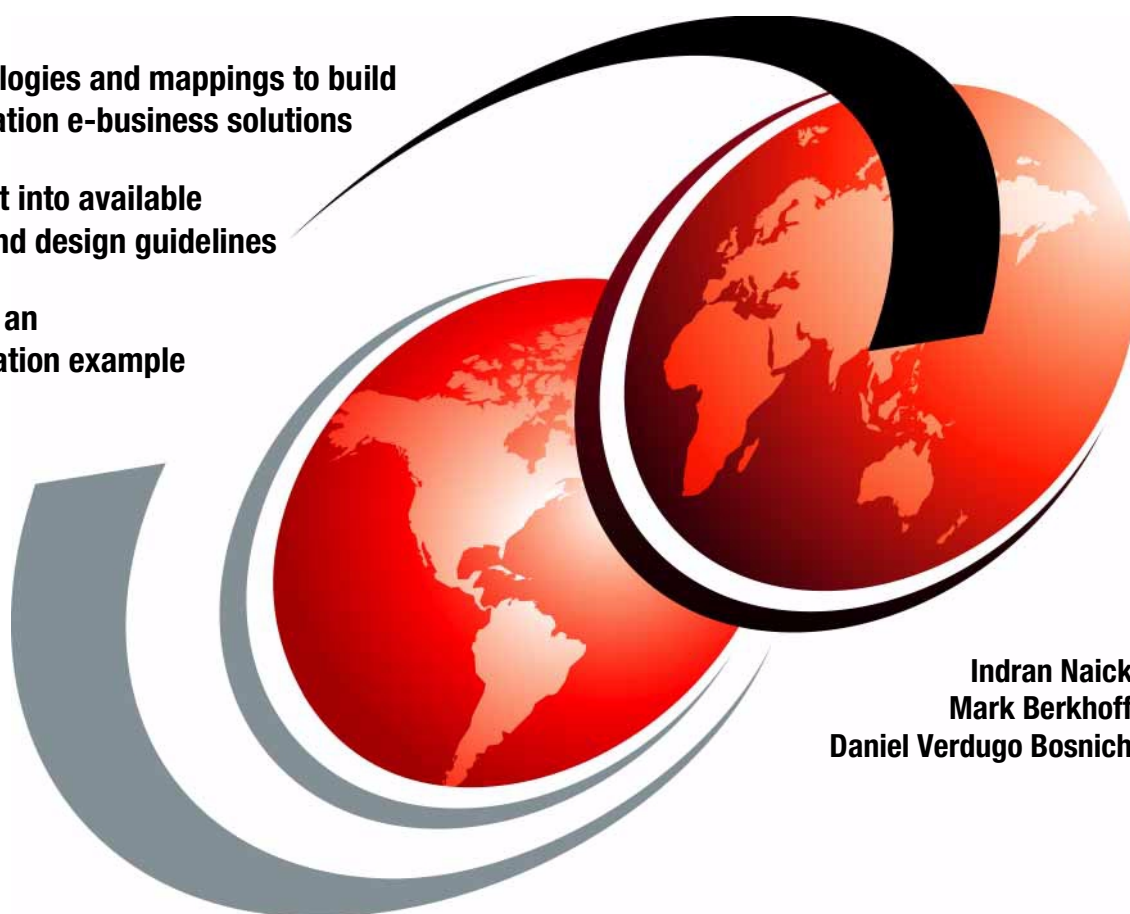


# **Business-to-Business Integration Using MQSeries and MQSI Patterns for e-business Series**

**Select topologies and mappings to build  
B2B Integration e-business solutions**

**Gain insight into available  
products and design guidelines**

**Learn from an  
implementation example**



**Indran Naick  
Mark Berkhoff  
Daniel Verdugo Bosnich**

**[ibm.com/redbooks](http://ibm.com/redbooks)**

# **Redbooks**





International Technical Support Organization

**Business-to-Business Integration  
Using MQSeries and MQSI  
Patterns for e-business Series**

December 2000

**Take Note!**

Before using this information and the product it supports, be sure to read the general information in Appendix D, "Special notices" on page 317.

**First Edition (December 2000)**

Comments may be addressed to:  
IBM Corporation, International Technical Support Organization  
Dept. JN9B Building 003 Internal Zip 2834  
11400 Burnet Road  
Austin, Texas 78758-3493

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2000. All rights reserved.  
Note to U.S Government Users – Documentation related to restricted rights – Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

---

## Contents

<b>Figures</b> .....	xiii
<b>Tables</b> .....	xvii
<b>Preface</b> .....	xix
The team that wrote this redbook .....	xix
Comments welcome .....	xxi
<hr/>	
<b>Part 1. Introduction</b> .....	1
<b>Chapter 1. e-business and B2B integration</b> .....	3
1.1 About e-business .....	3
1.1.1 Business transformation and Innovation .....	4
1.1.2 e-business value .....	6
1.2 e-business applications: A simplified classification scheme .....	7
1.2.1 Intra-business applications .....	8
1.2.2 Inter-business applications .....	9
1.3 Inter-business: B2Bi .....	12
1.3.1 EDI and B2B Integration .....	12
1.3.2 B2B and A2A (Application-to-Application) .....	13
1.3.3 B2B integration challenges .....	13
1.3.4 B2B integration solution components .....	14
1.4 Summary .....	16
<b>Chapter 2. Introduction to business patterns</b> .....	17
2.1 Patterns for e-business .....	17
2.1.1 Patterns for e-business and design patterns .....	18
2.1.2 Components of the patterns for e-business .....	19
2.1.3 Defined patterns for e-business .....	19
2.1.4 How to use these patterns .....	22
2.1.5 Patterns for e-business Web sites .....	22
2.2 The Business-to-Business Integration pattern .....	23
2.3 The Application Framework for e-business .....	25
2.4 Structure of this redbook .....	25
<b>Chapter 3. Choosing the application topology</b> .....	27
3.1 Influencing factors .....	27
3.1.1 Business context .....	28
3.1.2 Functional components .....	28
3.2 Application topology overview .....	28
3.3 Application Topology 1: Document exchange .....	30

3.3.1 Business driver . . . . .	31
3.3.2 Considerations . . . . .	31
3.3.3 Examples . . . . .	31
3.4 Topology 2: Direct with adapter/bridge . . . . .	32
3.4.1 Business driver . . . . .	33
3.4.2 Considerations . . . . .	33
3.4.3 Examples . . . . .	33
3.5 Topology 3: Message broker . . . . .	34
3.5.1 Business driver . . . . .	34
3.5.2 Considerations . . . . .	34
3.5.3 Examples . . . . .	35
3.6 Topology 4: Managed business protocol . . . . .	35
3.6.1 Business driver . . . . .	36
3.6.2 Considerations . . . . .	37
3.6.3 Examples . . . . .	37
3.7 Topology 5: Managed business protocol and process . . . . .	37
3.7.1 Business driver . . . . .	39
3.7.2 Considerations . . . . .	39
3.7.3 Examples . . . . .	40
3.8 Topologies summary . . . . .	40
<b>Chapter 4. Choosing the runtime topology . . . . .</b>	<b>41</b>
4.1 Runtime topologies . . . . .	41
4.2 Topology 1: Document exchange . . . . .	42
4.2.1 Illustrative Example 1: Document runtime topology . . . . .	42
4.2.2 Illustrative Example 2: EDI - Internet Runtime topology . . . . .	44
4.2.3 Summary . . . . .	46
4.3 Topology 2 - Direct with adapter/bridge . . . . .	46
4.3.1 Illustrative Example 1- Shared middleware . . . . .	47
4.3.2 Illustrative Example 2- Open standards . . . . .	49
4.3.3 Summary . . . . .	50
4.4 Topology 3 - Message broker . . . . .	51
4.4.1 Illustrative Example 1- Shared middleware . . . . .	52
4.4.2 Illustrative Example 2 - Open standards . . . . .	52
4.4.3 Summary . . . . .	53
4.5 An introduction to the node types . . . . .	53
4.5.1 Extranet . . . . .	53
4.5.2 Intranet . . . . .	54
4.5.3 DMZ . . . . .	54
4.5.4 EDI translation package . . . . .	54
4.5.5 VAN . . . . .	55
4.5.6 VAN access point . . . . .	55
4.5.7 VAN mailbox . . . . .	55

4.5.8	EDI/MIME translator . . . . .	55
4.5.9	SMTP server . . . . .	56
4.5.10	Internet gateway . . . . .	56
4.5.11	Directory and security services . . . . .	56
4.5.12	Queue manager . . . . .	58
4.5.13	Virtual Private Network (VPN). . . . .	59
4.5.14	Protocol domain firewall nodes . . . . .	59
4.5.15	Public Key Infrastructure (PKI) . . . . .	59
4.5.16	Domain Name Service (DNS) node. . . . .	60
4.5.17	Existing applications and data node . . . . .	60
4.5.18	Message broker . . . . .	60
4.5.19	HTTPS adapter for MQ middleware . . . . .	60
4.5.20	MQ server . . . . .	61
4.5.21	MQ adapter . . . . .	61
4.5.22	Web application server . . . . .	61
4.5.23	Database server node . . . . .	62
4.5.24	Load balancer node . . . . .	62
4.5.25	Web server redirector node . . . . .	62
4.5.26	Application server node . . . . .	63
4.5.27	Other open standards adapters . . . . .	63
4.5.28	Communication Interface . . . . .	63
4.5.29	Adapter node . . . . .	64
4.6	Summary . . . . .	64
<b>Chapter 5.</b>	<b>Technology options . . . . .</b>	<b>67</b>
5.1	Introduction . . . . .	67
5.2	Classifying technologies . . . . .	67
5.2.1	Framework categories . . . . .	69
5.2.2	Identifying key technology selection influences. . . . .	70
5.3	Partner (Client). . . . .	70
5.3.1	Choosing the partner technologies . . . . .	71
5.3.2	XML and the partner. . . . .	71
5.4	Web application server. . . . .	72
5.4.1	Internet/Web services. . . . .	72
5.4.2	Application services . . . . .	73
5.4.3	Illustrative examples . . . . .	74
5.5	Network-based infrastructure services . . . . .	76
5.5.1	Illustrative examples . . . . .	77
5.6	Integration services . . . . .	78
5.6.1	Database connectivity . . . . .	79
5.6.2	Packaged application API integration . . . . .	79
5.6.3	Middleware integration . . . . .	79
5.6.4	Component model integration. . . . .	79

5.6.5 Custom integration service development kit . . . . .	79
5.6.6 Application integration approaches . . . . .	79
5.6.7 Illustrative examples . . . . .	81
5.7 Web application programming model . . . . .	81
5.7.1 Influence of the component model . . . . .	82
5.7.2 Influence of architectural design patterns . . . . .	82
5.7.3 Illustrative examples . . . . .	83
5.8 e-business application services . . . . .	83
5.8.1 Illustrative examples . . . . .	84
5.9 Systems management . . . . .	84
5.9.1 System management model . . . . .	85
5.9.2 Cross-enterprise systems management . . . . .	86
5.9.3 Illustrative examples . . . . .	87
5.10 The development environment . . . . .	88
5.10.1 e-business application development team roles . . . . .	88
<b>Chapter 6. B2B integration protocols and standards . . . . .</b>	<b>89</b>
6.1 Overview . . . . .	89
6.1.1 Transporting the messages . . . . .	89
6.1.2 Content . . . . .	90
6.1.3 Business processes . . . . .	90
6.2 B2B Frameworks . . . . .	90
6.3 More on protocols . . . . .	94
6.3.1 OBI . . . . .	94
6.3.2 Rosettanet . . . . .	95
6.3.3 cXML . . . . .	96
6.3.4 Simple Object Access Protocol (SOAP) . . . . .	96
6.3.5 SET . . . . .	97
6.3.6 Commerce Business Library (CBL) . . . . .	97
6.3.7 Product Information Exchange (PIX) . . . . .	98
6.3.8 Information and Content Exchange (ICE) . . . . .	98
6.3.9 Internet Open Trading Protocol (IOTP) . . . . .	98
6.3.10 Open Financial Exchange (OFX) . . . . .	98
6.3.11 Platform for Privacy Preferences Project (P3P) . . . . .	98
6.3.12 Open Trading Protocol (OTP) . . . . .	98
6.3.13 XML/EDI . . . . .	99
<b>Chapter 7. IBM product guide . . . . .</b>	<b>101</b>
7.1 Foundation . . . . .	102
7.1.1 WebSphere Application Server . . . . .	102
7.1.2 MQSeries . . . . .	102
7.2 Foundation extensions . . . . .	102
7.2.1 VisualAge for Java . . . . .	103



7.2.2	VisualAge Application Rules . . . . .	103
7.2.3	VisualAge Generator . . . . .	103
7.2.4	WebSphere Studio . . . . .	103
7.2.5	WebSphere Homepage Builder . . . . .	104
7.2.6	WebSphere Business Components . . . . .	104
7.2.7	WebSphere Transcoding Publisher . . . . .	104
7.2.8	WebSphere Voice Server . . . . .	104
7.2.9	WebSphere Portal Server . . . . .	104
7.2.10	WebSphere Everyplace Suite . . . . .	105
7.2.11	WebSphere Personalization . . . . .	105
7.2.12	MQSeries Integrator . . . . .	105
7.2.13	WebSphere Edge Server . . . . .	105
7.2.14	WebSphere Site Analyzer . . . . .	105
7.2.15	WebSphere Host Integration Solution . . . . .	106
7.2.16	Tivoli Policy Director . . . . .	106
7.3	Application Accelerators . . . . .	106
7.3.1	WebSphere Commerce Suite . . . . .	106
7.3.2	Lotus Domino . . . . .	107
7.3.3	MQSeries WorkFlow . . . . .	107
7.3.4	WebSphere Business-to-Business Integrator . . . . .	107
7.3.5	Universal Description, Discovery, and Integration (UDDI) . . . . .	107
7.4	Customer and partner applications . . . . .	108
<b>Chapter 8. MQSeries and MQSeries integrator . . . . .</b>		<b>109</b>
8.1	Business integration and the MQSeries Family . . . . .	110
8.2	MQSeries primer . . . . .	111
8.3	What is Messaging and Queuing? . . . . .	112
8.4	About messages . . . . .	114
8.4.1	Message segmenting and grouping . . . . .	115
8.4.2	Distribution lists . . . . .	115
8.4.3	Message types . . . . .	115
8.4.4	Persistent and non-persistent messages . . . . .	116
8.4.5	The message descriptor . . . . .	116
8.5	About the Queue Manager . . . . .	118
8.6	About Queue Manager clusters . . . . .	120
8.7	About Queue Manager objects . . . . .	122
8.7.1	Queues . . . . .	123
8.7.2	Channels . . . . .	123
8.8	About message queues . . . . .	124
8.8.1	Local queue . . . . .	125
8.8.2	Cluster queue . . . . .	125
8.8.3	Remote queue . . . . .	125
8.8.4	Transmission queue . . . . .	126

