BS ISO/IEC 19500-1:2012

BSI Standards Publication

Information technology — Object Management Group — Common Object Request Broker Architecture (CORBA)

Part 1: Interfaces

bsi.

...making excellence a habit.™

Document Sharing Hub

Full document details: BS ISO/IEC 19500-1:2012

National foreword

This British Standard is the UK implementation of ISO/IEC 19500-1:2012.

The UK participation in its preparation was entrusted to Technical Committee IST/15, Software and systems engineering.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

© The British Standards Institution 2012. Published by BSI Standards Limited 2012

ISBN 978 0 580 65980 5

ICS 35.080

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 November 2012.

Amendments issued since publication

Date	Text affected

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19500-1:2012

INTERNATIONAL STANDARD

ISO/IEC 19500-1

First edition
2012-04-15

Information technology — Object Management Group — Common Object Request Broker Architecture (CORBA) —

Part 1: Interfaces

Technologies de l'information — OMG (Object Management Group) — CORBA (Common Object Request Broker Architecture) —

Partie 1: Interfaces

Reference number
ISO/IEC 19500-1:2012(E)

Documents
Document Sharing Hub
© ISO/IEC 2012

Full document details: BS ISO/IEC 19500-1:2012

COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2012

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

© ISO/IEC 2012 – All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19500-1:2012

Table of Contents

Foreword xvii
Introduction xix

1 Scope 1

2 Conformance and Compliance 1

3 Normative References 1

4 Additional Information 2
- 4.1 Outline of Contents 2
- 4.2 Keywords for Requirement Statements 3

5 The Object Model 5
- 5.1 General 5
- 5.2 Overview 5
- 5.3 Object Semantics 6
  - 5.3.1 Objects 6
  - 5.3.2 Requests 6
  - 5.3.3 Object Creation and Destruction 7
  - 5.3.4 Types 7
  - 5.3.5 Interfaces 8
  - 5.3.6 Value Types 9
  - 5.3.7 Abstract Interfaces 9
  - 5.3.8 Operations 9
  - 5.3.9 Attributes 11
- 5.4 Object Implementation 11
  - 5.4.1 The Execution Model: Performing Services 11
  - 5.4.2 The Construction Model 12

6 CORBA Overview 13
- 6.1 General 13

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub

6.2 Structure of an Object Request Broker 13
6.2.1 Object Request Broker 17
6.2.2 Clients 18
6.2.3 Object Implementations 18
6.2.4 Object References 18
6.2.5 OMG Interface Definition Language 19
6.2.6 Mapping of IDL to Programming Languages 19
6.2.7 Client Stubs 19
6.2.8 Dynamic Invocation Interface 19
6.2.9 Implementation Skeleton 20
6.2.10 Dynamic Skeleton Interface 20
6.2.11 Object Adapters 20
6.2.12 ORB Interface 20
6.2.13 Interface Repository 21
6.2.14 Implementation Repository 21
6.3 Example ORBs 21
6.3.1 Client- and Implementation-resident ORB 21
6.3.2 Server-based ORB 21
6.3.3 System-based ORB 22
6.3.4 Library-based ORB 22
6.4 Structure of a Client 22
6.5 Structure of an Object Implementation 23
6.6 Structure of an Object Adapter 25
6.7 CORBA Required Object Adapter 26
6.7.1 Portable Object Adapter 26
6.8 The Integration of Foreign Object Systems 26

7 IDL Syntax and Semantics 29
7.1 Overview 29
7.2 Lexical Conventions 30
7.2.1 Tokens 33
7.2.2 Comments 33
7.2.3 Identifiers 33
7.2.4 Keywords 35
7.2.5 Literals 36
7.3 Preprocessing 38
7.4 IDL Grammar 38

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

7.5 IDL Specification 45
7.6 Import Declaration 45
7.7 Module Declaration 46
7.8 Interface Declaration 47
7.8.1 Interface Header 47
7.8.2 Interface Inheritance Specification 47
7.8.3 Interface Body 48
7.8.4 Forward Declaration 48
7.8.5 Interface Inheritance 49
7.8.6 Abstract Interface 51
7.8.7 Local Interface 51
7.9 Value Declaration 52
7.9.1 Regular Value Type 52
7.9.2 Boxed Value Type 54
7.9.3 Abstract Value Type 55
7.9.4 Value Forward Declaration 55
7.9.5 Valuetype Inheritance 55
7.10 Constant Declaration 57
7.10.1 Syntax 57
7.10.2 Semantics 58
7.11 Type Declaration 61
7.11.1 Basic Types 62
7.11.2 Constructed Types 64
7.11.3 Template Types 68
7.11.4 Complex Declarator 69
7.11.5 Native Types 69
7.11.6 Deprecated Anonymous Types 70
7.12 Exception Declaration 73
7.13 Operation Declaration 73
7.13.1 Operation Attribute 74
7.13.2 Parameter Declarations 74
7.13.3 Raises Expressions 74
7.13.4 Context Expressions 75
7.14 Attribute Declaration 76
7.15 Repository Identity Related Declarations 77
7.15.1 Repository Identity Declaration 77
7.15.2 Repository Identifier Prefix Declaration 78
7.15.3 Repository Id Conflict 79

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19500-1:2012

7.16 Event Declaration 79
7.16.1 Regular Event Type 79
7.16.2 Abstract Event Type 80
7.16.3 Event Forward Declaration 80
7.16.4 Eventtype Inheritance 80
7.17 Component Declaration 80
7.17.1 Component 80
7.17.2 Component Header 81
7.17.3 Component Body 82
7.17.4 Event Sources—publishers and emitters 84
7.17.5 Event Sinks 84
7.17.6 Basic and Extended Components 85
7.18 Home Declaration 85
7.18.1 Home 85
7.18.2 Home Header 86
7.18.3 Home Body 87
7.19 CORBA Module 88
7.20 Names and Scoping 89
7.20.1 Qualified Names 89
7.20.2 Scoping Rules and Name Resolution 90
7.20.3 Special Scoping Rules for Type Names 93
8 ORB Interface 95
8.1 Overview 95
8.2 The ORB Operations 95
8.2.1 ORB Identity 101
8.2.2 Converting Object References to Strings 101
8.2.3 Getting Service Information 102
8.2.4 Creating a New Context 102
8.2.5 Thread-Related Operations 102
8.3 Object Reference Operations 105
8.3.1 Determining the Object Interface 107
8.3.2 Duplicating and Releasing Copies of Object References 107
8.3.3 Nil Object References 107
8.3.4 Equivalence Checking Operation 108
8.3.5 Probing for Object Non-Existence 108
8.3.6 Object Reference Identity 108
8.3.7 Type Coercion Considerations 110
8.3.8 Getting Policy Associated with the Object 110

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

8.3.9 Overriding Associated Policies on an Object Reference 111
8.3.10 Validating Connection 112
8.3.11 Getting the Domain Managers Associated with the Object 112
8.3.12 Getting Component Associated with the Object 113
8.3.13 Getting the ORB 113
8.3.14 LocalObject Operations 113

8.4 ValueBase Operations 114

8.5 ORB and OA Initialization and Initial References 115
8.5.1 ORB Initialization 115
8.5.2 Obtaining Initial Object References 117
8.5.3 Configuring Initial Service References 120

8.6 Context Object 122
8.6.1 Introduction 122
8.6.2 Context Object Operations 122

8.7 Current Object 125

8.8 Policy Object 126
8.8.1 Definition of Policy Object 126
8.8.2 Creation of Policy Objects 127
8.8.3 Usages of Policy Objects 129
8.8.4 Policy Associated with the Execution Environment 129
8.8.5 Specification of New Policy Objects 130
8.8.6 Standard Policies 131

8.9 Management of Policies 131
8.9.1 Client Side Policy Management 131
8.9.2 Server Side Policy Management 132
8.9.3 Policy Management Interfaces 132

8.10 Management of Policy Domains 134
8.10.1 Basic Concepts 134
8.10.2 Domain Management Operations 136

8.11 TypeCodes 138
8.11.1 The TypeCode Interface 138
8.11.2 TypeCode Constants 142
8.11.3 Creating TypeCodes 143

8.12 Exceptions 148
8.12.1 Definition of Terms 148
8.12.2 System Exceptions 148
8.12.3 Standard System Exception Definitions 150
8.12.4 Standard Minor Exception Codes 156

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19500-1:2012

9 Value Type Semantics 157
9.1 Overview 157
9.2 Architecture 157
9.2.1 Abstract Values 158
9.2.2 Operations 158
9.2.3 Value Type vs. Interfaces 159
9.2.4 Parameter Passing 159
9.2.5 Substitutability Issues 160
9.2.6 Widening/Narrowing 161
9.2.7 Value Base Type 161
9.2.8 Life Cycle issues 161
9.2.9 Security Considerations 162
9.3 Standard Value Box Definitions 162
9.4 Language Mappings 163
9.4.1 General Requirements 163
9.4.2 Language Specific Marshaling 163
9.4.3 Language Specific Value Factory Requirements 163
9.4.4 Value Method Implementation 164
9.5 Custom Marshaling 164
9.5.1 Implementation of Custom Marshaling 164
9.5.2 Marshaling Streams 165
9.6 Access to the Sending Context Run Time 171

10 Abstract Interface Semantics 173
10.1 Overview 173
10.2 Semantics of Abstract Interfaces 173
10.3 Usage Guidelines 174
10.4 Example 174
10.5 Security Considerations 175
10.5.1 Passing Values to Trusted Domains 175

11 Dynamic Invocation Interface 177
11.1 Overview 177
11.1.1 Common Data Structures 177
11.1.2 Memory Usage 179
11.1.3 Return Status and Exceptions 179

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

11.2 Request Operations ... 179
11.2.1 create_request ... 180
11.2.2 add_arg ... 182
11.2.3 invoke ... 182
11.2.4 delete ... 183
11.2.5 send ... 183
11.2.6 poll_response ... 183
11.2.7 get_response ... 183
11.2.8 sendp ... 184
11.2.9 prepare ... 184
11.2.10 sendc ... 184
11.3 ORB Operations ... 185
11.3.1 send_multiple_requests ... 185
11.3.2 get_next_response and poll_next_response ... 185
11.4 Polling ... 186
11.4.1 Abstract Valuetype Pollable ... 187
11.4.2 Abstract Valuetype DIIPollable ... 188
11.4.3 interface PollableSet ... 188
11.5 List Operations ... 189
11.5.1 create_list ... 190
11.5.2 add_item ... 190
11.5.3 free ... 191
11.5.4 free_memory ... 191
11.5.5 get_count ... 191
11.5.6 create_operation_list ... 191
12 Dynamic Skeleton Interface ... 193
12.1 Introduction ... 193
12.2 Overview ... 193
12.3 ServerRequestPseudo-Object ... 194
12.3.1 ExplicitRequest State: ServerRequestPseudo-Object ... 194
12.4 DSI: Language Mapping ... 195
12.4.1 ServerRequest's Handling of Operation Parameters ... 195
12.4.2 Registering Dynamic Implementation Routines ... 195
13 Dynamic Management of Any Values ... 197
13.1 General ... 197
13.2 Overview ... 197

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19500-1:2012

13.3 DynAny API 198
13.3.1 Creating a DynAny Object 204
13.3.2 The DynAny Interface 206
13.3.3 The DynFixed Interface 210
13.3.4 The DynEnum Interface 210
13.3.5 The DynStruct Interface 211
13.3.6 The DynUnion Interface 212
13.3.7 The DynSequence Interface 214
13.3.8 The DynArray Interface 215
13.3.9 The DynValueCommon Interface 216
13.3.10 The DynValue Interface 216
13.3.11 The DynValueBox Interface 217
13.4 Usage in C++ Language 218
13.4.1 Dynamic Creation of CORBA::Any values 218
13.4.2 Dynamic Interpretation of CORBA::Any values 219

