

Lecture Notes in Computer Science

1169

Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Advisory Board: W. Brauer D. Gries J. Stoer

Manfred Broy Stephan Merz
Katharina Spies (Eds.)

Formal Systems Specification

The RPC-Memory Specification
Case Study



Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany

Juris Hartmanis, Cornell University, NY, USA

Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Manfred Broy

Stephan Merz

Katharina Spies

Technische Universität München, Institut für Informatik

Arcisstr. 21, D-80290 München, Germany

E-mail: {broy/merz/spies}@informatik.tu-muenchen.de

Cataloging-in-Publication data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Formal systems specification : the RPC-memory specification case study / Manfred Broy ... (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Budapest ; Hong Kong ; London ; Milan ; Paris ; Santa Clara ; Singapore ; Tokyo : Springer, 1996

(Lecture notes in computer science ; Vol. 1169)

ISBN 3-540-61984-4

NE: Broy, Manfred [Hrsg.]; GT

CR Subject Classification (1991): C.2.4, D.2.1, D.2.4, D.2.10, F.3.1

ISSN 0302-9743

ISBN 3-540-61984-4 Springer-Verlag 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, re-use of illustrations, recitation, broadcasting, reproduction on microfilms 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-Verlag. Violations are liable for prosecution under the German Copyright Law.

© Springer-Verlag Berlin Heidelberg 1996

Printed in Germany

Typesetting: Camera-ready by author

SPIN 10549153 06/3142 - 5 4 3 2 1 0 Printed on acid-free paper

Preface

The growing interest in a scientifically based method for the specification, refinement, implementation, and verification of distributed, concurrent, interacting systems is caused by the wide areas of their applications. These include operating systems, computer networks, telecommunications, wide-area nets, such as the Internet or the World Wide Web, and all kinds of reactive systems embedded in technical devices such as in the automotive and avionics industries. Other typical applications are business applications and hardware/software codesign. New applications are emerging rapidly. As it turned out in the 1970's, providing good methods, techniques, models, and notations for system development is not a trivial task at all. Therefore the area of the formal foundation of the development of concurrent interacting systems has attracted many researchers since the stimulating work by Edsger Dijkstra, Tony Hoare, and others.

The subject of distributed interactive systems also leads to intricate theoretical problems, which have stimulating relationships to many areas of mathematics, logics, and electrical engineering. Not surprisingly, some of the work in these areas became more and more focused on isolated problems rather separated from the original motivation, namely to provide a scientifically well-founded and practically applicable development method that includes specification, refinement, and verification. A second problem is that over the years a large and confusing variety of different approaches have been developed. This richness, reflecting the large amount of creativity and new ideas of researchers, provides us with an overwhelming variety of aspects for the design and the analysis of systems. On one hand, this is good, since it gives a richer source of material on which practical techniques could be based. Unfortunately, this can be confusing for practical engineers who are interested in improving their methods by providing formal foundations. Even for the more research-oriented engineers in that area, it is not easy to keep track of all the different approaches and to understand their advantages and disadvantages.

Case studies can provide a basis for comparing, relating, and evaluating various approaches. There have been several actions in the past to provide case studies and to compare different methods on the basis of these case studies. Some of them provided a variety of different cases for the application. As a result, for every case study only a small number of approaches were applied. Therefore the comparison between a large number of methods was not supported. In another experiment, the case study was rather large, so not too many solutions were presented due to the overhead needed to work out the case study.

But even in these cases, it became clear that case studies provide a very valuable opportunity to improve our possibilities to compare and further develop our methods. First of all, they provide a real basis to evaluate approaches. And, more than the examples presented by the various methods by themselves, they really test how far a method is able to adapt to particular requirements and methodological needs. More important, there is another remarkable effect. The test cases guide the attention of the researchers in the applicability of their

methods. As a result of several case studies, researchers had to admit that in their current form their methods were actually not applicable and were forced to extend and modify their theories, notations, and techniques. This way, an interesting feedback was set up between the attempt to provide a solution to a case study and the further development of the methods. Obviously, case studies help to focus the attention of the researchers onto the critical issues in the development task.

