CAMELOT AND AVALON

A Distributed Transaction Facility

Edited by
Jeffrey L. Eppinger
Lily B. Mummert
Alfred Z. Spector

CAMELOT AND AVALON

A Distributed Transaction Facility

Edited by

Jeffrey L. Eppinger
Transarc Corporation

Lily B. Mummert
Carnegie Mellon University

Alfred Z. Spector
Transarc Corporation

MORGAN KAUFMANN PUBLISHERS, INC.
SAN MATEO, CA

Sponsoring Editor Bruce Spatz
Production Editor Sharon Montooth
Cover Designer Victoria Ann Philp

Border of the cover design taken from Simon Vostre's *Heures à l'Usage de Rome* of 1948, a reproduction of which was printed by O. Jouaust and published by L. Gauthier, 1890, France. Reproduced with permission of the publishers from PRINTING TYPES: THEIR HISTORY, FORMS, AND USE: A STUDY IN SURVIVALS, Volume II, Second Edition, by Daniel Berkeley Updike, Copyright © 1922 and 1937 by Harvard University Press, Cambridge, Mass.

Library of Congress Cataloging-in-Publication Data

Camelot and Avalon : a distributed transaction facility / edited by Jeffrey L. Eppinger, Lily B. Mummert, Alfred Z. Spector.
p. cm. -- (The Morgan Kaufmann series in data management systems, ISSN 1046-1698)
Includes bibliographical references and index.
ISBN 1-55860-185-6
1. Distributed data bases. 2. Avalon. 3. Camelot (Computer file)
I. Eppinger, Jeffrey L. II. Mummert, Lily B. III. Spector, Alfred Z. IV. Series.
QA76.9.D3C363 1991
005.75'8--dc20 90-24944
CIP

Morgan Kaufmann Publishers, Inc.
2929 Campus Drive, Suite 260
San Mateo, CA 94403

95 94 93 92 91 5 4 3 2 1 **RRD**

This research was sponsored by IBM and by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976 (Amendment 20), under contract F33615-87-C-1499 monitored by the Avionics Laboratory, Wright Air Force Aeronautical Laboratories, Wright-Patterson Air Force Base. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any of the sponsoring agencies or of the United States Government.

CAMELOT AND AVALON

A Distributed Transaction Facility

THE MORGAN KAUFMANN SERIES IN DATA MANAGEMENT SYSTEMS

Series Editor, Jim Gray

Camelot and Avalon: A Distributed Transaction Facility
Edited by Jeffrey L. Eppinger (Transarc Corporation), Lily B. Mummert (Carnegie Mellon University), and Alfred Z. Spector (Transarc Corporation)

Database Modeling and Design: The Entity-Relationship Approach
Toby J. Teorey (University of Michigan)

Readings in Object-Oriented Database Systems
Edited by Stanley B. Zdonik (Brown University) and David Maier (Oregon Graduate Center)

Readings in Database Systems
Edited by Michael Stonebraker (University of California, Berkeley)

Deductive Databases and Logic Programming
Jack Minker (University of Maryland)

Foreword

Camelot and Avalon are landmark systems – they show how the transaction concept can be layered atop operating system kernels and how transaction semantics can be integrated with conventional programming languages. The result is a transactional execution environment for all applications. With this design, programming languages can easily provide persistent data types; implementors of new types can make the types transactional; and implementors of applications can structure the applications as collections of transactions. It is now widely accepted that transactions are the key to constructing reliable distributed objects and computations. In the past, transactions were used almost exclusively in commercial database applications. Such applications use transactions to get the *ACID* execution properties:

- Atomicity: The transaction consists of a collection of actions. The system provides the all-or-nothing illusion that either all these operations are performed or none of them are performed – the transaction either commits or aborts.

- Consistency: Transactions are assumed to perform correct transformations of the abstract system state. The transaction concept allows the programmer to declare such consistency points and allows the system to validate them by application-supplied checks.

