 end
```

In a nutshell, Algorithm A.1 creates an $\mathsf{IMC}_{\mathsf{Reo}}$ for each channel in the given stochastic coordination pattern. It does so by using an $\mathsf{IMC}_{\mathsf{Reo}}$ template generated via function $IMC_{Reo}(\ldots)$, for each channel type. The rates in transitions of such template, are, of course, incorrect. By inspecting the information in the (origin and target) states of each transition in the template, one is able to decide which ports

are requesting IO operations or transmitting data. Based on these ports, either from the environment or from the channel flows, the correct stochastic value is retrieved and associated to that transition. Finally the concretised templates are composed to deliver the combined $\mathsf{IMC}_{\mathsf{Reo}}$.

Algorithm A.2 assumes the datatypes presented in Algorithm A.1, and (re)defines two of its own.

Algorithm A.2: Converting stochastic coordination pattern into $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$.

```
1  datatype State : R × T × ×E × D × Q
2  datatype DIMC_Reo : State ↦ Trans
3
4  input: CoordPat ρ, Env env
5  output: DIMC_Reo
6  begin
7    dimcs ← {}
8    envs ← {}
9
10   foreach c in channels(ρ)
11     dimc ← DIMC_Reo(type(c))
12     foreach st in keys(dimc)
13       foreach t in dimc[ st ]
14         if construct(t) = Mark
15           if T(state(t)) ≠ T(st) ∧ T(st)\T(state(t))∈ map(π1,flows(c))
16             rate(t) = the{γ ∈ f | f ∈ flow(c) ∧ π1(f) = T(st)\T(state(t))}
17           end
18         end
19       end
20     end
21     dimcs ← dimcs ∪ dimc
22   end
23
24   foreach nd in nodes(ρ)
25     dimcs ← dimcs ∪ DIMC_Reo(nd, ð_nd^enq, ð_nd^deq)
26   end
27
28   foreach ⟨nd,rt⟩ in keys(env)
29         envs ← envs ∪ DIMC_Reo(nd,rt)
30   end
31
32   return deploy(compose(dimcs), envs)
33 end
```

In summary, Algorithm A.2 creates a $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ for each channel and node in the given stochastic coordination pattern, and to each entry in the environment. Overloaded function $DIMC_{Reo}(\ldots)$ is used to obtain the template $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ of each such entity, provided that are suitable parameters. In the case of nodes and environment entries, the function is assumed to generate the templates with the correct rates, since these are given in the parameters. In the end, nodes and channels are composed together; the resulting $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ is then *deployed* in the generated $\mathcal{D}\mathsf{IMC}_{\mathsf{Reo}}$ environment.

Appendix B

CooPLa Grammar

In this appendix, the full concrete grammar for the CooPLa language is given in *extended Backus-Naur form (eBNF)*. In order to increase its readability, the grammar was divided into four logical parts: Listing B.1 presents the main grammar that glues the principal structures; Listing B.2, B.3 and B.4 present the specific grammar for such structures: channels, patterns and stochastic instances (respectively).

By convention, capital letter words refer to typical regular expressions.

Listing B.1: CooPLa main grammar

```
coopla   → import* element*
element  → channel_def
         |  pattern_def
         |  stochastic_def
import   → 'import' FILE_PATH ';'
```

Listing B.2: CooPLa channels grammar

```
channel_def  → 'channel' channel_sig extension? '{' channel_body '}'
channel_sig  → ID dimensions? '(' ports? ':' ports? (':' ID '=' condition)? ')'
dimensions   → '@' ( ID | INT )
             |  '~' ( ID | INT ) ( ',' ( ID | INT ) )*
ports        → ID ( ',' ID )*
condition    → '<' ID ( ',' ID )* '>'
extension    → 'extends' ID
channel_body → state_def? ( flow_def ';' )+
state_def    → 'state' ':' ID ';' 'observers' ':' ID ( ',' ID )* ';'
flow_def     → requests '->' flow_type
requests     → request ( ',' request )*
request      → '!'? ID
flow_type    → normal_flow ( '|' normal_flow) ?
             |  ID '?' flow_type : flow_type
normal_flow  → 'flow' ( ID | 'NULL' ) 'to' '@'? ( ID | 'NULL' )
```

Listing B.3: CooPLa coordination patterns grammar

```
pattern_def       → 'pattern' pattern_signature '{' pattern_body '}'
pattern_signature → ID '(' ports? ':' ports? ')'
pattern_body      → 'use' ':' pattern_decls 'in' ':' pattern_comps
pattern_decls     → ( reference_sig 'as' ID (',' ID)* ';' )+
reference_sig     → ('(' ID ')')? channel_sig
pattern_comps     → ( port_definition ';' )+ ( join_operation ';' )+
port_definition   → ID '=' (p_acc | join_operation)
p_acc             → ID '.' ID
join_operation    → 'join' port_access_list 'as' ID
                  |  'xor' port_access_list2 'as' ID
port_access_list  → '[' p_acc ( ',' p_acc )* ']'
port_access_list2 → '[' p_acc+ ':' p_acc ',' p_acc (',' p_acc)* ']'
```

Listing B.4: CooPLa stochastic instances grammar

```
1 stochastic_def   → 'stochastic' ID '@' stochastic_list ID
2 stochastic_list  → '{' stoch_elem+ '}'
3 stoch_elem       → ID ('#' ID)? '=' stoch_val ';'
4 stoch_val        → FLOAT
5                  |  '(' FLOAT ',' FLOAT ')'
```

Appendix C

ReCooPLa Grammar

In this appendix, the concrete grammar for the ReCooPLa language is given in eBNF. Additionally, the ReCooPLa implementation of the set of reconfiguration patterns introduced in Chapter 6 is presented.

C.1 The Grammar

In order to increase the readability of the ReCooPLa grammar, it was divided into three parts: Listing C.1 presents the main grammar that glues the principal parts: reconfigurations and their application presented in Listing C.2 and C.3, respectively. By convention, capital letter words refer to typical regular expressions.

Listing C.1: ReCooPLa main grammar

```
1 recoopla   → import* content
2 import     → 'import' FILE_PATH ';'
3 content    → reconf_def main_def?
```

Listing C.2: ReCooPLa reconfiguration grammar

```
1 reconf_def   → 'reconfiguration' ID '(' args_def ')' block
2 args_def     → arg_def (',' arg_def)*
3 arg_def      → datatype list_ids
4 datatype     → 'Pattern' | 'Channel' | 'Name' | 'Node' | 'XOR'
5              | other_type '<' datatype '>'
6 other_type   → 'Set' | 'Pair' | 'Triple'
7 list_ids     → ID (',' ID)*
8 block        → '{' instruction+ '}'
9 instruction  → ( decl | assign | apply | for ) ';'
10 decl        → datatype var_def (',' var_def)*
11 var_def     → ID | assign
12 assign      → ID '=' assigner
13 assigner    → expr | apply
14 apply       → ID? '@' rec_call
15 rec_call    → ('join' | 'split' | 'par' | 'remove' | 'const' | 'id' | ID) op_args
16 op_args     → '(' args? ')'
17 args        → expr (',' expr)*
18 for         → 'forall' '(' datatype ID ':' ID ')' block
19 expr        → factor ( ('+' | '&' | '-' ) factor)?
20 factor      → ID | oper | cons
21 oper        → ID ('#' ID)? '.' att_call
22 cons        → ( 'S' | 'P' | 'T' ) op_args
23 att_call    → ('in' | 'out') ( '(' INT ')' )?
24             | 'name' | 'nodes' | 'names' | 'fat' | 'and' | 'trd'
```

Listing C.3: ReCooPLa application grammar

```
1 main_def → 'main' '[' m_args? ']' m_block
2 m_args   → m_decl (';' m_decl)*
3 m_decl   → ID ids
4 ids      → ID (',' ID)*
5 m_block  → '{' m_instr+ '}'
6 m_instr  → ( m_assign | m_apply) ';'
7 m_assig  → ID ids '=' m_apply
8 m_apply  → ID '@' rec_call
```

C.2 Reconfiguration Patterns in ReCooPLa

The following presents the ReCooPLa implementation of the reconfiguration patterns introduced in Section 6.2.3.

