

Caromel · Henrio  
A Theory of Distributed Objects

Denis Caromel · Ludovic Henrio

# A Theory of Distributed Objects

Asynchrony – Mobility – Groups – Components

Preface by Luca Cardelli

With 114 Figures and 48 Tables

Denis Caromel

University of Nice Sophia Antipolis  
I3S CNRS – INRIA  
Institut universitaire de France  
2004 Rt. des Lucioles, BP 93  
06902 Sophia Antipolis Cedex, France  
*e-mail: Denis.Caromel@inria.fr*

Ludovic Henrio

University of Westminster  
Harrow School of Computer Science  
Watford Rd, Northwick Park  
Harrow HA1 3TP, UK  
*e-mail: Ludovic.Henrio@m4x.org*

Library of Congress Control Number: 2005923024

ISBN-10 3-540-20866-6 Springer Berlin Heidelberg New York

ISBN-13 978-3-540-20866-2 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable for prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media  
[springeronline.com](http://springeronline.com)

© Springer-Verlag Berlin Heidelberg 2005  
Printed in Germany

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Typeset by the authors using a Springer  $\text{\TeX}$  macro package  
Production: LE- $\text{\TeX}$  Jelonek, Schmidt & Vöckler GbR, Leipzig  
Cover design: KünnelLopka, Heidelberg

Printed on acid-free paper 45/3142/YL - 5 4 3 2 1 0

*To Isa, Ugo, Tom,*

*Taken from us by the Tsunami, Sri Lanka, December 26, 2004*

*Isabelle, my wife, my lover, my fellow intellect, I miss you so badly.  
My soul, my body, my brain, all hurt for you, all cry out for you.  
Your smile, your spirit would bring joy and light to all around you.  
Your plans were to do voluntary work to help humanity,  
I know you would have given courage and cheer to so many.*

*Ugo, my 8 year old boy, you could not wait to understand the world.  
You even found your own definition of infinity: God!  
I will remember forever when you would call me "Papaa ? ..."  
with that special tone, to announce a tough question.*

*Tom, my 5 year old boy, you could fight so hard and yet be so sweet.  
You were so strong, and you could be so gentle.  
Your determination was impressive, but clearly becoming thoughtful.  
I will remember forever when after a fight,  
you would jump up on my lap and give me a sweet, loving hug.*

*So many years of happiness and joy,  
May your spirits be with us and in me forever*

*Denis,  
Nice hospital,  
January 6, 2005*

*To Françoise, Marc, Laurianne and Sébastien,  
and my precious friends*

*Ludovic,  
December 10, 2004*

---

## Preface

With the advent of wide-area networks such as the Internet, distributed computing has to expand from its origins in shared-memory computing and local-area networks to a wider context. A large part of the additional complexity is due to the need to manage asynchrony, which is an unavoidable aspect of high-latency networks. Harnessing asynchronous communications is still an open area of research.

This monograph studies a natural programming model for distributed object-oriented programming. In this model, objects make asynchronous method invocations to other objects, and then concurrently carry on until the results of the requests are needed. Only at that point may they have to wait for the results to be completely computed; this delayed wait is called wait-by-necessity. Aspects of such a model have been proposed and formalized in the past: futures have been built into early concurrent languages, and various distributed object calculi have been investigated. However, this is the first time the two features, futures and distributed objects, have been studied formally together.

The result is a natural and disciplined programming model for asynchronous computing, one worthy of study. For example, it is important to understand under which conditions asynchronous execution produces predictable outcomes, without the usual combinatorial explosion of concurrent execution. Even the simplest sequential program becomes highly concurrent under wait-by-necessity execution, and yet such concurrency does not always imply that multiple outcomes are possible. One of the main technical contributions of the monograph, beyond the formalization of the programming model, is a sufficient condition for deterministic evaluation (confluence) of programs.

This monograph addresses problems that have been long identified as fundamental stumbling blocks in writing correct distributed programs. It constitutes a significant step forward, particularly in the area of formalizing and generalizing some of the best ideas proposed so far, coming up with new techniques, and providing a solid foundation for further study. The techniques studied here also have a very practical potential.

Cambridge, 2004-11-15  
Luca Cardelli

---

# Contents

<b>Preface by Luca Cardelli</b> . . . . .	<b>VII</b>
<b>Table of Contents</b> . . . . .	<b>IX</b>
<b>Lists of Figures, Tables, Definitions and Properties</b> . . . .	<b>XV</b>
<b>Prologue</b> . . . . .	<b>XXV</b>
<b>Reading Path and Teaching</b> . . . . .	<b>XXIX</b>

---

## Part I Review

---

<b>1 Analysis</b> . . . . .	<b>3</b>
1.1 A Few Definitions . . . . .	3
1.2 Distribution, Parallelism, Concurrency . . . . .	5
1.2.1 Parallel Activities . . . . .	5
1.2.2 Sharing . . . . .	6
1.2.3 Communication . . . . .	6
1.2.4 Synchronization . . . . .	10
1.2.5 Reactive vs. Proactive vs. Synchronous . . . . .	11
1.3 Objects . . . . .	14
1.3.1 Object vs. Remote Reference and Communication . .	14
1.3.2 Object vs. Parallel Activity . . . . .	14
1.3.3 Object vs. Synchronization . . . . .	15
1.4 Summary and Orientation . . . . .	17
<b>2 Formalisms and Distributed Calculi</b> . . . . .	<b>21</b>
2.1 Basic Formalisms . . . . .	21
2.1.1 Functional Programming and Parallel Evaluation . .	21
2.1.2 Actors . . . . .	23