14 The Interface Repository 221
14.1 Overview 221
14.2 Scope of an Interface Repository 221
14.3 Implementation Dependencies 223
14.3.1 Managing Interface Repositories 223
14.4 Basics 224
14.4.1 Names and Identifiers 224
14.4.2 Types and TypeCodes 225
14.4.3 Interface Repository Objects 225
14.4.4 Structure and Navigation of the Interface Repository 226
14.5 Interface Repository Interfaces 228
14.5.1 Supporting Type Definitions 229
14.5.2 IRObject 230
14.5.3 Contained 231
14.5.4 Container 233
14.5.5 IDLType 238
14.5.6 Repository 238
14.5.7 ModuleDef 240
14.5.8 ConstantDef 240
14.5.9 TypedefDef 241
14.5.10 StructDef 241
14.5.11 UnionDef 242
14.5.12 EnumDef 243
14.5.13 AliasDef 243

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

14.5.14 PrimitiveDef 244
14.5.15 StringDef 244
14.5.16 WstringDef 244
14.5.17 FixedDef 245
14.5.18 SequenceDef 245
14.5.19 ArrayDef 245
14.5.20 ExceptionDef 246
14.5.21 AttributeDef 247
14.5.22 ExtAttributeDef 247
14.5.23 OperationDef 248
14.5.24 InterfaceDef 250
14.5.25 ExtInterfaceDef 252
14.5.26 AbstractInterfaceDef 253
14.5.27 ExtAbstractInterfaceDef 254
14.5.28 LocalInterfaceDef 255
14.5.29 ExtLocalInterfaceDef 256
14.5.30 ValueMemberDef 256
14.5.31 ValueDef 257
14.5.32 ExtValueDef 260
14.5.33 ValueBoxDef 262
14.5.34 NativeDef 262

14.6 Component Interface Repository Interfaces 262
14.6.1 ComponentIR::Container 262
14.6.2 ComponentIR::Repository 264
14.6.3 ComponentIR::ProvidesDef 265
14.6.4 ComponentIR::UsesDef 265
14.6.5 ComponentIR::EventDef 266
14.6.6 ComponentIR::EventPortDef 266
14.6.7 ComponentIR::EmitsDef 267
14.6.8 ComponentIR::PublishesDef 268
14.6.9 ComponentIR::ConsumesDef 268
14.6.10 ComponentIR::ComponentDef 268
14.6.11 ComponentIR::FactoryDef 271
14.6.12 ComponentIR::FinderDef 272
14.6.13 ComponentIR::HomeDef 272

14.7 RepositoryIds 274
14.7.1 IDL Format 275
14.7.2 RMI Hashed Format 275
14.7.3 DCE UUID Format 277
14.7.4 LOCAL Format 277
14.7.5 Pragma Directives for RepositoryId 277
14.7.6 For More Information 282

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19500-1:2012

14.7.7 RepositoryIDs for OMG-Specified Types 282
14.7.8 Uniqueness Constraints on Repository IDs 283
14.8 IDL for Interface Repository 284

15 The Portable Object Adapter 303

15.1 Overview 303
15.2 Abstract Model Description 303
15.2.1 Model Components 303
15.2.2 Model Architecture 305
15.2.3 POA Creation 306
15.2.4 Reference Creation 307
15.2.5 Object Activation States 308
15.2.6 Request Processing 308
15.2.7 Implicit Activation 309
15.2.8 Multi-threading 310
15.2.9 Dynamic Skeleton Interface 311
15.2.10 Location Transparency 312
15.3 Interfaces 312
15.3.1 The Servant IDL Type 313
15.3.2 POAManager Interface 314
15.3.3 POAManagerFactory Interface 318
15.3.4 AdapterActivator Interface 319
15.3.5 ServantManager Interface 320
15.3.6 ServantActivator Interface 321
15.3.7 ServantLocator Interface 323
15.3.8 POA Policy Objects 325
15.3.9 POA Interface 328
15.3.10 Current Operations 337
15.4 IDL for PortableServer Module 338
15.5 UML Description of PortableServer 344
15.6 Usage Scenarios 346
15.6.1 Getting the Root POA 346
15.6.2 Creating a POA 347
15.6.3 Explicit Activation with POA-assigned Object Ids 347
15.6.4 Explicit Activation with User-assigned Object Ids 348
15.6.5 Creating References before Activation 349
15.6.6 Servant Manager Definition and Creation 349
15.6.7 Object Activation on Demand 351
15.6.8 Persistent Objects with POA-assigned Ids 352

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

15.6.9 Multiple Object Ids Mapping to a Single Servant 352
15.6.10 One Servant for All Objects 352
15.6.11 Single Servant, Many Objects and Types, Using DSI 355

16 Portable Interceptors 359

16.1 Introduction 359
16.1.1 Object Creation 359
16.1.2 Client Sends Request 360
16.1.3 Server Receives Request 361
16.1.4 Server Sends Reply 361
16.1.5 Client Receives Reply 362

16.2 General Behavior of Local Objects 362

16.3 Interceptor Interface 362

16.4 Request Interceptors 363
16.4.1 Design Principles 363
16.4.2 General Flow Rules 364
16.4.3 The Flow Stack Visual Model 364
16.4.4 The Request Interceptor Points 365
16.4.5 Client-Side Interceptor 365
16.4.6 Client-Side Interception Points 365
16.4.7 Client-Side Interception Point Flow 367
16.4.8 Server-Side Interceptor 370
16.4.9 Server-Side Interception Points 370
16.4.10 Server-Side Interception Point Flow 372
16.4.11 Request Information 375
16.4.12 RequestInfo Interface 375
16.4.13 ClientRequestInfo Interface 379
16.4.14 ServerRequestInfo Interface 382
16.4.15 ForwardRequest Exception 386

16.5 Portable Interceptor Current 386
16.5.1 Overview 386
16.5.2 Obtaining the Portable Interceptor Current 386
16.5.3 Portable Interceptor Current Interface 387
16.5.4 Use of Portable Interceptor Current 388

16.6 IOR Interceptor 392
16.6.1 Overview 392
16.6.2 An Abstract Model for Object Adapters 392
16.6.3 Object Reference Template 394
16.6.4 IORInterceptor Interface 396
16.6.5 IORInfo Interface 397

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

16.7 Interceptor Policy Objects 400
16.7.1 ProcessingMode Policy 400
16.8 PolicyFactory 401
16.8.1 PolicyFactory Interface 401
16.9 Registering Interceptors 401
16.9.1 ORBInitializer Interface 401
16.9.2 ORBInitInfo Interface 402
16.9.3 register_orb_initializer Operation 406
16.9.4 Notes about Registering Interceptors 408
16.10 Dynamic Initial References 409
16.10.1 register_initial_reference 409
16.11 Module Dynamic 410
16.11.1 NVList PIDL Represented by ParameterList IDL 410
16.11.2 ContextList PIDL Represented by ContextList IDL 410
16.11.3 ExceptionList PIDL Represented by ExceptionList IDL 410
16.11.4 Context PIDL Represented by RequestContext IDL 410
16.12 Consolidated IDL 410
16.12.1 Dynamic 410
16.12.2 Portions of IOP Relevant to Portable Interceptor 411
16.12.3 PortableInterceptor 412

17 CORBA Messaging 417
17.1 General 417
17.2 Quality of Service 417
17.3 Messaging Quality of Service 417
17.3.1 Rebind Support 419
17.3.2 Synchronization Scope 420
17.3.3 Request and Reply Priority 421
17.3.4 Request and Reply Timeout 422
17.3.5 Routing 424
17.3.6 Queue Ordering 425
17.4 Propagation of Messaging QoS 426
17.4.1 Structures 426
17.4.2 Messaging QoS Profile Component 426
17.4.3 Messaging QoS Service Context 426
17.5 Messaging Programming Model 427
17.6 Running Example 428

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

17.7 Async Operation Mapping 428
17.7.1 Callback Model Signatures (sendc) 429
17.7.2 Polling Model Signatures (sendp) 431
17.8 Exception Delivery in the Callback Model 433
17.8.1 Messaging::ExceptionHolder valuetype 433
17.9 Type-Specific ReplyHandler Mapping 433
17.9.1 ReplyHandler Operations for NO_EXCEPTION Replies 434
17.9.2 ReplyHandler Operations for Exceptional Replies 435
17.9.3 Example 435
17.10 Generic Poller Value 436
17.10.1 operation_target 437
17.10.2 operation_name 437
17.10.3 associated_handler 437
17.10.4 is_from_poller 437
17.11 Type-Specific Poller Mapping 437
17.11.1 Basic Type-Specific Poller 438
17.11.2 Persistent Type-Specific Poller 440
17.11.3 Example 440
17.12 Example Programmer Usage 441
17.12.1 Example Programmer Usage (Examples Mapped to C++) 441
17.12.2 Client-Side C++ Example for the Asynchronous Method Signatures 441
17.12.3 Client-Side C++ Example of the Callback Model 442
17.12.4 Client-Side C++ Example of the Polling Model 449
17.12.5 Server Side 454
17.13 Message Routing Interoperability 455
17.14 Routing Object References 456
17.15 Message Routing 457
17.15.1 Structures 459
17.15.2 Interfaces 460
17.15.3 Routing Protocol 462
17.16 Router Administration 467
17.16.1 Constants 470
17.16.2 Exceptions 470
17.16.3 Valuetypes 471
17.16.4 Interfaces 471
17.17 CORBA Messaging IDL 472
17.17.1 Messaging Module 472
17.17.2 MessageRouting Module 475

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

Chapter 17 Annexes

A.1 QoS Abstract Model Design 480
A.2 Model Components 480
A.2.1 Component Relationships 481
A.2.2 Component Design 481
A.3 AMI/TII Abstract Model Design 482
A.3.1 Asynchronous Method Invocation Components 482
A.3.2 Time-Independent Invocation Components 483
A.3.3 Component Relationships 483
A.3.4 Callback Model Detailed Design 486
A.3.5 Poller/PersistentRequest Detailed Design 487
A.4 Message Routing Abstract Model Design 488
A.4.1 Model Components 489
A.4.2 Component Relationships 489
A.4.3 Router Administration Design 489
B.1 Conformance Issues 491
B.2 Compatibility Issues 491
B.2.1 Transaction Service 491
B.2.2 Changes to Current OTS Behavior 491
B.2.3 Security Service 492

Annex A - IDL Tags and Exceptions 493
Annex B - Legal Information 507

© ISO/IEC 2012 - All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 19500-1 was prepared by Technical Committee ISO/IEC JTC1, Information technology, in collaboration with the Object Management Group (OMG), following the submission and processing as a Publicly Available Specification (PAS) of the OMG Common Object Request Broker Architecture (CORBA) specification Part 1 Version 3.1 CORBA Interfaces.

ISO/IEC 19500-1 is related to:

- ITU-T Recommendation X.902 (1995) | ISO/IEC 10746-2:1996, Information Technology - Open Distributed Processing - Reference Model: Foundations
- ITU-T Recommendation X.903 (1995) | ISO/IEC 10746-3:1996, Information Technology - Open Distributed Processing - Reference Model: Architecture
- ITU-T Recommendation X.920 (1997) | ISO/IEC 14750:1997, Information Technology - Open Distributed Processing - Interface Definition Language
- ISO/IEC 19500-2, Information Technology - Open Distributed Processing - CORBA Specification Part 2: CORBA Interoperability
- ISO/IEC 19500-3, Information Technology - Open Distributed Processing - CORBA Specification Part 3: CORBA Components

ISO/IEC 19500 consists of the following parts, under the general title *Information technology - Open distributed processing - CORBA specification*:

- Part 1: CORBA Interfaces
- Part 2: CORBA Interoperability
- Part 3: CORBA Components

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

It is the common core of the CORBA specification. Optional parts of CORBA, such as mappings to particular programming languages, Real-time CORBA extensions, and the minimum CORBA profile for embedded systems are documented in the other specifications that together comprise the complete CORBA specification. Please visit the CORBA download page at http://www.omg.org/technology/documents/corba_spec_catalog.htm to find the complete CORBA specification set.