removeP (Cs)

```
1 reconfiguration removeP (Set<Name> Cs ) {
2     forall ( Name n : Cs) {
3         @ remove(n);
4     }
5 }
```

overlapP (ρ, X)

```
1 reconfiguration overlapP(Pattern p; Set<Pair<Node>> X) {
2     @ par (p);
3     forall(Pair<Node> n : X) {
4         Node n1, n2;
5         n1 = n.fst;
6         n2 = n.snd;
7         Set<Node> E = S(n1, n2);
8         @ join(E);
9     }
10 }
```

insertP (ρ, n, m_i, m_o)

```
1 reconfiguration insertP (Pattern p; Node n,mi,mo) {
2     Pattern p1 = @ par(p) ;
3     Pattern p2 = @ split(n) ;
4     Set<Node> Isp = p2.in - p1.in ;
5     Set<Node> Osp = p2.out - p1.out;
6     Set<Node> Ospmi = Osp + S(mi);
7     Set<Node> Ispmo = Isp + S(mo);
8     @ join(Ospmi) ;
9     @ join(Ispmo) ;
10 }
```

replaceP (ρ, X, Cs)

```
1 reconfiguration replaceP(
2         Pattern p; Set<Pair<Node>> X; Set<Name> Cs ) {
3     @ removeP(Cs) ;
4     @ overlapP(p,X);
5 }
```

implodeP (Cs)

```
1 reconfiguration implodeP(Set<Name> Cs) {
2     Pattern p = @ id();
3     Pattern p1 = @ removeP (Cs);
4     Set<Node> Nds = p1.nodes - p.nodes ;
5     @ join( Nds ) ;
6 }
```

moveP
(ch, e, n)

```
1  reconfiguration moveP(Name ch; Node e, n) {
2      Pattern p = @ id() ;
3      Set<Node> x = p#ch.in + p#ch.out;
4      x = x - S(e) ;
5      Pattern p1 = @ split(e);
6      Set<Node> E = p1#ch.in + p1#ch.out ;
7      Set<Node> Isp = p1.in - p.in ;
8      Set<Node> Osp = p1.out - p.out ;
9      Set<Node> IOsp = Isp + Osp
10     Set<Node> E2 = E - S(x);
11     Set<Node> IOspE = IOsp - E;
12     Set<Node> E2n = E2 + S(n);
13     @ join( IOspE );
14     @ join( E2n );
15 }
```

References

[1] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. "Workflow Patterns". In: *Distributed and Parallel Databases* 14.1 (July 1, 2003), pp. 5–51.

[2] Franz Achermann, Markus Lumpe, Jean-Guy Schneider, and Oscar Nierstrasz. "PICCOLA—a Small Composition Language". In: *Formal Methods for Distributed Processing*. Ed. by Howard Bowman and John Derrick. New York, NY, USA: Cambridge University Press, 2001, pp. 403–426.

[3] Aditya Agrawal, Gabor Karsai, and Feng Shi. "A UML-based graph transformation approach for implementing domain-specific model transformations". In: *International Journal on Software and Systems Modeling* (2003), pp. 1–19.

[4] Jonathan Aldrich, Craig Chambers, and David Notkin. "Architectural Reasoning in ArchJava". In: *Object-Oriented Programming (ECOOP 2002)*. Ed. by Boris Magnusson. Vol. 2374. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, 2002, pp. 334–367.

[5] Jonathan Aldrich, Craig Chambers, and David Notkin. "ArchJava: connecting software architecture to implementation". In: *Proceedings of the 24th International Conference on Software Engineering*. ICSE 2002. New York, NY, USA: ACM, 2002, pp. 187–197.

[6] Luca de Alfaro and Thomas A. Henzinger. "Interface Automata". In: *ACM SIGSOFT Software Engineering Notes* 26.5 (Sept. 2001), pp. 109–120.

[7] Robert Allen. "A Formal Approach to Software Architecture". PhD thesis. Carnegie Mellon, School of Computer Science, Jan. 1997.

[8] Robert Allen, Rémi Douence, and David Garlan. "Specifying and Analyzing Dynamic Software Architectures". In: *Configurations* 1382 (1998), pp. 1–15.

[9] Robert Allen and David Garlan. "Formalizing Architectural Connection". In: *Proceedings of the 16th International Conference on Software Engineering*. ICSE 1994. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994, pp. 71–80.

[10] Françoise André, Erwan Daubert, and Guillaume Gauvrit. "Distribution and self-adaptation of a framework for dynamic adaptation of services". In: *The Sixth Inte,national Conference on Internet and Web Applications and Services*. ICIW 2011. Red Hook, NY, USA: IARIA, Mar. 2011, pp. 16–21.

[11] Farhad Arbab. "What Do You Mean, Coordination?" In: *Bulletin of the Dutch Association for Theoretical Computer Science (NVTI)*. Mar. 1998, pp. 11–22.

[12] Farhad Arbab. "Coordination of Mobile Components". In: *Electronic Notes in Theoretical Computer Science* 54 (Aug. 2001), pp. 1–16.

[13] Farhad Arbab. "Abstract Behavior Types: A Foundation Model for Components and Their Composition". In: *Formal Methods for Components and Objects*. Ed. by Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul de Roever. Vol. 2852. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003. Chap. 2, pp. 33–70.

[14] Farhad Arbab. "Reo: a channel-based coordination model for component composition". In: *Mathematical Structures in Computer Science* 14.3 (June 2004), pp. 329–366.

[15] Farhad Arbab. *Composition by Interaction*. Tech. rep. Leiden University, Oct. 28, 2005.

[16] Farhad Arbab, Tom Chothia, Rob van der Mei, Sun Meng, YoungJoo Moon, and Chrétien Verhoef. "From Coordination to Stochastic Models of QoS". In: *Coordination Models and Languages*. Ed. by John Field and Vasco Vasconcelos. Vol. 5521. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009. Chap. 14, pp. 268–287.

[17] Farhad Arbab, Christian Krause, Ziyan Maraikar, Young-Joo Moon, and José Proença. "Modeling, Testing and Executing Reo Connectors with the Eclipse Coordination Tools". In: *proceedings of the International Workshop on Formal Aspects of Component Software*. FACS 2008. Salamanca, Spain, Sept. 2008.

[18] Farhad Arbab and Farhad Mavaddat. "Coordination through Channel Composition". In: *Coordination Models and Languages*. Ed. by Farhad Arbab and Carolyn Talcott. Vol. 2315. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, Mar. 14, 2002. Chap. 6, pp. 275–297.

[19] Farhad Arbab and Jan Rutten. "A Coinductive Calculus of Component Connectors". In: *Recent Trends in Algebraic Development Techniques*. Ed. by Martin Wirsing, Dirk Pattinson, and Rolf Hennicker. Vol. 2755. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003. Chap. 2, pp. 34–55.

[20] Carlos Areces and Juan Heguiabehere. "HyLoRes 1.0: Direct Resolution for Hybrid Logics". In: *Automated Deduction. CADE-18*. Ed. by Andrei Voronkov. Vol. 2392. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2002, pp. 156–160.

[21] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. "Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations". In: *Model Driven Engineering Languages and Systems*. Ed. by DorinaC Petriu, Nicolas Rouquette, and Øystein Haugen. Vol. 6394. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, 2010, pp. 121–135.

[22] Juan C. Augusto. "Ambient Intelligence: Basic Concepts and Applications". In: *Software and Data Technologies*. Ed. by Joaquim Filipe, Boris Shishkov, and Markus Helfert. Vol. 10. Communications in Computer and Information Science. Berlin, Heidelberg: Springer, 2008, pp. 16–26.

[23] Juan C. Augusto. "Past, Present and Future of Ambient Intelligence and Smart Environments". In: *Agents and Artificial Intelligence*. Ed. by Joaquim Filipe, Ana Fred, and Bernadette Sharp. Vol. 67. Communications in Computer and Information Science. Berlin, Heidelberg: Springer, 2010, pp. 3–15.

[24] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. "Model-checking continuous-time Markov chains". In: *Transactions on Computational Logic* 1 (July 2000), pp. 162–170.

[25] Jos C. M. Baeten. "A brief history of process algebra". In: *Theoretical Computer Science* 335.2-3 (May 23, 2005), pp. 131–146.

[26] Elham Bafrooi. "Specification and Implementation of Workflow Control Patterns In Reo". M.Sc. thesis. Waterloo, Ontario, Canada: University of Waterloo, 2006.

[27] Christel Baier, Tobias Blechmann, Joachim Klein, and Sascha Klüppelholz. "A Uniform Framework for Modeling and Verifying Components and Connectors". In: *Coordination Models and Languages*. Ed. by John Field and Vasco T. Vasconcelos. Vol. 5521. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009. Chap. 13, pp. 247–267.

[28] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost P. Katoen. "Model-Checking Algorithms for Continuous-Time Markov Chains". In: *IEEE Transactions on Software Engineering* 29.6 (2003), pp. 524–541.

[29] Christel Baier, Marjan Sirjani, Farhad Arbab, and Jan J. M. M. Rutten. "Modeling component connectors in Reo by constraint automata". In: *Science of Computer Programming* 61.2 (2006), pp. 75–113.