2.1.3	$\pi$ -calculus . . . . .	26
2.1.4	Process Networks . . . . .	30
2.1.5	$\varsigma$ -calculus . . . . .	31
2.2	Concurrent Calculi and Languages . . . . .	35
2.2.1	MultiLisp . . . . .	35
2.2.2	PICT . . . . .	37
2.2.3	Ambient Calculus . . . . .	40
2.2.4	Join-calculus . . . . .	42
2.2.5	Other Expressions of Concurrency . . . . .	43
2.3	Concurrent Object Calculi and Languages . . . . .	45
2.3.1	ABCL . . . . .	45
2.3.2	Obliq and Øjeblik . . . . .	49
2.3.3	The $\pi o \beta \lambda$ Language . . . . .	51
2.3.4	Gordon and Hankin Concurrent Calculus: <b>conc</b> $\varsigma$ -calculus . . . . .	54
2.4	Synthesis and Classification . . . . .	56

---

## Part II ASP Calculus

---

<b>3</b>	<b>An Imperative Sequential Calculus . . . . .</b>	<b>63</b>
3.1	Syntax . . . . .	63
3.2	Semantic Structures . . . . .	65
3.2.1	Substitution . . . . .	65
3.2.2	Store . . . . .	66
3.2.3	Configuration . . . . .	66
3.3	Reduction . . . . .	66
3.4	Properties . . . . .	68
<b>4</b>	<b>Asynchronous Sequential Processes . . . . .</b>	<b>69</b>
4.1	Principles . . . . .	69
4.2	New Syntax . . . . .	71
4.3	Informal Semantics . . . . .	71
4.3.1	Activities . . . . .	72
4.3.2	Requests . . . . .	73
4.3.3	Futures . . . . .	73
4.3.4	Serving Requests . . . . .	73
<b>5</b>	<b>A Few Examples . . . . .</b>	<b>75</b>
5.1	Binary Tree . . . . .	76
5.2	Distributed Sieve of Eratosthenes . . . . .	77
5.3	From Process Networks to ASP . . . . .	79
5.4	Example: Fibonacci Numbers . . . . .	80
5.5	A Bank Account Server . . . . .	81



---

**Part III Semantics and Properties**


---

<b>6</b>	<b>Parallel Semantics</b>	<b>87</b>
6.1	Structure of Parallel Activities	87
6.2	Parallel Reduction	89
6.2.1	More Operations on Store	89
6.2.2	Reduction Rules	91
6.3	Well-formedness	98
<b>7</b>	<b>Basic ASP Properties</b>	<b>101</b>
7.1	Notation and Hypothesis	101
7.2	Object Sharing	104
7.3	Isolation of Futures and Parameters	105
<b>8</b>	<b>Confluence Property</b>	<b>107</b>
8.1	Configuration Compatibility	107
8.2	Equivalence Modulo Future Updates	111
8.2.1	Principles	113
8.2.2	Alias Condition	114
8.2.3	Sufficient Conditions	115
8.3	Properties of Equivalence Modulo Future Updates	117
8.4	Confluence	118
<b>9</b>	<b>Determinacy</b>	<b>121</b>
9.1	Deterministic Object Networks	121
9.2	Toward a Static Approximation of DON Terms	124
9.3	Tree Topology Determinism	126
9.4	Deterministic Examples	126
9.4.1	The Binary Tree	126
9.4.2	The Fibonacci Number Example	127
9.5	Discussion: Comparing Request Service Strategies	130

---

**Part IV A Few More Features**


---

<b>10</b>	<b>More Confluent Features</b>	<b>137</b>
10.1	Delegation	137
10.2	Explicit Wait	141
10.3	Method Update	141
<b>11</b>	<b>Non-Confluent Features</b>	<b>143</b>
11.1	Testing Future Reception	143
11.2	Non-blocking Services	144
11.3	Testing Request Reception	145
11.4	Join Patterns	146
11.4.1	Translating Join Calculus Programs	146

11.4.2	Extended Join Services in ASP	147
<b>12</b>	<b>Migration</b>	<b>151</b>
12.1	Migrating Active Objects	151
12.2	Optimizing Future Updates	153
12.3	Migration and Confluence	154
<b>13</b>	<b>Groups</b>	<b>157</b>
13.1	Groups in an Object Calculus	157
13.2	Groups of Active Objects	160
13.3	Groups, Determinism, and Atomicity	162
<b>14</b>	<b>Components</b>	<b>169</b>
14.1	From Objects to Components	169
14.2	Hierarchical Components	170
14.3	Semantics	172
14.4	Deterministic Components	175
14.5	Components and Groups: Parallel Components	176
14.6	Components and Futures	178
<b>15</b>	<b>Channels and Reconfigurations</b>	<b>181</b>
15.1	Genuine ASP Channels	181
15.2	Process Network Channels in ASP	183
15.3	Internal Reconfiguration	184
15.4	Event-Based Reconfiguration	186
<hr/>		
<b>Part V Implementation Strategies</b>		
<hr/>		
<b>16</b>	<b>A Java API for ASP: ProActive</b>	<b>189</b>
16.1	Design and API	189
16.1.1	Basic API and ASP Equivalence	190
16.1.2	Mapping Active Objects to JVMs: Nodes	191
16.1.3	Basic Patterns for Using Active Objects	192
16.1.4	Migration	192
16.1.5	Group Communications	195
16.2	Examples	198
16.2.1	Parallel Binary Tree	198
16.2.2	Eratosthenes	201
16.2.3	Fibonacci	206
<b>17</b>	<b>Future Update</b>	<b>213</b>
17.1	Future Forwarding	213
17.2	Update Strategies	215
17.2.1	ASP and Generalization: Encompassing All Strategies	215
17.2.2	No Partial Replies and Requests	217