Based on these considerations I suggested to Leslie Lamport in fall 1993 that we again do an experiment by providing a further case study. For a fair comparison, of course, case studies should be sufficiently large to meet the actual requirements of practical applications. Moreover a variety of case studies is needed since the different approaches may be better suited for different areas of applications and therefore a comparison on the basis of only one specific case study is certainly not fair. After several meetings and discussions, Leslie and I decided to provide just one case study and to choose its size to be not too large, however, to make sure that several steps of development were part of the exercise. The reason was that we were interested in looking not only at the pure specification task but also at refinement and verification.

Finally, we decided to choose an example that is typical for the area of operating systems and hardware applications, at least on the logical level of hardware architecture, and to include in this example a number of typically important issues such as failures, fairness, and real time. Leslie wrote a first version of the problem description and, after a few iterations, the problem statement was sent out together with the invitations to a workshop in Dagstuhl.

The workshop, which took place in September 1994, was very stimulating. Originally, we were afraid that after two days the participants would be completely fed up with the problem. As it turned out, this was not the case at all. On the contrary, new aspects of the problem were discovered and discussed. Striking was the fact that for many of the presented solutions, the ideas on modeling the problem and solution proved more important than the chosen methods and notations. Correspondingly, the individual methods were often just used as a representation vehicle and did not govern the basic specification, refinement, and verification ideas. This showed to a large extent that the notation and theoretical basis should not be over-emphasized and that the methods and techniques to find good ideas for modeling, representing, and refining systems should also attract enough attention.

After the workshop, we went through several iterations to collect written versions of the presented solutions. We went through two stages of a careful cross-refereeing process and finally came to the text presented here.

After many discussions on the issue of providing a careful comparison and evaluation of the written solutions we very modestly decided to include only a short synopsis paper in this volume in which the approaches were roughly explained and related. Clearly much more work can and needs to be done here. However, being involved in some of the solutions ourselves, it would not have been fair for us to draw too far-reaching conclusions. So we leave it to the reader

VII

and to future activities to provide a more careful analysis and comparison of the results of our experiment.

Looking back at the whole activity, I consider this experiment a success and a helpful contribution to the research community as well as to the engineering community for those interested in experimenting with formally based approaches to the design of systems.

It is a pleasure to thank the people who helped put this experiment forward. My thanks go to Leslie Lamport for an effective and stimulating cooperation, to Schloss Dagstuhl for providing an excellent place for the workshop, to Stephan Merz who helped a lot in organizing the workshop and carefully read the contributions in the refereeing process, and to Katharina Spies who helped in many ways in correspondence with the authors and who showed an admirable patience with all the little problems and difficulties that arose in the refereeing process. Finally my thanks go to Springer-Verlag who helped in a very gentle and efficient way to publish this experiment as a volume in the Lecture Notes in Computer Science.

Munich, July 1996

Manfred Broy

Acknowledgement

There is no book without authors. This is especially true for this collection of articles since we misused authors as reviewers for the contributions. We did this move since we considered the authors of articles as the best experts of the specified problem. Moreover, we were interested in an interaction between the proponents of the quite different methods. For all these reasons, we decided to establish a cross-refereeing process.

It is our pleasure to thank all the reviewers

M. Abadi,
E. Astesiano,
D. Barnard,
E. Best,
J. Cuéllar,
R. Gotzhein,
J. Hooman,
H. Hungar,
B. Jonsson,
J.N. Kok,
R. Kurki-Suonio,
L. Lamport,
K. Larsen,
S. Merz,
O. Müller,
M. Nielsen,
G. Reggio,
M. Rinderspacher,
B. Steffen,
K. Stølen,
K. Sunesen,
C. Weise

for their prompt and precise work in the last few months. Only because of their readiness for fruitful and mostly peaceful discussions and cooperation with the authors we can present contributions of selected quality in this collection. We are especially grateful for the cross-refereeing efforts to prepare assessments of the articles which were very helpful in the preparation of the synopsis you find in this volume.