[30] Christel Baier and Verena Wolf. "Stochastic reasoning about channel-based component connectors". In: *Coordination Models and Languages*. Vol. 4038. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 1–15.

[31] Simonetta Balsamo, Vittoria de Nitto Personè, and Paola Inverardi. "A Review on Queueing Network Models with Finite Capacity Queues for Software Architectures Performance Prediction". In: *Performance Evaluation* 51.2-4 (Feb. 2003), pp. 269–288.

[32] Simonetta Balsamo and Vittoria de Nitto Personè. "A survey of product form queueing networks with blocking and their equivalences". In: *Annals of Operations Research* 48.1 (1994), pp. 31–61.

[33] Luciano Baresi, Reiko Heckel, Sebastian Thöne, and Dániel Varró. "Modeling and Analysis of Architectural Styles Based on Graph Transformation". In: *Proceedings of the 6th ICSE Workshop on Component-Based Software Engineering: Automated Reasoning and Prediction*. CBSE 2003. May 2003, pp. 67–72.

[34] Ananda Basu, Saddek Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, and Joseph Sifakis. "Rigorous Component-Based System Design Using the BIP Framework". In: *IEEE Software* 28.3 (May 2011), pp. 41–48.

[35] Ananda Basu, Marius Bozga, and Joseph Sifakis. "Modeling Heterogeneous Real-time Components in BIP". In: *Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*. SEFM 2006. Washington, DC, USA: IEEE Computer Society, 2006, pp. 3–12.

[36] Thais Batista, Ackbar Joolia, and Geoff Coulson. "Managing Dynamic Reconfiguration in Component-Based Systems". In: *Software Architecture*. Ed. by Ron Morrison and Flavio Oquendo. Vol. 3527. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005. Chap. 1, pp. 1–17.

[37] Matthias Becker, Markus Luckey, and Steffen Becker. "Performance Analysis of Self-adaptive Systems for Requirements Validation at Design-time". In: *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures*. QoSA 2013. New York, NY, USA: ACM, 2013, pp. 43–52.

[38] Steffen Becker, Heiko Koziolek, and Ralf Reussner. "The Palladio Component Model for Model-driven Performance Prediction". In: *Journal of Systems and Software* 82.1 (Jan. 2009), pp. 3–22.

[39] Maurice ter Beek, Antonio Bucchiarone, and Stefania Gnesi. "Dynamic Software Architecture Development: Towards an Automated Process". In: *Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications*. SEAA 2009. IEEE, Aug. 2009, pp. 105–108.

[40] Saddek Bensalem, Marius Bozga, Benoit Delahaye, Cyrille Jegourel, Axel Legay, and Ayoub Nouri. "Statistical Model Checking QoS Properties of Systems with SBIP". In: *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*. Ed. by Tiziana Margaria and Bernhard Steffen. Vol. 7609. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 327–341.

[41] Albert Benveniste, Claude Jard, Ajay Kattepur, Sidney Rosario, and John A. Thywissen. "QoS-aware management of monotonic service orchestrations". In: *Formal Methods in System Design* 44.1 (2014), pp. 1–43.

[42] Jan A. Bergstra and Jan W. Klop. "ACP_τ a universal axiom system for process specification". In: *Algebraic Methods: Theory, Tools and Applications*. Ed. by Martin Wirsing and Jan A. Bergstra. Vol. 394. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1989, pp. 445–463.

[43] Marco Bernardo, Paolo Ciancarini, and Lorenzo Donatiello. "Architecting Families of Software Systems with Process Algebras". In: *ACM Transactions on Software Engineering and Methodology* 11.4 (Oct. 2002), pp. 386–426.

[44] Marco Bernardo and Roberto Gorrieri. "Extended Markovian Process Algebra". In: *Concurrency Theory. CONCUR'96*. Vol. 1119. Lecture Notes in Computer Science. Springer, 1996, pp. 315–330.

[45] Antonia Bertolino and Raffaela Mirandola. "CB-SPE Tool: Putting Component-Based Performance Engineering into Practice". In: *Component-Based Software Engineering*. Ed. by Ivica Crnkovic, JudithA Stafford, HeinzW Schmidt, and Kurt Wallnau. Vol. 3054. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 233–248.

[46] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. "Semiring-based constraint satisfaction and optimization". In: *Journal of the ACM* 44.2 (1997), pp. 201–236.

[47] Patrick Blackburn. "Representation, Reasoning, and Relational Structures: a Hybrid Logic Manifesto". In: *Logic Journal of IGPL* 8.3 (2000), pp. 339–365.

[48] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Sept. 30, 2002.

[49] Patrick Blackburn and Jerry Seligman. "Hybrid Languages". In: *Journal of Logic, Language and Information* 4.3 (1995), pp. 251–272.

[50] Tobias Blechmann and Christel Baier. "Checking Equivalence for Reo Networks". In: *Electronic Notes in Theoretical Computer Science* 215 (June 2008), pp. 209–226.

[52] Egor Bondarev, Michel Chaudron, and Peter With. "A Process for Resolving Performance Trade-Offs in Component-Based Architectures". In: *Component-Based Software Engineering*. Ed. by Ian Gorton, George T. Heineman, Ivica Crnković, Heinz W. Schmidt, Judith A. Stafford, Clemens Szyperski, and Kurt Wallnau. Vol. 4063. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 254–269.

[53] Egor Bondarev, Johan Muskens, Peter de With, Michel Chaudron, and Johan Lukkien. "Predicting real-time properties of component assemblies: a scenario-simulation approach". In: *Proceedings of the 30th Euromicro Conference*. Aug. 2004, pp. 40–47.

[54] Marcello M. Bonsangue, Dave Clarke, and Alexandra Silva. "A model of context-dependent component connectors". In: *Science of Computer Programming* 77.6 (June 17, 2012), pp. 685–706.

[55] Marius Bozga, Mohamad Jaber, Nikolaos Maris, and Joseph Sifakis. "Modeling Dynamic Architectures Using Dy-BIP". In: *Software Composition*. Ed. by Thomas Gschwind, Flavio De Paoli, Volker Gruhn, and Matthias Book. Vol. 7306. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 1–16.

[56] Torben Braüner. *Hybrid Logic and its Proof-Theory*. 2011th ed. Applied Logic Series. Springer, Nov. 30, 2010.

[57] Torben Braüner and Valeria de Paiva. "Intuitionistic hybrid logic". In: *Journal of Applied Logic* 4.3 (2006), pp. 231–255.

[58] Mario Bravetti. "Revisiting Interactive Markov Chains". In: *Electronic Notes in Theoretical Computer Science* 68.5 (May 2003), pp. 65–84.

[59] Mario Bravetti, Cinzia Di Giusto, Jorge A. Pérez, and Gianluigi Zavattaro. "Adaptable processes". In: *Logical Methods in Computer Science* 8.4 (Nov. 2012), pp. 1–71.

[60] Mario Bravetti, Cinzia Giusto, Jorge A. Pérez, and Gianluigi Zavattaro. "Towards the Verification of Adaptable Processes". In: *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*. Ed. by Tiziana Margaria and Bernhard Steffen. Vol. 7609. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 269–283.

[61] Lothar Breuer and Dieter Baum. *An Introduction to Queueing Theory and Matrix-Analytic Method*. 2005th ed. Springer, Nov. 7, 2005.

[62] Yuriy Brun, Giovanna M. Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. "Engineering Self-Adaptive Systems Through Feedback Loops". In: *Software Engineering for Self-Adaptive Systems*. Ed. by Betty H. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee. Vol. 5525. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 48–70.

[63] Dario Bruneo, Salvatore Distefano, Francesco Longo, and Marco Scarpa. "Stochastic Evaluation of QoS in Service-Based Systems". In: *IEEE Transactions on Parallel and Distributed Systems* 24.10 (Oct. 2013), pp. 2090–2099.

[64] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean B. Stefani. "The FRACTAL Component Model and Its Support in Java: Experiences with Auto-adaptive and Reconfigurable Systems". In: *Software: Practice and Experience* 36.11-12 (Sept. 2006), pp. 1257–1284.

[65] Roberto Bruni, Antonio Bucchiarone, Stefania Gnesi, Dan Hirsch, and Alberto Lluch Lafuente. "Graph-Based Design and Analysis of Dynamic Software Architectures Concurrency, Graphs and Models". In: *Concurrency, Graphs and Models*. Ed. by Pierpaolo Degano, Rocco Nicola, and José Meseguer. Vol. 5065. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008. Chap. 4, pp. 37–56.