17.2.3 Forward-Based . . . . .	219
17.2.4 Message-Based . . . . .	220
17.2.5 Lazy Future Update . . . . .	222
17.3 Synthesis and Comparison of the Strategies . . . . .	223
<b>18 Loosing Rendezvous . . . . .</b>	<b>225</b>
18.1 Objectives and Principles . . . . .	225
18.2 Asynchronous Without Guarantee . . . . .	227
18.3 Asynchronous Point-to-Point FIFO Ordering . . . . .	229
18.4 Asynchronous One-to-All FIFO Ordering . . . . .	232
18.5 Conclusion . . . . .	235
<b>19 Controlling Pipelining . . . . .</b>	<b>237</b>
19.1 Unrestricted Parallelism . . . . .	238
19.2 Pure Demand Driven . . . . .	238
19.3 Controlled Pipelining . . . . .	239
<b>20 Garbage Collection . . . . .</b>	<b>241</b>
20.1 Local Garbage Collection . . . . .	241
20.2 Futures . . . . .	242
20.3 Active Objects . . . . .	242
<hr/> <b>Part VI Final Words</b> <hr/>	
<b>21 ASP Versus Other Concurrent Calculi . . . . .</b>	<b>245</b>
21.1 Basic Formalisms . . . . .	245
21.1.1 Actors . . . . .	245
21.1.2 $\pi$ -calculus and Related Calculi . . . . .	246
21.1.3 Process Networks . . . . .	248
21.1.4 $\varsigma$ -calculus . . . . .	249
21.2 Concurrent Calculi and Languages . . . . .	249
21.2.1 MultiLisp . . . . .	249
21.2.2 Ambient Calculus . . . . .	250
21.2.3 join-calculus . . . . .	250
21.3 Concurrent Object Calculi and Languages . . . . .	250
21.3.1 Obliq and Øjeblik . . . . .	250
21.3.2 The $\pi o \beta \lambda$ Language . . . . .	251
<b>22 Conclusion . . . . .</b>	<b>253</b>
22.1 Summary . . . . .	253
22.2 A Dynamic Property for Determinism . . . . .	254
22.3 ASP in Practice . . . . .	255
22.4 Stateful Active Objects vs. Immutable Futures . . . . .	256
22.5 Perspectives . . . . .	257
<b>23 Epilogue . . . . .</b>	<b>261</b>

---

**Appendices**


---

<b>A</b>	<b>Equivalence Modulo Future Updates</b>	<b>269</b>
A.1	Renaming	269
A.2	Reordering Requests ( $R_1 \equiv_R R_2$ )	269
A.3	Future Updates	270
A.3.1	Following References and Sub-terms	270
A.3.2	Equivalence Definition	273
A.4	Properties of $\equiv_F$	276
A.5	Sufficient Conditions for Equivalence	281
A.6	Equivalence Modulo Future Updates and Reduction	283
A.7	Another Formulation	288
A.8	Decidability of $\equiv_F$	290
A.9	Examples	291
<b>B</b>	<b>Confluence Proofs</b>	<b>295</b>
B.1	Context	295
B.2	Lemmas	296
B.3	Local Confluence	298
B.3.1	Local vs. Parallel Reduction	299
B.3.2	Creating an Activity	300
B.3.3	Localized Operations (SERVE, ENDSERVICE)	301
B.3.4	Concurrent Request Sending: REQUEST/REQUEST	304
B.4	Calculus with service based on activity name: <i>Serve</i> ( $\alpha$ )	305
B.5	Extension	306
	<b>References</b>	<b>309</b>
	<b>Notation</b>	<b>321</b>
	<b>Syntax of ASP Calculus</b>	<b>327</b>
	<b>Operational Semantics</b>	<b>329</b>
	<b>Overview of Properties</b>	<b>331</b>
	<b>Overview of ASP Extensions</b>	<b>333</b>
	<b>Index</b>	<b>343</b>

---

## List of Figures

1	Suggested reading paths . . . . .	XXX
2.1	Classification of calculi (informal) . . . . .	22
2.2	A binary tree in CAML . . . . .	23
2.3	A factorial actor [9] . . . . .	24
2.4	Execution of the sieve of Eratosthenes in Process Networks	31
2.5	Sieve of Eratosthenes in Process Networks [100] . . . . .	32
2.6	Sieve of Eratosthenes in $\varsigma$ -calculus [3] . . . . .	35
2.7	Binary tree in $\varsigma$ -calculus [3] . . . . .	36
2.8	A simple Fibonacci example in PICT [130] . . . . .	39
2.9	A factorial example in the core PICT language [130] . . .	39
2.10	Locks in ambients [47] . . . . .	41
2.11	Channels in ambients [47] . . . . .	41
2.12	A cell in the join-calculus [68] . . . . .	43
2.13	Bounded buffer in ABCL [161] . . . . .	46
2.14	The three communication types in ABCL . . . . .	47
2.15	Prime number sieve in Obliq . . . . .	50
2.16	Binary tree in (a language inspired by) $\pi o\beta\lambda$ [110] . . .	53
2.17	$\pi o\beta\lambda$ parallel binary tree, equivalent to Fig. 2.16 [110] .	54
4.1	Objects and activities topology . . . . .	70
4.2	Example of a parallel configuration . . . . .	72
5.1	Example: a binary tree . . . . .	76
5.2	Topology and communications in the parallel binary tree	77
5.3	Example: sieve of Eratosthenes (pull) . . . . .	78
5.4	Topology of sieve of Eratosthenes (pull) . . . . .	78
5.5	Example: sieve of Eratosthenes (push) . . . . .	79
5.6	Topology of sieve of Eratosthenes (push) . . . . .	79
5.7	Process Network vs. object network . . . . .	79
5.8	Fibonacci number processes . . . . .	80