8.8.5	Dynamic queue	126
8.8.6	Alias queue	126
8.8.7	Model queue	127
8.8.8	Initiation queue	127
8.8.9	Reply-to-queue	127
8.8.10	Dead-letter queue	127
8.8.11	Repository queue	128
8.8.12	Creating a Queue Manager	128
8.9	Manipulating Queue Manager objects	129
8.10	Clients and servers	130
8.11	How MQSeries works	132
8.12	Communication between queue managers	134
8.12.1	How to define a connection between two systems	135
8.12.2	How to start communication manually	137
8.13	How to trigger applications	140
8.14	Communication between client and server	142
8.14.1	How to define a client/server connection	142
8.14.2	How a Client/Server connection works	143
8.14.3	How a Client sends a request	144
8.14.4	How the server receives a request	145
8.14.5	How the server sends a reply	146
8.14.6	How the client receives a reply	146
8.15	The Message Queuing Interface (MQI)	147
8.16	A code fragment	148
8.17	MQSeries integrator components	151
8.17.1	The Configuration Manager	153
8.17.2	The Control Center	154
8.17.3	The Message Broker	156
8.17.4	Controller	159
8.17.5	The User Name Server	161
8.17.6	Security subsystem	162
8.17.7	Databases	162
8.17.8	Dependencies	162
8.17.9	Message domains, message sets, message types	164
8.18	XML and MQSeries	166
8.18.1	Importance for the MQSeries family	166
8.18.2	Use of XML within MQSeries Integrator	167

---

<b>Part 2. B2B integration guidelines</b>	<b>169</b>
---	------------

<b>Chapter 9. Application design guidelines</b>	<b>171</b>
9.1 Application elements	171
9.2 Communicating between applications	173

9.2.1 Synchronous communication . . . . .	174
9.2.2 Asynchronous communication . . . . .	174
9.2.3 Synchronous and asynchronous communication . . . . .	174
9.2.4 Comparison in a two-system update . . . . .	175
9.3 General principles . . . . .	179
9.4 Connecting to a store and forward mechanism . . . . .	180
9.4.1 Invasive insertion . . . . .	180
9.4.2 Passive adaptation . . . . .	180
9.4.3 Placing the adapter . . . . .	181
9.4.4 The role of the adapter . . . . .	182
9.5 Hub and Spoke integration architecture . . . . .	184
9.5.1 Where to do the transformation . . . . .	188
9.6 Application design summary . . . . .	189
9.7 Using MQSeries . . . . .	189
9.8 Connecting to the business application using MQSeries or MQSI . . . . .	190
9.8.1 Application types . . . . .	190
9.8.2 The MQSeries Adapter Offering . . . . .	192
9.9 General MQSeries guidelines . . . . .	194
9.10 Application style . . . . .	195
9.11 Application Programming Interface options . . . . .	196
9.11.1 MQI . . . . .	196
9.11.2 AMI . . . . .	197
9.11.3 Java-based APIs . . . . .	197
9.12 Considerations for the Partner Interface using MQSeries . . . . .	200
9.12.1 MQ Queue to MQ Queue: Intercommunication . . . . .	200
9.12.2 Summary of interface options . . . . .	206
9.13 Building the hub and spoke architecture using MQSI . . . . .	206
9.13.1 MQSeries Integrator applications . . . . .	208
9.13.2 Multiple hubs? . . . . .	209
9.13.3 Database resilience . . . . .	209
9.13.4 Message routing - Basis . . . . .	210
<b>Chapter 10. Application development guidelines . . . . .</b>	<b>213</b>
10.1 The development process . . . . .	213
10.2 The scope of this chapter . . . . .	214
10.3 The application and architecture domains . . . . .	215
10.4 Solution outline . . . . .	216
10.5 Macro design . . . . .	217
10.6 Micro design . . . . .	218
10.7 Build cycle . . . . .	219
10.8 Deployment . . . . .	220
10.9 Developing MQSeries applications . . . . .	221
10.9.1 Message Queue Interface (MQI) . . . . .	221

10.9.2 MQSeries classes for Java and MQSeries classes for JMS . . .	221
10.9.3 AMI. . . . .	222
10.10 Application development for MQSeries Integrator. . . . .	224
10.10.1 Terminology . . . . .	224
10.10.2 Overview of the requirements for a plug-in . . . . .	225
<b>Chapter 11. Performance guidelines . . . . .</b>	<b>229</b>
11.1 MQSeries and MQSI tuning, capacity planning, and performance . . .	230
11.1.1 Hardware and capacity. . . . .	230
11.1.2 MQSeries and MQSI application performance . . . . .	230
11.1.3 Additional performance information . . . . .	231
<b>Chapter 12. Systems management . . . . .</b>	<b>235</b>
12.1 Managing MQ? . . . . .	235
12.1.1 What should be managed in MQSeries networks? . . . . .	236
12.1.2 MQSeries Systems management products . . . . .	237
12.2 Security . . . . .	238
12.2.1 MQSeries security functions . . . . .	238
12.2.2 MQSeries messages . . . . .	240
12.2.3 Point-to-point security. . . . .	240
12.2.4 End-to-end security . . . . .	241
12.2.5 Where to find more information. . . . .	242
<hr/> <b>Part 3. An application example . . . . .</b>	<hr/> <b>243</b>
<b>Chapter 13. Getting a single customer view with MQSeries . . . . .</b>	<b>245</b>
13.1 Outbound flow . . . . .	246
13.1.1 RB_SCV_1message flow . . . . .	246
13.1.2 RB_SCV_Request_Endow message flow . . . . .	267
13.1.3 RB_SCV_Request_House message flow . . . . .	290
13.1.4 RB_SCV_Request_Motor message flow. . . . .	293
13.2 Inbound flow. . . . .	298
13.2.1 RB_SCV_Backend_Reply message flow . . . . .	299
13.2.2 RB_SCV_Backend_Reply_House&Motor message flow . . . . .	301
<b>Appendix A. Hardware and software specifications . . . . .</b>	<b>305</b>
<b>Appendix B. MQSeries Internet pass-thru . . . . .</b>	<b>307</b>
B.1 Introduction . . . . .	307
B.2 Overview of how Internet pass-thru works . . . . .	310
B.3 HTTP support . . . . .	311
B.4 Supported channel configurations . . . . .	312
13.3 Normal termination and failure conditions. . . . .	313

B.5 Security considerations . . . . .	313
<b>Appendix C. Using the additional material . . . . .</b>	<b>315</b>
C.1 Locating the additional material on the Internet . . . . .	315
C.2 Using the Web material . . . . .	315
C.2.1 System requirements for downloading the Web material . . . . .	315
C.2.2 How to use the Web material . . . . .	315
<b>Appendix D. Special notices . . . . .</b>	<b>317</b>
<b>Appendix E. Related publications . . . . .</b>	<b>321</b>
E.1 IBM Redbooks . . . . .	321
E.2 IBM Redbooks collections. . . . .	321
E.3 Other resources . . . . .	322
E.4 Referenced Web sites. . . . .	323
<b>How to get IBM Redbooks . . . . .</b>	<b>325</b>
IBM Redbooks fax order form . . . . .	326
<b>Glossary . . . . .</b>	<b>327</b>
<b>Abbreviations and acronyms . . . . .</b>	<b>335</b>
<b>Index . . . . .</b>	<b>337</b>
<b>IBM Redbooks review . . . . .</b>	<b>343</b>



---

## Figures

1. e-business adoption process . . . . .	5
2. The e-business applications classification schema . . . . .	8
3. Separation between e-Markets and Hubs . . . . .	10
4. B2BI Solution Components . . . . .	15
5. Patterns for e-business . . . . .	19
6. e-business integration . . . . .	21
7. General B2Bi pattern . . . . .	24
8. Diagram conventions for B2Bi . . . . .	29
9. Application Topology 1 - EDI . . . . .	30
10. Topology 2 - Direct connection with adapter/bridge. . . . .	32
11. Application Topology 3 - Message broker . . . . .	34
12. Application Topology 4 - Managed Business Protocol. . . . .	36
13. Managed Business Protocol and Process . . . . .	38
14. Application Topology 1: Runtime topology. . . . .	42
15. Runtime topology: VAN-Document . . . . .	43
16. Runtime topology: EDI - Internet . . . . .	45
17. Application Topology 2 - Runtime topology . . . . .	47
18. Illustrative example of topology two using shared middleware . . . . .	49
19. Illustrative example of topology two using open standards . . . . .	50
20. Application Topology 3: Runtime topology. . . . .	51
21. Illustrative example of Topology 3 using shared middleware . . . . .	52
22. Illustrative example of topology 3 using open standards . . . . .	53
23. Elements of a B2BI architecture . . . . .	68
24. Application integration approaches . . . . .	80
25. System management model . . . . .	85
26. OBI information flow. . . . .	94
27. IBM product suite. . . . .	101
28. Runtime topology 2 . . . . .	109
29. The MQSeries family for business integration . . . . .	111
30. MQSeries at run time . . . . .	112
31. Messages and Queues . . . . .	114
32. Program-to-program communication - One system. . . . .	118
33. Program-to-program communication - Two systems. . . . .	118
34. MQPUT to a remote queue . . . . .	120
35. MQPUT to a cluster queue . . . . .	121
36. Accessing cluster queues . . . . .	122
37. MQSeries channels . . . . .	124
38. MQI and message channels . . . . .	131
39. MQSeries - Parts and logic . . . . .	132
40. Communication between two queue managers. . . . .	136

41. Triggering channels . . . . .	139
42. Triggering an application . . . . .	140
43. MQSeries application trigger choices . . . . .	141
44. Client/Server connection . . . . .	142
45. Clients and server communicating. . . . .	145
46. A code fragment. . . . .	150
47. Overview of MQSeries integrator. . . . .	152
48. Relationship between the Control Center and Configuration Manager . .	154
49. Overview of a broker . . . . .	156
50. Role of the User Name Server. . . . .	161
51. Application topology 3: Runtime topology . . . . .	172
52. Application to application communication . . . . .	173
53. Synch communication:Distributed transaction with two phase commit . .	176
54. Asynch communication:Distributed transaction with two phase commit .	177
55. Distributed Systems: General principles . . . . .	179
56. Placing the adapter . . . . .	181
57. Process for a passive adapter . . . . .	182
58. Number of connections for 3, 4 and 10 integrated systems. . . . .	184
59. Number of Hub systems for 3, 4, and 10 integrated systems . . . . .	185
60. Complexity of Direct Connected Systems vs. Hub Connected Systems .	186
61. Comparison of direct connected systems vs hub connected systems. . .	187
62. Application to application communication . . . . .	190
63. Adapter Interface classification and differentiation . . . . .	191
64. Adapter builder tool . . . . .	193
65. Overview of the components of distributed queuing . . . . .	201
66. Sending messages . . . . .	203
67. Sending messages in both directions . . . . .	204
68. Mutiple application with multiple partners . . . . .	207
69. Development process overview . . . . .	214
70. Domain concept in IBM Global Services methodology . . . . .	216
71. Application topology 3, the runtime components . . . . .	229
72. Component overview . . . . .	245
73. RB_SCV_1 message flow . . . . .	246
74. SessReq MQInput node: Basic tab . . . . .	248
75. SessReq MQInput node: Default tab . . . . .	248
76. SessReqFail MQOutput node . . . . .	249
77. FilterRequest Filter node . . . . .	250
78. FilterReqFail MQOutput node . . . . .	251
79. FilterReqUnknown MQOutput node. . . . .	251
80. NolInfo Compute node . . . . .	252
81. NolInfo Compute node ESQ. . . . .	253
82. Node: FilterEndow . . . . .	254
83. FilterEndowFail MQOutput node . . . . .	255



84. FilterEndowUnknown MQOutput node . . . . .	255
85. FilterHouse Filter node. . . . .	256
86. FilterHouseFail MQOutput node . . . . .	257
87. FilterHouseUnknown MQOutput node . . . . .	257
88. FilterMotor Filter node . . . . .	258
89. FilterMotorFail MQOutput node . . . . .	259
90. FilterMotorUnknown MQOutput node . . . . .	260
91. MergerTrigger Compute node . . . . .	261
92. MergerTrigger Compute node ESQ L . . . . .	262
93. MergerTrigger Compute node ESQ L (continued) . . . . .	263
94. MergerTrigger Compute node ESQ L (continued) . . . . .	263
95. MergerTrigger Compute node ESQ L (continued) . . . . .	264
96. MergerTrigger Compute node ESQ L (continued) . . . . .	264
97. MergerTrigger Compute node ESQ L (continued) . . . . .	265
98. MergerTriggerFail MQOutput node . . . . .	266
99. MergerTriggerTrace Trace node . . . . .	266
100. MergerTriggerQ MQOutput node . . . . .	267
101. Flow RB_SCV_Request_Endow . . . . .	268
102. RequestEndow node . . . . .	269
103. The Compute node Properties pull-down menu . . . . .	269
104. Copying message headers only . . . . .	270
105. The Compute node add input task . . . . .	271
106. Adding input sources to a Compute node . . . . .	272
107. Adding a database table as a Compute node input source. . . . .	273
108. Entering data source and table names to the Compute node input . . . .	274
109. Selecting the MRM output message of the Compute node. . . . .	275
110. Selecting an MRM message as the output of the Compute node . . . . .	276
111. Selecting a Compute node output message from the MRM . . . . .	277
112. Compute node with inputs and outputs selected. . . . .	278
113. Compute node, generated ESQ L . . . . .	279
114. ESQ L to populate SCV_ENDOW_BACKEND_MSG properties . . . . .	280
115. ESQ L to populate SCV_ENDOW_BACKEND_MSG MQMD . . . . .	281
116. Compute node simple ESQ L . . . . .	282
117. Calculating an output message field value . . . . .	283
118. Using the Compute node ESQ L to populate a list. . . . .	284
119. Initializing unused repeating structure iterations . . . . .	285
120. Configuring a Trace node . . . . .	286
121. Configuring the Trace node: Selecting the trace destination. . . . .	286
122. Configuring the Trace node: Identifying the destination file . . . . .	287
123. Configuring the Trace node: what to print in the trace . . . . .	288
124. RB_SCV_Request_House message flow . . . . .	290
125. RequestEndow Compute node ESQ L . . . . .	292
126. ReqHouseFail MQOutput node . . . . .	292

127.ReqHouseTrace Trace node . . . . .	293
128.HouseBackend MQOutput node . . . . .	293
129.RB_SCV_Request_Motor message flow . . . . .	294
130.RequestMotor Compute node properties . . . . .	295
131.RequestMotor Compute node ESQL . . . . .	296
132.ReqMotorFail MQOutput node . . . . .	297
133.RequestMotorTrace Trace node . . . . .	297
134.MQOutput node . . . . .	298
135.RB_SCV_Backend_Reply message flow . . . . .	299
136.HouseBackendReply MQInput node default properties . . . . .	300
137.RB_SCV_Backend_Reply_House&Motor message flow . . . . .	301
138.FormatHouse&Motor Compute node . . . . .	302
139.FormatHouse&Motor Compute node ESQL . . . . .	303
140.FormatHouse&Motor Compute node ESQL . . . . .	303
141.FormatHouse&Motor Compute node ESQL . . . . .	303
142.FormatHouse&Motor Compute node ESQL . . . . .	304
143.Example of MQIPT as a channel concentrator . . . . .	308
144.Example of MQIPT with a “demilitarized zone” . . . . .	309
145.Example of MQIPT and HTTP tunneling . . . . .	309

---

## Tables

1. Patterns for e-business and e-business solutions . . . . .	21
2. Web application server technology examples . . . . .	74
3. Network-based infrastructure technology examples . . . . .	77
4. Integration technology examples . . . . .	81
5. Programming model technology examples . . . . .	83
6. e-business application service technology examples . . . . .	84
7. Systems management technology examples . . . . .	87
8. XML B2B framework initiatives . . . . .	91
9. Attributes of the message descriptor . . . . .	116
10. Queue types and their purposes . . . . .	124
11. MQSeries Objects defining connection between two queue managers. .	136
12. MQSeries APIs and their purposes . . . . .	147



---

## Preface

Patterns for e-business is a group of proven, reusable assets that can help speed up the process of developing applications. The pattern discussed in this book, Business-to-Business Integration patterns 2 and 3, forms the basis for many of the more complex and functional B2B patterns. It is relevant to all enterprises dealing with partner integration issues over the Internet.