[66] Roberto Bruni, Antonio Bucchiarone, Stefania Gnesi, and Hernán Melgratti. "Modelling Dynamic Software Architectures using Typed Graph Grammars". In: *Electron. Notes Theor. Comput. Sci.* 213.1 (May 5, 2008), pp. 39–53.

[67] Roberto Bruni, Howard Foster, Alberto L. Lafuente, Ugo Montanari, and Emilio Tuosto. "A Formal Support to Business and Architectural Design for Service Oriented Systems". In: *Rigorous software engineering for service-oriented systems*. Ed. by Martin Wirsing and Matthias Hölzl. Berlin, Heidelberg: Springer, 2011, pp. 133–152.

[68] Roberto Bruni, Alberto L. Lafuente, and Ugo Montanari. "Hierarchical Design Rewriting with Maude". In: *Electronic Notes in Theoretical Computer Science* 238.3 (June 29, 2009), pp. 45–62.

[69] Roberto Bruni, Alberto L. Lafuente, Ugo Montanari, and Emilio Tuosto. "Service Oriented Architectural Design". In: *Trustworthy Global Computing*. Ed. by Gilles Barthe and Cédric Fournet. Vol. 4912. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 186–203.

[70] Antonio Bucchiarone. "Dynamic Software Architectures for Global Computing Systems". PhD thesis. Lucca, Italy: IMT Institute for Advanced Studies, Lucca, 2008.

[71] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. "Choreography and Orchestration: A Synergic Approach for System Design". In: *Service-Oriented Computing. ICSOC 2005*. Ed. by Boualem Benatallah, Fabio Casati, and Paolo Traverso. Vol. 3826. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005. Chap. 18, pp. 228–240.

[72] Rajkumar Buyya, James Broberg, and Andrzej M. Goscinski. *Cloud Computing Principles and Paradigms*. Wiley Publishing, 2011.

[73] Rajkumar Buyya, Chee S. Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility". In: *Future Generation Computer Systems* 25.6 (June 2009), pp. 599–616.

[74] Radu Calinescu, Carlo Ghezzi, Marta Kwiatkowska, and Raffaela Mirandola. "Self-adaptive Software Needs Quantitative Verification at Runtime". In: *Communications of the ACM* 55.9 (Sept. 2012), pp. 69–77.

[75] Radu Calinescu, Lars Grunske, Marta Kwiatkowska, Raffaela Mirandola, and Giordano Tamburrelli. "Dynamic QoS Management and Optimization in Service-Based Systems". In: *IEEE Transactions on Software Engineering* 37.3 (May 2011), pp. 387–409.

[76] Radu Calinescu and Marta Kwiatkowska. "Using Quantitative Analysis to Implement Autonomic IT Systems". In: *Proceedings of the 31st International Conference on Software Engineering*. ICSE 2009. Vancouver, BC, Canada: IEEE Computer Society, May 2009, pp. 100–110.

[77] Carlos Canal, Javier Cámara, and Gwen Salaün. "Structural reconfiguration of systems under behavioral adaptation". In: *Science of Computer Programming* 78.1 (Nov. 2012), pp. 46–64.

[78] Carlos Canal, Ernesto Pimentel, and José M. Troya. "Specification and Refinement of Dynamic Software Architectures". In: *Proceedings of the TC2 First Working IFIP Conference on Software Architecture*. WICSA 1999. Deventer, The Netherlands: Kluwer, B.V., 1999, pp. 107–126.

[79] Carlos Canal, Ernesto Pimentel, and José M. Troya. "Compatibility and inheritance in software architectures". In: *Science of Computer Programming* 41.2 (Oct. 2001), pp. 105–138.

[80] Antonio Cansado, Carlos Canal, Gwen Salaün, and Javier Cubo. "A Formal Framework for Structural Reconfiguration of Components under Behavioural Adaptation". In: *Electronic Notes in Theoretical Computer Science* 263 (June 3, 2010), pp. 95–110.

[81] Bogdan A. Caprarescu and Dana Petcu. "A Self-Organizing Feedback Loop for Autonomic Computing". In: *Proceedings of Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*. COMPUTATIONWORLD 2009. IEEE, Nov. 2009, pp. 126–131.

[82] Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Stefano Iannucci, Francesco Lo Presti, and Raffaela Mirandola. "MOSES: A Framework for QoS Driven Runtime Adaptation of Service-Oriented Systems". In: *IEEE Transactions on Software Engineering* 38.5 (2012), pp. 1138–1159.

[83] Nicholas Carriero and David Gelernter. "Linda in context". In: *Communications of the ACM* 32.4 (Apr. 1989), pp. 444–458.

[86] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. "Probabilistic Weighted Automata". In: *Concurrency Theory. CONCUR 2009*. Ed. by Mario Bravetti and Gianluigi Zavattaro. Vol. 5710. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 244–258.

[87] Betty H. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee. "Software Engineering for Self-Adaptive Systems: A Research Roadmap". In: *Software Engineering for Self-Adaptive Systems*. Vol. 5525. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 1–26.

[88] Shang-Wen Cheng and David Garlan. "Stitch: A language for architecture-based self-adaptation". In: *Journal of Systems and Software* 85.12 (Mar. 2012), pp. 2860–2875.

[89] Allan Clark, Stephen Gilmore, Jane Hillston, and Mirco Tribastone. "Stochastic Process Algebras". In: *Formal Methods for Performance Evaluation*. Ed. by Marco Bernardo and Jane Hillston. Vol. 4486. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, pp. 132–179.

[90] Dave Clarke. *Reasoning About Connector Reconfiguration I: Equivalence of Constructions*. Tech. rep. Amsterdam: CWI - Centrum voor Wiskunde en Informatique, Feb. 2005.

[91] Dave Clarke. "Reasoning About Connector Reconfiguration II: Basic Reconfiguration Logic". In: *Electronic Notes in Theoretical Computer Science* 159 (May 24, 2006), pp. 61–77.

[92] Dave Clarke. "A Basic Logic for Reasoning about Connector Reconfiguration". In: *Fundamenta Informaticae* 82 (Feb. 2008), pp. 361–390.

[93] Dave Clarke, David Costa, and Farhad Arbab. "Connector Colouring I: Synchronisation and Context Dependency". In: *Electronic Notes in Theoretical Computer Science* 154 (May 2006), pp. 101–119.

[94] Edmund M. Clarke and E. Allen Emerson. "Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic". In: *Logic of Programs, Workshop*. London, UK: Springer, 1982, pp. 52–71.

[95] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.

[96] Alan Colman and Jun Han. "Coordination Systems in Role-Based Adaptive Software". In: *Coordination Models and Languages*. Ed. by Jean-Marie Jacquet and Gian P. Picco. Vol. 3454. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, 2005, pp. 63–78.

[97] Diane J. Cook, Juan C. Augusto, and Vikramaditya R. Jakkula. "Ambient intelligence: Technologies, applications, and opportunities". In: *Pervasive and Mobile Computing* 5.4 (Aug. 15, 2009), pp. 277–298.

[98] Andrea Corradini, Ugo Montanari, Francesca Rossi, Hartmut Ehrig, Reiko Heckel, and Michael Löwe. "Algebraic Approaches to Graph Transformation - Part I: Basic Concepts and Double Pushout Approach". In: *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. Ed. by Grzegorz Rozenberg. World Scientific, 1997, pp. 163–246.

[99] Ângelo Costa, José C. Castillo, Paulo Novais, Antonio Fernández-Caballero, and Ricardo Simões. "Sensor-driven agenda for intelligent home care of the elderly". In: *Expert Systems with Applications* 39.15 (Nov. 2012), pp. 12192–12204.

[100] David Costa. "Formal Models for Component Connectors". PhD thesis. Amsterdam: Vrije University, Oct. 2010.

[101] Juan C. Cruz and Stéphane Ducasse. "A Group Based Approach for Coordinating Active Objects". In: *Coordinatio Languages and Models*. Ed. by Paolo Ciancarini and AlexanderL Wolf. Vol. 1594. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1999, pp. 355–370.

[102] Carlos E. Cuesta, Pablo de la Fuente, Manuel Barrio-Solórzano, and Encarnación Beato. "Coordination in a Reflective Architecture Description Language". In: *Coordination Models and Languages*. Ed. by Farhad Arbab and Carolyn Talcott. Vol. 2315. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2002, pp. 141–148.

[103] Carlos E. Cuesta, Pablo de la Fuente, Manuel Barrio-Solórzano, and Encarnación Beato. "An "abstract process" approach to algebraic dynamic architecture description". In: *The Journal of Logic and Algebraic Programming* 63.2 (May 2005), pp. 177–214.