5.9	Example: Fibonacci numbers . . . . .	80
5.10	Topology of a bank application . . . . .	81
5.11	Example: bank account server . . . . .	82
6.1	Example of a deep copy: $copy(\iota, \sigma_\alpha)$ . . . . .	91
6.2	NEWACT rule . . . . .	93
6.3	A simple forwarder . . . . .	94
6.4	REQUEST rule . . . . .	94
6.5	SERVE rule . . . . .	95
6.6	ENDSERVICE rule . . . . .	96
6.7	REPLY rule . . . . .	97
6.8	Another example of configuration . . . . .	98
7.1	An informal property diagram . . . . .	102
7.2	Absence of sharing . . . . .	104
7.3	Store partitioning: future value, active store, request parameter . . . . .	105
8.1	Example of RSL . . . . .	109
8.2	Example of RSL compatibility . . . . .	112
8.3	Two equivalent configurations modulo future updates . . . . .	113
8.4	An example illustrating the alias condition . . . . .	116
8.5	Updates in a cycle of futures . . . . .	117
8.6	Confluence . . . . .	118
8.7	Confluence without cycle of futures . . . . .	119
9.1	A non-DON term . . . . .	124
9.2	Concurrent replies in the binary tree case . . . . .	127
9.3	Fibonacci number RSLs . . . . .	128
10.1	Explicit delegation in ASP . . . . .	140
10.2	Implicit delegation in ASP . . . . .	141
11.1	A join-calculus cell in ASP . . . . .	147
12.1	Chain of method calls and chain of corresponding futures . . . . .	153
13.1	A group of passive objects . . . . .	158
13.2	Request sending to a group of active objects . . . . .	161
13.3	An activated group of objects . . . . .	161
13.4	A confluent program if communications are atomic . . . . .	164
13.5	Execution with atomic group communications . . . . .	165
13.6	An execution without atomic group communications . . . . .	166
14.1	A primitive component . . . . .	170
14.2	Fibonacci as a composite component . . . . .	171

14.3	A definition of Fibonacci components . . . . .	172
14.4	Deployment of a composite component . . . . .	174
14.5	A parallel component using groups . . . . .	177
14.6	Components and futures . . . . .	178
15.1	Requests on separate channels do not interfere . . . . .	182
15.2	A non-deterministic merge . . . . .	183
15.3	A channel specified with an active object . . . . .	184
16.1	A simple mobile agent in ProActive . . . . .	193
16.2	Method call on group . . . . .	196
16.3	Dynamic typed group of active objects . . . . .	197
16.4	Sequential binary tree in Java . . . . .	199
16.5	Subclassing binary tree for a parallel version . . . . .	200
16.6	Main binary tree program in ProActive . . . . .	200
16.7	Execution of the parallel binary tree program of Fig. 16.6	201
16.8	Screenshot of the binary tree at execution . . . . .	202
16.9	Sequential Eratosthenes in Java . . . . .	203
16.10	Sequential <b>Prime</b> Java class . . . . .	203
16.11	Parallel Eratosthenes in Java ProActive . . . . .	204
16.12	Parallel <b>ActivePrime</b> class . . . . .	205
16.13	Graph of active objects in the Fibonacci program . . . . .	206
16.14	Main Fibonacci program in ProActive . . . . .	207
16.15	The class <b>Add</b> of the Fibonacci program . . . . .	208
16.16	The class <b>Cons1</b> of the Fibonacci program . . . . .	209
16.17	The class <b>Cons2</b> of the Fibonacci program . . . . .	210
16.18	Graphical visualization of the Fibonacci program using IC2D	211
17.1	A future flow example. . . . .	215
17.2	General strategy: any future update can occur . . . . .	217
17.3	No partial replies and requests . . . . .	218
17.4	Future updates for the forward-based strategy . . . . .	220
17.5	Message-based strategy: future received and update messages	222
17.6	Lazy future update: only needed futures are updated . . .	223
17.7	Future update strategies . . . . .	223
18.1	Example: activities synchronized by rendezvous . . . . .	226
18.2	ASP with rendezvous – message ordering . . . . .	227
18.3	Asynchronous communications without guarantee . . . . .	229
18.4	Asynchronous point-to-point FIFO communications . . .	231
18.5	Asynchronous one-to-all FIFO communications . . . . .	234
19.1	Strategies for controlling parallelism . . . . .	237
23.1	Potential queues, buffering, pipelining, and strategies in ASP	263
23.2	Classification of strategies for sending requests . . . . .	264

## XVIII List of Figures

23.3	Classification of strategies for future update . . . . .	265
A.1	Simple example of future equivalence . . . . .	271
A.2	The principle of the alias conditions . . . . .	275
A.3	Simple example of future equivalence . . . . .	291
A.4	Example of a “cyclic” proof . . . . .	292
A.5	Equivalence in the case of a cycle of futures . . . . .	292
A.6	Example of alias condition . . . . .	293
B.1	SERVE/REQUEST . . . . .	302
B.2	ENDSERVICE/REQUEST . . . . .	303
B.3	The diamond property (Property B.12) proof . . . . .	307
2	Diagram of properties . . . . .	331