- Isolation: While a transaction is updating shared data, that data may be temporarily inconsistent. Such inconsistent data must not be exposed to other transactions until the updater commits. The system must give each transaction the illusion that it is running in isolation; that is, it appears that all other transactions either ran previous to the start of this transaction or ran subsequent to its commit.

- Durability: Once a transaction commits, its updates must be durable. The new state of all objects it updated will be preserved, even in case of hardware or software failures.

These ACID properties were first used to protect data in centralized database applications. But with the advent of distributed databases, transactional RPC became the key technique for structuring distributed database queries and updates. About a decade ago, researchers began generalizing the transaction concept to the broader context of distributed computations. The Argus project at the Massachusetts Institute of Technology, and the TABS group at Carnegie Mellon University pioneered this work. The MIT group, led by Barbara Liskov, evolved the CLU language to Argus, a persistent programming language that included distributed and nested transactions. The CMU group, led by Alfred Spector, initially focused on integrating a general transaction model with an operating system kernel. Their goal was to structure the transaction manager as an "open" interface which could be used by any programming language and any resource manager. They implemented the

TABS system which provided an open, general-purpose nested-transaction mechanism on top of the Accent kernel. By 1985, both the Argus and TABS systems were operational and both demonstrated the validity of the approach, but both had disappointing performance. The thesis of TABS was that transactions could be efficiently layered on an operating system kernel as a general-purpose facility open to all resource managers (data servers in TABS/Camelot terminology). Camelot, the successor to TABS, corrected many of its performance problems, generalized many of the basic concepts, and demonstrated the original thesis – the Camelot transaction manager was efficient and derived much of its performance, flexibility, and generality from careful use of operating system kernel facilities. Included with Camelot is the Library, a collection of C procedures that ease resource manager (data server) implementation. Several resource managers have been written using the Camelot Library, including the Jack and Jill example in Chapter 5 of this book. The Avalon programming language, a persistent C++, is implemented as a layer on Camelot. Both Avalon and the Camelot Library show how persistent programming languages can be implemented atop an open transaction manager. It is fair to say that Camelot and Avalon have widely influenced other transaction processing systems. They made substantial contributions to algorithms in many areas. Notable examples are:

- transactional virtual memory integrated with the memory manager,
- transactional remote procedure call,
- commit protocols including lazy commit, non-blocking commit, and group commit,
- log replication protocols for a cluster of servers,
- parallel, nested transactions,
- transaction semantics integrated as a C Programming Language library to automate the standard aspects of constructing transactional clients and servers,
- persistent programming language data types, and
- a zero-knowledge approach to authenticating and authorizing clients and servers.

Their solutions in these areas are prototypes for future transaction processing systems. Open recovery managers such as IBM's MVS DBSR, DEC's VMS DECdtm, and X/Open DTP all have this open style, allowing new resource managers to implement new object types with transactional semantics.

This book tells ALL about Camelot and Avalon. It gives the design rationale, explains the key algorithms, and then describes how they are implemented. The presentation should satisfy even the most curious; it includes implementation details down to the C-structures of interfaces and control blocks. In addition, it includes a detailed study of the performance of the resulting system. As such, this is an excellent book for the novice and the guru, alike. It teaches the novice the basic concepts, and exemplifies these concepts with a concrete implementation. It presents the guru with a whole new world: one in which transactional RPC is not a special-purpose database-only technique, but rather a structuring principle for reliable distributed computations. It shows how the transaction mechanism interacts with the operating system kernel, the communication subsystem, and how it is externalized to its resource manager and application clients. In addition, it explains the system's performance and administration.

Jim Gray
Digital Equipment Corporation

Preface

This book presents details of a general-purpose distributed transaction facility. Future thinking information system specialists will find that this book addresses transactions from paradigms for modern programming languages to details of two-phase commitment protocols. Our goal is to provide the reader with a comprehensive view of an experimental distributed transaction facility so that he can develop a good *intuitive* understanding of these vital concepts as distributed systems comes of age.