[104] F. Curbera, Y. Goland, J. Klein, and F. Leymann. *Business Process Execution Language for Web-Services*. Tech. rep. IBM, 2002.

[105] Pedro R. D'Argenio, Holger Hermanns, Joost-Pieter Katoen, and Ric Klaren. "MoDeST — A Modelling and Description Language for Stochastic Timed Systems". In: *Process Algebra and Probabilistic Methods. Performance Modelling and Verification*. Ed. by Luca de Alfaro and Stephen Gilmore. Vol. 2165. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2001, pp. 87–104.

[106] Eric M. Dashofy, André van der Hoek, and Richard N. Taylor. "A Highly-Extensible, XML-Based Architecture Description Language". In: *Proceedings of the Working IEEE/IFIP Conference on Software Architecture*. WICSA 2001. Washington, DC, USA: IEEE Computer Society, 2001.

[107] Pierre-Charles David, Thomas Ledoux, Marc Léger, and Thierry Coupaye. "FPath and FScript: Language support for navigation and reliable reconfiguration of Fractal architectures". In: *Annals of Telecommunications* 64.1-2 (Feb. 1, 2009), pp. 45–63.

[108] Rocco De Nicola, Gianluigi Ferrari, Ugo Montanari, Rosario Pugliese, and Emilio Tuosto. "A Process Calculus for QoS-Aware Applications". In: *Coordination Models and Languages*. Ed. by Jean-Marie Jacquet and Gian P. Picco. Vol. 3454. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, 2005, pp. 33–48.

[109] Pierpaolo Degano and Ugo Montanari. "A model for distributed systems based on graph rewriting". In: *Journal of the ACM* 34.2 (Apr. 1987), pp. 411–449.

[110] Yuxin Deng and Matthew Hennessy. "On the semantics of Markov automata". In: *Information and Computation* 222 (Jan. 2013), pp. 139–168.

[111] Simon Dobson, Spyros Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. "A Survey of Autonomic Communications". In: *ACM Transactions on Autonomous and Adaptive Systems* 1.2 (Dec. 2006), pp. 223–259.

[112] Jim Dowling and Vinny Cahill. "The K-Component Architecture Meta-Model for Self-Adaptive Software". In: *Metalevel Architectures and Separation of Crosscutting Concerns*. Ed. by Akinori Yonezawa and Satoshi Matsuoka. Vol. 2192. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, Oct. 5, 2001. Chap. 6, pp. 81–88.

[113] Hartmut Ehrig, Michael Pfender, and Hans Jürgen Schneider. "Graph-Grammars: An Algebraic Approach". In: *Proceedings of the 14th Annual Symposium on Switching and Automata Theory*. 1973, pp. 167–180.

[114] Christian Eisentraut, Holger Hermanns, and Lijun Zhang. "On Probabilistic Automata in Continuous Time". In: *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science*. LICS 2010. IEEE, July 2010, pp. 342–351.

[115] Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pål Krogdahl, Min Luo, and Tony Newling. *Patterns: Service-Oriented Architecture and Web Services*. 1st ed. IBM Redbooks, Apr. 2004.

[116] Thomas Erl. *SOA Design Patterns*. 1st ed. Prentice Hall PTR, Jan. 9, 2009.

[117] William Feller. *An Introduction to Probability Theory and Its Applications*. 3rd. Vol. 1. Wiley, Jan. 1, 1968.

[118] Paulo Fernandes, Brigitte Plateau, and William J. Stewart. "Efficient Descriptor-vector Multiplications in Stochastic Automata Networks". In: *Journal of the ACM* 45.3 (May 1998), pp. 381–414.

[119] José L. Fiadeiro and Antónia Lopes. "CommUnity on the Move: Architectures for Distribution and Mobility". In: *Formal Methods for Components and Objects*. Ed. by Frank S. Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul Roever. Vol. 3188. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 177–196.

[120] José L. Fiadeiro and Antónia Lopes. "A model for dynamic reconfiguration in service-oriented architectures". In: *Software and Systems Modeling* 12.2 (Feb. 19, 2013), pp. 349–367.

[121] Thorsten Fischer, Jörg Niere, Lars Torunski, and Albert Zündorf. "Story Diagrams: A New Graph Rewrite Language Based on the Unified Modeling Language and Java". In: *Theory and Application of Graph Transformations*. Ed. by Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg. Vol. 1764. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2000. Chap. 21, pp. 296–309.

[122] Jacqueline Floch, Cristina Frà, Rolf Fricke, Kurt Geihs, Michael Wagner, Jorge L. Gallardo, Eduardo S. Cantero, Stephan Mehlhase, Nearchos Paspallis, Hossein Rahnama, Pedro A. Ruiz, and Ulrich Scholz. "Playing MUSIC — building context-aware and self-adaptive mobile applications". In: *Software: Practice and Experience* 43.3 (Mar. 1, 2013), pp. 359–388.

[124] Cédric Fournet and Georges Gonthier. "The Join Calculus: A Language for Distributed Mobile Programming". In: *Applied Semantics*. Ed. by Gilles Barthe, Peter Dybjer, Luís Pinto, and João Saraiva. Vol. 2395. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2002, pp. 268–332.

[125] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. "CADP 2011: a toolbox for the construction and analysis of distributed processes". In: *International Journal on Software Tools for Technology Transfer* (2012), pp. 1–19.

[126] David Garlan, Robert T. Monroe, and David Wile. "ACME: An Architecture Description Interchange Language". In: *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*. CASCON 1997. Toronto, Ontario, Nov. 1997, pp. 169–183.

[127] David Garlan, Shang-Wen Cheng, An C. Huang, Bradley Schmerl, and Peter Steenkiste. "Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure". In: *Computer* 37.10 (Oct. 2004), pp. 46–54.

[128] David Garlan, Robert T. Monroe, and David Wile. "ACME: Architectural Description of Component-Based Systems". In: *Foundations of Component-Based Systems*. Ed. by Gary T. Leavens and Murali Sitaraman. Cambridge University Press, 2000, pp. 47–68.

[129] David Garlan, Bradley Schmerl, and Shang-Wen Cheng. "Software Architecture-Based Self-Adaptation". In: *Autonomic Computing and Networking*. Ed. by Yan Zhang, Laurence T. Yang, and Mieso K. Denko. US: Springer, 2009. Chap. 2, pp. 31–55.

[130] David Garlan and Mary Shaw. "An Introduction to Software Architecture". In: *Advances in Software Engineering and Knowledge Engineering*. Ed. by Vincenzo Ambriola, Genoveffa Tortora, and Shi K. Chang. Vol. 1. River Edge, NJ, USA: World Scientific Publishing Company, 1993, pp. 1–39.

[131] Erann Gat. "Three-layer Architectures". In: *Artificial Intelligence and Mobile Robots*. Ed. by David Kortenkamp, R. Peter Bonasso, and Robin Murphy. Cambridge, MA, USA: MIT Press, 1998, pp. 195–210.

[132] Jean Gelissen. *Robocop: Robust open component based software architecture*. Nov. 2014. URL: `http://www.hitech-projects.com/euprojects/robocop/`.

[133] Hassan Gomaa and Mohamed Hussein. "Software reconfiguration patterns for dynamic evolution of software architectures". In: *Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture*. WICSA 2004. 2004, pp. 79–88.

[134] Vincenzo Grassi, Raffaela Mirandola, and Enrico Randazzo. "Model-Driven Assessment of QoS-Aware Self-Adaptation". In: *Software Engineering for Self-Adaptive Systems*. Ed. by Betty H. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee. Vol. 5525. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 201–222.

[135] Vincenzo Grassi, Raffaela Mirandola, Enrico Randazzo, and Antonino Sabetta. "KLAPER: An Intermediate Language for Model-Driven Predictive Analysis of Performance and Reliability". In: *The Common Component Modeling Example*. Ed. by Andreas Rausch, Ralf Reussner, Raffaela Mirandola, and František Plášil. Vol. 5153. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 327–356.

[136] Vincenzo Grassi, Raffaela Mirandola, and Antonino Sabetta. "From Design to Analysis Models: A Kernel Language for Performance and Reliability Analysis of Component-based Systems". In: *Proceedings of the 5th International Workshop on Software and Performance*. WOSP 2005. New York, NY, USA: ACM, 2005, pp. 25–36.

[137] Vincenzo Grassi, Raffaela Mirandola, and Antonino Sabetta. "A Model-driven Approach to Performability Analysis of Dynamically Reconfigurable Component-based Systems". In: *Proceedings of the 6th International Workshop on Software and Performance*. WOSP 2007. Buenes Aires, Argentina: ACM, 2007, pp. 103–114.