Apart from this Foreword, the text of this International Standard is identical with that for the OMG specification for CORBA, v3.1.1, Part 1.

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

Introduction

The rapid growth of distributed processing has led to a need for a coordinating framework for this standardization and ITU-T Recommendations X.901-904 | ISO/IEC 10746, the Reference Model of Open Distributed Processing (RM-ODP) provides such a framework. It defines an architecture within which support of distribution, interoperability and portability can be integrated.

RM-ODP Part 2 (ISO/IEC 10746-2) defines the foundational concepts and modeling framework for describing distributed systems. The scopes and objectives of the RM-ODP Part 2 and the UML, while related, are not the same and, in a number of cases, the RM-ODP Part 2 and the UML specification use the same term for concepts which are related but not identical (e.g., interface). Nevertheless, a specification using the Part 2 modeling concepts can be expressed using UML with appropriate extensions (using stereotypes, tags, and constraints).

RM-ODP Part 3 (ISO/IEC 10746-3) specifies a generic architecture of open distributed systems, expressed using the foundational concepts and framework defined in Part 2. Given the relation between UML as a modeling language and Part 3 of the RM-ODP standard, it is easy to show that UML is suitable as a notation for the individual viewpoint specifications defined by the RM-ODP.

This International Standard for CORBA Interfaces is a standard for the technology specification of an ODP system. It defines a technology to provide the infrastructure required to support functional distribution of an ODP system, specifying functions required to manage physical distribution, communications, processing and storage, and the roles of different technology objects in supporting those functions.

Context of CORBA

The key to understanding the structure of the CORBA architecture is the Reference Model, which consists of the following components:

- **Object Request Broker**, which enables objects to transparently make and receive requests and responses in a distributed environment. It is the foundation for building applications from distributed objects and for interoperability between applications in hetero- and homogeneous environments. The architecture and specifications of the Object Request Broker are described in this manual.
- **Object Services**, a collection of services (interfaces and objects) that support basic functions for using and implementing objects. Services are necessary to construct any distributed application and are always independent of application domains. For example, the Life Cycle Service defines conventions for creating, deleting, copying, and moving objects; it does not dictate how the objects are implemented in an application. Specifications for Object Services are contained in *CORBAservices*: *Common Object Services Specification*.
- **Common Facilities**, a collection of services that many applications may share, but which are not as fundamental as the Object Services. For instance, a system management or electronic mail facility could be classified as a common facility. Information about Common Facilities will be contained in *CORBAfacilities*: *Common Facilities Architecture*.
- **Application Objects**, which are products of a single vendor on in-house development group that controls their interfaces. Application Objects correspond to the traditional notion of applications, so they are not standardized by OMG. Instead, Application Objects constitute the uppermost layer of the Reference Model.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

The Object Request Broker, then, is the core of the Reference Model. It is like a telephone exchange, providing the basic mechanism for making and receiving calls. Combined with the Object Services, it ensures meaningful communication between CORBA-compliant applications.

The architecture and specifications described in this standard are aimed at software designers and developers who want to produce applications that comply with OMG specifications for the Object Request Broker (ORB), or this standard (ISO/IEC 19500). The benefit of compliance is, in general, to be able to produce interoperable applications that are based on distributed, interoperating objects. The ORB provides the mechanisms by which objects transparently make requests and receive responses. Hence, the ORB provides interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems.

This Part of this International Standard includes normative and non-normative annexes.

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

Information technology - Object Management Group Common Object Request Broker Architecture (CORBA), Interfaces

1 Scope

This part of ISO/IEC 19500 specifies the CORBA Object Model and uses concepts from that model to define the operation of the Object Request Broker (ORB). The ORB is the basic mechanism by which objects transparently make requests to, and receive responses from, each other on the same machine or across a network. A client need not be aware of the mechanisms used to communicate with or activate an object, how the object is implemented, or where the object is located.

2 Conformance and Compliance

The minimum required for a CORBA-compliant system is adherence to the specifications in this part of ISO/IEC 19500 and one mapping. Each additional language mapping is a separate, optional compliance point. Optional means users aren't required to implement these points if they are unnecessary at their site, but if implemented, they must adhere to the CORBA specifications to be called CORBA-compliant. For instance, if a vendor supports C++, their ORB must comply with the OMG IDL to C++ binding specified in the C++ Language Mapping Specification.

The CORBA Language Mappings have been separated from this standard and each language mapping is its own separate OMG specification.

3 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

- ITU-T Recommendation X.902 (1995) | ISO/IEC 10746-2:1996, Information Technology - Open Distributed Processing - Reference Model: Foundations
- ITU-T Recommendation X.903 (1995) | ISO/IEC 10746-3:1996, Information Technology - Open Distributed Processing - Reference Model: Architecture
- ITU-T Recommendation X.920 (1997) | ISO/IEC 14750:1997, Information Technology - Open Distributed Processing - Interface Definition Language
- ISO/IEC 14882:2003, Information Technology - Programming languages - C++
- ISO/IEC 9899:1999, Information Technology - Programming languages - C
- [OMA] Object Management Group, "Object Management Architecture Guide, revision 3.0," available from http://www.omg.org/oma/

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19500-1:2012

- [ASMOTS] Object Management Group, “Additional Structuring Mechanisms for the OTS,” available from http://www.omg.org/spec/OTS/
- [TRANS] Object Management Group, “Transaction Service,” available from http://www.omg.org/spec/TRANS/
- [FIREWALL] Object Management Group, “CORBA Firewall Traversal Specification,” available from http://www.omg.org/members/cgi-bin/doc?ptc/04-04-05.pdf
- [SCCP] Object Management Group, “CORBA / TC Interworking and SCCP-Inter ORB Protocol (SCCP),” available from http://www.omg.org/spec/SCCP
- [FTCORBA] Object Management Group, “Fault Tolerant Corba,” clause 23 of CORBA 3.0.3, available from http://www.omg.org/cgi-bin/doc?formal/2004-03-01
- [RTCORBA] Object Management Group, “Real-Time CORBA, version 1.2,” available from http://www.omg.org/spec/RT/
- [WATM] Object Management Group, “Wireless Access and Telecom Mobility in CORBA, Version 1.2,” available from http://www.omg.org/spec/WATM/
- [DCOMI] Object Management Group, “Interoperability with non-CORBA Systems” clause 20 of CORBA 3.0.3, available from http://www.omg.org/cgi-bin/doc?formal/2004-03-01
- [TSAS] Object Management Group “Telecommunications Service Access and Subscription Specification,” available from http://www.omg.org/spec/TSAS/
- [SECDOM] Object Management Group “Security Domain Membership Management Service,” available from http://www.omg.org/members/cgi-bin/doc?orbos/01-06-01.pdf
- [RFC2119] IETF RFC 2119, “Key words for use in RFCs to Indicate Requirement Levels,” S. Bradner, March 1997. Available from http://ietf.org/rfc/rfc2119

4 Additional Information

4.1 Outline of Contents

This part of ISO/IEC 19500 consists of the following:

1. The syntax and semantics of the OMG interface definition language (OMG IDL), which is used to describe the interfaces that client objects call and object implementations provide. Throughout this International Standard the abbreviation IDL is used, for brevity, as shorthand for OMG IDL.
2. The interface to the ORB functions that do not depend on object adapters: these operations are the same for all ORBs and object implementations.
3. The semantics of passing an object by value.
4. An IDL abstract interface, which provides the capability to defer the determination of whether an object is passed by reference or by value until runtime.
5. The Dynamic Invocation Interface (DII), the client’s side of the interface that allows dynamic creation and invocation of request to objects.

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

6. The Dynamic Skeleton Interface (DSI), the server's-side interface that can deliver requests from an ORB to an object implementation that does not have compile-time knowledge of the type of the object it is implementing.
7. The interface for the Dynamic Any type which allows statically-typed programming languages such as C and Java to create or receive values of type Any without compile-time knowledge that the typer contained in the Any.
8. The Interface Repository that manages and provides access to a collection of object definitions.
9. The Portable Object Adapter which defines a group of IDL interfaces that an implementation uses to access ORB functions.
10. ORB operations that allow services such as security to be inserted in the invocation path.
11. Messaging which covers: Quality of Service, Asynchronous Method Invocations (to include Time-Independent or "Persistent" Requests), and the specification of interoperable Routing interfaces to support the transport of requests asynchronously from the handling of their replies.

4.2 Keywords for Requirement Statements

The keywords "must," "must not," "shall," "shall not," "should," "should not," and "may" in this International Standard are to be interpreted as described in [RFC 2119].

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19500-1:2012

© ISO/IEC 2012 – All rights reserved
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

5 The Object Model

5.1 General

This clause describes the concrete object model that underlies the CORBA architecture. The model is derived from the abstract Core Object Model defined by the Object Management Group in the *Object Management Architecture Guide*.

5.2 Overview

The object model provides an organized presentation of object concepts and terminology. It defines a partial model for computation that embodies the key characteristics of objects as realized by the submitted technologies. The OMG object model is *abstract* in that it is not directly realized by any particular technology. The model described here is a *concrete* object model. A concrete object model may differ from the abstract object model in several ways:

- It may *elaborate* the abstract object model by making it more specific, for example, by defining the form of request parameters or the language used to specify types.
- It may *populate* the model by introducing specific instances of entities defined by the model, for example, specific objects, specific operations, or specific types.
- It may *restrict* the model by eliminating entities or placing additional restrictions on their use.

An object system is a collection of objects that isolates the requestors of services (clients) from the providers of services by a well-defined encapsulating interface. In particular, clients are isolated from the implementations of services as data representations and executable code.

The object model first describes concepts that are meaningful to clients, including such concepts as object creation and identity, requests and operations, types and signatures. It then describes concepts related to object implementations, including such concepts as methods, execution engines, and activation.

The object model is most specific and prescriptive in defining concepts meaningful to clients. The discussion of object implementation is more suggestive, with the intent of allowing maximal freedom for different object technologies to provide different ways of implementing objects.

There are some other characteristics of object systems that are outside the scope of the object model. Some of these concepts are aspects of application architecture, some are associated with specific domains to which object technology is applied. Such concepts are more properly dealt with in an architectural reference model. Examples of excluded concepts are compound objects, links, copying of objects, change management, and transactions. Also outside the scope of the object model are the details of control structure: the object model does not say whether clients and/or servers are single-threaded or multi-threaded, and does not specify how event loops are programmed nor how threads are created, destroyed, or synchronized.

This object model is an example of a *classical object model*, where a client sends a message to an object. Conceptually, the object interprets the message to decide what service to perform. In the classical model, a message identifies an object and zero or more actual parameters. As in most classical object models, a distinguished first parameter is required, which identifies the operation to be performed; the interpretation of the message by the object involves selecting a method based on the specified operation. Operationally, of course, method selection could be performed either by the object or the ORB.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19500-1:2012

5.3 Object Semantics

An object system provides services to clients. A *client* of a service is any entity capable of requesting the service. This sub clause defines the concepts associated with object semantics, that is, the concepts relevant to clients.

5.3.1 Objects

An object system includes entities known as objects. An *object* is an identifiable, encapsulated entity that provides one or more services that can be requested by a client.

5.3.2 Requests

Clients request services by issuing requests.

The term *request* is broadly used to refer to the entire sequence of causally related events that transpires between a client initiating it and the last event causally associated with that initiation. For example:

- the client receives the final response associated with that *request* from the server,
- the server carries out the associated operation in case of a oneway request, or
- the sequence of events associated with the *request* terminates in a failure of some sort. The initiation of a Request is an event.