Camelot and Avalon are research systems built at Carnegie Mellon University. Camelot is a transaction processing facility for C programmers. It is layered on top of the Mach operating system. The Camelot Library provides procedures and macros that naturally extend C for transaction semantics. Avalon is a language for C++ programmers that extends C++ to support transactions and highly concurrent access to shared data. Avalon is implemented as a layer on top of Camelot.

The book is organized into six parts to reflect the different levels of understanding on which the reader might focus:

- **Part I: The Extended Camelot Interface** presents the high-level interface for Camelot programmers. We assume a good understanding of UNIX[1] and C. The first chapters in this part introduce Camelot and Mach. The Camelot chapter also presents a brief introduction to transactions. The Mach chapter includes material on how to create transactional remote procedure call interfaces.

  Subsequent chapters in this part describe the Camelot Library and Node Configuration Application. The Camelot Library provides extensions to the C programming language implemented with macros and calls to library procedures. To help clarify the explanations, we repeatedly refer to an example application (Jack) and data server (Jill) that are distributed with Camelot. Jack provides an interactive interface that is invoked in response to user commands. Jill implements read and write operations on a recoverable array of non-negative integers.

- **Part II: The Primitive Camelot Interface** describes Camelot's low-level message interfaces. These interfaces are hidden by the Camelot Library and the Camelot Node Configuration Application. Chapters in this part describe the interfaces for Mach's message passing and thread of control primitives, as well as interfaces to Camelot's recoverable virtual memory, transaction management, and node management components. The material discussed in these chapters presents Camelot's structure in much greater detail and should aid in debugging Camelot programs. It also provides information needed to write new Camelot libraries.

[1] Registered trademark of AT&T.

- **Part III: Design Rationale** explains how Camelot is organized internally. The first chapter describes the overall design of Camelot. Subsequent chapters describe each of the Camelot components in detail.
- **Part IV: The Avalon Language** presents a new programming language that extends C++ to provide support for transactions. In Avalon, the recoverability attribute for storage is inherited via the type mechanism. The language also supports hybrid atomic transactions to allow highly concurrent access to data.
- **Part V: Advanced Features** presents three research packages that can be used with Camelot. The first package is an experimental version of the Camelot Library for Lisp. The second is the Strongbox security package. The third is the distributed log facility that allows Camelot nodes to spool log data to stable storage that is maintained on other nodes.
- **Part VI: The Appendices** describe how to debug Camelot programs, the Camelot abort codes, the Camelot interface specifications, and the Avalon grammar.

Readers that desire a high-level view of the system should focus on the early chapters in each part as they introduce issues each part addresses. The reader that desires a broad understanding of distributed transactions should read the first and third parts of the book as they present high-level concepts and design decisions of Camelot. Specialists that wish to learn how to implement a distributed transaction facility will find the functional separation of the Camelot components in Parts II and III makes the implementation details easier to grasp. Researchers will find discussions of the many novel aspects of the Camelot and Avalon systems throughout the book.

This document describes Camelot as of release 1.0(84) and Mach as of release 2.5. This book was a collaborative effort of Joshua Bloch, Stewart Clamen, Eric Cooper, Dean Daniels, David Detlefs, Richard Draves, Dan Duchamp, Jeffrey Eppinger, Maurice Herlihy, Elliot Jaffe, Karen Kietzke, Richard Lerner, Su-Yuen Ling, David McDonald, George Michaels, Lily Mummert, Sherri Nichols, Randy Pausch, Alfred Spector, Peter Stout, Dean Thompson, Doug Tygar, Jeannette Wing, and Bennet Yee. Kathryn Swedlow provided editorial assistance. Initially, this book was oriented as a Camelot manual. The focus later shifted to a comprehensive description of Camelot and related software. We thank Jim Gray, Norm Hutchinson, and Domenico Ferrari for their guidance during this refocusing.