---

## List of Tables

1.1	Aspects of distribution, parallelism, and concurrency . . .	17
1.2	Aspects of ASP . . . . .	19
2.1	The syntax of an Actors language [9] . . . . .	24
2.2	Aspects of Actors . . . . .	25
2.3	The syntax of $\pi$ -calculus . . . . .	26
2.4	$\pi$ -calculus structural congruence . . . . .	27
2.5	$\pi$ -calculus reaction rules . . . . .	28
2.6	Aspects of $\pi$ -calculus . . . . .	30
2.7	Aspects of Process Networks . . . . .	31
2.8	The syntax of <b>imp</b> $\varsigma$ -calculus [3] . . . . .	33
2.9	Well-formed store . . . . .	33
2.10	Well-formed stack . . . . .	34
2.11	Semantics of <b>imp</b> $\varsigma$ -calculus (big-step, closure based) . . .	34
2.12	Aspects of MultiLisp . . . . .	37
2.13	A syntax for PICT [132] . . . . .	38
2.14	Aspects of PICT . . . . .	40
2.15	The syntax of Ambient calculus . . . . .	40
2.16	Aspects of Ambients . . . . .	42
2.17	The syntax of the join-calculus . . . . .	43
2.18	Main rules defining evaluation in the Join calculus . . . .	43
2.19	Aspects of the Join-Calculus . . . . .	44
2.20	Aspects of ABCL . . . . .	49
2.21	The syntax of Øjeblik . . . . .	50
2.22	Aspects of Obliq and Øjeblik . . . . .	52
2.23	Aspects of $\pi o \beta \lambda$ . . . . .	54
2.24	The syntax of <b>conc</b> $\varsigma$ -calculus [78] . . . . .	55
2.25	Aspects of <b>conc</b> $\varsigma$ -calculus . . . . .	56
2.26	Summary of a few calculi and languages . . . . .	58
3.1	Syntax of ASP sequential calculus . . . . .	64

3.2	Sequential reduction . . . . .	67
4.1	Syntax of ASP parallel primitives . . . . .	71
4.2	Syntax of ASP calculus . . . . .	71
6.1	Deep copy . . . . .	90
6.2	Parallel reduction (used or modified values are non-gray) .	92
10.1	Rules for delegation (DELEGATE) . . . . .	138
13.1	Reduction rules for groups . . . . .	159
13.2	Atomic reduction rules for groups . . . . .	163
16.1	Relations between ASP constructors and ProActive API .	190
16.2	Migration primitives in ProActive . . . . .	193
17.1	Generalized future update . . . . .	216
17.2	No partial replies and requests protocol . . . . .	218
17.3	Forward-based protocol . . . . .	219
17.4	Message-based protocol for future update . . . . .	221
22.1	Duality active objects (stateful) and futures (immutable)	257
A.1	Reordering requests . . . . .	270
A.2	Path definition . . . . .	272
A.3	Equivalence rules . . . . .	289
1	Sequential reduction . . . . .	329
2	Deep copy . . . . .	330
3	Parallel reduction (used or modified values are non-gray) .	330

---

## List of Definitions and Properties

Definition	1.1	Parallelism .....	3
Definition	1.2	Concurrency .....	3
Definition	1.3	Distribution .....	4
Definition	1.4	Asynchronous systems .....	4
Definition	1.5	Future .....	11
Definition	1.6	Reactive system .....	12
Definition	1.7	Synchrony hypothesis .....	12
Definition	1.8	Wait-by-necessity .....	16
Definition	3.1	Well-formed sequential configuration .....	66
Definition	3.2	Equivalence on sequential configurations .....	66
Property	3.3	Well-formed sequential reduction .....	68
Property	3.4	Determinism .....	68
Definition	6.1	Copy and merge .....	91
Property	6.2	Copy and merge .....	91
Definition	6.3	Future list .....	99
Definition	6.4	Well-formedness .....	99
Property	6.5	Well-formed parallel reduction .....	99
Definition	7.1	Potential services .....	103
Property	7.2	Store partitioning .....	105
Definition	8.1	Request Sender List .....	108
Definition	8.2	RSL comparison $\trianglelefteq$ .....	110
Definition	8.3	RSL compatibility: $RSL(\alpha) \bowtie RSL(\beta)$ .....	110
Definition	8.4	Configuration compatibility: $P \bowtie Q$ .....	110
Definition	8.5	Cycle of futures .....	116
Definition	8.6	Parallel reduction modulo future updates .....	117
Property	8.7	Equivalence modulo future updates and reduction .....	117
Property	8.8	Equivalence and generalized parallel reduction .....	118
Definition	8.9	Confluent configurations: $P_1 \nabla P_2$ .....	118
Theorem	8.10	Confluence .....	118
Definition	9.1	DON .....	122

Property	9.2	DON and compatibility .....	122
Theorem	9.3	DON determinism .....	123
Definition	9.4	Static DON .....	125
Property	9.5	Static approximation .....	125
Theorem	9.6	SDON determinism .....	125
Theorem	9.7	Tree determinacy, TDON .....	126
Definition	10.1	Well-formedness .....	139
Definition	14.1	Primitive component .....	169
Definition	14.2	Composite component .....	170
Definition	14.3	Deterministic Primitive Component (DPC) .....	175
Definition	14.4	Deterministic Composite Component (DCC) .....	175
Property	14.5	DCC determinism .....	176
Definition	17.1	Forwarded futures .....	214
Property	17.2	Origin of futures .....	214
Property	17.3	Forwarded futures flow .....	214
Property	17.4	No forwarded futures .....	218
Property	17.5	Forward-based future update is eager .....	219
Property	17.6	Message-based strategy is eager .....	221
Definition	18.1	Triangle pattern .....	231
Definition	A.1	$a \xrightarrow{\alpha}_L b$ .....	271
Definition	A.2	$a \xrightarrow{\alpha^*}_L b$ .....	272
Lemma	A.3	$\xrightarrow{\alpha}_L$ and $\xrightarrow{\alpha^*}_L$ .....	273
Lemma	A.4	Uniqueness of path destination .....	273
Definition	A.5	Equivalence modulo future updates: $P \equiv_F Q$ .....	274
Property	A.6	Equivalence relation .....	275
Definition	A.7	Equivalence of sub-terms .....	275
Lemma	A.8	Sub-term equivalence .....	276
Property	A.9	Equivalence and compatibility .....	276
Lemma	A.10	$\equiv_F$ and store update .....	276
Lemma	A.11	$\equiv_F$ and substitution .....	279
Lemma	A.12	A characterization of deep copy .....	279
Lemma	A.13	Copy and merge .....	279
Lemma	A.14	$\equiv_F$ and store merge .....	280
Property	A.15	REPLY and $\equiv_F$ .....	281
Property	A.16	Sufficient condition for equivalence .....	283
Property	A.17	$\equiv_F$ and reduction(1) .....	283
Definition	A.18	Parallel reduction modulo future updates .....	284
Property	A.19	$\equiv_F$ and reduction(2) .....	284
Corollary	A.20	$\equiv_F$ and reduction .....	287
Definition	A.21	Equivalence modulo future updates (2) .....	288
Property	A.22	Equivalence of the two equivalence definitions .....	290
Property	A.23	Decidability .....	290
Property	B.1	Confluence .....	295
Lemma	B.2	$\mathcal{Q}$ and compatibility .....	296