The information associated with a request consists of an operation, a target object, zero or more (actual) parameters, and an optional request context.

A *request form* is a description or pattern that can be evaluated or performed multiple times to cause the issuing of requests. As described in the IDL Syntax and Semantics clause, request forms are defined by particular language bindings. An alternative request form consists of calls to the dynamic invocation interface to create an invocation structure, add arguments to the invocation structure, and to issue the invocation (refer to the *Dynamic Invocation Interface* clause for descriptions of these request forms).

A *value* is anything that may be a legitimate (actual) parameter in a request. More particularly, a value is an instance of an IDL data type. There are non-object values, as well as values that reference objects.

An *object reference* is a value that reliably denotes a particular object. Specifically, an object reference will identify the same object each time the reference is used in a request (subject to certain pragmatic limits of space and time). An object may be denoted by multiple, distinct object references.

A request may have parameters that are used to pass data to the target object; it may also have a request context that provides additional information about the request. A request context is a mapping from strings to strings.

A request causes a service to be performed on behalf of the client. One possible outcome of performing a service is returning to the client the results, if any, defined for the request.

If an abnormal condition occurs during the performance of a request, an exception is returned. The exception may carry additional return parameters particular to that exception.

The request parameters are identified by position. A parameter may be an input parameter, an output parameter, or an input-output parameter. A request may also return a single *return result value*, as well as the results stored into the output and input-output parameters.

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

The following semantics hold for all requests:

- Any aliasing of parameter values is neither guaranteed removed nor guaranteed to be preserved.
- The order in which aliased output parameters are written is not guaranteed.
- The return result and the values stored into the output and input-output parameters are undefined if an exception is returned.

For descriptions of the values and exceptions that are permitted, see Types on page 7 and Exceptions on page 10.

5.3.3 Object Creation and Destruction

Objects can be created and destroyed. From a client's point of view, there is no special mechanism for creating or destroying an object. Objects are created and destroyed as an outcome of issuing requests. The outcome of object creation is revealed to the client in the form of an object reference that denotes the new object.

5.3.4 Types

A *type* is an identifiable entity with an associated predicate (a single-argument mathematical function with a boolean result) defined over entities. An entity *satisfies* a type if the predicate is true for that entity. An entity that satisfies a type is called a *member of the type*.

Types are used in signatures to restrict a possible parameter or to characterize a possible result.

The *extension of a type* is the set of entities that satisfy the type at any particular time.

An *object type* is a type whose members are object references. In other words, an object type is satisfied only by object references.

Constraints on the data types in this model are shown in this sub clause.

5.3.4.1 Basic types

- 16-bit, 32-bit, and 64-bit signed and unsigned 2's complement integers.
- Single-precision (32-bit), double-precision (64-bit), and double-extended (a mantissa of at least 64 bits, a sign bit and an exponent of at least 15 bits) IEEE floating point numbers.
- Fixed-point decimal numbers of up to 31 significant digits.
- Characters, as defined in ISO Latin-1 (8859.1) and other single- or multi-byte character sets.
- A boolean type taking the values TRUE and FALSE.
- An 8-bit opaque detectable, guaranteed to *not* undergo any conversion during transfer between systems.
- Enumerated types consisting of ordered sequences of identifiers.
- A string type, which consists of a variable-length array of characters; the length of the string is a non-negative integer, and is available at run-time. The length may have a maximum bound defined.
- A wide character string type, which consists of a variable-length array of (fixed width) wide characters; the length of the wide string is a non-negative integer, and is available at run-time. The length may have a maximum bound defined.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19500-1:2012

- A container type “any,” which can represent any possible basic or constructed type.
- Wide characters that may represent characters from any wide character set.
- Wide character strings, which consist of a length, available at runtime, and a variable-length array of (fixed width) wide characters.

5.3.4.2 Constructed types

- A record type (called struct), which consists of an ordered set of (name,value) pairs.
- A discriminated union type, which consists of a discriminator (whose exact value is always available) followed by an instance of a type appropriate to the discriminator value.
- A sequence type, which consists of a variable-length array of a single type; the length of the sequence is available at run-time.
- An array type, which consists of a fixed-shape multidimensional array of a single type.
- An interface type, which specifies the set of operations that an instance of that type must support.
- A value type, which specifies state as well as a set of operations that an instance of that type must support.

Entities in a request are restricted to values that satisfy these type constraints. The legal entities are shown in Figure 5.1. No particular representation for entities is defined.

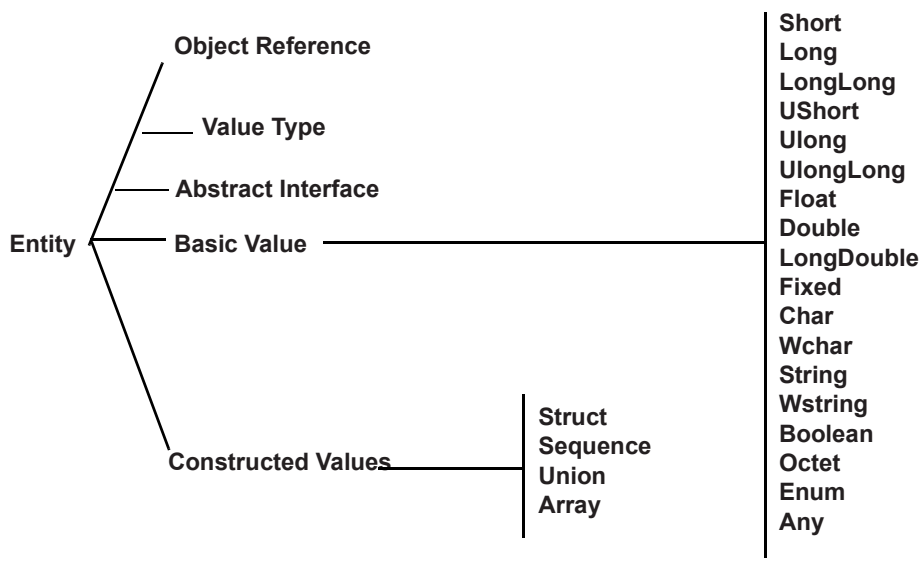

Figure 5.1 - Legal Values

5.3.5 Interfaces

An *interface* is a description of a set of possible operations that a client may request of an object, through that interface. It provides a syntactic description of how a service provided by an object supporting this interface, is accessed via this set of operations. An object *satisfies* an interface if it provides its service through the operations of the interface according to the specification of the operations (see Operations on page 9).

© ISO/IEC 2012 - All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

The *interface type* for a given interface is an object type, such that an object reference will satisfy the type, if and only if the referent object also satisfies the interface.

Interfaces are specified in IDL. Interface inheritance provides the composition mechanism for permitting an object to support multiple interfaces. The *principal interface* is simply the most-specific interface that the object supports, and consists of all operations in the transitive closure of the interface inheritance graph.

Interfaces satisfy the Liskov substitution principle. If interface A is derived from interface B, then a reference to an object that supports interface A can be used where the formal type of a parameter is declared to be B.

5.3.6 Value Types

A *value type* is an entity, which shares many of the characteristics of interfaces and structs. It is a description of both a set of operations that a client may request and of state that is accessible to a client. Instances of a value type are always local concrete implementations in some programming language.

A value type, in addition to the operations and state defined for itself, may also inherit from other value types, and through multiple inheritance support other interfaces.

Value types are specified in IDL.

An *abstract value type* describes a value type that is a "pure" bundle of operations with no state.

5.3.7 Abstract Interfaces

An *abstract interface* is an entity, which may at runtime represent either a regular interface (see Interfaces on page 8) or a value type (see Value Types on page 9). Like an abstract value type, it is a pure bundle of operations with no state. Unlike an abstract value type, it does not imply pass-by-value semantics, and unlike a regular interface type, it does not imply pass-by-reference semantics. Instead, the entity's runtime type determines which of these semantics are used.

5.3.8 Operations

An *operation* is an identifiable entity that denotes the indivisible primitive of service provision that can be requested. The act of requesting an operation is referred to as *invoking the operation*. An operation is identified by an *operation identifier*.

An operation has a *signature* that describes the legitimate values of request parameters and returned results. In particular, a *signature* consists of:

- A specification of the parameters required in requests for that operation.
- A specification of the result of the operation.
- An identification of the user exceptions that may be raised by an invocation of the operation.
- A specification of additional contextual information that may affect the invocation.
- An indication of the execution semantics the client should expect from an invocation of the operation.

Operations are (potentially) *generic*, meaning that a single operation can be uniformly invoked on objects with different implementations, possibly resulting in observably different behavior. Genericity is achieved in this model via interface inheritance in IDL and the total decoupling of implementation from interface specification.

© ISO/IEC 2012 - All rights reserved

AVA Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19500-1:2012

The general form for an operation signature is:

[oneway] <op_type_spec> <identifier> (param1, ..., paramL)
[raises(except1,...,exceptN)] [context(name1, ..., nameM)]

where:

- The optional **oneway** keyword indicates that best-effort semantics are expected of requests for this operation; the default semantics are exactly-once if the operation successfully returns results or at-most-once if an exception is returned.
- The **<op_type_spec>** is the type of the return result.
- The **<identifier>** provides a name for the operation in the interface.
- The operation parameters needed for the operation; they are flagged with the modifiers **in**, **out**, or **inout** to indicate the direction in which the information flows (with respect to the object performing the request).
- The optional **raises** expression indicates which user-defined exceptions can be signaled to terminate an invocation of this operation; if such an expression is not provided, no user-defined exceptions will be signaled.
- The optional **context** expression indicates which request context information will be available to the object implementation; no other contextual information is required to be transported with the request.

Parameters

A parameter is characterized by its mode and its type. The *mode* indicates whether the value should be passed from client to server (**in**), from server to client (**out**), or both (**inout**). The parameter's type constrains the possible value, which may be passed in the directions dictated by the mode.

Return Result

The return result is a distinguished **out** parameter.

Exceptions

An *exception* is an indication that an operation request was not performed successfully. An exception may be accompanied by additional, exception-specific information.

The additional, exception-specific information is a specialized form of record. As a record, it may consist of any of the types described in Types on page 7.

All signatures implicitly include the system exceptions; the standard system exceptions are described in System Exceptions on page 148.

Contexts

A *request context* provides additional, operation-specific information that may affect the performance of a request.

Execution Semantics

Two styles of execution semantics are defined by the object model:

- At-most-once: if an operation request returns successfully, it was performed exactly once; if it returns an exception indication, it was performed at-most-once.

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

- Best-effort: a best-effort operation is a request-only operation (i.e., it cannot return any results and the requester never synchronizes with the completion, if any, of the request).

The execution semantics to be expected is associated with an operation. This prevents a client and object implementation from assuming different execution semantics.

Note that a client is able to invoke an at-most-once operation in a synchronous or deferred-synchronous manner.

5.3.9 Attributes

An interface may have attributes. An attribute is logically equivalent to declaring a pair of accessor functions: one to retrieve the value of the attribute and one to set the value of the attribute.

An attribute may be read-only, in which case only the retrieval accessor function is defined.

5.4 Object Implementation

This sub clause defines the concepts associated with object implementation (i.e., the concepts relevant to realizing the behavior of objects in a computational system).

The implementation of an object system carries out the computational activities needed to effect the behavior of requested services. These activities may include computing the results of the request and updating the system state. In the process, additional requests may be issued.

The implementation model consists of two parts: the execution model and the construction model. The execution model describes how services are performed. The construction model describes how services are defined.

5.4.1 The Execution Model: Performing Services

A requested service is performed in a computational system by executing code that operates upon some data. The data represents a component of the state of the computational system. The code performs the requested service, which may change the state of the system.