The developers of Camelot are deeply indebted to the initial users of the system, who have helped to uncover a number of bugs and design failures. These users are Steven Berman, Gregory Bruell, Mark Hahn, Andrew Hastings, Scott Jones, Toshihiko Kato, Jay Kistler, Puneet Kumar, Maria Okasaki, Mahadev Satyanarayanan, Ellen Siegel, and David Steere. Prose from various Mach Project documents has been incorporated into the text, with permission of the authors (including Mary Thompson and Rick Rashid).

Jeffrey L. Eppinger
Transarc Corporation

Lily B. Mummert
Carnegie Mellon University

Alfred Z. Spector
Transarc Corporation

Contents

Foreword v

Preface vii

I The Extended Camelot Interface 1

1 Introduction to Camelot 3
- 1.1 Background . . . 3
- 1.2 A Transaction Example . . . 4
- 1.3 Overview of the Camelot Distributed Transaction Facility . . . 7
- 1.4 Major Camelot Functions . . . 8
- 1.5 Camelot from a User's Point of View . . . 10

2 An Introduction to Mach for Camelot Users 13
- 2.1 Tasks and Threads . . . 13
- 2.2 Virtual Memory Management . . . 14
- 2.3 Interprocess Communication . . . 15
- 2.4 Mach Interface Generator . . . 16

3 The Camelot Library 21
- 3.1 Introduction . . . 21
- 3.2 Using the Camelot Library . . . 22
- 3.3 Application Basics . . . 23
- 3.4 Server Basics . . . 26
- 3.5 Caveats . . . 33
- 3.6 Advanced Constructs . . . 35

4 Camelot Node Configuration 57
- 4.1 Server Maintenance . . . 57
- 4.2 Account Maintenance . . . 59
- 4.3 Accessing a Remote Node Server . . . 59
- 4.4 The Node Server Database . . . 59
- 4.5 Commands Listed . . . 60

5 A Sample Camelot Application and Server 63
5.1 Introduction . . . 63
5.2 Sample Execution . . . 63
5.3 The Application . . . 67
5.4 The Server . . . 71
5.5 Installation . . . 76

II The Primitive Camelot Interface 79

6 The Structure of Camelot 81
6.1 The Camelot Architecture . . . 81
6.2 An Example Message Flow . . . 84

7 Mach for Camelot Implementors 93
7.1 Interprocess Communication . . . 93
7.2 The External Memory Management Interface . . . 98
7.3 C Threads . . . 98

8 Recoverable Storage Management in Camelot 111
8.1 Recoverable Segments and Regions . . . 111
8.2 Initialization . . . 113
8.3 Mapping . . . 114
8.4 Forward Processing . . . 115
8.5 The Shared Memory Queues . . . 117
8.6 Recovery Processing . . . 120

9 Transaction Management in Camelot 121
9.1 The Nested Transaction Model . . . 121
9.2 Transaction Services for Applications . . . 122
9.3 Transaction Services for Servers . . . 129

10 Camelot Node Management 139
10.1 The NA interface . . . 140

III Design Rationale 149

11 The Design of Camelot 151
11.1 Introduction . . . 151
11.2 Architecture . . . 157
11.3 Algorithms . . . 158
11.4 Related Systems Work . . . 159
11.5 Conclusions . . . 161

12 The Design of the Camelot Library **163**
12.1 Introduction . . . 163
12.2 Architecture . . . 165
12.3 Related Work . . . 185
12.4 Conclusions . . . 185

13 The Design of the Camelot Local Log Manager **189**
13.1 Introduction . . . 189
13.2 Architecture . . . 190
13.3 Algorithms . . . 191
13.4 Related Work . . . 195
13.5 Conclusions . . . 196

14 The Design of the Camelot Disk Manager **197**
14.1 Introduction . . . 197
14.2 Architecture . . . 199
14.3 Algorithms and Data Structures . . . 202
14.4 Related Work . . . 234
14.5 Discussion . . . 234