This redbook discusses two application topologies of the Business-to-Business patterns. Application topology 2 describes a scenario in which messages are passed between two enterprise applications and no routing is performed. Topology 3 extends topology 2 to describe the scenario in which routing is required for multiple cross enterprise applications to communicate.

Part 1., "Introduction" on page 1, takes you through the process of choosing an application topology and a runtime topology. It then gives you possible product mappings for implementation of the chosen runtime topology. This part introduces all the topologies, even though only two of the patterns are covered in this book.

Part 2., "B2B integration guidelines" on page 169, is a set of guidelines based on topologies 2 and 3 for building your e-business application. It includes information about application design, technology options, application development, performance, and security.

Part 3., "An application example" on page 243, takes you through a working example showing the simple implementation of an integration pattern using application topology 3.

---

### The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

**Indran Naick** is a Project Leader and Senior IT Specialist at the International Technical Support Organization, Austin Center. He has 10 years of industry experience. He writes extensively and teaches IBM classes worldwide on a number of client/server topics. Before joining the ITSO in 1999, Indran worked for the IBM Software Group, Southern Africa, as a Software Solutions Architect. He holds a Bachelors degree in Computer Science from the University of the Witwatersrand in South Africa.

**Mark Berkhoff** is an IT Architect in the Netherlands with over 10 years experience in several positions in the IT industry. He holds a degree in Computer Science, and his areas of expertise include the Internet, security, and cryptography. He has written extensively on Public Key Infrastructure (PKI) issues.

**Daniel Verdugo Bosnich** is an e-Business IT/Architect in Chile. He has four years of experience on e-business multi-platform integration projects and has worked at IBM for six years. His areas of expertise include Object Oriented Design and MQSeries and Websphere solution design.

Part three of this book was produced by a team of specialists working at the International Technical Support Organization, Raleigh Center. We would like to thank Colin Brett, Paul Sehorne, Sharon Stubblebine, and Geert Van de Putte for their contribution.

Special thanks go to the following people:

**Keith Edwards**, who provided much of the material that makes up the design chapter

**Dieter Wackerow**, who provided the introduction for the MQSeries chapter reproduced in this book

**Kareem Yusuf**, who provided invaluable guidance and input on all the subjects throughout the course of this project

Thanks to the following people for their invaluable contributions to this project:

Ron Aguirre, Milos Radosavljevic  
International Technical Support Organization, Austin Center

Carla Sadler, Geert Van de Putte  
International Technical Support Organization, Raleigh Center

Luis Ennser  
International Technical Support Organization, San Jose Center

Barbara Van Laeken  
IBM U.S., Dallas

David Hardcastle, Pete Murphy, Rodric Yates  
IBM UK

Jonathan Adams, Joel Farrell  
IBM

---

## Comments welcome

### **Your comments are important to us!**

We want our Redbooks to be as helpful as possible. Please send us your comments about this or other Redbooks in one of the following ways:

- Fax the evaluation form found in “IBM Redbooks review” on page 343 to the fax number shown on the form.
- Use the online evaluation form found at [ibm.com/redbooks](http://ibm.com/redbooks)
- Send your comments in an Internet note to [redbook@us.ibm.com](mailto:redbook@us.ibm.com)





---

## Part 1. Introduction

The following chapters describe business-to-business concepts and their effect on the business and technology dimensions of an organization. This is a vast topic, and the information presented here is in no way comprehensive. The intent is to serve as an introduction and cover concepts that are used later in the book.

Chapter 1, “e-business and B2B integration” on page 3, defines B2B Integration in the context of the broader B2B and e-business marketplace. It also describes the components of a B2BI interaction.

Chapter 2, “Introduction to business patterns” on page 17, describes the patterns for e-business and the different business topologies. Each business topology has an associated application topology which in-turn has an associated runtime topology. A mapping of the chapters within this book is also described in this chapter.

Chapter 3, “Choosing the application topology” on page 27, introduces all the application topologies for the Business-to-Business Integration pattern. With very accessible notation, these application topologies capture the essential “shape” of the application solution.

Chapter 4, “Choosing the runtime topology” on page 41, discusses the runtime topologies for these application topologies. It includes discussions of variations of these topologies that are appropriate for scalability and availability.

Chapter 5, “Technology options” on page 67, maps the runtime topology to various technology components. These components are required to satisfy the functional requirements within the runtime topology.

Chapter 6, “B2B integration protocols and standards” on page 89, goes on to further describe some of the protocols and the standards that are prevalent in the B2B Integration space.

Chapter 7, “IBM product guide” on page 101, provides a high-level description of the IBM Suite of products. These are described in the context of their operation and functional aspects.

Chapter 8, “MQSeries and MQSeries integrator” on page 109, describes MQSeries and MQSeries Integrator in some detail. An understanding of these components is essential because they form the core of IBM offerings for topologies 2 and 3.



---

## Chapter 1. e-business and B2B integration

This chapter provides an introduction to the e-business model and its many components. Within this model we focus on business-to-business (B2B) commerce. B2B solutions focus on using the Internet and/or extranet to improve partnerships and transform inter-organizational relationships.

These interactions can be classified by the properties of the relationship between the businesses.

This book focuses on B2B Integration, which is a subset of the B2B commerce model. The distinction between these two markets will become more clear in the chapters that follow.

---

### 1.1 About e-business

At the beginning of the Internet era, IBM invented the term e-business to give a proper name to a new class of powerful software applications and services that, in its vision, needed to be developed in the following years. This class of applications derive their power from combining the universal access and standards of the Internet with the reliability, security, and availability of existing content, core business processes, and applications.

In very simplified terms, e-business refers to the usage of Internet technologies to improve and transform key enterprise processes. Most organizations understand this and have begun the transformation from traditional applications to their e-business counterparts. This transformation has begun to Web-enable core processes to strengthen customer service operations, streamline supply chains, and reach existing and new customers.

Probably one of the well-known applications of e-business is e-commerce, which refers to buying and selling activities over a digital medium. However, as we will soon clarify, e-business embraces e-commerce and includes Intranet applications. e-business is the overall strategy, although e-commerce is an extremely important subset of e-business.

In what ways does e-business differ from the old order? Some characteristics of the e-business model that set it apart from legacy business systems are:

- e-business facilitates transactions with a much wider group of respondents.
- Communication and other transactions are instantaneous.
- Customers are empowered.

- Competition is fierce.
- Customer communities will emerge.

Companies that are creating value when they implement e-business realize it is not simply a matter of establishing a Web site or a single, discrete, online application. It arises from an e-business vision and an e-business roadmap that begins with an initial solution that is extendable into other high-value areas of the business. e-business is not just about e-commerce transactions; it is about redefining old business models with the aid of technology.

To summarize, an e-business is an organization that connects critical business systems directly to their customers, employees, partners, and suppliers, via intranet, extranet, and the Web. As customers, employees, suppliers, and distributors are connected to the business systems and information they need, e-business actually transforms, innovates, and integrates key business processes.

### **1.1.1 Business transformation and Innovation**

The convergence of Internet technologies, IT systems, and business processes creates new business models – e-business models that is. Thus, e-business poses the most significant challenge to the traditional business model. Computers have increased business speed by automating tasks, but they have not fundamentally altered the business foundation; e-business has. If any entity in the value chain begins to do business electronically, similar companies must follow suit or risk being eliminated. Rethinking and redesigning the business model is not an option but a necessary step to surviving in the information age.

Companies have found that success in e-business is typically based on building efficient, value-added relationships with their customers. Those companies exhibiting the highest degree of satisfaction and success with e-business consistently focus their strategy on improving their performance for customers. Whether simply making it efficient for a customer to place an order for a product or service, Web-linking and customizing access for a supplier network, or integrating a real-time collaborative product design/development process, the goal is to serve customers better.

Regardless of size or industry, companies follow a similar pattern of e-business adoption. Figure 1 on page 5 illustrates the typical e-business adoption roadmap.

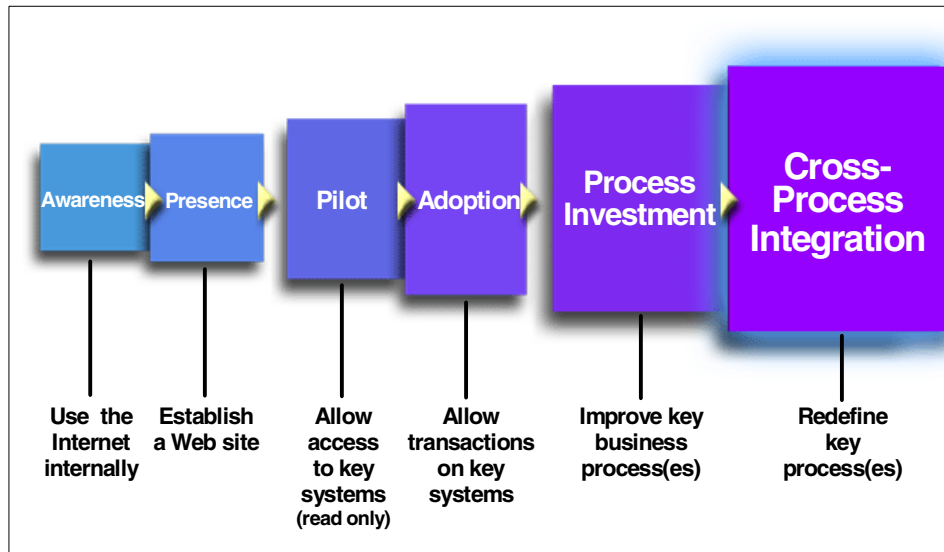


Figure 1. e-business adoption process

The first two steps are the introduction to using Web technologies: Deploying Intranet applications to communicate with employees and establishing external Web presence with information about the company and its offerings.

This experience has led companies to take the first steps toward real e-business: The establishment of pilots that give customers and suppliers access to the information contained in key business systems. For example, opening up the customer database that a service representative views and providing this information to customers directly.

The next step in the cycle extends legacy transaction capability to customers. The travel industry, for example, offers online bookings while retail banking offers online banking.

The final two steps in the process offer the greatest opportunity for return as companies transform their business processes by integrating multiple back-end systems to create a common user experience. For example, an airline has integrated all travel systems and customer processes for bookings, upgrades, seat and meal selection, awards redemption, and frequent flyer account management.

During these last two steps, not only can a customer buy/transact online whenever they wish, but the company gains valuable insight and knowledge about the specific needs and behavior of its customers. They can also

segment customers from a behavioral perspective as well as from a value perspective. However, without the initial effort to make the data customer-ready, value for both the customer and the enterprise, although real, would be minimal. Many companies have begun offering online transactions to reduce their costs without considering the value for their customers and, therefore, have not achieved sustained customer commitment to the system.

In the process investment phase, armed with this customer knowledge and with the integration of business intelligence tools, analyses, and insights, companies can improve the customer relationship process by personalizing customer interactions online and integrating the customer view for their entire enterprise. Through these personalized interactions, customer retention and loyalty analysis can be applied to serving the customer more effectively at the point of need rather than integrating this information after the fact.

Cross process integration, the final phase in e-business evolution, focuses on integrating across all the processes in an enterprise as well as across customer processes. For the enterprise, this means integrating all the customer touchpoints across all operational systems from supply to demand fulfillment and through customer satisfaction. For the customer, it means linking the systems of all the suppliers involved in their process. Airlines, for example, have integrated their booking systems with their frequent traveler services internally and have linked into their travel partner systems to begin offering passengers a single convenient point of fulfillment; retail banks are integrating their services into a much broader single service access point for their on-line customers.

### **1.1.2 e-business value**

There are several trends that are shaping e-business necessity and, at the same time, its value:

- Businesses of all sizes are impacted by globalization and deregulation, which lowers barriers to entry and dramatically reshapes the competitive landscape.
- Customers now have a broader array of choices and, therefore, are becoming more sophisticated and more demanding – both in what they want from a supplier and how they choose to acquire goods and services.
- As a consequence of the fact that markets are becoming increasingly fragmented (see the first two points above), mass marketing is fading in importance as mass customization becomes the path to serving discriminating customers.

- Technology continues to evolve rapidly to support this environment. The global reach of the World Wide Web enables companies to reach customers anywhere and connect to employees, suppliers, and trading partners wherever they are. This will create an expanding amount of data, which can then be mined for insight leading to better decisions and help create ways to know and serve your customers better and more profitably and gain a competitive advantage.

e-business can dramatically improve competitiveness and create new paths to customer loyalty.

---

## **1.2 e-business applications: A simplified classification scheme**

e-business activities can be classified in a number of ways. In this section, they are divided based on the scope of the applications and on the players at each end of the transaction.

Concerning the application scope, we have two main categories:

- Intra-business applications
- Inter-business applications

The first category includes all e-business applications that have a company/organization internal scope. These applications lie in the company/organization boundary and are connected to internal business activities. For example, this class includes the realization of an intranet information Web site for company employees.

The second category includes all applications that require some kind of interaction between the company/organization and some other external entities, such as a customer, a company trading partner, or a financial institution. For example, as we will see more clearly in the following sections, an e-commerce application that models a buying/selling activity between a company and its customers over the Internet is an Inter-business application as well as a Web-bank application where bank customers can view their account balances, recent transactions, and so on through the Web.

#### Note

One could argue that, sometimes, the boundary between an Intra-business application and an Inter-enterprise application cannot be easily defined. However, just for the purpose of this classification, we are referring to Intra-business applications as all applications that are interfaced/used directly by company employees and/or other systems belonging to the company and isolated from the external world. Complementary inter-business applications are all those that can be interfaced by external end-users (for example a customer) and/or external applications (trading partner applications).

Classification of e-business activities, or business processes, is described in more detail in the following sections. It is important to note that although the following sections make a distinction between Intra and Inter-business applications, from the infrastructure solution point of view, this distinction can disappear. In fact, IBM defined another convenient way to describe the architectural nature of the e-business solutions that can be applied to both Intra and Inter-enterprise models. As the next chapter will describe, IBM recently introduced patterns for e-business that will allow IT architects in 80 percent of cases to quickly develop 80 percent of their required infrastructure by the reuse of proven architecture patterns, design patterns, runtime patterns, design, development and deployment guidelines, and code. Figure 2 illustrates the e-business application classification schema.