[138] Orna Grumberg and Helmut Veith, eds. *25 Years of Model Checking: History, Achievements, Perspectives*. Berlin, Heidelberg: Springer, 2008.

[139] Dennis Guck, Tingting Han, Joost-Pieter Katoen, and Martin R. Neuhäuer. "Quantitative Timed Analysis of Interactive Markov Chains". In: *NASA Formal Methods*. Ed. by Alwyn E. Goodloe and Suzette Person. Vol. 7226. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 8–23.

[140] Svein O. Hallsteinsen, Kurt Geihs, Nearchos Paspallis, Frank Eliassen, Geir Horn, Jorge Lorenzo, Alessandro Mamelli, and George A. Papadopoulos. "A development framework and methodology for self-adapting applications in ubiquitous computing environments". In: *Journal of Systems and Software* 85.12 (Dec. 2012), pp. 2840–2859.

[141] Moshe Haviv. *Queues: A Course in Queueing Theory*. Vol. 191. International Series in Operations Research & Management Science. Berlin, Heidelberg: Springer, June 3, 2013.

[142] Luke Herbert and Robin Sharp. "Using Stochastic Model Checking to Provision Complex Business Services". In: *Proceedings of the IEEE 14th International Symposium on High-Assurance Systems Engineering*. HASE 2012. IEEE, Oct. 2012, pp. 98–105.

[143] Holger Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*. Vol. 2428. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2002.

[144] Holger Hermanns and Joost-Pieter Katoen. "The how and why of interactive Markov chains". In: *Proceedings of the 8th international conference on Formal methods for components and objects*. FMCO 2010. Berlin, Heidelberg: Springer, 2010, pp. 311–337.

[145] Ulrich Herzog. "Formal Description, Time and Performance Analysis a Framework". In: *Entwurf und Betrieb verteilter Systeme*. Ed. by Theo Härder, Hartmut Wedekind, and Gerhard Zimmermann. Vol. 264. Informatik-Fachberichte. Berlin, Heidelberg: Springer, 1990, pp. 172–190.

[146] Jane Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[147] Dan Hirsch, Paola Inverardi, and Ugo Montanari. "Modeling Software Architectures and Styles with Graph Grammars and Constraint Solving". In: *Software Architecture*. Ed. by Patrick Donohoe. Vol. 12. IFIP — The International Federation for Information Processing. US: Springer, 1999, pp. 127–143.

[148] Petr Hnětynka and František Plášil. "Dynamic Reconfiguration and Access to Services in Hierarchical Component Models". In: *Component-Based Software Engineering*. Ed. by Ian Gorton, George T. Heineman, Ivica Crnković, Heinz W. Schmidt, Judith A. Stafford, Clemens Szyperski, and Kurt Wallnau. Vol. 4063. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006. Chap. 27, pp. 352–359.

[149] C. A. R. Hoare. "Communicating Sequential Processes". In: *Commun. ACM* 21.8 (Aug. 1978), pp. 666–677.

[150] André van Hoorn, Matthias Rohr, Asad Gul, and Wilhelm Hasselbring. "An Adaptation Framework Enabling Resource-efficient Operation of Software Systems". In: *Proceedings of the Warm Up Workshop for ACM/IEEE ICSE 2010*. WUP 2009. New York, NY, USA: ACM, 2009, pp. 41–44.

[151] Nikolaus Huber, André van Hoorn, Anne Koziolek, Fabian Brosig, and Samuel Kounev. "Modeling run-time adaptation at the system architecture level in dynamic service-oriented environments". In: *Service Oriented Computing and Applications* 8.1 (Mar. 2014), pp. 73–89.

[152] Markus C. Huebscher and Julie A. McCann. "A Survey of Autonomic Computing—Degrees, Models, and Applications". In: *ACM Computer Surveys* 40.3 (Aug. 2008), pp. 1–28.

[153] IBM Corp. *An architectural blueprint for autonomic computing*. USA: IBM Corp., Oct. 2004.

[154] Mamdouh H. Ibrhaim, Kerrie Holley, Nicolai M. Josuttis, Brenda Michelson, Dave Thomas, and John Devadoss. "The future of SOA: what worked, what didn't, and where is it going from here?" In: *Companion to the 22nd ACM SIGPLAN conference on Object oriented programming systems and applications companion*. OOPSLA 2007. New York, NY, USA: ACM, 2007, pp. 1034–1038.

[155] Daniel Jackson. "Alloy: A Lightweight Object Modelling Notation". In: *ACM Transactions on Software Engineering and Methodology* 11.2 (Apr. 2002), pp. 256–290.

[156] Sung-Shik T. Q. Jongmans and Farhad Arbab. "Overview of Thirty Semantic Formalisms for Reo". In: *Scientific Annals of Computer Science* 22.1 (2012), pp. 201–251.

[157] Sung-Shik T. Q. Jongmans, Francesco Santini, and Farhad Arbab. "Partially-Distributed Coordination with Reo". In: *22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing*. PDP 2014. IEEE, Feb. 2014, pp. 697–706.

[158] Nicolai M. Josuttis. *SOA in Practice: The Art of Distributed System Design*. 1st ed. O'Reilly Media, Aug. 31, 2007.

[159] Jaber Karimpour, Ayaz Isazadeh, and Habib Izadkhah. "Early performance assessment in component-based software systems". In: *IET Software* 7.2 (Apr. 2013), pp. 118–128.

[160] Jeffrey O. Kephart and David M. Chess. "The Vision of Autonomic Computing". In: *Computer* 36.1 (Jan. 14, 2003), pp. 41–50.

[161] Jeffrey O. Kephart and William E. Walsh. "An artificial intelligence perspective on autonomic computing policies". In: *5th IEEE International Workshop on Policies for Distributed Systems and Networks*. POLICY 2004. IEEE, June 2004, pp. 3–12.

[162] Sascha Klüppelholz. "Verification of Branching-Time and Alternating-Time Properties for Exogenous Coordination Models". PhD thesis. Dresden, Germany: Technische Universität Dresden, Mar. 19, 2012.

[163] Hisashi Kobayashi and Brian L. Mark. "Generalized Loss Models and Queueing-loss Networks". In: *International Transactions in Operational Research* 9.1 (Jan. 1, 2002), pp. 97–112.

[164] Christian Koehler, Farhad Arbab, and Erik de Vink. "Reconfiguring Distributed Reo Connectors". In: *Recent Trends in Algebraic Development Techniques*. Ed. by Andrea Corradini and Ugo Montanari. Vol. 5486. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 221–235.

[165] Christian Koehler, David Costa, José Proença, and Farhad Arbab. "Reconfiguration of Reo Connectors Triggered by Dataflow". In: *Proceedings of the 7th International Workshop on Graph Transformation and Visual Modeling Techniques. GT-VMT'08*. Vol. 10. Electronic Communications of the EASST. 2008, pp. 1–13.

[166] Christian Koehler, Alexander Lazovik, and Farhad Arbab. "Connector Rewriting with High-Level Replacement Systems". In: *Electronic Notes in Theoretical Computer Science* 194.4 (Apr. 2008), pp. 77–92.

[167] Dexter Kozen. "Results on the propositional μ-calculus". In: *Theoretical Computer Science* 27.3 (Jan. 1983), pp. 333–354.

[168] Heiko Koziolek. "Performance evaluation of component-based software systems: A survey". In: *Performance Evaluation* 67.8 (Aug. 3, 2010), pp. 634–658.

[169] Jeff Kramer and Jeff Magee. "Dynamic Configuration for Distributed Systems". In: *IEEE Transactions on Software Engineering* 11.4 (1985), pp. 424–436.

[170] Christian Krause. "Reconfigurable Component Connectors". PhD thesis. Amsterdam, The Netherlands: Leiden University, 2011.

[171] Christian Krause, Ziyan Maraikar, Alexander Lazovik, and Farhad Arbab. "Modeling dynamic reconfigurations in Reo using high-level replacement systems". In: *Science of Computer Programming* 76.1 (2011), pp. 23–36.

[172] Marta Kwiatkowska, Gethin Norman, and David Parker. "Stochastic Model Checking". In: *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*. Ed. by Marco Bernardo and Jane Hillston. Vol. 4486. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, pp. 220–270.

[173] Marta Kwiatkowska, Gethin Norman, and David Parker. "A Framework for Verification of Software with Time and Probabilities". In: *Proceedings of the 8th International Conference on Formal Modelling and Analysis of Timed Systems. FORMATS'10*. Ed. by Krishnendu Chatterjee and Thomas A. Henzinger. Vol. 6246. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 25–45.

[174] Martin Lange. "Model Checking for Hybrid Logic". In: *Journal of Logic, Language and Information* 18.4 (2009), pp. 465–491.