Code that is executed to perform a service is called a *method*. A method is an immutable description of a computation that can be interpreted by an execution engine. A method has an immutable attribute called a *method format* that defines the set of execution engines that can interpret the method. An *execution engine* is an abstract machine (not a program) that can interpret methods of certain formats, causing the described computations to be performed. An execution engine defines a dynamic context for the execution of a method. The execution of a method is called a *method activation*.

When a client issues a request, a method of the target object is called. The input parameters passed by the requestor are passed to the method and the output and input-output parameters and return result value (or exception and its parameters) are passed back to the requestor.

Performing a requested service causes a method to execute that may operate upon an object's persistent state. If the persistent form of the method or state is not accessible to the execution engine, it may be necessary to first copy the method or state into an execution context. This process is called *activation*; the reverse process is called *deactivation*.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

5.4.2 The Construction Model

A computational object system must provide mechanisms for realizing behavior of requests. These mechanisms include definitions of object state, definitions of methods, and definitions of how the object infrastructure is to select the methods to execute and to select the relevant portions of object state to be made accessible to the methods. Mechanisms must also be provided to describe the concrete actions associated with object creation, such as association of the new object with appropriate methods.

An *object implementation*—or *implementation*, for short—is a definition that provides the information needed to create an object and to allow the object to participate in providing an appropriate set of services. An implementation typically includes, among other things, definitions of the methods that operate upon the state of an object. It also typically includes information about the intended types of the object.

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

6 CORBA Overview

6.1 General

The Common Object Request Broker Architecture (CORBA) is structured to allow integration of a wide variety of object systems. The motivation for some of the features may not be apparent at first, but as we discuss the range of implementations, policies, optimizations, and usages we expect to encompass, the value of the flexibility becomes clearer.

6.2 Structure of an Object Request Broker

Figure 6.1 shows a request being sent by a client to an object implementation. The Client is the entity that wishes to perform an operation on the object and the Object Implementation is the code and data that actually implements the object.

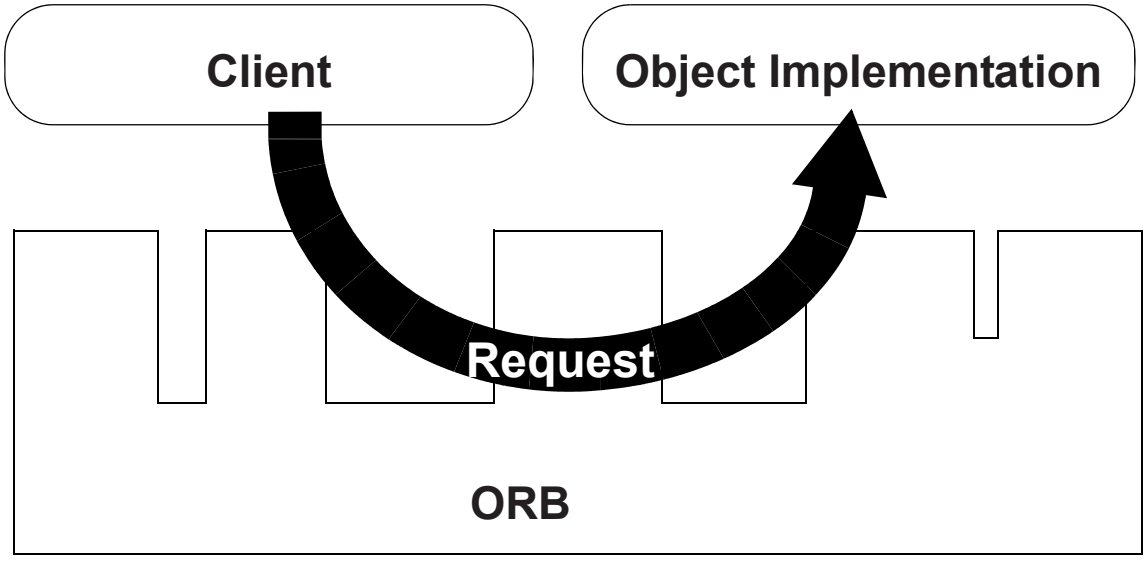

Figure 6.1 - A Request Being Sent Through the Object Request Broker

The ORB is responsible for all of the mechanisms required to find the object implementation for the request, to prepare the object implementation to receive the request, and to communicate the data making up the request. The interface the client sees is completely independent of where the object is located, what programming language it is implemented in, or any other aspect that is not reflected in the object's interface.

Figure 6.2 shows the structure of an individual Object Request Broker (ORB). The interfaces to the ORB are shown by striped boxes, and the arrows indicate whether the ORB is called or performs an up-call across the interface.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

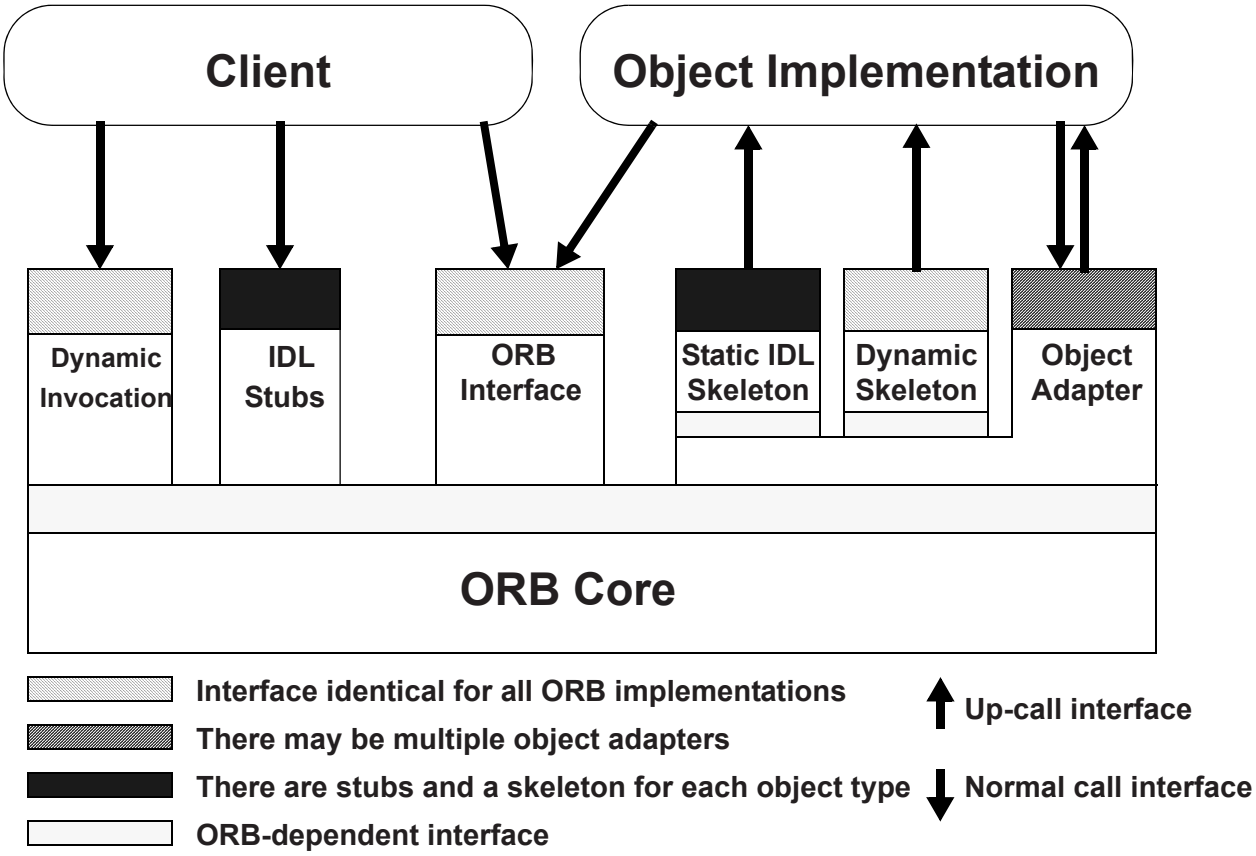

Figure 6.2 - The Structure of Object Request Interfaces

To make a request, the Client can use the Dynamic Invocation interface (the same interface independent of the target object's interface) or an IDL stub (the specific stub depending on the interface of the target object). The Client can also directly interact with the ORB for some functions.

The Object Implementation receives a request as an up-call either through the IDL generated skeleton or through a dynamic skeleton. The Object Implementation may call the Object Adapter and the ORB while processing a request or at other times.

Definitions of the interfaces to objects can be defined in two ways. 1) Interfaces can be defined statically in an interface definition language, called the OMG Interface Definition Language (IDL). This language defines the types of objects according to the operations that may be performed on them and the parameters to those operations. 2) Alternatively, or in addition, interfaces can be added to an Interface Repository service. This service represents the components of an interface as objects, permitting run-time access to these components. In any ORB implementation, the Interface Definition Language (which may be extended beyond its definition in this document) and the Interface Repository have equivalent expressive power.

The client performs a request by having access to an Object Reference for an object and knowing the type of the object and the desired operation to be performed. The client initiates the request by calling stub routines that are specific to the object or by constructing the request dynamically (see Figure 6.3).

© ISO/IEC 2012 - All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

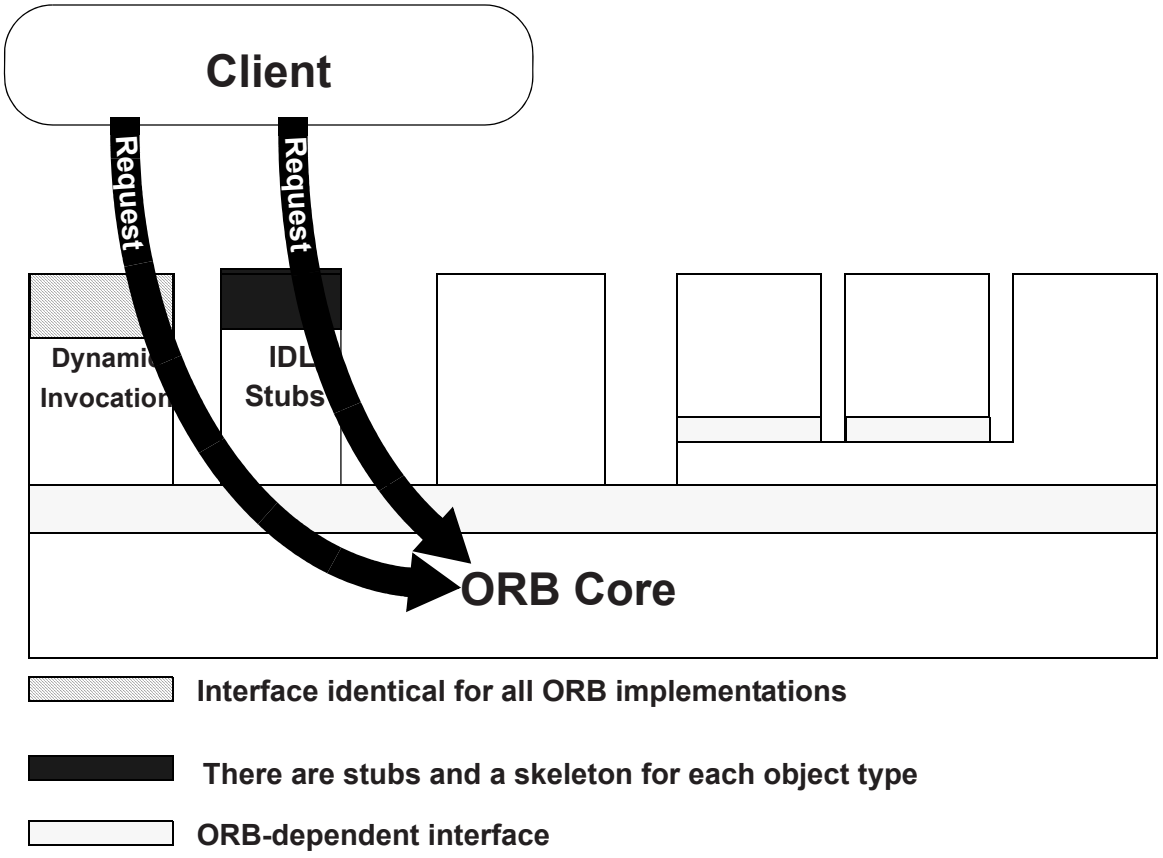

Figure 6.3 - A Client Using the Stub or Dynamic Invocation Interface

The dynamic and stub interface for invoking a request satisfy the same request semantics, and the receiver of the message cannot tell how the request was invoked.