Table of Contents

Manfred Broy, Leslie Lamport:

The RPC-Memory Specification Problem – Problem Statement	1
1 The Procedure Interface	1
2 A Memory Component	1
3 Implementing the Memory	2
3.1 The RPC Component	2
3.2 The Implementation	3
4 Implementing the RPC Component	3
4.1 A Lossy RPC	3
4.2 The RPC Implementation	4

Manfred Broy, Stephan Merz, Katharina Spies:

The RPC-Memory Case Study: A Synopsis	5
1 About this Book	5
2 The Problem	6
3 The Solutions	8
3.1 Criteria for Classification	8
3.2 Reviews of Individual Contributions	11
4 Conclusion	19

Martín Abadi, Leslie Lamport, Stephan Merz:

A TLA Solution to the RPC-Memory Specification Problem	21
Introduction	21
1 The Procedure Interface	22
1.1 The Module and its Parameters	23
1.2 State Functions, State Predicates, and Actions	24
1.3 Temporal Formulas	25
2 A Memory Component	27
2.1 The Parameters	27
2.2 The Memory Specification	28
2.3 Solution to Problem 1	34
3 Implementing the Memory	36
3.1 The RPC Component	36
The Parameters Module	37
Problem 2: The RPC Component's Specification	39
3.2 The Implementation	41

The Memory Clerk	41
The Implementation Proof	42
4 Implementing the RPC Component	52
4.1 A Lossy RPC	53
4.2 The RPC Implementation	54
The RPC Clerk	54
The Implementation Proof	56
References	63
Index	65

Egidio Astesiano, Gianna Reggio:

A Dynamic Specification of the RPC-Memory Problem	67
1 Introduction	67
2 Dynamic Specifications (DSPECs)	69
2.1 The formal model for dynamic systems	69
2.2 Dynamic structures	70
2.3 A logic for DSPECs	71
2.4 Dynamic specifications	74
3 A Development Process for Dynamic Systems	75
3.1 Development process	75
3.2 How to write a DSPEC	76
Simple dynamic system	77
Concurrent dynamic system	78
3.3 Specification presentation	79
4 MC Requirement Specification (PHASE 1)	79
Natural description	79
Border determination	79
Basic data structures	79
Interactions	81
States	81
Activity	82
Shadow spots	84
5 The Reusable Specification of RPC	85
The acceptable actual parameters	85
Interactions	86
States	87
Activity	87
6 MC Development (First Step of PHASE 2)	89
Natural description	89
Basic data structures	90
Components of MC	93

The reliable memory component RMC	93
The remote procedure caller RPC	93
The clerk CLERK	94
Interactions	94
States	95
Activity	95
Interactions (of MC)	98
States (of MC)	98
Activity (of MC)	99
Correctness justification	102
7 Discussing our Solution	104

Eike Best:

A Memory Module Specification Using Composable High-Level Petri Nets	109
1 Introduction	109
2 The M-net Model	110
2.1 Definition of M-nets	110
2.2 Transition rule and unfolding	112
2.3 Composition operations	114
3 Behavioural Notions	117
3.1 Occurrence sequences and permutations	117
3.2 Collapsing transitions	119
3.3 Implementation relation	120
4 Memory Specification	123
4.1 Notation	123
4.2 Reliable and unreliable memory	124
5 Memory Implementation	130
5.1 The <i>RPC</i> module	131
5.2 The implementation	133
5.3 Main result	135
5.4 Progress and fairness	137
6 A Lossy Remote Procedure Call Component	139
6.1 Time M-nets	140
6.2 Lossy <i>RPC</i> module	140
7 Discussion	143
7.1 In retrospect	143
7.2 Answers to [6]	145
7.3 Safety and liveness specifications	146
A Proofs	148

Johan Blom, Bengt Jonsson:

Constraint Oriented Temporal Logic Specification	161
1 Introduction	161
2 Temporal Logic	163
2.1 Syntax and Informal Semantics	163
3 Semantics	164
4 Using LTL to Specify Concurrent Systems	165
4.1 Specification of State Changes	165
4.2 State transition diagrams	166
4.3 Specifying Components	167
4.4 Combining components	168
5 Overall Structure of the Specification	168
6 The Memory Cells	169
7 The Memory Interface	171
7.1 Unreliable Memory Interface	171
7.2 Reliable Memory Interface	174
7.3 Solution to Problem 1	174
8 Problem 2 – The RPC Component	176
8.1 Solution to Problem 2	177
9 Problem 3 – The Clerk Component	177
9.1 Solution to Problem 3	180