[175] Marc Léger, Thomas Ledoux, and Thierry Coupaye. "Reliable Dynamic Reconfigurations in the Fractal Component Model". In: *Proceedings of the 6th International Workshop on Adaptive and Reflective Middleware*. ARM 2007. New York, NY, USA: ACM, 2007.

[176] Marin Litoiu, Mircea Mihaescu, Dan Ionescu, and Bogdan Solomon. "Scalable Adaptive Web Services". In: *Proceedings of the 2nd international workshop on Systems development in SOA environments*. SDSOA 2008. New York, NY, USA: ACM, Feb. 2008, pp. 47–52.

[177] David C. Luckham, John J. Kenney, Larry M. Augustin, James Vera, Doug Bryan, and Walter Mann. "Specification and analysis of system architecture using Rapide". In: *IEEE Transactions on Software Engineering* 21.4 (Apr. 1995), pp. 336–354.

[178] David C. Luckham and James Vera. "An event-based architecture definition language". In: *IEEE Transactions on Software Engineering* 21.9 (Sept. 6, 1995), pp. 717–734.

[180] Jeff Magee, Naranker Dulay, Susan Eisenbach, and Jeff Kramer. "Specifying Distributed Software Architectures". In: *Proceedings of the 5th European Software Engineering Conference*. Ed. by Wilhelm Schäfer and Pere Botella. Vol. 989. Lecture Notes in Computer Science. London, UK: Springer, 1995, pp. 137–153.

[181] Jeff Magee and Jeff Kramer. "Dynamic structure in software architectures". In: *Proceedings of the 4th ACM SIGSOFT symposium on Foundations of software engineering*. Vol. 21. SIGSOFT 1996. New York, NY, USA: ACM, Nov. 1996, pp. 3–14.

[182] Michal Malohlava and Tomáš Bureš. "Language for reconfiguring runtime infrastructure of component-based systems". In: *Proceedings of the Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*. MEMICS 2008. Znojmo, Czech Republic, Nov. 2008.

[183] Marco A. Marsan. "Stochastic Petri nets: An elementary introduction". In: *Advances in Petri Nets 1989*. Ed. by Grzegorz Rozenberg. Vol. 424. Lecture Notes in Computer Science. Springer, 1990, pp. 1–29.

[184] Marco A. Marsan, Gianfranco Balbo, Gianni Conte, Susanna Donatelli, and Giuliana Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. 1st. New York, NY, USA: John Wiley & Sons, Inc., 1994.

[185] Robert von Massow, André van Hoorn, and Wilhelm Hasselbring. "Performance Simulation of Runtime Reconfigurable Component-Based Software Architectures". In: *Software Architecture*. Ed. by Ivica Crnkovic, Volker Gruhn, and Matthias Book. Vol. 6903. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 43–58.

[186] Radu Mateescu and Wendelin Serwe. "Model checking and performance evaluation with CADP illustrated on shared-memory mutual exclusion protocols". In: *Science of Computer Programming* 78.7 (2013), pp. 843–861.

[187] E. Michael Maximilien and Munindar P. Singh. "A framework and ontology for dynamic Web services selection". In: *IEEE Internet Computing* 8.5 (Sept. 2004), pp. 84–93.

[188] Nenad Medvidovic and Richard N. Taylor. "A classification and comparison framework for software architecture description languages". In: *IEEE Transactions on Software Engineering* 26.1 (Jan. 2000), pp. 70–93.

[189] Stephan Merz. "Model Checking: A Tutorial Overview". In: *Modeling and Verification of Parallel Processes*. Ed. by Franck Cassez, Claude Jard, Brigitte Rozoy, and MarkDermot Ryan. Vol. 2067. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2001, pp. 3–38.

[190] Robin Milner. *A Calculus of Communicating Systems*. Vol. 92. Lecture Notes in Computer Science. Springer, 1980.

[191] Robin Milner. *Communicating and Mobile Systems: The π-calculus*. New York, NY, USA: Cambridge University Press, 1999.

[192] Naftaly H. Minsky and Victoria Ungureanu. "Law-governed Interaction: A Coordination and Control Mechanism for Heterogeneous Distributed Systems". In: *ACM Transactions on Software Engineering and Methodology* 9.3 (July 2000), pp. 273–305.

[193] Jayadev Misra and William R. Cook. "Computation Orchestration: A Basis for Wide-area Computing". In: *Software and Systems Modeling* 6.1 (Mar. 2007), pp. 83–110.

[194] Michael K. Molloy. "On the Integration of Delay and Throughput Measures in Distributed Processing Models". PhD thesis. University of California, 1981.

[195] Robert T. Monroe and Armani Overview. *Capturing Software Architecture Design Expertise with Armani*. Tech. rep. Pittsburgh, PA: Carnegie Mellon University, Jan. 2001.

[196] Ugo Montanari and Francesca Rossi. "Graph Rewriting, Constraint Solving and Tiles for Coordinating Distributed Systems". In: *Applied Categorical Structures* 7.4 (1999), pp. 333–370.

[197] Francisco Moo-Mena and Khalil Drira. "Reconfiguration of Web Services Architectures: A model-based approach". In: *Proceedings of the 12th IEEE Symposium on Computers and Communications*. ISCC 2007. IEEE, July 2007, pp. 357–362.

[198] Young-Joo Moon. "Stochastic Models for Quality of Service of Component Connectors". PhD thesis. Universiteit Leiden, Oct. 2011.

[199] Young-Joo Moon, Alexandra Silva, Christian Krause, and Farhad Arbab. "A Compositional Semantics for Stochastic Reo Connectors". In: *Proceedings of the 9th International Workshop on the Foundations of Coordination Languages and Software Architectures*. 2010, pp. 93–107.

[200] Young-Joo Moon, Alexandra Silva, Christian Krause, and Farhad Arbab. "A compositional model to reason about end-to-end QoS in Stochastic Reo connectors". In: *Science of Computer Programming* 80 (Jan. 2014), pp. 3–24.

[201] Vivek Nallur and Rami Bahsoon. "A decentralized self-adaptation mechanism for service-based applications in the cloud". In: *IEEE Transactions on Software Engineering* 39.5 (May 2013), pp. 591–612.

[202] Nils J. Nilsson. *Principles of Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1980.

[204] Nuno Oliveira and Luís S. Barbosa. "Reconfiguration Mechanisms for Service Coordination". In: *Web Services and Formal Methods*. Ed. by Maurice H. ter Beek and Niels Lohmann. Vol. 7843. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 134–149.

[210] Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimhigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. "An architecture-based approach to self-adaptive software". In: *IEEE Intelligent Systems* 14.3 (May 1999), pp. 54–62.

[211] Peyman Oreizy, Nenad Medvidovic, and Richard N. Taylor. "Architecture-based runtime software evolution". In: *Proceedings of the 20th international conference on Software engineering*. ICSE 1998. Washington, DC, USA: IEEE Computer Society, 1998, pp. 177–186.

[212] Peyman Oreizy and Richard N. Taylor. "On the role of software architectures in runtime system reconfiguration". In: *Proceedings of the 4th International Conference on Configurable Distributed Systems*. Annapolis, MA, USA: IEEE, May 1998, pp. 61–70.

[213] Carolina Osorio and Michel Bierlaire. "An analytic finite capacity queueing network model capturing the propagation of congestion and blocking". In: *European Journal of Operational Research* 196.3 (Aug. 1, 2009), pp. 996–1007.

[214] Jonathan Ozik and Michael North. "Modeling Endogenous Coordination Using a Dynamic Language". In: *Simulating Interacting Agents and Social Phenomena*. Ed. by Shu-Heng Chen, Claudio Cioffi-Revilla, Nigel Gilbert, Hajime Kita, Takao Terano, Keiki Takadama, Claudio Cioffi-Revilla, and Guillaume Deffuant. Vol. 7. Agent-Based Social Systems. Japan: Springer, 2010, pp. 265–276.

[215] George A. Papadopoulos and Farhad Arbab. "Dynamic Reconfiguration in Coordination Languages". In: *High Performance Computing and Networking*. Ed. by Marian Bubak, Hamideh Afsarmanesh, Bob Hertzberger, and Roy Williams. Vol. 1823. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2000, pp. 197–206.

[216] George A. Papadopoulos and Farhad Arbab. "Configuration and dynamic reconfiguration of components using the coordination paradigm". In: *Future Generation Computer Systems* 17.8 (June 2001), pp. 1023–1038.

[217] Carlos Parra, Xavier Blanc, and Laurence Duchien. "Context Awareness for Dynamic Service-oriented Product Lines". In: *Proceedings of the 13th International Software Product Line Conference*. SPLC 2009. Pittsburgh, PA, USA: Carnegie Mellon University, 2009, pp. 131–140.