Lemma	B.3	Independent stores .....	296
Lemma	B.4	Extensibility of local reduction .....	296
Lemma	B.5	<i>copy</i> and locations .....	296
Lemma	B.6	Multiple copies .....	297
Lemma	B.7	Copy and store update .....	297
Corollary	B.8	Copy and store update .....	298
Property	B.9	Diamond property .....	298
Lemma	B.10	$\equiv_F$ and $\mathcal{Q}(Q, Q')$ .....	306
Lemma	B.11	REPLY vs. other reduction .....	306
Property	B.12	Diamond property with $\equiv_F$ .....	306

---

## Prologue

Distributed objects are becoming ubiquitous. *Communicating objects* interact at various levels (application objects, Web and middleware services), and in a wide range of environments (mobile devices, local area networks, Grid, and P2P). These objects send messages, call methods on each other's interfaces, and receive requests and replies.

Why would we employ objects to act as interacting entities? An answer with a religious twist would be that *object orientation* has, so far, won the language crusade. However, a technical answer has more substance: objects are stateful abstractions. Any globally-distributed computation must rely on various levels of state, somehow acting as a cache for improved locality, leading to greater scalability and performance. In a multi-tier application server, for instance, objects representing persistent data (e.g., Entity Beans) act as a cache for data within the  $n$ -tier database.

Thus, stateful objects interact with each other. Why should they communicate with *method calls* rather than with messages traveling over channels? One answer is that this is exactly what objects are all about: distributed systems should not abandon such a critical feature for software structuring. *Remote method invocation* in industrial platforms, following 15 years of research in academia, has taken off, and appears to be a practical and effective solution. Moreover, method calls are also about safety and verification, a highly desirable feature for distributed, multi-principal, multi-domain applications. Because method calls and the interface imply the emergence of *types*, remote method invocations fall within the scope of type theories and practical verifications – including static analyses, which rely heavily on inter-procedural analysis.

With distribution spanning the world ever more widely, an intrinsic characteristic of communication is high latency, with an unbreakable barrier of 70 milliseconds for a signal to go half-way around the world at the speed of light. Large systems, with potentially thousands of interacting entities, cannot accommodate the high coupling induced by synchronous calls, because

such coupling can lead to a blocked chain of remote method calls spanning a large number of entities. An extreme case that requires non-synchronous invocation is the handling of the disconnected mode in wireless settings. In sum, high latency and low coupling call for asynchronous interactions, as in the case of distributed objects: *asynchronous method calls*. But if we want method calls to retain their full capacity, one-way calls on their own are insufficient. Asynchronous method calls with returns are needed, leading to an emerging abstraction: namely, *futures*, the expected result of a given asynchronous method call. Futures turn out to be a very effective abstraction for large distributed systems, preserving both low coupling and high structuring.

To summarize the argument, scalable distributed object systems cannot be effective without interactions based on asynchronous method calls, with respect to mastering both complexity and efficiency. While acknowledged theories have been proposed for both asynchronous message passing (e.g.,  $\pi$ -calculus) and objects (e.g.,  $\varsigma$ -calculus), no formal framework has been proposed for objects communicating solely with non-blocking method calls. This is exactly the ambition of the current book: to define a theory for distributed objects interacting with asynchronous method calls.

Starting from widely adopted object theory, the  $\varsigma$ -calculus [3], a syntactically lightweight extension is proposed to take distribution into account. Two simple primitives are proposed: *Active* and *Serve*. The former turns an object into an independent and potentially remote activity; the latter allows such an active object to execute (serve) a pending remote call. On activation, an object becomes a remotely accessible entity with its own thread of control: *an active object*. In accordance with the above reasoning, we have chosen to make method calls to active objects systematically asynchronous. Synchronization is ensured with a natural dataflow principle: *wait-by-necessity*. An active object is blocked on the invocation of a not yet available result, i.e., a strict operation on an unknown future. A further level of asynchrony and low coupling is reached with the first-class nature of futures within wait-by-necessity; they can be passed between active objects as method parameters and returned as results.