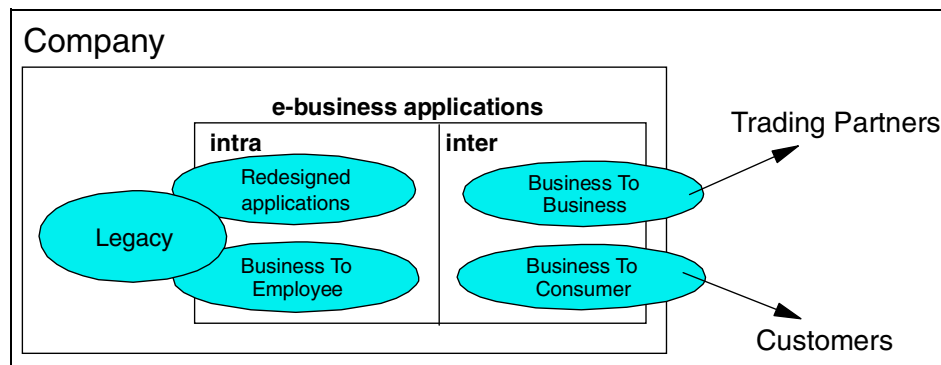


Figure 2. The e-business applications classification schema

### 1.2.1 Intra-business applications

Most Intra-enterprise solutions are based on an intranet infrastructure where the purpose of an intranet is to share company information and computing



resources among employees. Usually, this class of applications is known as business-to-employee (B2E). An intranet can also be used to facilitate working in groups and for teleconferences.

There are four general categories of Intra-business applications:

- Internal communication creates an involved and communicative environment by making information accessible to a company's/organization's employees/members.
- Knowledge management is the identification and analysis of available and required knowledge assets and knowledge asset-related processes. Knowledge assets are knowledge regarding markets, products, technologies, and organizations that a business owns or needs to own and which enable its business processes to generate profits and provide value.

Knowledge management means facilitating an environment where knowledge, such as standards, common processes, and best practices, is shared and exploited by any group, department, or division in an organization. This involves the process of capturing and leveraging a company's intellectual capital, which is a valuable commodity in many companies.

Knowledge management includes collaboration that exploits Web technology to create team-oriented work environments that are more productive, higher quality, and enjoyable.

Business intelligence means developing applications that are able to individuate information that is conclusive, fact-based, and actionable. Typically, Business Intelligence solutions combine consulting & services, applications and, technologies to gather, manage and analyze data, transform it into usable information to develop the insight and, understanding needed to make informed decisions.

Using the latest e-business technologies, this intelligence can then be distributed around your company or around the world helping to make crucial and profitable decisions.

- Business process redesign means transforming inefficient practices and processes inside the organization, from supply chain management to internal procurement or expense reports.

### **1.2.2 Inter-business applications**

Inter-business applications are all applications that require some kind of interaction between a company/organization and some other external entity, such as a customer, a company trading partner, a financial institution, or public administration department. Inter-business applications can be divided

into two main subclasses: Business-to-consumer (B2C) and business-to-business (B2B).

B2B applications focus on using the Internet and/or extranet to improve business-to-business partnerships and transform inter-organizational relationships. B2B trade may be conducted between a company/organization and its supply chain as well as between a company/organization and other company/organization end-customers. Trading can be conducted directly between buyers and sellers and/or supported by a third party (an intermediary) within an eMarketPlace. B2B describes two styles of inter-business to business: eMarketPlaces and B2Bi. Figure 3 shows the breakdown between eMarketPlaces and B2Bi based on the number of buyers and sellers. In the case of B2Bi there is usually a one-to-one relationship between buyers and sellers. Any other relationship, such as one-to-many, many-to-many, or many-to-one, would fall into the eMarketPlace category. Each of the eMarketPlace categories has its own unique characteristics, and these are reflected in their implementation. For example, in the case where there is one seller and many buyers, the interaction is similar to that of a traditional auction; so, the digital model has to allow for multiple disclosed bids.

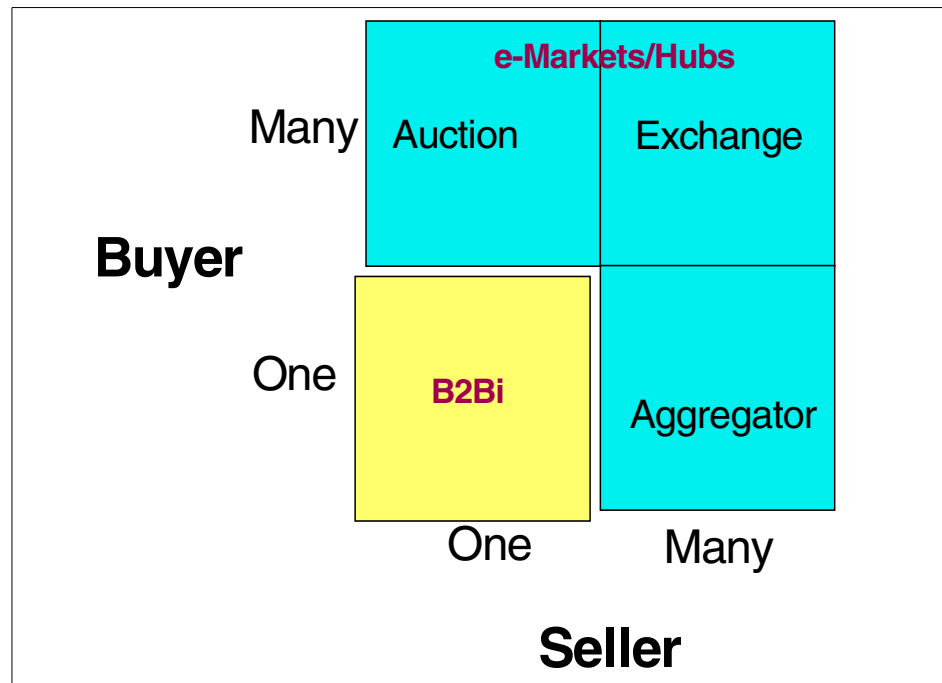


Figure 3. Separation between e-Markets and Hubs

#### **1.2.2.1 B2Bi**

B2Bi covers programmatic links between arms-length businesses (where a trading partner agreement may, potentially, be appropriate). A good example of this would be Supply/Chain applications.

The remainder of this book focuses on B2Bi. The following section is a short description of the second style of inter-business to business. Another redbook, due out later this year, will cover eMarketPlaces interaction.

#### **1.2.2.2 B2M2B (eMarketPlace)**

The second style covers the eMarketPlace where the model supports B2M2B. The M represents the eMarketPlace, which supports multiple buyers and suppliers. The buying function may be performed online or programmatically.

The traditional B2B model, centered around the buyer-seller transaction paradigm, is showing its limitations: it is definite in scale and displays only partial efficiency in terms of market economics. B2M2B overcomes these limitations and leverages existing B2B applications and technology. The eMarketPlace or online trading communities assist multiple buyers and suppliers to exchange information and transactions.

Trading communities are Internet-based hubs that focus on specific industry verticals (see, for example, the recent hub, Component Knowledge, launched by IBM Global Services for the electronic components market) or specific industry processes and use various market making mechanisms, such as auctions, exchanges, and aggregation, to mediate any-to-any transactions among businesses.

Through the trading communities hubs, buyers and sellers can trade electronically with established partners and, at the same time, get access to new markets and new parts of the supply chain. eMarketPlaces can be a public, interactive buying and selling community where all members participate in the open, or they can be private, invitation-only communities whose members participate in special pricing arrangements and/or product and service offerings. Online trading communities have the potential to create excellent and efficient markets.

Among the best-known e-market makers are Chemdex, VerticalNet, Altra Energy Technologies, Paper Exchange, Instill, PlasticsNet, and Commerce One's Marketsite.net.

---

## 1.3 Inter-business: B2Bi

B2Bi, as just described, excludes eMarketPlaces. It describes e-commerce where the relationships between businesses is one-to-one. In the May 2000 issue of the EAI journal (<http://www.eaijournal.com>), Greg Olsen described the laymen's view of B2Bi as being one of the following:

“That's EDI (Electronic Data Interchange). It's about agreeing on standard datasets using X12, EDIFACT, or XML and then exchanging the data over Value-Added Networks (VANs) or the Internet.

B2B integration is application integration extended outside a single company. It's about using middleware technologies, such as distributed objects, remote procedure calls, message queueing, data transformation, and publish/subscribe, to connect different applications with the added complication of getting through firewalls.

B2B Integration is about using the Web to share data across company boundaries. B2B integration is accomplished by putting a Web front end on your applications so information can be shared with suppliers, customers, and partners.

As mentioned in the article, each of these perspectives may be appropriate in certain circumstances, but each presents a narrow view and misses many fundamental B2B integration requirements.”

### 1.3.1 EDI and B2B Integration

EDI is an existing example of B2B Integration that uses a more proprietary set of technologies. Facilities, such as Electronic Data Interchange (EDI), have successfully provided electronic document interchange between companies and their suppliers for a number of years. However, EDI's high cost and inflexible structure have always proved a barrier to adoption by all but the largest enterprises. While EDI will continue to evolve, utilizing pervasive networks, such as the Internet, to reduce costs, complementary technologies are emerging that are able to provide some of the key capabilities necessary to enable dynamic business process integration. The basis of these technologies is the formulation of:

- A “common language” that can be employed by existing or potential trading partners to specify how they will interact
- An “electronic contract” that employs this common language in order to define and enforce the interaction protocols with which they will do business.

### 1.3.2 B2B and A2A (Application-to-Application)

In a recent eBizQ article, Steve Scala writes: “The Drive to EAI and B2B Convergence: a Conversation with Roy Shulte offered this recipe for B2B integration: Start with a dependable A2A integration broker, stir in generous portions of XML, add a pinch of security, sprinkle liberally with HTTP, and serve up to enterprises hungry for a taste of B2B e-commerce.

There is a definite convergence between A2A or EAI and B2Bi. Both problems are about application integration. However B2Bi does present many challenges that are unique to itself, challenges that have never been addressed by Application Integration.”

The articles mentioned above (available at <http://eai.ebizQ.net/>) offer some insight into some of the issues that differentiate the two business problems.

### 1.3.3 B2B integration challenges

Several issues have to be faced in designing B2B systems. These issues are concerned with privacy, autonomy, heterogeneity in software and platforms, and, more importantly, managing complexity of interactions.

Some of these issues, such as the heterogeneity of programming languages and platforms in which the application components are developed, are also addressed in the automation of business internal processes and integrating application components. Total knowledge and control in the design of the business process within an organization makes this a manageable task.

Component architectures, such as CORBA and Enterprise Java Beans, provide middleware for integrating application components written in different languages. For the purpose of interaction, an application component needs to know only the interfaces to other components written in a suitable middleware integration language (for example, the Interface Definition Language or IDL in CORBA). In such environments, typically, the applications are executed as short ACID (atomicity, consistency, isolation, durability) transactions. The underlying middleware provides necessary runtime services, for instance, naming, transaction, and resource allocation.

For the automation of the B2B interactions, methodologies that automate internal processes of individual businesses are not directly applicable. First of all, no common, shared, underlying middleware can be assumed for distributed applications spanning organizational boundaries and using a public network, such as the Internet. Setting up such a common software bus requires tight coupling of the business partners' software platforms (for

example, consider the issues of security, naming, and component registration).

Even if such a software bus can be established, ACID and/or complex extended transaction models of stateful interactions are not appropriate for such B2B interactions. First, implementation of such protocols necessitates tight coupling of operational states across business applications, which is highly undesirable. The application components in one organization may hold locks and resources in other organizations for an extended period of time, resulting in a loss of autonomy. Rollback and/or compensation of application steps is no longer under the control of a single organization. Finally, in real-world business operations, the states always move forward, and explicit recourse actions are taken by business partners to move to a more desirable operational state. An example is cancellation of a prior purchase or reservation.

The invocation of application components across organizational boundaries needs to be controlled and monitored. First, without rigorous testing and cooperation in software development across organizations, the correct execution of such complex distributed applications cannot be assumed. Second, in such automated interactions, trust becomes an overarching concern. During runtime, explicit checks are necessary to ensure that business partners are not violating any policy constraints; for instance, cancellation of a reservation must be within the allowable time window.

In addition, additional services for supporting long running applications, for example, application development, asynchronous event driven execution, compensation framework, maintaining correlation of conversations, and logging and querying the activity on a conversation are required.

This book will not explore solutions to all the problems described here; instead, we will focus on specific implementations and expand on the specifics of the problems posed by those topologies.

#### **1.3.4 B2B integration solution components**

Each B2BI solution will differ based on the software, hardware, and services selected to fulfill the system requirements. At a high level, the approach taken in the EAI journal describes four elements as being common across most B2B integration solutions as shown in Figure 4 on page 15.

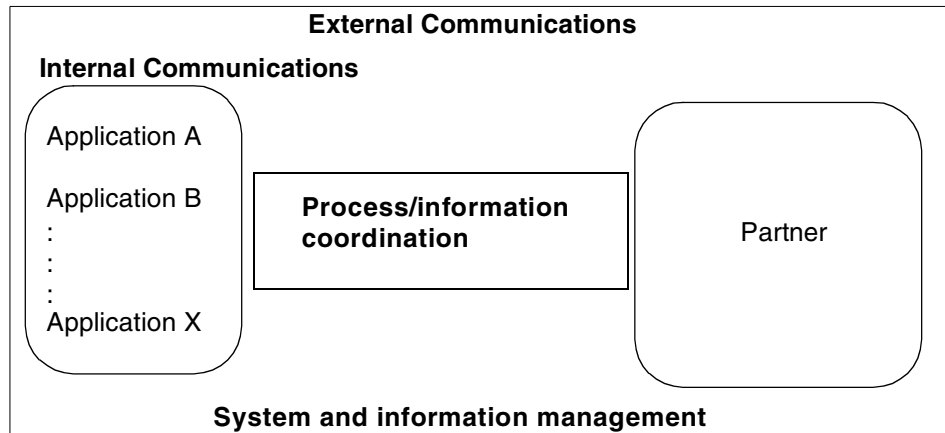


Figure 4. B2BI Solution Components

- External communications

This element describes the communication between trading partners. Since B2BI implicitly involves communication with external organizations, there are likely to be many partners, each having different requirements.

- Internal communications

For organizations with existing infrastructure, a mechanism to communicate with existing applications is going to be a requirement. This is usually very similar to Inter Enterprise Application Integration, where you have an external application.

- Process/information coordination

Process /information coordination represents the heart or the logic of the interaction between internal communication and external communication. You can characterize this broadly as an information-brokering function.

- System and information management

For B2Bi, the ability to manage the solution is a critical success factor. The solution will also require managing several information components, including partner profile information, data and process definitions, communications and security settings, and user information.

In the chapters that follow, we describe the details of some of these the components relative to the selected topologies.

---

## 1.4 Summary

e-business encompasses all transactions over a digital medium. e-business applications can be classified broadly into two types: Inter-business and intra-business.

Inter-business applications include two styles of business-to-business commerce: An eMarketplace and B2B Integration. B2Bi is very similar to internal application integration but has many of its own challenges.

Almost all large corporations and even smaller companies today are implementing a B2BI solution. This is in addition to the large number of enterprises currently using EDI. This number will increase as infrastructure and standards evolve. The rate of adoption will be a function of the costs and benefits that this new digital medium will provide. Almost all major corporations engage in some B2BI.