15 The Design of the Camelot Recovery Manager **237**
15.1 Introduction . . . 237
15.2 Architecture . . . 240
15.3 Algorithms . . . 244
15.4 Related Work . . . 250
15.5 Conclusions . . . 250

16 The Design of the Camelot Transaction Manager **251**
16.1 Introduction . . . 251
16.2 Architecture . . . 253
16.3 Algorithms . . . 260
16.4 Related Work . . . 285
16.5 Conclusions . . . 285

17 The Design of the Camelot Communication Manager **287**
17.1 Introduction . . . 287
17.2 Architecture . . . 287
17.3 Algorithms . . . 288
17.4 Related Work . . . 290
17.5 Conclusions . . . 290

18 Performance of Select Camelot Functions **293**
18.1 Performance Metrics . . . 294
18.2 Library Costs . . . 294
18.3 Recoverable Virtual Memory Costs . . . 300
18.4 Recovery Costs . . . 302

IV The Avalon Language **303**

19 A Tutorial Introduction **305**
19.1 Terminology . . . 305
19.2 Array of Atomic Integers . . . 306
19.3 FIFO Queue . . . 315
19.4 Atomic Counters . . . 320

20 Reference Manual **335**
20.1 Lexical Considerations . . . 335
20.2 Servers . . . 335
20.3 Base Classes . . . 337
20.4 Control Structures . . . 341
20.5 Transmission of Data . . . 344

21 Library **347**
21.1 Non-atomic Avalon/C++ Types and Type Generators . . . 347
21.2 Atomic Types . . . 351
21.3 Catalog Server . . . 353

22 Guidelines for Programmers **357**
22.1 Choosing Identifiers . . . 357
22.2 Using and Implementing Avalon Types . . . 357
22.3 Constructing an Avalon Program . . . 363
22.4 For Experts Only . . . 364

V Advanced Features **369**

23 Common Lisp Interface **371**
23.1 Introduction . . . 371
23.2 Accessing Camelot Servers from Lisp . . . 372
23.3 Examples . . . 373
23.4 The Lisp Recoverable Object Server . . . 376
23.5 Summary and Future Work . . . 380

24 Strongbox **381**
24.1 Introduction . . . 381
24.2 Design Goals . . . 382
24.3 Strongbox Architecture . . . 383
24.4 Converting Camelot Clients and Servers to be Secure . . . 387
24.5 Secure Loader and White Pages Server . . . 391
24.6 Interfaces . . . 392
24.7 Security Algorithms . . . 393
24.8 Special Issues . . . 398
24.9 Conclusions . . . 400

25 The Design of the Camelot Distributed Log Facility **401**
25.1 Introduction . . . 401
25.2 Architecture . . . 402
25.3 Algorithms . . . 404
25.4 Related Work . . . 422
25.5 Conclusions . . . 423

VI Appendices **425**

A Debugging **427**
A.1 Avoiding Bugs . . . 427
A.2 Tools and Techniques . . . 430

B Abort Codes **435**
B.1 System Abort Codes . . . 435
B.2 Library Abort Codes . . . 436

C Camelot Interface Specification **437**
C.1 AT Interface . . . 437
C.2 CA Interface . . . 438
C.3 CS Interface . . . 438
C.4 CT Interface . . . 439
C.5 DL Interface . . . 439
C.6 DN Interface . . . 440
C.7 DR Interface . . . 442
C.8 DS Interface . . . 447
C.9 DT Interface . . . 451
C.10 LD Interface . . . 452
C.11 MD Interface . . . 455
C.12 MR Interface . . . 456
C.13 MT Interface . . . 456
C.14 MX Interface . . . 457
C.15 NA Interface . . . 457
C.16 ND Interface . . . 464
C.17 RD Interface . . . 467
C.18 RT Interface . . . 469
C.19 SR Interface . . . 469
C.20 ST Interface . . . 470
C.21 TA Interface . . . 472
C.22 TC Interface . . . 474
C.23 TD Interface . . . 475
C.24 TR Interface . . . 477
C.25 TS Interface . . . 477