The proposed calculus is named *Asynchronous Sequential Processes* (ASP), reflecting an important property: the sequentiality of active objects. Processes denote the potentially coarse-grain nature of active objects. Such processes are usually formed with a set of standard objects under the exclusive control of a root object. The proposed theory allows us to express a fundamental condition for *confluence*, alleviating for the programmer of the unscalable need to consider the interleaving of all instructions and communications. Furthermore, a property ensures *determinism*, stating that, whatever the order of communications, whatever the order of future updates, even in the presence of cycles, some systems converge towards a determinate global state. Apart from Process Networks [99, 100, 159], now close to 40 years old, few calculi

and languages ensure determinism, and even fewer in the context of stateful distributed objects interacting with asynchronous method calls. The potential of the proposed theory is further demonstrated by the capacity to cope with more advanced issues such as mobility, groups, and components.

One objective of the proposed theory is to be a *practical* one. Implementation strategies are covered. Several chapters explore a number of solutions, adapted to various settings (high-speed local area networks with buffer saving in mind, wide area networks with latency hiding as a primary goal, etc.), but each one still preserving semantics and properties. An illustration of such practicability is available under an open source Java API and environment, ProActive [134], which implements the proposed theory using a strategy designed to hide latency in the setting of wide area networks.

The first part of this book analyzes the issues at hand, reviewing existing languages and calculi.

Parts II and III formally introduce the proposed framework, defining the main properties of confluence and determinism.

Part IV reaches a new frontier and discusses issues at the cutting edge of software engineering, namely migration, reconfiguration, and component-based systems. From the proposed framework, we suggest a path that can lead to reconfigurable components. It demonstrates how we can go from asynchronous distributed objects to asynchronous distributed components, including collective remote method invocations (group communications), while retaining determinism.

With practicality in mind, Part V analyzes implementation issues, and suggests a number of strategies. We are aware that large-scale distributed systems encounter large variations in conditions, due to both localization in space and dynamic changes over time. Thus, potentially adaptive strategies for buffering and pipelining are proposed.

Finally, after a comparative evaluation of related formalisms, Part VI concludes and suggests directions for the future.



## Acknowledgments

*We are pleased to acknowledge discussions, collaboration, and joint work with many people as a crucial inspiration and contribution to the pages herein.*

*Without Isabelle Attali – Isa, Project Leader of the OASIS team until December 26th 2004, this book would not be in your hands.*

*Bernard Serpette significantly contributed to the development of the ASP calculus and related proofs.*

*All the other senior OASIS team members, Françoise Baude and Eric Madelaine, were very supportive and contributed in many aspects.*

*This book is also the result of many fruitful interactions between theory and practice. Many inspiring ideas came from the practical development of the ProActive library, and from contributors.*

*Special thanks go to Fabrice Huet and Julien Vayssière, the first two ProActive contributors, implementors, and testers. Many others recently had key contributions, especially the younger researchers and engineers in our team: Laurent Baduel, Tomás Barros, Rabéa Ameur-Boulifa, Javier Bustos, Arnaud Contes, Alexandre Di Costanzo, Christian Delbé, Felipe Luna Del Aguila, Matthieu Morel, and last, but not least, Romain Quilici.*

*Former Master's, Ph.D. students, engineers, were also a key source of maturation and inspiration: Alexandre Bergel, Roland Bertuli, Florian Doyon, Sidi Ould Ehmety, Alexandre Fau, Wilfried Klauser, Emmanuel Léty, Lionel Mestre, Olivier Nano, Arnaud Poizat, Yves Roudier, Marjorie Russo, David Sagnol.*

*Finally, colleagues from around the world, Martín Abadi, Gul Agha, Gérard Boudol, Luca Cardelli, David Crookall, Davide Sangiorgi, Akinori Yonezawa, and Andrew Wendelborn, made very useful comments on early drafts of the book or related research papers.*

*Nice - Sophia Antipolis  
London  
February 2005*

*Denis Caromel  
Ludovic Henrio*

---

# Reading Paths and Teaching

## Extra Material and Dependencies

You will find at the end of this book a list of notations and a summary of ASP syntax and semantics that should provide a convenient quick reference (Index of Notations, Syntax, Operational Semantics). This is followed by a graphical view of ASP properties (page 331), and the syntax of ASP extensions (Synchronizations, Migration, Groups, Components).

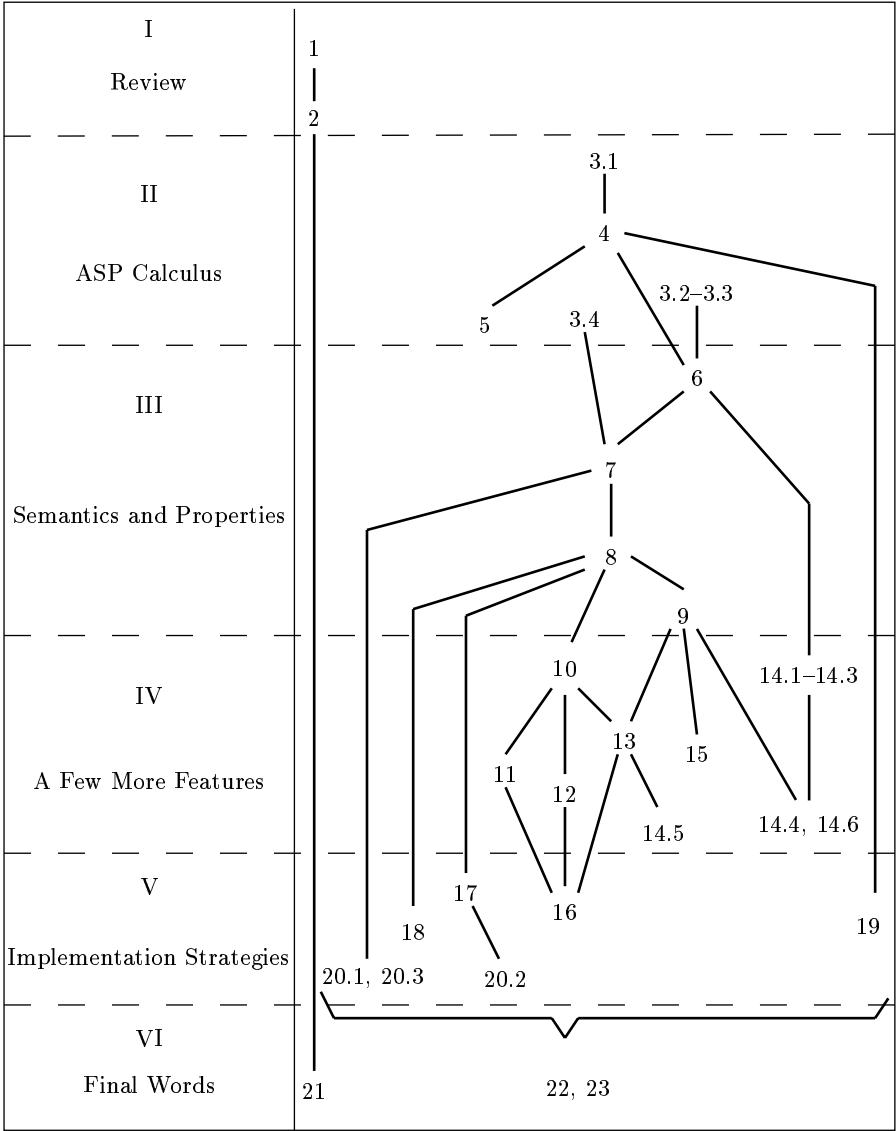
The Appendices detail formal definitions and proofs of the main theorems and properties introduced in Part III.