The ORB locates the appropriate implementation code, transmits parameters, and transfers control to the Object Implementation through an IDL skeleton or a dynamic skeleton (see Figure 6.4). Skeletons are specific to the interface and the object adapter. In performing the request, the object implementation may obtain some services from the ORB through the Object Adapter. When the request is complete, control and output values are returned to the client.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19500-1:2012

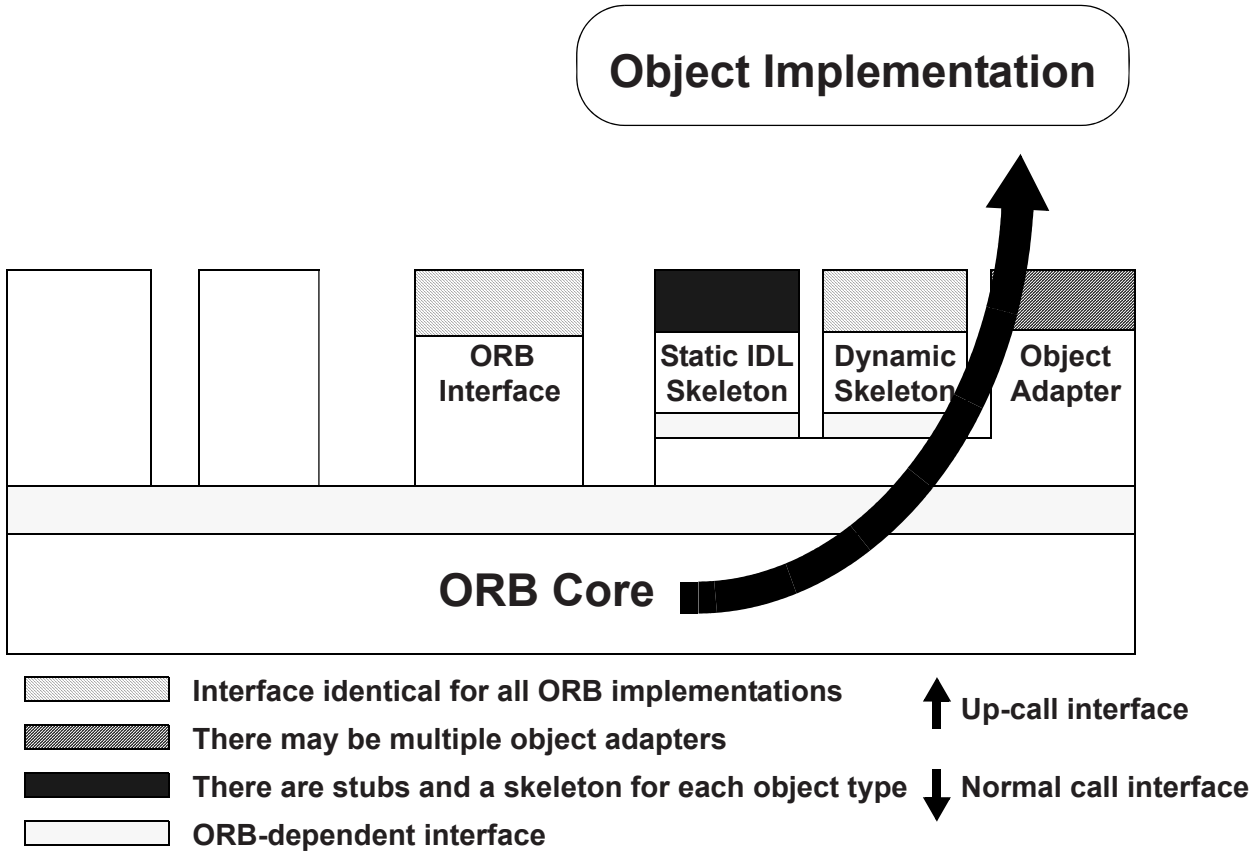

Figure 6.4 - An Object Implementation Receiving a Request

The Object Implementation may choose which Object Adapter to use. This decision is based on what kind of services the Object Implementation requires.

Figure 6.5 shows how interface and implementation information is made available to clients and object implementations. The interface is defined in IDL and/or in the Interface Repository; the definition is used to generate the client Stubs and the object implementation Skeletons.

© ISO/IEC 2012 - All rights reserved
Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

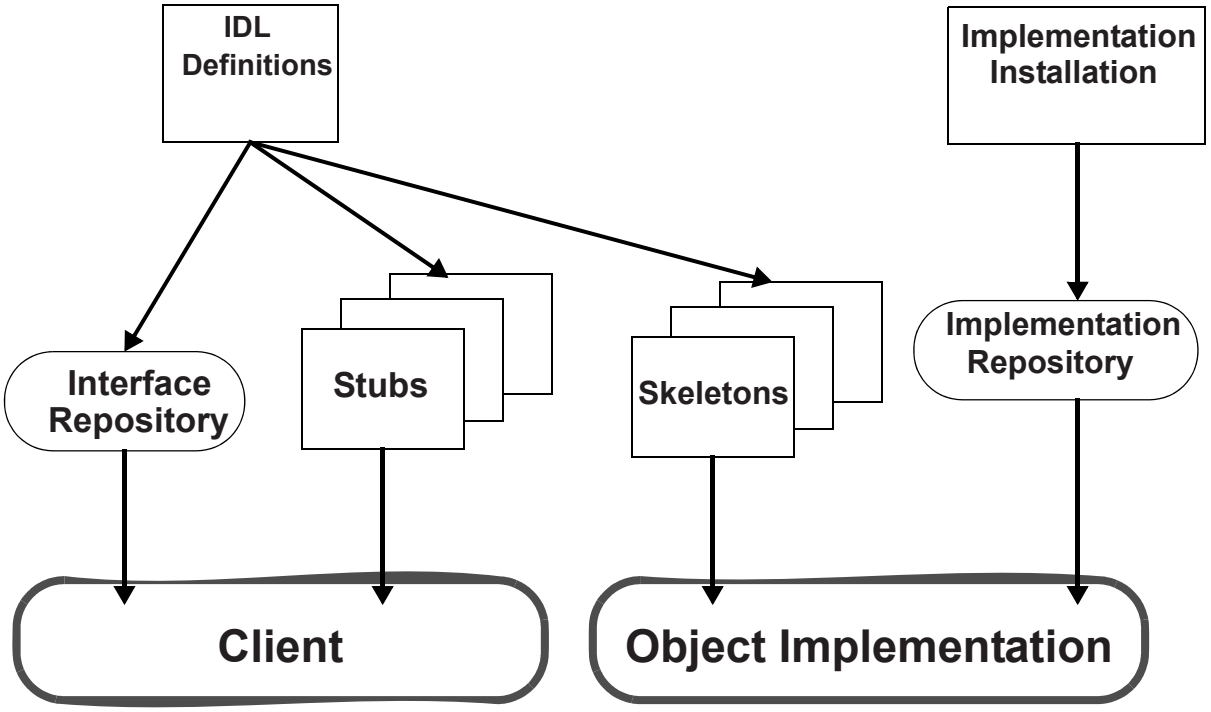

Figure 6.5 - Interface and Implementation Repositories

The object implementation information is provided at installation time and is stored in the Implementation Repository for use during request delivery.

6.2.1 Object Request Broker

In the architecture, the ORB is not required to be implemented as a single component, but rather it is defined by its interfaces. Any ORB implementation that provides the appropriate interface is acceptable. The interface is organized into three categories:

1. Operations that are the same for all ORB implementations.
2. Operations that are specific to particular types of objects.
3. Operations that are specific to particular styles of object implementations.

Different ORBs may make quite different implementation choices, and, together with the IDL compilers, repositories, and various Object Adapters, provide a set of services to clients and implementations of objects that have different properties and qualities.

There may be multiple ORB implementations (also described as multiple ORBs), which have different representations for object references and different means of performing invocations. It may be possible for a client to simultaneously have access to two object references managed by different ORB implementations. When two ORBs are intended to work together, those ORBs must be able to distinguish their object references. It is not the responsibility of the client to do so.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

The ORB Core is that part of the ORB that provides the basic representation of objects and communication of requests. CORBA is designed to support different object mechanisms, and it does so by structuring the ORB with components above the ORB Core, which provide interfaces that can mask the differences between ORB Cores.

6.2.2 Clients

A client of an object has access to an object reference for the object, and invokes operations on the object. A client knows only the logical structure of the object according to its interface and experiences the behavior of the object through invocations. Although we will generally consider a client to be a program or process initiating requests on an object, it is important to recognize that something is a client relative to a particular object. For example, the implementation of one object may be a client of other objects.

Clients generally see objects and ORB interfaces through the perspective of a language mapping, bringing the ORB right up to the programmer's level. Clients are maximally portable and should be able to work without source changes on any ORB that supports the desired language mapping with any object instance that implements the desired interface. Clients have no knowledge of the implementation of the object, which object adapter is used by the implementation, or which ORB is used to access it.

6.2.3 Object Implementations

An object implementation provides the semantics of the object, usually by defining data for the object instance and code for the object's methods. Often the implementation will use other objects or additional software to implement the behavior of the object. In some cases, the primary function of the object is to have side-effects on other things that are not objects.

A variety of object implementations can be supported, including separate servers, libraries, a program per method, an encapsulated application, an object-oriented database, etc. Through the use of additional object adapters, it is possible to support virtually any style of object implementation.

Generally, object implementations do not depend on the ORB or how the client invokes the object. Object implementations may select interfaces to ORB-dependent services by the choice of Object Adapter.

6.2.4 Object References

An Object Reference is the information needed to specify an object within an ORB. Both clients and object implementations have an opaque notion of object references according to the language mapping, and thus are insulated from the actual representation of them. Two ORB implementations may differ in their choice of Object Reference representations.

The representation of an object reference handed to a client is only valid for the lifetime of that client.

All ORBs must provide the same language mapping to an object reference (usually referred to as an Object) for a particular programming language. This permits a program written in a particular language to access object references independent of the particular ORB. The language mapping may also provide additional ways to access object references in a typed way for the convenience of the programmer.

There is a distinguished object reference, guaranteed to be different from all object references, that denotes no object.

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

6.2.5 OMG Interface Definition Language

The OMG Interface Definition Language (IDL) defines the types of objects by specifying their interfaces. An interface consists of a set of named operations and the parameters to those operations. Note that although IDL provides the conceptual framework for describing the objects manipulated by the ORB, it is not necessary for there to be IDL source code available for the ORB to work. As long as the equivalent information is available in the form of stub routines or a run-time interface repository, a particular ORB may be able to function correctly.

IDL is the means by which a particular object implementation tells its potential clients what operations are available and how they should be invoked. From the IDL definitions, it is possible to map CORBA objects into particular programming languages or object systems.

6.2.6 Mapping of IDL to Programming Languages

Different object-oriented or non-object-oriented programming languages may prefer to access CORBA objects in different ways. For object-oriented languages, it may be desirable to see CORBA objects as programming language objects. Even for non-object-oriented languages, it is a good idea to hide the exact ORB representation of the object reference, method names, etc. A particular mapping of IDL to a programming language should be the same for all ORB implementations. Language mapping includes definition of the language-specific data types and procedure interfaces to access objects through the ORB. It includes the structure of the client stub interface (not required for object-oriented languages), the dynamic invocation interface, the implementation skeleton, the object adapters, and the direct ORB interface.