D Avalon Grammar **479**
D.1 Expressions 479
D.2 Declarations 480
D.3 Statements 482
D.4 External Definitions 482

Bibliography **483**

Index **493**

List of Figures

1.1 A transaction to transfer $100 from savings to checking. 5
1.2 A transaction that adds monthly interest to savings account. 6
1.3 A Non-serializable Schedule 6
1.4 Client Server Model in Camelot 8
1.5 A User's View of Camelot 11

2.1 The Operation of a Mach Remote Procedure Call 17
2.2 The MIG Specification file `jill.defs` 18

3.1 COFOR Example 41

5.1 Jack - Main Procedure 67
5.2 Jack - `jill_transaction` Function 68
5.3 Jack - `jill_transaction` Function 69
5.4 Jack - Read and Write Functions 70
5.5 Jack - Help Function 71
5.6 Jill - MIG Interface Description File 72
5.7 Jill - Client Header File 72
5.8 Jill - Internal Header File 73
5.9 Jill - Main Procedure 74
5.10 Jill - Initialization Procedure 74
5.11 Jill - Operation Procedures 75

6.1 Interfaces to Camelot 82
6.2 The Structure of a Camelot Node 83
6.3 Messages for the BEGIN_TRANSACTION Statement 85
6.4 Messages for the `jill_read` Server Call 86
6.5 Messages for the `jill_write` Server Call 87
6.6 Messages for the END_TRANSACTION Statement 88
6.7 Messages for a Network Server Call 89
6.8 Messages for Distributed Commitment 90
6.9 Messages for Abort 92

7.1 Mapping an External Memory Object 99
7.2 A Page Fault an External Memory Object 99

8.1 Camelot Segment Descriptor 112
8.2 Camelot Region Pointer 112

9.1 Example Family of Transactions 122
9.2 Applications and the Transaction Manager 123
9.3 Servers and the Transaction Manager – Top Level Transactions 130
9.4 Servers and the Transaction Manager – Nested Transactions 131
9.5 Alternatives for Lock Anti-Inheritance 133
9.6 Abort Protocol 134
9.7 Commitment Protocol – Read-only Independent Vote 136
9.8 Commitment Protocol – Read-only Dependent Vote 137
9.9 Commitment Protocol – Yes Vote, Successful Commit 137
9.10 Commitment Protocol – Yes or No Vote, Abort During Commit 137

11.1 Tasks in Camelot 156

12.1 Simplified Expansion of a Transaction Block 178
12.2 Simplified Locking Procedure 181
12.3 Simplified House Cleaning Procedure 183

13.1 Threads vs. I/O calls 194

14.1 A Shared Memory Queue 203
14.2 The Grid Tracks Records in the Log 207
14.3 Page Record 208
14.4 Transaction Record 209
14.5 Grid Record 210
14.6 Patch Records in the Grid 219
14.7 Patch Record 220
14.8 Server Record 224
14.9 Camelot Chunk Descriptor 228

16.1 Camelot Transaction Identifier 256
16.2 Timestamps in Camelot 257
16.3 Site Grid for a Transaction Family 258
16.4 Local Commitment Protocol 261
16.5 Distributed Two-Phase Commitment without Errors 264
16.6 Non-blocking Commitment without Errors 268
16.7 Non-blocking Commitment with Coordinator Crash 270
16.8 Distributed abort of a top-level transaction 277
16.9 Distributed abort of an active nested transaction 278
16.10 FPI Internal Structure 282
16.11 Procedure for Assigning New Serial-Sequence Pair in FPI 282
16.12 FPI Comparison Macro 283
16.13 Using Timestamps for Crash Detection 284