[221] Jim Pitman. *Probability*. Springer, June 1, 1999.

[222] František Plášil, Dušan Bálek, and Radovan Janeček. "SOFA/DCUP: architecture for component trading and dynamic updating". In: *Proceedings of the 4th International Conference on Configurable Distributed Systems*. CDS 1998. Washington, DC, USA: IEEE Computer Society, May 1998, pp. 43–51.

[223] Corrado Priami. "Stochastic π-Calculus". In: *The Computer Journal* 38.7 (1995), pp. 578–589.

[224] Jean P. Queille and Joseph Sifakis. "Specification and verification of concurrent systems in CESAR". In: *International Symposium on Programming*. Ed. by Mariangiola D. Ciancaglini and Ugo Montanari. Vol. 137. Lecture Notes in Computer Science. Springer, 1982, pp. 337–351.

[225] Andres J. Ramirez and Betty H. Cheng. "Design Patterns for Developing Dynamically Adaptive Systems". In: *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS 2010. New York, NY, USA: ACM, 2010, pp. 49–58.

[226] Shangping Ren, Yue Yu, Nianen Chen, Kevin Marth, Pierre-Etienne Poirot, and Limin Shen. "Actors, Roles and Coordinators - A Coordination Model for Open Distributed and Embedded Systems". In: *Coordination Models and Languages*. Ed. by Paolo Ciancarini and Herbert Wiklicky. Vol. 4038. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006. Chap. 16, pp. 247–265.

[227] Flávio Rodrigues. "An Engine for Coordination-based Architectural Reconfigurations". M.Sc. thesis. Braga, Portugal: Departamento de Informática, Universidade do Minho, Dec. 2014.

[228] Flávio Rodrigues, Nuno Oliveira, and Luís S. Barbosa. "ReCooPLa: a DSL for Coordination-based Reconfiguration of Software Architectures". In: *3rd Symposium on Languages, Applications and Technologies*. Ed. by Maria J. V. Pereira, José P. Leal, and Alberto Simões. Vol. 38. OpenAccess Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, June 2014, pp. 61–76.

[230] Grzegorz Rozenberg, ed. *Handbook of Graph Grammars and Computing by Graph Transformation: Foundations*. Vol. 1. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 1997.

[231] Nick Russell, Arthur H. M. ter Hofstede, Wil M. P. van der Aalst, and Natalya Mulyar. *Workflow Control-Flow Patterns: A Revised View*. Tech. rep. BPMcenter.org, 2006.

[232] Mazeiar Salehie and Ladan Tahvildari. "Self-adaptive Software: Landscape and Research Challenges". In: *ACM Transactions on Autonomous and Adaptive Systems* 4.2 (May 2009), pp. 1–42.

[233] Alejandro Sanchez, Luís S. Barbosa, and Daniel Riesco. "A language for behavioural modelling of architectural patterns". In: *Proceedings of the Third Workshop on Behavioural Modelling*. BMFA 2011. New York, NY, USA: ACM, 2011, pp. 17–24.

[234] Alejandro Sanchez, Luís S. Barbosa, and Daniel Riesco. "Bigraphical modelling of architectural patterns". In: *Formal Aspects of Component Software. FACS'2011*. Ed. by Farhad Arbab and Peter C. Ölveczky. Vol. 7253. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 313–330.

[236] Lionel Seinturier, Philippe Merle, Romain Rouvoy, Daniel Romero, Valerio Schiavoni, and Jean-Bernard Stefani. "A Component-Based Middleware Platform for Reconfigurable Service-Oriented Architectures". In: *Software: Practice and Experience* 42.5 (2011), pp. 559–583.

[237] Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

[238] John Simpson. *XPath and XPointer: Locating Content in XML Documents*. 1st ed. O'Reilly Media, Aug. 10, 2002.

[239] Borja Sotomayor, Ruben S. Montero, Ignacio M. Llorente, and Ian T. Foster. "Virtual Infrastructure Management in Private and Hybrid Clouds". In: *IEEE Internet Computing* 13.5 (Sept. 9, 2009), pp. 14–22.

[240] William J. Stewart, Karim Atif, and Brigitte Plateau. "The numerical solution of stochastic automata networks". In: *European Journal of Operational Research* 86.3 (Nov. 1995), pp. 503–525.

[241] Gabriele Taentzer. "AGG: A Graph Transformation Environment for Modeling and Validation of Software". In: *Applications of Graph Transformations with Industrial Relevance*. Ed. by John Pfaltz, Manfred Nagl, and Boris Böhlen. Vol. 3062. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004. Chap. 35, pp. 446–453.

[242] Carolyn L. Talcott. "Coordination Models Based on a Formal Model of Distributed Object Reflection". In: *Electronic Notes in Theoretical Computer Science* 150 (Mar. 2006), pp. 143–157.

[243] Richard N. Taylor, Nenad Medvidovic, and Peyman Oreizy. "Architectural styles for runtime software adaptation". In: *Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture. WICSA/ECSA 2009*. IEEE, Sept. 2009, pp. 171–180.

[244] Microsoft Team. *Microsoft Application Architecture Guide (Patterns & Practices)*. Second Edition. Microsoft Press, Nov. 22, 2009.

[245] Wolfgang Thomas. "Computation tree logic and regular ω-languages". In: *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*. Ed. by Jaco W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg. Vol. 354. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1989, pp. 690–713.

[246] Matthias Tichy and Benjamin Klöpper. "Planning Self-adaption with Graph Transformations". In: *Applications of Graph Transformations with Industrial Relevance*. Ed. by Andy Schürr, Dániel Varró, and Gergely Varró. Vol. 7233. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 137–152.

[247] Henk C. Tijms. *A First Course in Stochastic Models*. 2nd. Wiley, Apr. 18, 2003.

[248] Luis M. Vaquero, Luis R. Merino, Juan Caceres, and Maik Lindner. "A Break in the Clouds: Towards a Cloud Definition". In: *ACM SIGCOMM Computer Communication Review* 39.1 (Dec. 2008), pp. 50–55.

[249] Norha M. Villegas Machado, Hausi A. Müller, and Gabriel Tamura Morimitsu. "On Designing Self-Adaptive Software Systems". In: *Sistemas & Telemática* 9.18 (2011), pp. 29–51.

[250] Pieter Vromant, Danny Weyns, Sam Malek, and Jesper Andersson. "On Interacting Control Loops in Self-adaptive Systems". In: *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS 2011. New York, NY, USA: ACM, 2011, pp. 202–207.

[251] Michel A. Wermelinger. "Specification of Software Architecture Reconfiguration". PhD thesis. Lisboa, Portugal: Universidade Nova de Lisboa, 1999.

[252] Michel A. Wermelinger and José L. Fiadeiro. "Algebraic software architecture reconfiguration". In: *Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering*. ESEC/FSE 1999. London, UK: Springer-Verlag, 1999, pp. 393–409.

[253] Michel A. Wermelinger, Antónia Lopes, and José L. Fiadeiro. "A Graph Based Architectural (Re)Configuration Language". In: *Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. Vol. 26. ESEC/FSE 2001 5. New York, NY, USA: ACM, Sept. 2001, pp. 21–32.

[254] Danny Weyns, M. Usman Iftikhar, and Joakim Söderlund. "Do External Feedback Loops Improve the Design of Self-adaptive Systems? A Controlled Experiment". In: *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS 2013. Piscataway, NJ, USA: IEEE Press, 2013, pp. 3–12.

[255] Danny Weyns, Bradley Schmerl, Vincenzo Grassi, Sam Malek, Raffaela Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese, and Karl M. Göschka. "On Patterns for Decentralized Control in Self-Adaptive Systems". In: *Software Engineering for Self-Adaptive Systems II*. Ed. by Rogério de Lemos, Holger Giese, Hausi A. Müller, and Mary Shaw. Vol. 7475. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 76–107.

[256] Yunni Xia, Yi Liu, Ji Liu, and Qingsheng Zhu. "Modeling and Performance Evaluation of BPEL Processes: A Stochastic-Petri-Net-Based Approach". In: *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans* 42.2 (Mar. 2012), pp. 503–510.

[257] Uwe Zdun, Carsten Hentrich, and Wil M. P. van der Aalst. "A survey of patterns for Service-Oriented Architectures". In: *International Journal of Internet Protocol Technology* 1 (May 2006), pp. 132–143.

[258] Lijun Zhang and Martin Neuhäuer. "Model Checking Interactive Markov Chains". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Javier Esparza and Rupak Majumdar. Vol. 6015. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 53–68.