A language mapping also defines the interaction between object invocations and the threads of control in the client or implementation. The most common mappings provide synchronous calls, in that the routine returns when the object operation completes. Additional mappings may be provided to allow a call to be initiated and control returned to the program. In such cases, additional language-specific routines must be provided to synchronize the program's threads of control with the object invocation.

6.2.7 Client Stubs

Generally, the client stubs will present access to the IDL-defined operations on an object in a way that is easy for programmers to predict once they are familiar with IDL and the language mapping for the particular programming language. The stubs make calls on the rest of the ORB using interfaces that are private to, and presumably optimized for, the particular ORB Core. If more than one ORB is available, there may be different stubs corresponding to the different ORBs. In this case, it is necessary for the ORB and language mapping to cooperate to associate the correct stubs with the particular object reference.

6.2.8 Dynamic Invocation Interface

An interface is also available that allows the dynamic construction of object invocations, that is, rather than calling a stub routine that is specific to a particular operation on a particular object, a client may specify the object to be invoked, the operation to be performed, and the set of parameters for the operation through a call or sequence of calls. The client code must supply information about the operation to be performed and the types of the parameters being passed (perhaps obtaining it from an Interface Repository or other run-time source). The nature of the dynamic invocation interface may vary substantially from one programming language mapping to another.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

6.2.9 Implementation Skeleton

For a particular language mapping, and possibly depending on the object adapter, there will be an interface to the methods that implement each type of object. The interface will generally be an up-call interface, in that the object implementation writes routines that conform to the interface and the ORB calls them through the skeleton.

The existence of a skeleton does not imply the existence of a corresponding client stub (clients can also make requests via the dynamic invocation interface).

It is possible to write an object adapter that does not use skeletons to invoke implementation methods. For example, it may be possible to create implementations dynamically for languages such as Smalltalk.

6.2.10 Dynamic Skeleton Interface

An interface is available, which allows dynamic handling of object invocations. That is, rather than being accessed through a skeleton that is specific to a particular operation, an object's implementation is reached through an interface that provides access to the operation name and parameters in a manner analogous to the client side's Dynamic Invocation Interface. Purely static knowledge of those parameters may be used, or dynamic knowledge (perhaps determined through an Interface Repository) may also be used, to determine the parameters.

The implementation code must provide descriptions of all the operation parameters to the ORB, and the ORB provides the values of any input parameters for use in performing the operation. The implementation code provides the values of any output parameters, or an exception, to the ORB after performing the operation. The nature of the dynamic skeleton interface may vary substantially from one programming language mapping or object adapter to another, but will typically be an up-call interface.

Dynamic skeletons may be invoked both through client stubs and through the dynamic invocation interface; either style of client request construction interface provides identical results.

6.2.11 Object Adapters

An object adapter is the primary way that an object implementation accesses services provided by the ORB. There are expected to be a few object adapters that will be widely available, with interfaces that are appropriate for specific kinds of objects. Services provided by the ORB through an Object Adapter often include: generation and interpretation of object references, method invocation, security of interactions, object and implementation activation and deactivation, mapping object references to implementations, and registration of implementations.

The wide range of object granularities, lifetimes, policies, implementation styles, and other properties make it difficult for the ORB Core to provide a single interface that is convenient and efficient for all objects. Thus, through Object Adapters, it is possible for the ORB to target particular groups of object implementations that have similar requirements with interfaces tailored to them.

6.2.12 ORB Interface

The ORB Interface is the interface that goes directly to the ORB, which is the same for all ORBs and does not depend on the object's interface or object adapter. Because most of the functionality of the ORB is provided through the object adapter, stubs, skeleton, or dynamic invocation, there are only a few operations that are common across all objects. These operations are useful to both clients and implementations of objects.

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

6.2.13 Interface Repository

The Interface Repository is a service that provides persistent objects that represent the IDL information in a form available at run-time. The Interface Repository information may be used by the ORB to perform requests. Moreover, using the information in the Interface Repository, it is possible for a program to encounter an object whose interface was not known when the program was compiled, yet, be able to determine what operations are valid on the object and make an invocation on it.

In addition to its role in the functioning of the ORB, the Interface Repository is a common place to store additional information associated with interfaces to ORB objects. For example, debugging information, libraries of stubs or skeletons, routines that can format or browse particular kinds of objects might be associated with the Interface Repository.

6.2.14 Implementation Repository

The Implementation Repository contains information that allows the ORB to locate and activate implementations of objects. Although most of the information in the Implementation Repository is specific to an ORB or operating environment, the Implementation Repository is the conventional place for recording such information. Ordinarily, installation of implementations and control of policies related to the activation and execution of object implementations is done through operations on the Implementation Repository.

In addition to its role in the functioning of the ORB, the Implementation Repository is a common place to store additional information associated with implementations of ORB objects. For example, debugging information, administrative control, resource allocation, security, etc., might be associated with the Implementation Repository.

6.3 Example ORBs

There are a wide variety of ORB implementations possible within the Common ORB Architecture. This sub clause will illustrate some of the different options. Note that a particular ORB might support multiple options and protocols for communication.

6.3.1 Client- and Implementation-resident ORB

If there is a suitable communication mechanism present, an ORB can be implemented in routines resident in the clients and implementations. The stubs in the client either use a location-transparent IPC mechanism or directly access a location service to establish communication with the implementations. Code linked with the implementation is responsible for setting up appropriate databases for use by clients.

6.3.2 Server-based ORB

To centralize the management of the ORB, all clients and implementations can communicate with one or more servers whose job it is to route requests from clients to implementations. The ORB could be a normal program as far as the underlying operating system is concerned, and normal IPC could be used to communicate with the ORB.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

6.3.3 System-based ORB

To enhance security, robustness, and performance, the ORB could be provided as a basic service of the underlying operating system. Object references could be made unforgeable, reducing the expense of authentication on each request. Because the operating system could know the location and structure of clients and implementations, it would be possible for a variety of optimizations to be implemented, for example, avoiding marshalling when both are on the same machine.

6.3.4 Library-based ORB

For objects that are light-weight and whose implementations can be shared, the implementation might actually be in a library. In this case, the stubs could be the actual methods. This assumes that it is possible for a client program to get access to the data for the objects and that the implementation trusts the client not to damage the data.

6.4 Structure of a Client

A client of an object has an object reference that refers to that object. An object reference is a token that may be invoked or passed as a parameter to an invocation on a different object. Invocation of an object involves specifying the object to be invoked, the operation to be performed, and parameters to be given to the operation or returned from it.

The ORB manages the control transfer and data transfer to the object implementation and back to the client. In the event that the ORB cannot complete the invocation, an exception response is provided. Ordinarily, a client calls a routine in its program that performs the invocation and returns when the operation is complete.

Clients access object-type-specific stubs as library routines in their program (see Figure 6.6). The client program thus sees routines callable in the normal way in its programming language. All implementations will provide a language-specific data type to use to refer to objects, often an opaque pointer. The client then passes that object reference to the stub routines to initiate an invocation. The stubs have access to the object reference representation and interact with the ORB to perform the invocation. (See the *C Language Mapping* specification for additional, general information on language mapping of object references.)

© ISO/IEC 2012 – All rights reserved
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

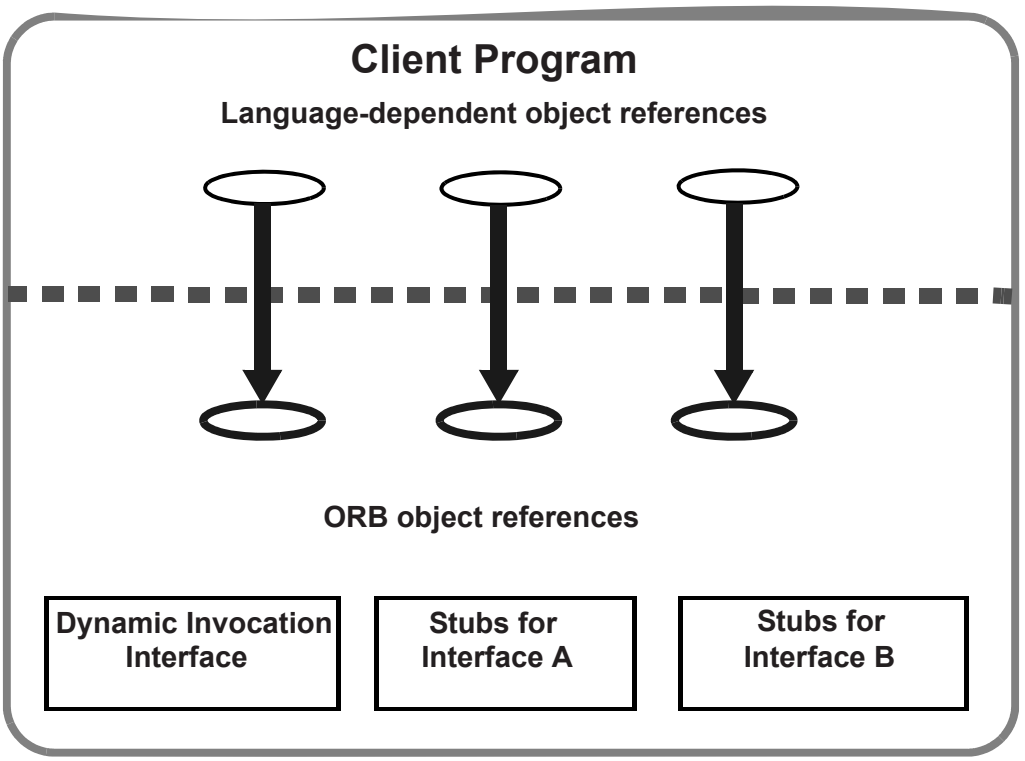

Figure 6.6 - The Structure of a Typical Client

An alternative set of library code is available to perform invocations on objects, for example when the object was not defined at compile time. In that case, the client program provides additional information to name the type of the object and the method being invoked, and performs a sequence of calls to specify the parameters and initiate the invocation.

Clients most commonly obtain object references by receiving them as output parameters from invocations on other objects for which they have references. When a client is also an implementation, it receives object references as input parameters on invocations to objects it implements. An object reference can also be converted to a string that can be stored in files or preserved or communicated by different means and subsequently turned back into an object reference by the ORB that produced the string.

6.5 Structure of an Object Implementation

An object implementation provides the actual state and behavior of an object. The object implementation can be structured in a variety of ways. Besides defining the methods for the operations themselves, an implementation will usually define procedures for activating and deactivating objects and will use other objects or non-object facilities to make the object state persistent, to control access to the object, as well as to implement the methods.