---

## Chapter 2. Introduction to business patterns

We are all familiar with the pace of development of the computer industry during its relatively brief history. The rapid advances in computer hardware have been driven in no small part by the use of standards and well specified components for assembly. The desire to apply these same approaches to software construction gave rise to object-oriented software, design patterns and component-based development.

The idea of design patterns has gained acceptance by software designers and developers because it enables an efficiency in both the communication and implementation of software design, based upon a common vocabulary and reference. Information technology architects, encouraged by the success of design patterns, and facing challenges in systematic and repeatable description of systems, have also explored the idea of architectural patterns.

The Enterprise Solution Structure (ESS) work (see “Enterprise Solutions Structure” in *IBM Systems Journal, Volume 38, No. 1, 1999* at <http://www.research.ibm.com/journal/sj38-1.html>) looked at patterns for complete end-to-end system architectures. ESS is now part of the IBM Global Services methodology.

The following publications are interesting reading for more information on design patterns and their background:

- *Design Patterns - Elements of Reusable Object-Oriented Software*, ISBN 0-2016-3361-2, by E. Gamma, R. Helm, R. Johnson, J. Vlissides
- *A Pattern Language*, ISBN 0-1950-1919-9, by C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, S. Angel.
- *Pattern-Oriented Software Architecture - A System of Patterns*, ISBN 0-4719-5869-7, by Buschmann, et al.
- *Pattern Hatching - Design Patterns Applied*, ISBN 0-2014-3293-5, by J. Vlissides

---

### 2.1 Patterns for e-business

The patterns for e-business aim is to communicate in a highly accessible fashion the business pattern, systems architecture (application and runtime topologies), product mappings, and guidelines required for different classes of applications. For the some patterns there is also an associated Pattern Development Kit, which provides sample application code to illustrate effective use of those patterns.

The goal is to provide the smallest number of patterns for e-business which will allow IT architects in 80 percent of cases to quickly develop 80 percent of their required infrastructure by the reuse of proven:

- Architecture patterns
- Design patterns
- Runtime patterns
- Application development and systems management patterns
- Design, development, and deployment guidelines
- Code

### **2.1.1 Patterns for e-business and design patterns**

Design patterns describe atomic elements that are reused to create architectural structures. These structures could make up part or all of an application. Some structures are common enough to themselves constitute a reusable pattern.

The patterns for e-business falls into the latter category of patterns. They are made up of atomic elements and represent the 80 percent of applications that are made up of a common set of elements.

### 2.1.2 Components of the patterns for e-business

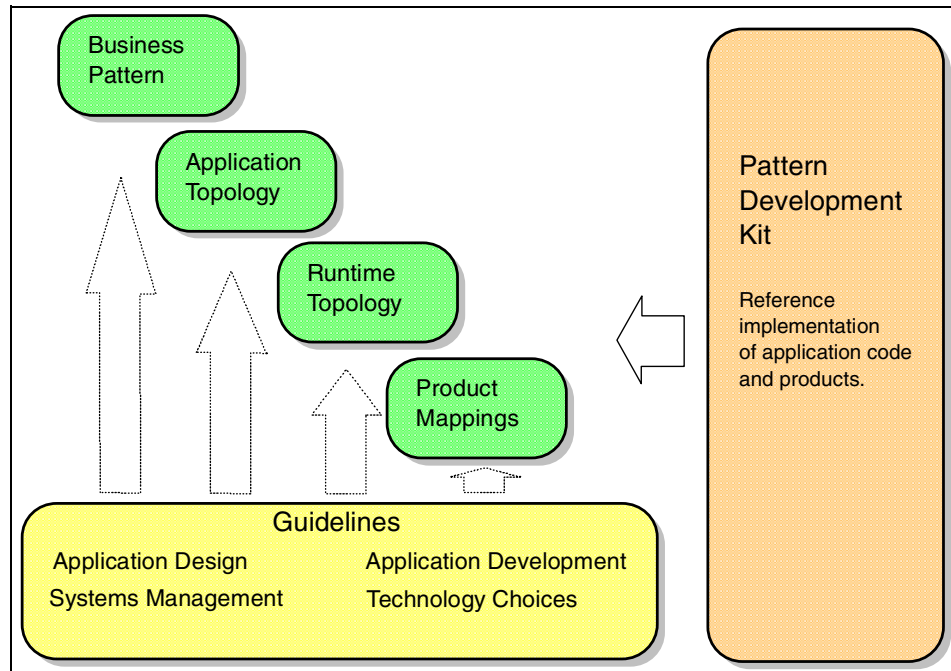


Figure 5. Patterns for e-business

Business patterns describe the interaction between the participants in an e-business solution.

The application topology illustrates the various ways to configure the interaction between users, applications, and data. Choosing an application topology will lead to an underpinning runtime topology.

The runtime topology uses nodes to group functional requirements. The nodes are interconnected to solve a business problem. An application topology leads to an underpinning runtime topology.

Product mappings show possible combinations of products used to instantiate the runtime topology.

The guidelines outline and define the processes used to build the e-business application.

### 2.1.3 Defined patterns for e-business

There are currently six defined patterns for e-business.

**User-to-Business** is the general case of users (internal or external) interacting with enterprise transactions and data. In particular, it is relevant to those enterprises that deal with goods and services that cannot be listed and sold from a catalog. It can also be thought of as covering all user to business interactions not covered by the User-to-Online Buying pattern.

**User-to-Online Buying** is used to describe the special case (a subset of the User to Business pattern) in which packaged goods, for example, are sold through a catalog using a shopping cart, a wallet, and so on. This includes both consumers purchasing goods or online buyers purchasing goods from a single supplier. It can also include links to backend systems to allow for inventory updates and credit checking.

**Business-to-Business** is used to describe two styles of inter-business-to-business. (Intra-business-to business is covered under Application Integration as follows).

The first style, B2Bi, covers programmatic links between arms-length businesses (where, potentially, a trading partner agreement may be appropriate). A good example of this would be Supply/Chain applications.

The second style covers the eMarketPlace where the model supports B2M2B. The M represents the eMarketPlace, which supports multiple buyers and suppliers. The buying function may be performed online or programmatically.

**User-to-User** is used to describe users collaborating with one another by email, shared documents, and so on.

**User-to-Data** is used to describe users needing to take large volumes of data, text, images, video, and so on and use tools to extract useful information from it.

**Application Integration** is used to link applications together within a business (like ERP with existing applications). This can be used within a business pattern or between business patterns.

As the following diagram shows, IBM views e-business as an integration of many application domains into systems that connect a business with its customers, partners, and suppliers.

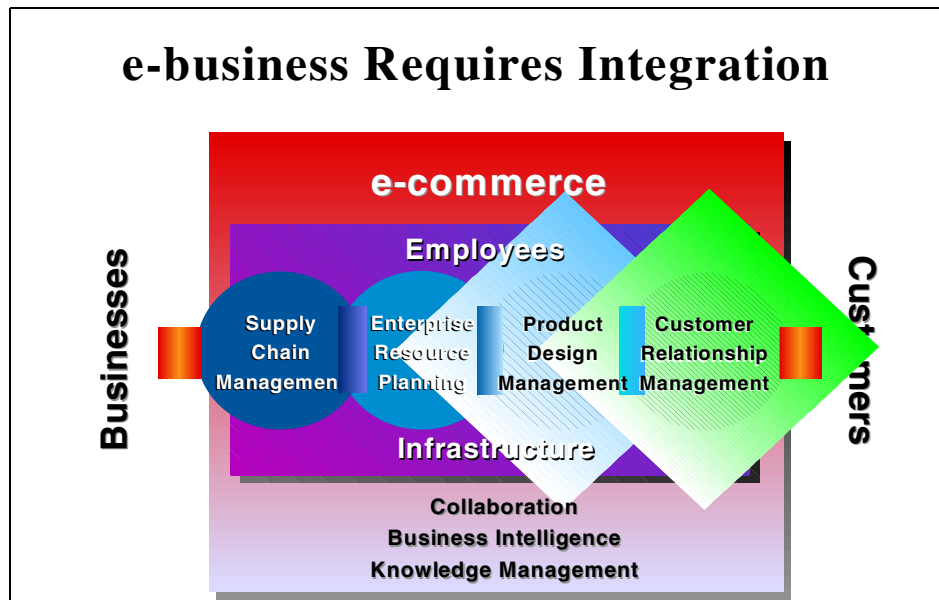


Figure 6. e-business integration

These systems are not confined to Web interfaces, although, increasingly, many of the user interfaces to the combined system will use Web technology.

The common set of node descriptions in the patterns for e-business enable communication between architects and designers from very different application domains and will suggest areas for shared nodes and infrastructure.

This is similar to the process of using design patterns to solve a programming design problem, where classes in the composed pattern play multiple roles derived from the source patterns. It is different, however, in that design pattern composition is, by nature, based on class diagrams and white box, whereas composing architectural patterns is more component-based.

The patterns for e-business may be applied to e-business solution areas. Table 1 is a guide to where you may find them most applicable.

Table 1. Patterns for e-business and e-business solutions

e-business solution area	Business pattern
Customer relationship management	User-to-Business Pattern
e-commerce	User-to-Online Buying Pattern

Supply chain management, e-Marketplace	Business-to-Business Pattern
Collaboration	User-to-User Pattern
Business Intelligence; Knowledge Management	User-to-Data Pattern
Business application integration	Application Integration Pattern

#### 2.1.4 How to use these patterns

The patterns for e-business are particularly focused on addressing common business application problems and providing answers to frequently-asked architecture, design, and implementation questions.

You can use the patterns for e-business in a number of ways according to your needs:

- As a starting point for an end-to-end system architecture
- As a detailed example and prescriptive approach following the product mappings and guidance provided
- As a way of designing more complex, multi-channel systems when several patterns are used together

As with the design patterns and ESS work, we anticipate that architects and designers will want to combine these patterns to compose solutions to more complex system architectures.

We recommend that you use the patterns for e-business together with an appropriate development methodology that considers the full set of requirements that are to be understood and implemented, whether these requirements concern the function of the solution or its operational characteristics, such as availability, scalability, or performance.

#### 2.1.5 Patterns for e-business Web sites

The patterns for e-business are published on IBM developerWorks, a portal for developers, and can be located at:

<http://www.ibm.com/software/developer/web/patterns>

This interactive patterns site acts as a guide to aid you in the selection of the pattern and topologies most relevant to your needs. While you can navigate

by way of shortcuts to the information you need most, the site is structured to enable you to drill down into the material as you:

1. Select a business pattern
2. Select an application topology
3. Review runtime topologies
4. Review product mappings
5. Review guidelines

At the time of writing, the Web site has material for the user-to-business and user-to-online buying patterns, with material for the other business patterns in the process of development.

You can also register at this site for pattern-related updates, which will include the Pattern Development Kit for User-to-Business when it is available.

---

## **2.2 The Business-to-Business Integration pattern**

The Business-to-Business Integration Patterns address the interaction of business processes between organizations. It can be viewed as taking enterprise application integration one step further to the integration of applications between businesses.

In addition to user-initiated transactions that follow patterns similar to those in the User-to-Online-Buying scenarios, B2B integration adds programmatic interactions that are key to automating B2B e-commerce across business boundaries. The patterns presented here should, therefore, be considered complementary to the User-to-Online-Buying patterns presented elsewhere. A good example of business-to-business integration is supply chain execution in which automated processes work across a supplier network.

The general problem addressed by these patterns is illustrated in Figure 7 on page 24.

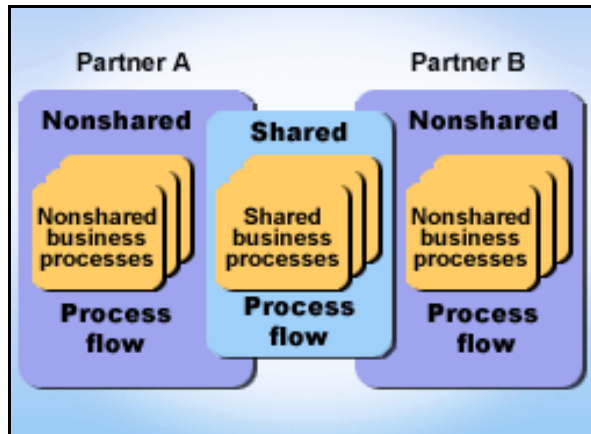


Figure 7. General B2Bi pattern

Interactions between partners form a shared process or, potentially, multiple distinct shared processes. Each of these must be integrated into the private business process flows implemented by each partner. Such integration might be as simple as passing data to a particular application or as sophisticated as initiating or resuming a multistep workflow involving several applications and user interactions.

The following are some industry examples where the Business-to-Business Integration pattern would provide the appropriate application and runtime topologies to fit each particular need.

#### **Manufacturing**

- Supply chain planning
- Supply chain execution

#### **Travel**

- Checking flight or room availability
- Making reservations
- Modifying reservations

#### **Retail**

- Checking supplier inventory
- Placing replenishment orders
- Paying suppliers automatically

#### **Financial**

- Transferring payments
- Checking account balances



- Obtaining credit information
- Processing securities

---

## 2.3 The Application Framework for e-business

The advent of e-business, with the requirement for interoperability that it brings, has been a major catalyst for the more rapid adoption of standards by the industry.

The IBM Application Framework for e-business establishes:

- A recommended approach for building systems, embodied in the patterns for e-business
- Innovative technology delivered in a rich product portfolio
- Cross-platform standards including Java and XML

The Framework, along with the standards it prescribes for e-business systems and their components, can be applied to:

- Custom application code
- Application packages
- Software products

The patterns for e-business are an integral part of the IBM Application Framework for e-business. The Patterns make it easy to apply the technologies, standards, and products of the Application Framework to provide an e-business solution.

---

## 2.4 Structure of this redbook

There are currently five logical application topologies associated with the Business-to-Business Integration pattern. In Chapter 3, “Choosing the application topology” on page 27, you will be given the information needed to choose the application topology that best suits your needs.

The structure we will be following is:

Chapter 3, “Choosing the application topology” on page 27, introduces all the application topologies for the Business-to-Business Integration pattern. With very accessible notation, these application topologies capture the essential “shape” of the application solution.

Chapter 4, “Choosing the runtime topology” on page 41, discusses the runtime topologies for these application topologies. It includes a discussion of

variations of these topologies that are appropriate for scalability and availability.

Chapter 5, “Technology options” on page 67, maps the runtime topology to various technology components. These components are required to satisfy the functional requirements within the runtime topology.

Chapter 6, “B2B integration protocols and standards” on page 89, goes on to further describe some of the protocols and the standards that are prevalent in the B2B Integration space.

Chapter 7, “IBM product guide” on page 101, provides a high-level description of the IBM suite of products. These are described in the context of their operation and functionality.

Chapter 8, “MQSeries and MQSeries integrator” on page 109, describes in MQSeries and MQSeries Integrator some detail. An understanding of these components is essential because they form the core of the IBM offerings for topologies 2 and 3.

Chapter 9, “Application design guidelines” on page 171, introduces consideration of the functional components of the application within the context of the runtime topologies.

Chapter 10, “Application development guidelines” on page 213, provides guidelines for application development, considering the roles, processes, and tools that are required.