Manfred Broy:

A Functional Solution to the RPC-Memory Specification Problem	183
1 Introduction	183
2 The Basic System Model	184
3 Basic Message Sets Involved	186
4 The Unreliable and the Reliable Memory Component	187
5 The Remote Procedure Call Component	191
6 Implementation with the Help of a Clerk	193
7 The Composed System	194
8 Specification of the Lossy RPC Component	195
9 A Clerk for the Lossy RPC Component	196
10 Composing the Clerk and the Lossy RPC Component	197
11 Conclusion	197

A Mathematical Basis	198
B State Transition Specification of the Memory Component . .	200
C A Trace Solution to the Memory Specification Problem . . .	202
D Proof of the Verification Conditions	204

Jorge Cuéllar, Dieter Barnard, Martin Huber:

A Solution Relying on the Model Checking of Boolean Transition Systems	213
0 Introduction	213
1 The Procedure Interface	215
1.1 Data types	215
1.2 Interfaces in TLT	216
2 A Memory Component	220
2.1 The Memory Specification	222
2.2 The Server Specification	223
2.3 Result and UserConnector Modules	225
2.4 Conjunction of Modules	226
2.5 Solution to Problem 1(a)(i):	228
2.6 The Reliable Server Specification	228
2.7 The Refinement of Server	229
3 A RPC Component	229
3.1 The RPC Specification	230
3.2 The RPC Implementation	232
3.3 The Refinement of Server	234
4 A Lossy RPC Component	235
4.1 Timed Instructions	235
4.2 The Lossy RPC Specification	237
4.3 The Supervising Clerk	240
4.4 Proof of the Real-time Property in Problem 5(b)	241
5 Conclusion	242
A An Introduction to TLT	244
A.1 Concrete States and Transitions	244
A.2 Boolean Algebras	244
A.3 Boolean Transition Systems	245
A.4 Conjunction of Boolean Transition Systems	245
A.5 Parameterization of Boolean Transition Systems	247
A.6 Translating the Program Notation	250

Reinhard Gotzhein:

Applying a Temporal Logic to the RPC-Memory Specification Problem	253
1 Introduction	253
2 The Formalism	254
2.1 Elementary concepts	254
2.2 System architecture	255
2.3 System behaviour	256
2.4 System	262
2.5 System refinement	262
3 The Memory Component	263
3.1 A temporal logic	264
3.2 Specification of the unreliable memory component	265
3.3 Refinements of the unreliable memory component	269
4 Are Architectural Requirements Important?	271
5 Conclusion	272

Jozef Hooman:

Using PVS for an Assertional Verification of the RPC-Memory Specification Problem	275
0 Introduction	275
0.1 Formal Background	276
0.2 Basic PVS Framework	277
1 The Procedure Interface	282
2 A Memory Component	284
2.1 Basic Properties of Memory	284
2.2 Inner Memory	285
Equivalent Formulation of the Inner Memory	286
2.3 Memory	288
2.4 Simplifying the Memory Specification	290
2.5 Reliable Memory	292
Correctness of Reliable Memory	293
2.6 Failing Memory	293
Correctness of Failing Memory	293
3 Implementing the Memory	294
3.1 The RPC Component	294
3.2 The Implementation	296
Correctness of the Implementation	298
4 Implementing the RPC Component	299
4.1 A Lossy RPC	299

4.2 The RPC Implementation	300
Correctness of the RPC Implementation	302
5 Concluding Remarks	303

Hardi Hungar:

Specification and Verification Using a Visual Formalism on Top of Temporal Logic	305
1 Specifics of this Solution	305
1.1 Methodology	305
1.2 Problem Coverage	306
2 The Implementation Language	308
2.1 Processes and Programs	308
2.2 The Semantics of Programs	309
3 The Specification Language	312
3.1 The Logic	312
3.2 Symbolic Timing Diagrams	313
4 Specification of the Memory Component	316
4.1 The Protocol of Calls and Replies	316
4.2 Specification of the Functional Behavior	318
4.3 Reliability	321
5 Implementing the Memory	322
5.1 Specification and Verification of the Protocol	322
5.2 Specification and Verification of the Functional Behavior	324
6 Verifying an Implementation	328
6.1 Practical Experiments	335
7 Observations	336
7.1 Timing Diagrams Versus Temporal Logic	336
7.2 Adequateness of Temporal Logic	336
8 Conclusion	337