The object implementation (see Figure 6.7) interacts with the ORB in a variety of ways to establish its identity, to create new objects, and to obtain ORB-dependent services. It primarily does this via access to an Object Adapter, which provides an interface to ORB services that is convenient for a particular style of object implementation.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

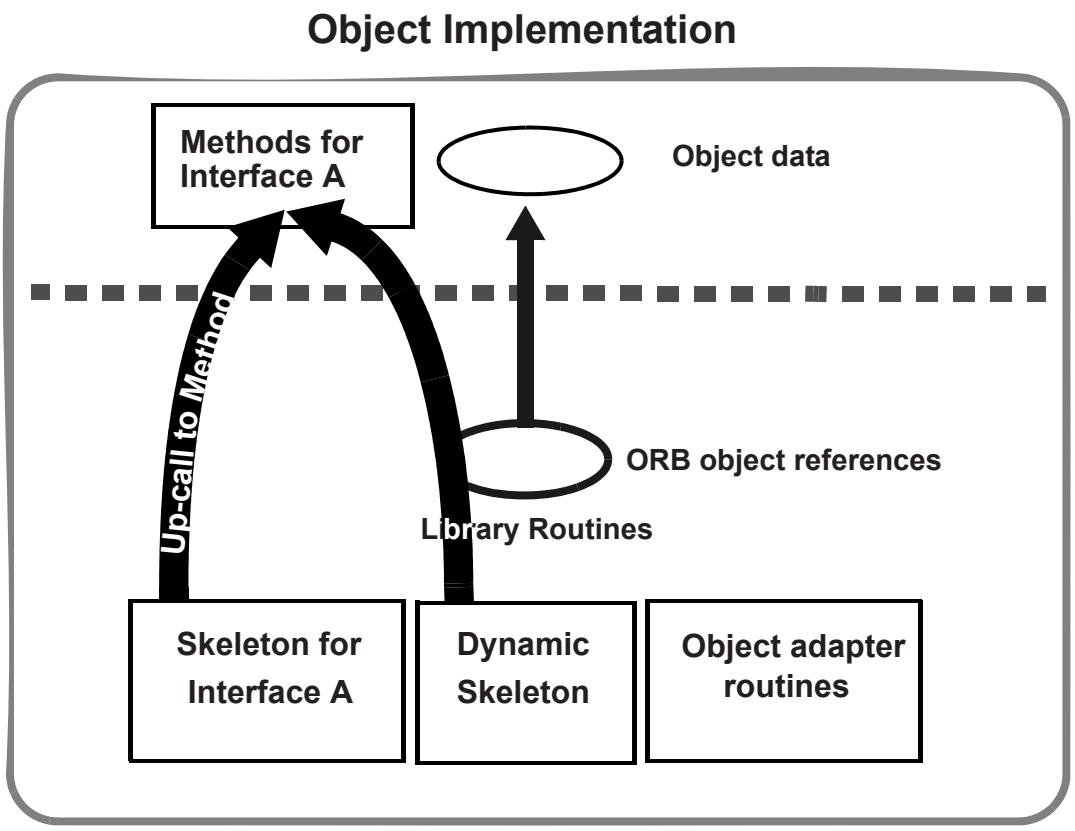

Figure 6.7 - The Structure of a Typical Object Implementation

Because of the range of possible object implementations, it is difficult to be definitive about how an object implementation is structured. See the Portable Object Adapter clauses.

When an invocation occurs, the ORB Core, object adapter, and skeleton arrange that a call is made to the appropriate method of the implementation. A parameter to that method specifies the object being invoked, which the method can use to locate the data for the object. Additional parameters are supplied according to the skeleton definition. When the method is complete, it returns, causing output parameters or exception results to be transmitted back to the client.

When a new object is created, the ORB may be notified so that it knows where to find the implementation for that object. Usually, the implementation also registers itself as implementing objects of a particular interface, and specifies how to start up the implementation if it is not already running.

Most object implementations provide their behavior using facilities in addition to the ORB and object adapter. For example, although the Portable Object Adapter provides some persistent data associated with an object (its OID or Object ID), that relatively small amount of data is typically used as an identifier for the actual object data stored in a storage service of the object implementation's choosing. With this structure, it is not only possible for different object implementations to use the same storage service, it is also possible for objects to choose the service that is most appropriate for them.

© ISO/IEC 2012 - All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

6.6 Structure of an Object Adapter

An object adapter (see Figure 6.8) is the primary means for an object implementation to access ORB services such as object reference generation. An object adapter exports a public interface to the object implementation, and a private interface to the skeleton. It is built on a private ORB-dependent interface.

Object adapters are responsible for the following functions:

- Generation and interpretation of object references
- Method invocation
- Security of interactions
- Object and implementation activation and deactivation
- Mapping object references to the corresponding object implementations
- Registration of implementations

These functions are performed using the ORB Core and any additional components necessary. Often, an object adapter will maintain its own state to accomplish its tasks. It may be possible for a particular object adapter to delegate one or more of its responsibilities to the Core upon which it is constructed.

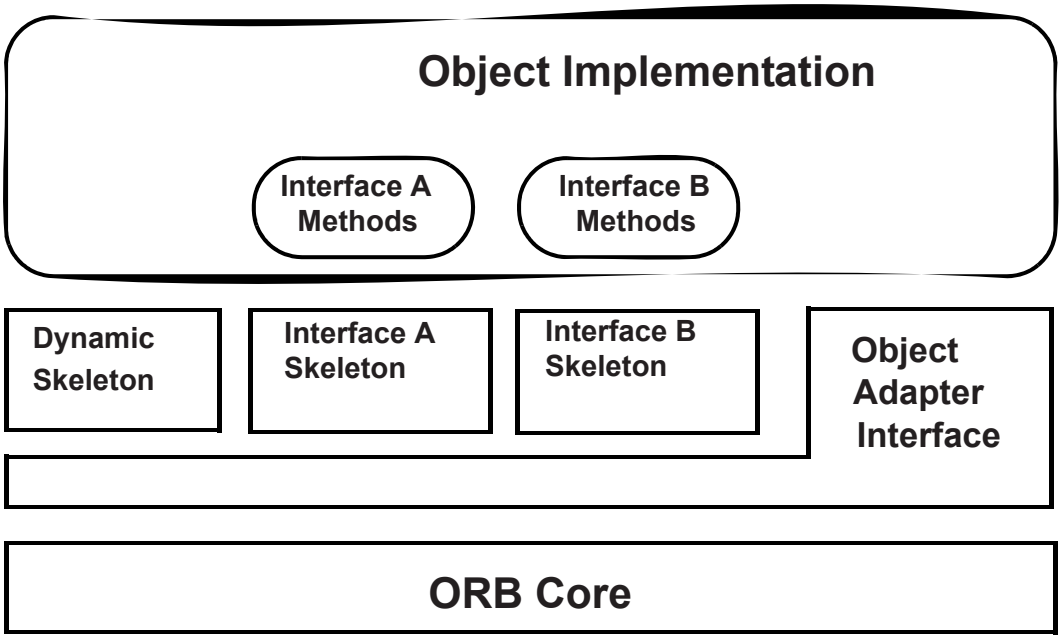

Figure 6.8 - The Structure of a Typical Object Adapter

As shown in Figure 6.8, the Object Adapter is implicitly involved in invocation of the methods, although the direct interface is through the skeletons. For example, the Object Adapter may be involved in activating the implementation or authenticating the request.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

The Object Adapter defines most of the services from the ORB that the Object Implementation can depend on. Different ORBs will provide different levels of service and different operating environments may provide some properties implicitly and require others to be added by the Object Adapter. For example, it is common for Object Implementations to want to store certain values in the object reference for easy identification of the object on an invocation. If the Object Adapter allows the implementation to specify such values when a new object is created, it may be able to store them in the object reference for those ORBs that permit it. If the ORB Core does not provide this feature, the Object Adapter would record the value in its own storage and provide it to the implementation on an invocation. With Object Adapters, it is possible for an Object Implementation to have access to a service whether or not it is implemented in the ORB Core — if the ORB Core provides it, the adapter simply provides an interface to it; if not, the adapter must implement it on top of the ORB Core. Every instance of a particular adapter must provide the same interface and service for all the ORBs it is implemented on.

It is also not necessary for all Object Adapters to provide the same interface or functionality. Some Object Implementations have special requirements. For example, an object-oriented database system may wish to implicitly register its many thousands of objects without doing individual calls to the Object Adapter. In such a case, it would be impractical and unnecessary for the object adapter to maintain any per-object state. By using an object adapter interface that is tuned towards such object implementations, it is possible to take advantage of particular ORB Core details to provide the most effective access to the ORB.

6.7 CORBA Required Object Adapter

There are a variety of possible object adapters; however, since the object adapter interface is something that object implementations depend on, it is desirable that there be as few as practical. Most object adapters are designed to cover a range of object implementations, so only when an implementation requires radically different services or interfaces should a new object adapter be considered. In this sub clause, we briefly describe the object adapter defined in this International Standard.

6.7.1 Portable Object Adapter

This International Standard defines a Portable Object Adapter that can be used for most ORB objects with conventional implementations. (See the *Portable Object Adapter* clause for more information.) The intent of the POA, as its name suggests, is to provide an Object Adapter that can be used with multiple ORBs with a minimum of rewriting needed to deal with different vendors' implementations.

This International Standard allows several ways of using servers but it does not deal with the administrative issues of starting server programs. Once started, however, there can be a servant started and ended for a single method call, a separate servant for each object, or a shared servant for all instances of the object type. It allows for groups of objects to be associated by means of being registered with different instances of the POA object and allows implementations to specify their own activation techniques. If the implementation is not active when an invocation is performed, the POA will start one. The POA is specified in IDL, so its mapping to languages is largely automatic, following the language mapping rules. (The primary task left for a language mapping is the definition of the Servant type.)

6.8 The Integration of Foreign Object Systems

The Common ORB Architecture is designed to allow interoperation with a wide range of object systems (see Figure 6.9). Because there are many existing object systems, a common desire will be to allow the objects in those systems to be accessible via the ORB. For those object systems that are ORBs themselves, they may be connected to other ORBs through the mechanisms described throughout this manual.

© ISO/IEC 2012 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

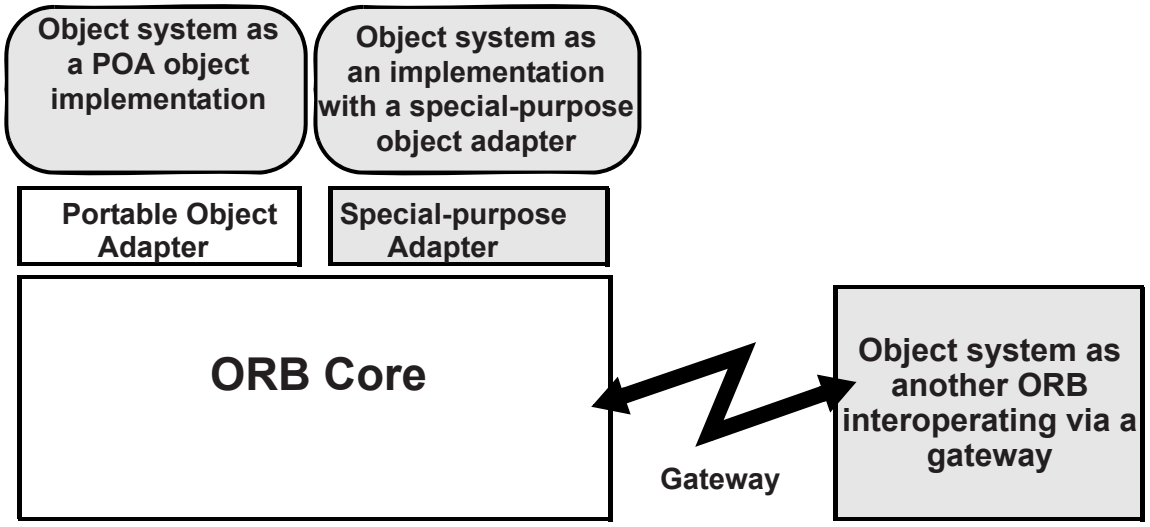

Figure 6.9 - Different Ways to Integrate Foreign Object Systems

For object systems that simply want to map their objects into ORB objects and receive invocations through the ORB, one approach is to have those object systems appear to be implementations of the corresponding ORB objects. The object system would register its objects with the ORB and handle incoming requests, and could act like a client and perform outgoing requests.

In some cases, it will be impractical for another object system to act like a POA object implementation. An object adapter could be designed for objects that are created in conjunction with the ORB and that are primarily invoked through the ORB. Another object system may wish to create objects without consulting the ORB, and might expect most invocations to occur within itself rather than through the ORB. In such a case, a more appropriate object adapter might allow objects to be implicitly registered when they are passed through the ORB.

© ISO/IEC 2012 - All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 19500-1:2012

© ISO/IEC 2012 – All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 19500-1:2012