Figure 1 exhibits the dependencies between chapters and sections. Each chapter is best read after the preceding chapters. For example, in order to fully understand the group communication in ASP (Chap. 13), one should read Chaps 3, 4, Part III (Chaps. 6, 7, 8, 9), and Chap.10. Going down the lines (Fig. 1), one can follow the outcomes of chapters. For instance, still for group communication in Chap. 13, immediate benefits are parallel components (Sect. 14.5), and a practical implementation of typed group communication within ProActive (Chap. 16).

## Text Book

Besides researchers and middleware designers, the material here can also be used as a text book for courses related to *models, calculi, languages for concurrency, parallelism, and distribution*. The focus is clearly on recent advances, especially object-orientation and asynchronous communications. Such courses can provide theoretical foundations, together with a perspective on practical programming and software engineering issues, such as distributed components.

The courses cover classical calculi such as CSP [88] and  $\pi$ -calculus [119, 120, 144], object-orientation using  $\varsigma$ -calculus [3, 1, 2], and ASP [52], and advanced issues such as mobility, groups, and components. Overall, the objectives are threefold:



**Fig. 1.** Suggested reading paths

- (1) study and analyze existing models of concurrency and distribution,
- (2) survey their formal definitions within a few calculi,
- (3) understand the implications on programming issues.

Depending on the objectives, the courses can be aimed at more theoretical aspects, up to proofs of convergence and determinacy within  $\pi$ -calculus and ASP, or targeted at more pragmatic grounds, up to practical programming

sessions using software such as PICT [132, 131] or *ProActive* [134].

Below is a suggested outline for a semester course, with references to online material, and chapters or sections of this book:

### Models, Calculi, Languages for Concurrency, Parallelism, and Distribution

1.	Introduction to Distribution, Parallelism, Concurrency General Overview of Basic formalisms	1 [39]
2.	CCS, and/or Pi-Calculus	2.1.3 [73]
3.	Other Concurrent Calculi and Languages (Process Network, Multilist, Ambient, Join, ...)	2.1.4, 2.2 [125]
4.	Object-Oriented calculus: $\varsigma$ -calculus	2.1.5 [4]
5.	Overview of Concurrent Object Calculi (Actors, ABCL, Obliq and Øjeblik, $\pi o\beta\lambda$ , <b>conc</b> $\varsigma$ -calculus, ...)	2.1.2, 2.3 [39]
6.	Asynchronous Method Calls and Wait-by-necessity ASP: Asynchronous Sequential Processes	3, 4, 5
7.	Semantics, Confluence, Determinacy	6, 7, 8, 9
8.	Advanced issues I: Confluent and non-confluent features, mobility	10, 11, 12 [125]
9.	Advanced issues II: Groups, Components	13, 14
10.	Open issue: reconfiguration Conclusion, Perspective, Wrap-up	15, 21, 22, 23

The Web page [39] gathers a broad range of information aimed at concurrent systems, also featuring parallel and distributed aspects. Valuable material for teaching models of concurrent computation, including CCS and  $\pi$ -calculus can be found at [73]. The Web page [4] is dedicated to the book *A Theory of Objects* [3]; it references pointers to courses using  $\varsigma$ -calculus, some with teaching material available online. Finally, a comprehensive set of resources related to calculi for mobile processes is available at [125].

Assignments can include proofs of the confluence or non-confluence natures of a few features (e.g., delegation, explicit wait, method update, testing future or request reception, non-blocking services, join constructs, etc.). More practical assignments can involve designing and evaluating new future-update strategies, new request delivery protocols, or new schemes for pipelining control. Practicality can reach as far as implementing examples or prototypes, using PICT [132, 131], *ProActive* [134], or other programming frameworks.

## **A Theory of Distributed Objects online**

We intend to maintain a Web page for general information, typos, etc. Extra material is also expected to be added (slides, exercises and assignments, contributions, reference to new related papers, etc.). This page is located at:

<http://www.inria.fr/oasis/caromel/TDO>

Do not hesitate to contact us to comment or to exchange information!

## Part I

---

## Review