Nils Klarlund, Mogens Nielsen, Kim Sunesen:

A Case Study in Verification Based on Trace Abstractions . . .	341
1 Introduction	341
2 Monadic Second-Order Logic on Strings	345
2.1 FIDO	347
2.2 Automated translation and validity checking	349
3 Systems	350
3.1 Composition	351
3.2 Implementation	353

4 Relational Trace Abstractions	354
4.1 Decomposition	355
5 The RPC-Memory Specification Problem	356
5.1 The procedure interface	356
5.2 A memory component	357
Problem 1	362
6 Implementing the Memory	362
6.1 The RPC component	363
Problem 2	367
6.2 The implementation	367
Problem 3	369
7 Verifying the Implementation	369

Reino Kurki-Suonio:

Incremental Specification with Joint Actions:

The RPC-Memory Specification Problem	375
---	-----

1 Introduction	375
2 Foundations in TLA	376
2.1 Correspondence to Canonical TLA Expressions	376
2.2 Objects and Actions	377
2.3 Refinement by Superposition	378
2.4 Combination of Specifications	379
2.5 Modeling vs. Specification	380
3 Components and Processes	381
3.1 Component and Process Classes	381
3.2 Computational Steps by Processes	381
4 Procedure Calls	382
4.1 Interface Processes	382
4.2 Calls and Returns	383
4.3 Notes on the Formalism	386
4.4 Safety Properties	387
4.5 Reliable Procedure Calls	387
4.6 Call Structures	388
5 Memory Components	388
5.1 Memory Class	388
5.2 Memory Actions	389
5.3 Liveness Properties	390
5.4 Reliable Memory	390
5.5 Simplified Memory	391
6 The RPC Component	391
6.1 RPC Component Class	391

6.2 RPC Actions	392
7 Implementing the Memory	393
7.1 RPC-Memory Classes	393
7.2 RPC-Memory Actions	393
7.3 Correctness of the Implementation	394
8 Lossy RPC Component	397
8.1 Superposing Real-Time Properties	397
8.2 Zeno Behaviors	397
8.3 Classes for Lossy RPC	398
8.4 Actions for Lossy RPC	398
9 RPC Implementation	399
9.1 Classes for RPC Implementation	399
9.2 Actions for RPC Implementation	400
9.3 Receivers with Bounded Response Times	401
9.4 Correctness of the Implementation	401
10 Concluding Remarks	402
 <i>Kim G. Larsen, Bernhard Steffen, Carsten Weise:</i>	
The Methodology of Modal Constraints	405
1 Introduction	405
2 Modal Transition Systems	407
3 Actions, Design Patterns and Projective Views	410
3.1 Abstract Actions	410
3.2 Projective Views	411
3.3 Generalizations	413
3.4 Sufficient Proof Condition	414
4 Specification of the Untimed Problem	415
4.1 Specification of the Memory Component	415
4.2 Specification of the RPC Mechanism	420
4.3 The Clerk	423
5 Applying the Method	425
5.1 Formal Specification of the Implementation	425
5.2 Proof of Correctness	426
Step 1: Application of the Sufficient Proof Condition	426
Step 2: Skolemization	427
Step 3(a): Abstraction by Refinement	427
Step 3(b): Abstraction by Factorization	427
6 Timed Modal Specification	429
7 Specification of the Timed Problem	430

8 Applying the Method	431
9 Conclusion and Future Work	433

Judi Romijn:

Tackling the RPC-Memory Specification Problem with I/O Automata	437
1 Introduction	437
1.1 Notes on the problem specification	438
Ambiguities	438
Observable versus internal behaviour	438
Fairness and real time	438
1.2 Notes on the I/O automata model	438
Benefits	438
Imperfections	439
What we added to the classic model	439
1.3 Further remarks	440
Acknowledgements	440
2 Preliminaries	440
2.1 Fair I/O automata	440
2.2 Details on fair I/O automata	440
Specification	440
Presentation	441
Proofs	441
3 Specifications and Verifications for Problem 1	441
3.1 Problem 1(a): Specification of two Memory Components	441
Data types	442
The Memory component	442
Liveness	443
The Reliable Memory component	445
Liveness	445
3.2 Problem 1(b): <i>RelMemory</i> implements <i>Memory</i>	445
Safety	445
Deadlock freeness	446
Implementation	446
3.3 Problem 1(c): Nothing but <i>MEM_FAILURE_P</i> actions?	446
4 Specifications and Verifications for Problem 2	447
4.1 Problem 2: Specification of the RPC component	447
Data types	447
Specification	447
Liveness	448
5 Specifications and Verifications for Problem 3	448
5.1 Problem 3: Specification of the composition	448

Data types	448
A front end for the RPC component	449
Liveness	449
Renaming component <i>RelMemory</i>	449
Liveness	451
The implementation	451
Liveness	451
5.2 Set-up for the verification	453
An intermediate automaton	453
Liveness	453
5.3 Problem 3: <i>MemoryImp</i> implements <i>Memory</i>	453
<i>Memory*</i> implements <i>Memory</i>	453
<i>MemoryImp</i> implements <i>Memory*</i>	456
Invariants	456
Safety	458
Deadlock freeness	459
The main result	460
6 Specifications for Problem 4	460
6.1 Problem 4: Specification of a lossy RPC	460
Data types	461
7 Specifications and Verifications for Problem 5	461
7.1 Problem 5(a): The RPC implementation <i>RPCImp</i>	461
Data types	461
Specification	461
The composition	463
7.2 Problem 5(b): <i>RPCImp</i> implements <i>RPC</i>	463
A new implementation and specification	465
Admissible trace inclusion	466
Weak refinement	467
Fairness is preserved	468
A Safe and Fair I/O Automata	471
A.1 Safe I/O automata	471
Enabling of actions	471
Executions	471
Traces	471
A.2 Fair I/O automata	472
Enabling of sets	472
Fair executions	472
Fair traces	472
Implementation relation	472
Fairness as a liveness condition	472
B Timed and Fair Timed I/O Automata	473
B.1 Timed I/O automata	473
Timed Traces	473

B.2 Fair timed I/O automata	474
Enabling of sets	474
Fair executions	475
Fair timed traces	475
Implementation relation	475

Ketil Stølen:

Using Relations on Streams to Solve the RPC-Memory Specification Problem	477
1 Introduction	477
2 Streams and Operators on Streams	478
3 Problem I: The Memory Component	479
3.1 Basic Definitions	480
3.2 The Sequential Memory Component	481
3.3 The Concurrent Memory Component	482
3.4 The Repetitive Memory Component	483
3.5 Timed Streams and Operators on Timed Streams	485
3.6 The Reliable Memory Component	486
3.7 The Unreliable Memory Component	488
3.8 Implementation	490
4 Problem II: The RPC Component	490
4.1 Basic Definitions	490
4.2 The Sequential RPC Component	491
4.3 The RPC Component	494
5 Problem III: Implementing the Unreliable Memory Component	495
5.1 Basic Assumptions	495
5.2 The Sequential Clerk Component	496
5.3 The Clerk Component	497
5.4 The Implementation	497
5.5 Verification	498
6 Problem IV: The Lossy RPC Component	506
6.1 The Sequential Lossy RPC Component	506
7 Problem V: Implementing the RPC Component	508
7.1 The Sequential RPC Clerk Component	508
7.2 The Implementation	510
7.3 Verification	510
8 Conclusions	514
9 Acknowledgements	515
A Specifying the Hand-Shake Protocol	515

Rob T. Uddink, Joost N. Kok:

The RPC-Memory Specification Problem: UNITY + Refinement Calculus	521
1 Introduction	521
2 The ImpUNITY Programming Language	522
3 The ImpUNITY Logic	525
4 Program Refinement	527
5 The RPC-Memory Specification Problem	529
5.1 Implementing the RPC Component	533
6 Refinement Steps for The Memory Component	533
7 Conclusions	539
 List of Authors	 541