Chapter 11, “Performance guidelines” on page 229, introduces performance guidelines by considering the components of the topologies under discussion that are particularly relevant to performance.

Chapter 12, “Systems management” on page 235, looks at the management of MQSeries and related components.

Chapter 13, “Getting a single customer view with MQSeries” on page 245, the third part of the book, looks at a solution with MQSeries, MQSeries Integrator, WebSphere, and DB2. This solution consists of a Web application that finds information about a customer on a number of separate systems.

---

## Chapter 3. Choosing the application topology

In Chapter 2, “Introduction to business patterns” on page 17, we described the patterns for e-business and how they can be used in different situations. This chapter describes the topologies that apply to the B2B integration pattern. Each topology is applied based on the solution requirements of the company.

An application topology shows the principal layout of the application, focusing on the shape of the application, the application logic, and the associated data. It does not show middleware or the files or databases where Web pages may be stored. Nor is the application design described in the application topology. For more information on application design see Chapter 9, “Application design guidelines” on page 171.

This chapter describes the application topologies for the B2B Integration pattern.

In Section 3.1, “Influencing factors” on page 27, we analyze the factors that would influence the solution architecture. This is not an extensive list of factors but some of the most common that would occur in most engagements.

In the sections that follow, we analyze how the topologies map to specific business requirements. Current and future requirements determine the level of function that needs to be implemented within the various components of the solution.

Usually, based on the function you require, a particular topology or variation of it will present itself. Each application topology has associated runtime topologies. The runtime topologies are covered in greater detail in the chapter that follows.

---

### 3.1 Influencing factors

It is clear that e-business is not just about technology. The business component of e-business is about redefining business models, reinventing business processes, changing corporate cultures, and raising relationships with customers and suppliers to unprecedented levels of intimacy. The technology component is the infrastructure put together to enable business.

In a B2B Integration solution, the business component is the business context for which the solution is developed. The technology component of the

solution manifests itself as the components required to achieve the required level of interaction for that solution.

### **3.1.1 Business context**

By definition, B2B Integration is the exchange of data between an application in one computer and an application in another computer. In B2B Integration, the second computer is in another company.

The following are some of the more obvious contexts of B2B Integration opportunities.

- Business to business integration, commerce
- Merger and acquisition integration
- Supply chain integration
- Customer relationship management integration
- Enterprise resource planning (ERP) packaged application integration
- Straight-through processing
- Web integration

### **3.1.2 Functional components**

To create inter-company processes, a number of functional components need to exist to enable the interaction. The location, the application interface, the infrastructure, and so on, are all functional components.

The exact nature of each of these components is usually a function of the relationship between the organizations wanting to interact. Existing components, dominance, and other factors will strongly influence the architecture of each of the functional components.

---

## **3.2 Application topology overview**

The current Business-to-Business Integration practices, typified by Electronic Data Interchange, and the emerging XML-based approaches, are formalized in the following five application topologies. These diagram conventions, shown in Figure 8 on page 29, are used to describe the topologies.

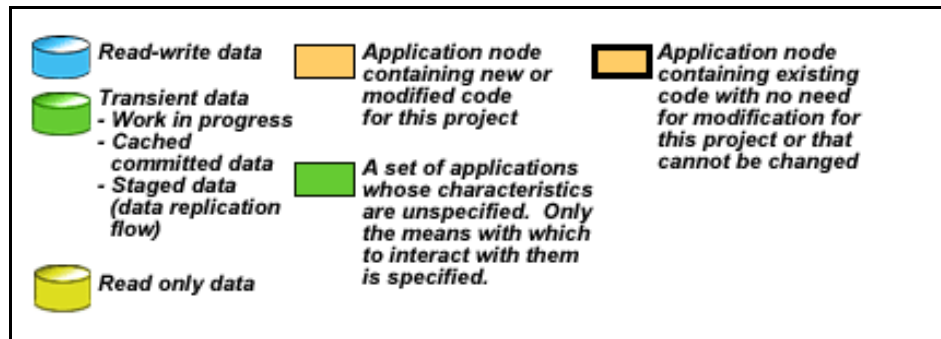


Figure 8. Diagram conventions for B2Bi

The five Business-to-Business Integration application topologies are presented here in order of increasing flexibility and sophistication. Topologies 1 and 2 focus on basic transport and data interchange. Topology 3, which builds on topology 2, employs application integration approaches that simplify interactions involving multiple applications. The interactions described in these topologies are stateless. In a stateless approach, the logic performed by the application does not depend on the state of the interaction between the applications. That is, the application's response to receiving a message depends solely on the message content.

A stateless scenario may include message content transformation and intelligent routing. Message content transformations refer to translating a message from one format to another format. Intelligent routing refers to controlling the message destinations. The routing decision can be based on the message content, source (system) or type among other possibilities.

Topologies 4 and 5 add formalized management of business protocols and other elements of the agreement between partners. They also address the problems of integrating inter-business processes with the intra-business processes implemented internally by each partner. These topologies represent a more complex scenario. Here, the applications may wish to have long-running conversations. That is, a complete transaction may be made up of a number of messages, each message being driven by an event. These applications may themselves be complex workflow or transaction-oriented applications.

This introduces state into the interaction. In a stateful interaction, the interaction logic performed by the integration mediator depends on previous application interactions. A stateful interaction is when the integration mediator accumulates events and sends a message to a destination application when



Topology 1 has the following characteristics:

- Both partners subscribe to a VAN and communicate using the infrastructure provided by the VAN. A VAN is a networking service that leases communication lines to subscribers and adds extra services or capability, such as security, error detection, guaranteed message delivery, and a message buffer.
- All partners in the interaction develop their own interfaces to their applications.
- Requests are usually batched.
- Mutually agreed upon messages, such as EDI transaction sets, are exchanged via mechanisms that allow the messages to be retrieved from a persistent buffer. The transmission format of the data is translated into a format usable by the internal business processes of the receiving organization. This transmission is performed by the VAN. VANs may provide additional services or software to translate messages.

### **3.3.1 Business driver**

Two partners are usually engaged in this application-to-application interaction to reduce administrative costs, improve the timeliness and accuracy of data, and promote a closer trading partner relationship.

Often, the factor driving this topology is a large EDI user placing a requirement on its partners to use EDI over a particular Value Added Network (VAN). Prior to the growth of the Internet, using EDI over a VAN represented the only mechanism for the reliable exchange of information.

Although it is an inflexible topology, two partners can use this topology to maximize the performance of the interaction, particularly by batching requests.

### **3.3.2 Considerations**

EDI is a well-established standard, but it is deployed by only a small number of companies. It applies mostly to partners who need to participate in an existing EDI-based network. For businesses that need flexibility in connecting to multiple partners that might have different IT infrastructure capabilities, this topology might not be appropriate.

### **3.3.3 Examples**

A producer of automobile parts enters into an agreement to be a supplier to a major auto manufacturer. The manufacturer manages an established EDI

network through which its entire supply chain operates. Because the specification given to the parts provider includes automotive industry X12 EDI transaction sets to be used in the EDI interactions, the supplier chooses Topology 1 because it directly supports the EDI approach.

### 3.4 Topology 2: Direct with adapter/bridge

Topology 2 makes a set of applications available for direct access by outside organizations. An extended message-based interaction includes an adapter or bridge that converts the mutually agreed to messages into API calls to existing applications.

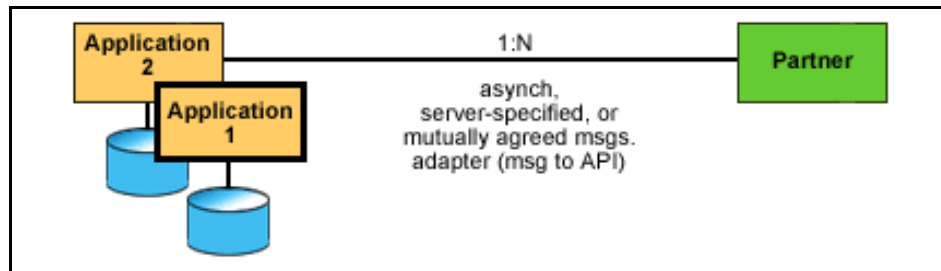


Figure 10. Topology 2 - Direct connection with adapter/bridge

This integrates an existing application across organizational boundaries. New applications, represented by Application 2, are integrated without the adapter by directly using messaging interactions. The types of interactions are limited by the functions and by the restrictions of the particular applications made available to the partner. This topology has the same motivation as the adapter design pattern - To convert from a given (server) interface into an interface that a client expects (message transformation). In a non-intrusive adapter design, the server application doesn't know of the adapter. In an intrusive adapter design, the server application is modified in some way.

The types of interactions are limited by the functions offered by the standalone applications designated as available. More sophisticated business interactions, however, might require several back-end applications participating jointly in a single message exchange giving rise to the more complex topologies described further in this chapter.

The application could have an existing EDI interface that needs to be extended to include more open standards, or it could be an application that had no external interface.



This topology provides a flexible approach to exporting legacy application services.

This topology usually requires that the partners agree on the external communications protocols. The internal communications are the responsibility of each of the partners. This topology usually does not have a highly complex Process/information coordination function.

#### **3.4.1 Business driver**

For low-complexity message exchanges, Topology 2 provides interaction with one or more specific applications. It applies when a more sophisticated interaction between the partners is not necessary or when this interaction is a specialized case within a larger cooperation. Because it is based on reliable messaging, for example, message queuing, Topology 2 provides message availability and manageability.

#### **3.4.2 Considerations**

The message definitions should be made as general as possible to promote some flexibility in changing the interface of the application without affecting the agreed-upon message definitions. Still, even with this abstraction, this topology is not easily generalized to more sophisticated integration of business processes and is sensitive to changes in the applications.

#### **3.4.3 Examples**

A small manufacturer is contracted to produce a set of parts for a major industrial company that produces aircraft engines. The contract specifies not only the product to be produced but the mechanisms for scheduling production, reporting production status, and producing shipment status as well. The aircraft engine company has an internal message queuing network and specifies, as part of the agreement, that the partner must communicate with it over the same message-oriented middleware. The small manufacturer has a production scheduling application that also provides production status and another application that generates shipping reports. These can be integrated with message-oriented middleware using adapters. Because only two applications are involved and the interactions with the aircraft engine manufacturer are very limited, no additional integration approaches, such as message brokering, are needed. Topology 2 is a simple solution to this small manufacturer's integration problem.

### 3.5 Topology 3: Message broker

Topology 3 exposes applications or sets of applications to an outside organization as a set of services. A message broker routes and transforms messages as appropriate for the application or applications implementing a service, employing adapters as necessary. This message broker maximizes both isolation of business processes from the outside organization and the flexibility to change the processes and the applications that implement them. It also allows a combination of applications or business processes to handle a single message from a partner. The “Decomp Rules” node in this topology represents the message broker message transformation and routing definitions supplied by the developer, based on the requirements and capabilities of the business applications. App 1 and App 2 represent applications that are integrated through a message-to-API adapter and a direct messaging interaction, respectively.

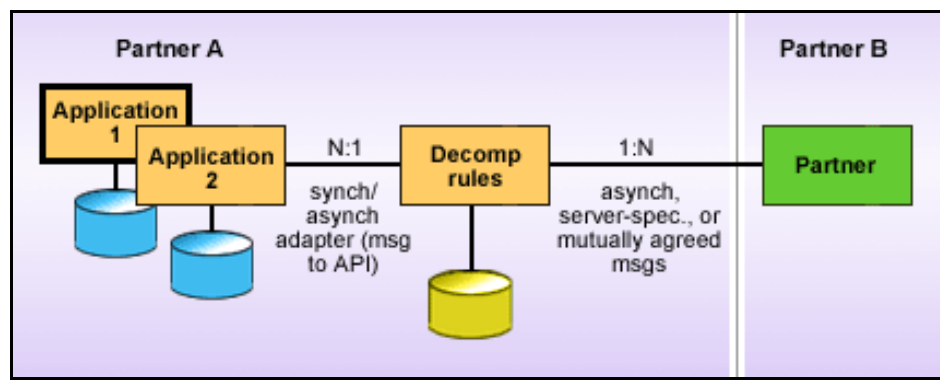


Figure 11. Application Topology 3 - Message broker

#### 3.5.1 Business driver

Message brokers and application adapters are common building blocks of enterprise application integration. This topology leverages the investment made in application integration to extend beyond the enterprise. It provides independence of implementation and configuration, and, because it is based on message queuing, it also provides availability and manageability of the message flow.

#### 3.5.2 Considerations

When enterprise integration approaches are extended outside the company, additional problems must be addressed, including the enforcement of business-to-business protocols. This might require adding handcrafted

protocol management code to the message flow prior to routing a request to business process applications. In addition, using message queuing interactions between partners requires that both partners use common message-oriented middleware.

The application topology shown in Figure 11 on page 34, embodies the concept of a services-oriented architecture in which applications or sets of applications are exposed to an outside organization as a set of services. Integral to this topology is a message broker that can route and transform messages as appropriate for the application or applications implementing a service, employing adapters as necessary. This maximizes both isolation of business processes from the outside organization and the flexibility to change the processes and the applications that implement them.

### **3.5.3 Examples**

Because Topology 3 is closely related to Topology 2, a similar example can be used. A small manufacturer is contracted to produce a set of parts for a major industrial company that produces aircraft engines. The contract specifies not only the product to be produced but the mechanisms for scheduling production, reporting production status, and even producing shipment status. In addition, the small manufacturer is required to participate in the production planning process. The aircraft engine company has an internal message queuing network and, as part of the agreement, specifies that the partner must communicate with it over the same message-oriented middleware. The small manufacturer has a production scheduling application that also provides production status and another application that generates shipping reports. The requirement to participate in production planning requires data to be extracted on demand from several enterprise resource planning applications. The applications can be integrated with message-oriented middleware through adapters, but any interactions with the partner require the participation of several applications to satisfy a single request. This can be addressed by a message broker corresponding to the "decomp rules" component of this topology. Topology 3 provides the extra integration flexibility needed to address the relationship with the manufacturing partner.

---

## **3.6 Topology 4: Managed business protocol**

The Managed Business Protocol topology combines the services and broker approach of the Message Broker topology with management of the business protocol between the two trading partners. It includes an executable contract, agreed to by the trading partners, that governs the business-to-business

interactions, specifying such things as the message protocol, security and encryption standards to be used, and permissible message sequences. The messages now form a conversation and are not independent.

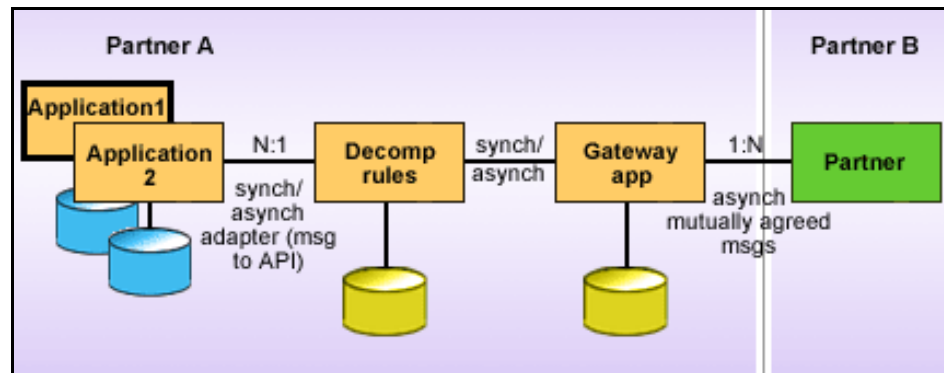


Figure 12. Application Topology 4 - Managed Business Protocol

As in Topology 3, the implementations of service interfaces might use brokering to interact with internal business processes.

