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

Springer

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

Springer is a part of Springer Science+Business Media
springeronline.com

Typeset by the authors using a Springer TEX macro package
Production: LE-TEX Jelonek, Schmidt & Vöckler GbR, Leipzig
Cover design: KünkelLopka, 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

Part III Semantics and Properties

Part IV A Few More Features

Appendices

List of Figures

List of Tables

List of Definitions and Properties

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., π-calculus) and objects (e.g., ς-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 ς-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 π-calculus [119, 120, 144], object-orientation using ς-calculus [3, 1, 2], and ASP [52], and advanced issues such as mobility, groups, and components. Overall, the objectives are threefold:

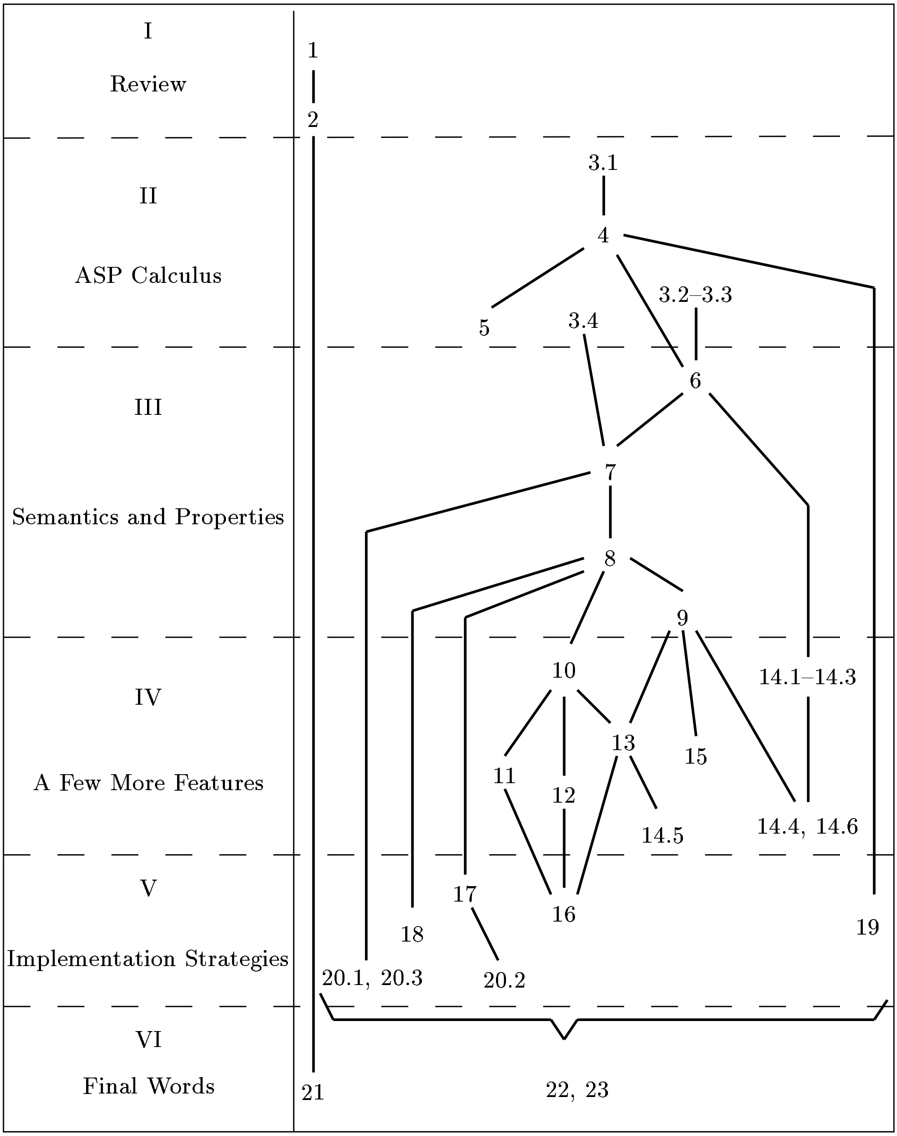

Fig. 1. Suggested reading paths

(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 π-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: ς-calculus	2.1.5 [4]
5.	Overview of Concurrent Object Calculi (Actors, ABCL, Obliq and Øjeblik, $\pi o\beta\lambda$, **conc**ς-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 π-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 ς-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