The “Gateway App” node in this topology includes the executable contract and services that turn the business-to-business protocol message into the appropriate message to the back-end applications, generally, through a message broker. The Decomp Rules node represents the message broker message transformation and routing definitions that are supplied by the developer based on the requirements and capabilities of the business applications. App 1 and App 2 represent applications that are integrated through a message-to-API adapter and direct messaging interaction, respectively.

This topology supports multiple partners through a single “Gateway App” node. For a specific partner, you can employ multiple business-to-business protocols.

### 3.6.1 Business driver

This topology maximizes flexibility while assuring that agreed-upon messaging protocols are followed. Like Topology 3, it leverages the investment made in application integration to extend beyond the enterprise. Using a gateway application, while implying additional infrastructure, addresses business-to-business protocol handling, such as enforcing communications and security protocols, ensuring that messages are delivered only one time, and authenticating partners. It extends the pure

message broker approach, enabling accommodation and control over a variety of communications protocols, security standards, and message choreographies without writing any additional code.

This topology maximizes flexibility while providing assurance that contractual agreements are being met. Like message broker topology, it leverages the investment made in application integration to extend beyond the enterprise. The use of interaction rules, while implying additional infrastructure, addresses the business-to-business problems not addressed by the current practice in enterprise application integration.

### **3.6.2 Considerations**

The contract implemented by the gateway should be based on model contracts that correspond to standardized business-to-business protocols, such as Open Buying on the Internet (OBI) and RosettaNet, a protocol for IT supply chain management. Business partners have a choice of either hand-coding a business-to-business protocol management system to ensure compliance with the Trading Partner Agreement (TPA) or installing an instance of the gateway, deploying the same XML script (TPA) to govern the B2B message exchange.

### **3.6.3 Examples**

Suppose a producer of computer components wants to integrate into the supply chains of several personal computer manufacturers. Because the Information Technology industry has jointly developed an IT supply chain standard, called RosettaNet, and the PC makers it wants to supply support this standard, the company decides to use it to integrate into these supply chains. RosettaNet is a sophisticated B2B protocol requiring considerable effort to implement. A B2B gateway is needed to map this protocol into interactions with internal applications and processes. The component producer in this example has a manageable number of applications involved, and the business processes concerned with the supply chain interactions are fairly simple; therefore, elaborate workflow management is not needed to integrate with the RosettaNet processes. Topology 4 supports such a deployment, in which the business-to-business processes are sophisticated, but the integration with internal applications is manageable using message decomposition.

---

## **3.7 Topology 5: Managed business protocol and process**

The Managed Business Protocol and Process topology combines the services and broker approach of the Message Broker topology with

management of the business protocol between the two trading partners under the umbrella of integrated B2B and internal business workflow.

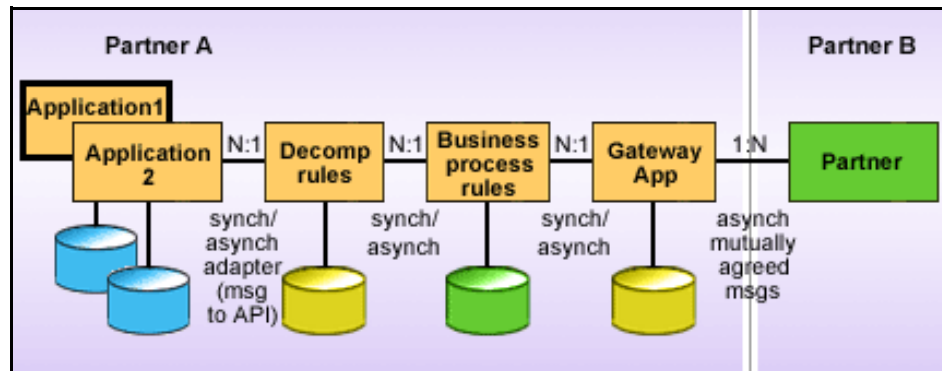


Figure 13. Managed Business Protocol and Process

Shared messaging protocol definitions (which can be encoded using executable XML "contracts" of "Trading Partner Agreements") mediate the instantaneous business-to-business interactions, while agreed-upon inter-business workflows govern the long-running transactions comprising entire business process cycles, such as Submit RFQ - Receive Price Quotation - Issue Purchase Order - Manage Fulfillment. Major steps in the B2B workflow are executed by business applications that can implement a micro workflow including all human and machine interactions needed to complete such a major step in the business process (approve Purchase Order, for example). This micro workflow then employs application integration techniques, including message brokering and application adapters, to execute its function.

As in Topology 4, the Gateway App node in this topology includes the executable contract and services that turn the business-to-business protocol message into the appropriate message to the backend applications. The Business Process Rules node includes the definitions of the long-running transactions (workflow) that trading partners have agreed upon and the micro workflow needed to execute them. This node drives the execution of transactions. The Decomposition Rules node represents the message broker message transformation and routing definitions that are supplied by the developer based on the requirements and capabilities of the business applications. App 1 and App 2 represent applications integrated using a message-to-API adapter and direct messaging interaction, respectively.

This topology supports multiple partners. For a specific partner, you can employ multiple business-to-business protocols and processes. Similarly, multiple internal business processes can be integrated with B2B workflow.

### **3.7.1 Business driver**

This topology maximizes flexibility while providing assurance that agreed-upon messaging protocols and workflows are being followed. Similar to Topologies 3 and 4, it leverages the investment made in application integration to extend beyond the enterprise. The use of both messaging protocol and business process management, while implying additional infrastructure, addresses business-to-business integration in virtually all its aspects. It includes enforcing agreed-upon message choreographies, message formats, communication and security protocols, and workflow steps. This approach also provides an extendable environment to model and deploy business-to-business processes. The flexibility gained by the business process definition lets you replace handcrafted business applications using predefined application building blocks knitted together through business workflow scripts.

The automated business process management topology also cuts down the application code required to execute business processes by separating the workflow logic from the modules executing business transactions. Like message broker approaches, it provides independence of implementation and configuration but extends this independence further to the long-running transactions managed by a workflow engine. In integrating business processes, workflow management provides flexibility in modeling and changing the sequence of actions, while message routing and transformation provides the same kind of flexibility in modeling and changing the flow and format of communications.

### **3.7.2 Considerations**

The messaging protocol implemented by the gateway should be based on standardized business-to-business protocols, such as Open Buying on the Internet (OBI) and RossettaNet, a protocol for IT supply chain management. Tools for business process modeling define the flow of action required to execute a business process within and across organizations. A holistic approach begins with a description of (1) the roles and responsibilities, (2) the task flows and processes, and (3) the data definitions and documents underlying the joint business processes. The approach then employs a set of tools to transform these descriptions into a consistent and coherent set of message definitions, workflow definitions, directory entries, database schemas, and so on, which can be deployed onto a runtime architecture

representing a software implementation of these processes. This "ultimate" approach to business-to-business interactions can be modified in the future with increasing levels of depth and technical detail.

### **3.7.3 Examples**

Suppose a computer manufacturer needs to manage its supply chain. Just like the supplier in the example in Topology 4, this manufacturer decides to use the RosettaNet standard to integrate with suppliers of IT technology. This manufacturer recognizes that RosettaNet is a sophisticated B2B protocol and will require a B2B gateway to map this protocol into interactions with internal applications and processes. Because the computer manufacturer has a number of significant business processes controlling production, parts inventory, procurement, and other critical activities, the integration of the B2B process flows with the company's private processes requires more than simple application mapping. Long-running inter-business workflows must be managed, and private processes must be woven into them. This requires the Business Process Rules application, which is a superset of the B2B gateway needed to handle the protocol. The computer manufacturer needs the full sophistication of Topology 5.

---

## **3.8 Topologies summary**

The aforementioned topologies represent some of the most commonly used patterns. However, they are still only a subset of the possible combinations of solutions that could be deployed.

The five Business-to-Business Integration application topologies were presented in order of increasing flexibility and sophistication. Topologies 1 and 2 focus on basic transport and data interchange. Topology 3, which builds on topology 2, employs application integration approaches that simplify interactions involving multiple applications. Topologies 4 and 5 add formalized management of business protocols and other elements of the agreement between partners. They also address the problems of integrating inter-business processes with the intra-business processes implemented internally by each partner. As the topologies build on each other, their capabilities and reliance on middleware increase, and they require less application development effort.



---

## Chapter 4. Choosing the runtime topology

Once the application topology has been chosen, it is time to choose the runtime topology. The runtime topology uses nodes to group functional and operational components. The nodes are interconnected to solve a business problem. An application topology leads to an underpinning runtime topology.

This chapter is divided into a three overall sections. The first section reviews the concept of a runtime topology and its elements.

The second section describes each of the five runtime topologies. Each of these topologies has two illustrative examples. Since there could be endless combinations, only two are described to illustrate the overall concept.

The third section provides an introduction to each of the nodes types.

For each topology, we introduce the topology, provide a graphical representation, describe its structure, and list some of its characteristics.

Every description of a variation, including the basic runtime topology, is self-contained.

---

### 4.1 Runtime topologies

Topology is defined as the arrangement in which the nodes are connected to each other. Runtime topology refers to the nodes that will be used to provide runtime support for the solution.

The runtime topology uses Nodes to group functional requirements, with each node representing a functional requirement. The nodes are interconnected to solve the business problem. The runtime topology will closely map to the physically-deployed solution. The node functionality is usually represented by a software and/or hardware component.

A physical hardware component, such as a server, could perform more than one function, thereby, representing, in the physical solution, more than one node.

Each of the nodes used in the B2Bi runtime topologies is described in Section 4.2, "Topology 1: Document exchange" on page 42.

---

## 4.2 Topology 1: Document exchange

The Document runtime topology represents the current practice for EDI interactions between businesses. Message interactions are mediated by a VAN service provider that delivers messages to mail boxes

The VAN provider would, in this topology, provide all the translation software and may also provide a translation service before the message is accepted into the receiving application.

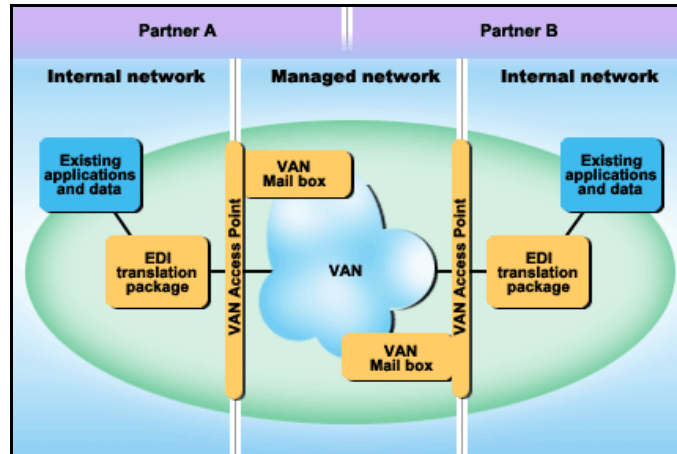


Figure 14. Application Topology 1: Runtime topology

Section 3.3, “Application Topology 1: Document exchange” on page 30, describes, in some depth, the characteristics of this application topology. Variation 1, described in Section 4.2.1, “Illustrative Example 1: Document runtime topology” on page 42, represents the typical VAN solution. Variation 2 describes an emerging alternative, sending EDI documents over the Internet.

### 4.2.1 Illustrative Example 1: Document runtime topology

Basically, the VAN-Document runtime topology represents a networking service that leases communication lines to subscribers and adds extra services or capability, such as security, error detection, guaranteed message delivery, and a message buffer between companies using EDI as a document exchange protocol.

The goal of Electronic Data Interchange (or EDI) is to provide an electronic transmission medium for the delivery of a standard business document in a

predefined format from one company's application to its trading partners's application.

Typically, the VAN is implemented by a third company, which provides the physical communication and services. In this way, the companies do not need to buy their own physical communication infrastructure. Keep in mind that this was prior to the Internet becoming a popular network infrastructure.

This runtime topology represent the current practice in business-to-business integration used principally by large corporations.

### **Structure**

Figure 15 depicts the VAN-Document runtime topology for the document exchange model, which represents the current practice for EDI interactions between businesses. Message interactions are mediated by a VAN service provider that delivers messages to mail boxes. Both companies' applications should use the EDI translation package and the VAN access point to translate and transmit their traditional business documents in EDI formats. Typically, these nodes are proprietary software installed into the organization system by the VAN service provider.

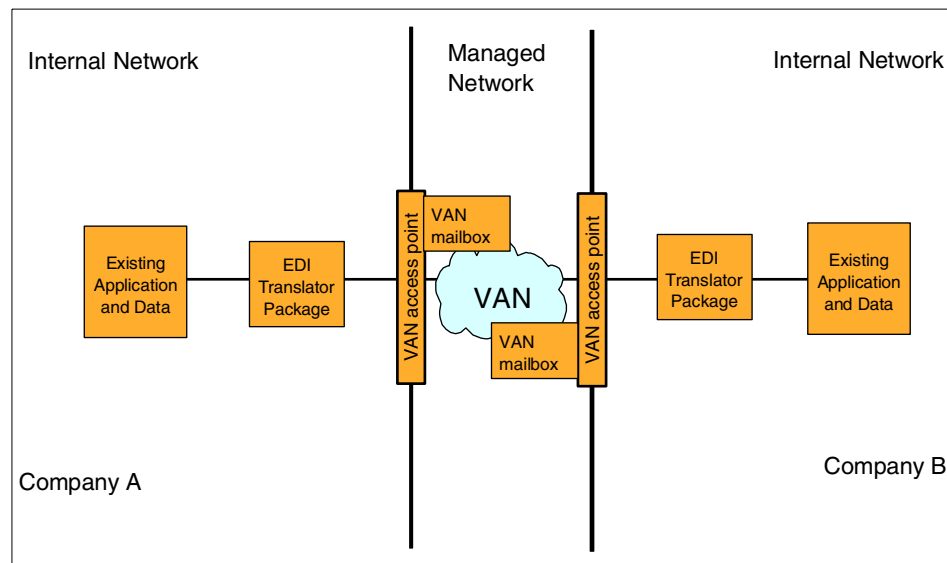


Figure 15. Runtime topology: VAN-Document

### **Characteristics**

Companies A and B have subscribed to a VAN. Users of a VAN send messages to and retrieve messages from a mailbox. This service of the VAN

holds messages until the receiver requests them. The VAN access point represents the networking endpoint of a VAN. It is the company's connection to the VAN. As its simplest form, an EDI translator converts EDI transaction sets (EDI messages) to and from flat files of a format needed by an enterprise's applications.

#### **4.2.2 Illustrative Example 2: EDI - Internet Runtime topology**

The most appealing benefit of using the Internet for EDI is the low cost. The VAN providers charge a great deal of money for their services; furthermore, the charge is, at times, relative to the number of transactions made by the company. An Internet Service Provider (ISP) is usually much less expensive than the VAN provider and does not charge by the transaction.

This makes it possible for small and medium companies to gain access to EDI technology and work with large corporations that use EDI with a VAN at a reasonable price. Consequently, the number of potential new users for these systems could increase significantly.

This variant defines an intermediate tier between the Internet world and the VAN environment to allow EDI Documents to be exchanged between companies that use the Internet and companies that use a VAN.

A technical paper from the IBM Institute of Advanced Commerce, titled *A Practical Approach to Web-Based Internet EDI*, is available at <http://www.ibm.com/iac/papers/icdcsws99/index.html> and introduces a Web-based EDI model.

##### **Structure**

Figure 16 on page 45 shows the EDI - Internet runtime topology and its nodes. The components between the Internet cloud and the VAN cloud are the gateway nodes that enable communication between Internet and the VAN.

The figure does not show the company that has direct access to a VAN, which would appear on the left of the figure. This part is similar to variation 1 (see Figure 15).

When an existing application in Company B need sends a document to another company connected to the VAN EDI system, it uses the EDI/MIME translator to map the application data to the EDI document structure. The document is then packaged in a MIME message and sent to an SMTP (Simple Mail Transfer Protocol) server.

The SMTP Server in Company B sends the message to the SMTP server at the VAN provider. Both SMTP Servers use the Directory and Security services to encrypt the message as well as to document and authenticate the message sender.

The Internet Gateway node retrieves messages stored in the SMTP Server and transform the MIME message into an EDI message, it then sends it to the corresponding VAN mailbox, using the VAN access point.

Any document stored in the VAN mailbox of company B is retrieved by the Internet Gateway and sent to the SMTP Server to be delivered to Company B.

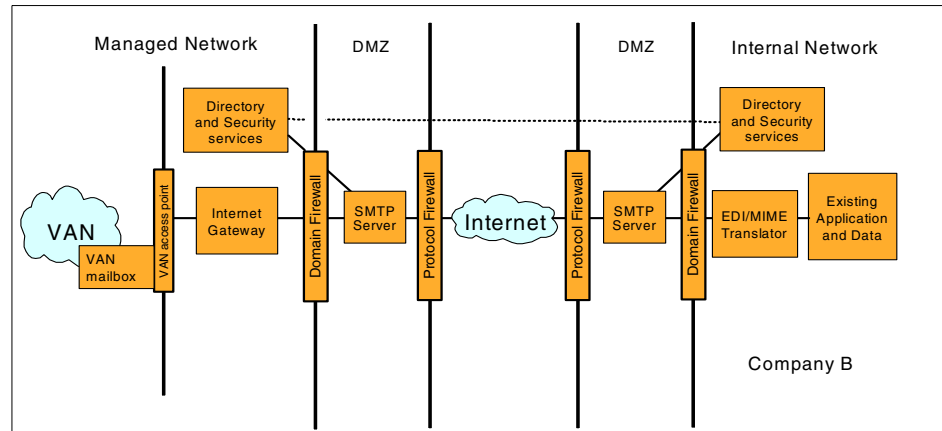


Figure 16. Runtime topology: EDI - Internet

### Characteristics

The following list contains the principal characteristics of this runtime topology:

- The EDI software should provide function equivalent to the EDI translator package described in Variation 1. Here, the principal difference is that the transport layer uses the SMTP/MIME format.
- The SMTP service could be provided by an ISP Company (the same VAN service provider could provide this service). Consequently, Company B would not require an SMTP server, which can be expensive for small companies. It could just use the e-mail server provided by the ISP company.
- In the Managed Network systems, the VAN access point is protected behind the Domain Firewall to prevent unwanted access.

- The Internet Gateway does not worry about the stored and forward problem; this is resolved by the SMTP server and the VAN mailbox node. It only transforms the inbound and outbound messages into the corresponding format (VAN format to SMTP/MIME format or vice versa).

### 4.2.3 Summary

This section describes how the VAN - Document runtime topology defines a communication standard based on traditional EDI Document formats that is used by a great number of large companies worldwide.

Variation 2, EDI - Internet describes a low-cost solution that allows small and medium companies to have access to the VAN-Documents using the Internet as a medium to transmit the documents and SMTP/MIME protocols to wrap and route the Documents to the VAN environment.

---

## 4.3 Topology 2 - Direct with adapter/bridge

This topology describes a direct integration without intermediate tiers. This is also called point-to-point integration. Non-integrated applications are adapted to understand mutually agreed upon messages; this allows the export of the services of these applications across organizational boundaries. Once exported, the services of these applications become reusable assets that can be much more effectively leveraged to meet business needs.

Two categories of integration can be identified: *Front-end* and *back-end*. Front-end integrations deal with user interfaces and are typically used in User-to-Business situations. In B2B situations, however, the integration is usually performed on the application's back-end. The effort needed to develop a back-end adapter is influenced heavily by the existing coupling between the applications front and back ends.

The partner's application invokes the services of the application through the adapter interface. The adapter doesn't know of the partner's application, and in a non-intrusive adapter design, the application doesn't know of the adapter.

Figure 17 on page 47 makes some assumptions. The first is that there is a symmetrical topology that both partners select to use a message queue. This is likely but may not necessarily happen in practice. Second, the use of a queue manager implies shared middleware, which, again, may not necessarily be done in practice. The protocol running over the communications link would be determined by the latter selections.

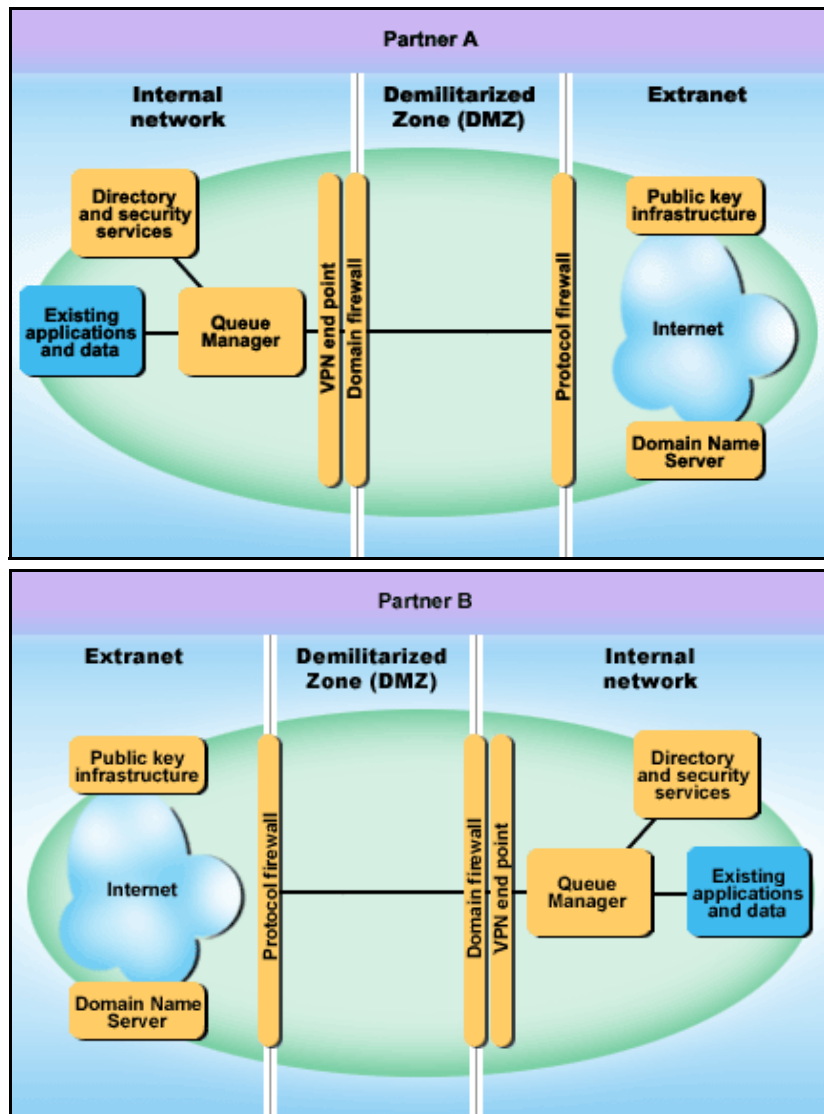


Figure 17. Application Topology 2 - Runtime topology

#### 4.3.1 Illustrative Example 1- Shared middleware

The middleware adapter adapts the backend of an existing application to an interface defined by an asynchronous Message Queuing (MQ) middleware product and implements mutually agreed upon messages that can be exchanged with partners that have deployed compatible middleware.

### **Synchronous middleware**

Adapters can also integrate applications with synchronous middleware products, such as object request brokers (CORBA/DCOM) or Distributed Computing Environment (DCE). Although the use of a synchronous adapter is not common in a B2B topology, it does make sense to leverage Enterprise Application Integration (EAI) investments made in such synchronous middleware, for example, using the Internet Inter ORB Protocol (IIOP) to integrate two CORBA implementations.

#### **Structure**

This variation is represented using the following assumptions: The shared middleware is configured as server-to-server. Both parties implement MQ middleware servers behind their firewalls; so, there are no nodes inside the DMZ. The firewalls have to be configured to allow the traversal of the MQ message channel. Alternatively, you could implement MQ Internet Passthru, which acts as a proxy and wrappers the MQ transmissions in HTTP, sending it through standard ports in the firewall. See Appendix B, “MQSeries Internet pass-thru” on page 307.

An alternative configuration could be a client-server one, which uses a single server and should allow the traversal of the MQ client channel through the firewalls.

Another assumption is that security is implemented on the network level by using a Virtual Private Networking (VPN) protocol, such as the Secure Internet Protocol (IPSec).



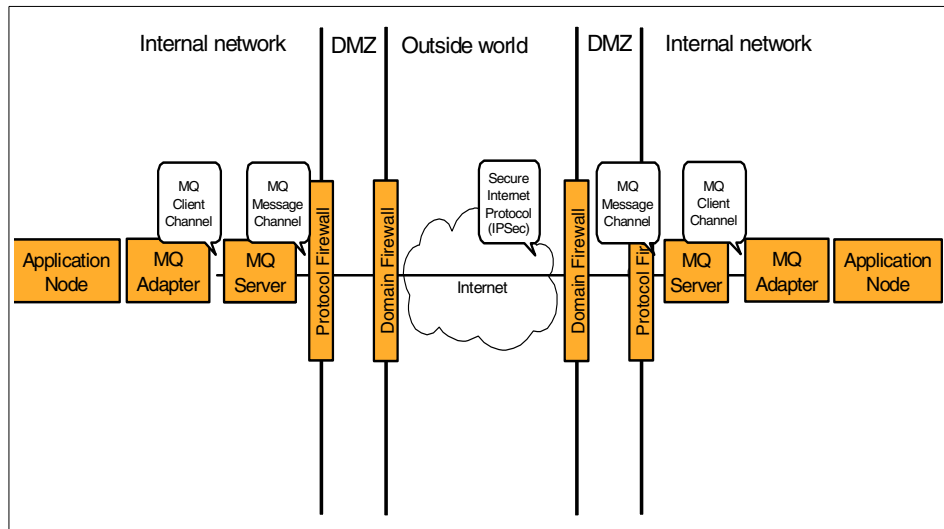


Figure 18. Illustrative example of topology two using shared middleware

### 4.3.2 Illustrative Example 2- Open standards

Although the shared middleware variation can provide a short term solution to meet interconnectivity requirements, in the long run, a solution that supports heterogeneous middleware is needed. One of the problems with current middleware technologies is that they often enforce the use of proprietary or uncommon protocols. Internet access that is being controlled by firewalls and filtering routers may need to be adapted to support these protocols. Only the most common protocols, such as HTTP and SMTP, are able to traverse typical firewall and router configurations.

The solution to this problem is to include an adapter that transforms the proprietary middleware protocol into HTTP-based protocols that are able to traverse typical firewall and router configurations. Such adapters will eventually become part of the middleware packages.

Several technologies address this issue by transforming internal middleware protocols to XML- and HTTP-based protocols that are able to traverse common firewall and router configurations. One of those technologies is the Simple Object Access Protocol (SOAP). Others addressing the same problem are XML-RPC, WDDX, and WIDL.

## SOAP

SOAP is a protocol specification for invoking methods on servers, services, components, and objects. SOAP codifies the existing practice of using XML and HTTP as a method invocation mechanism. The SOAP specification mandates a small number of HTTP headers that facilitate firewall/proxy filtering. The SOAP specification also mandates an XML vocabulary that is used to represent method parameters, return values, and exceptions.

## Structure

Figure 19 shows a topology using open standards.

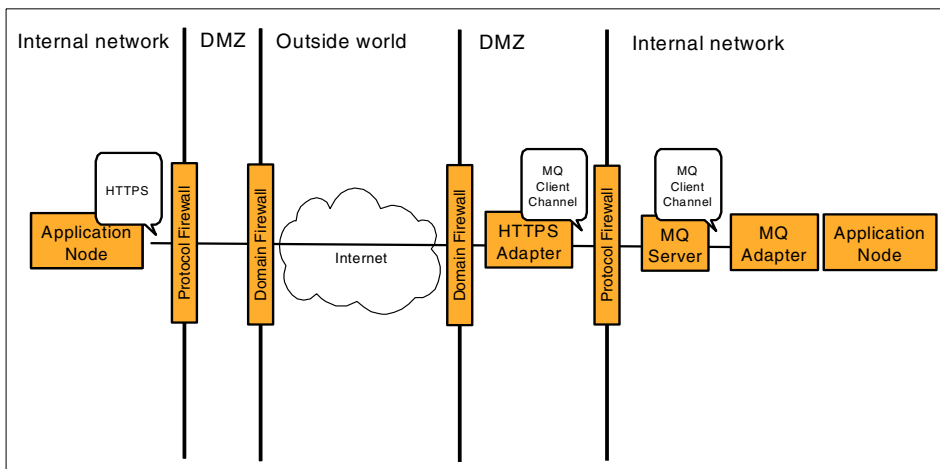


Figure 19. Illustrative example of topology two using open standards

### 4.3.3 Summary

The direct topology integrates applications without an intermediate tier. The open standards adapter provides an interface using common Internet protocols that will, typically, be able to traverse firewall and router configurations. The middleware adapter, however, adapts to middleware technology that is to be shared with the partner. Usually, this does not automatically traverse common firewall and router configurations and requires special care. Often, the use of middleware technology will increase the need for an intermediate tier, a broker, which is covered in topology 3.

#### 4.4 Topology 3 - Message broker

This topology describes an integration based on an intermediate tier, the message broker. Applications exchange messages with this broker only. The broker performs the necessary routing and transformation for each message in accordance with a predefined message flow (set of processing steps). A broker is often being deployed as the result of Enterprise Application Integration (EAI) investments. Figure 20 makes some assumptions. First, there is a symmetrical topology both partners select to use a message queue. This is likely but may not necessarily happen in practice. Second, the use of a queue manager implies shared middleware, which, again, may not necessarily be done in practice. The protocol running over the communications link would be determined by the latter selections.

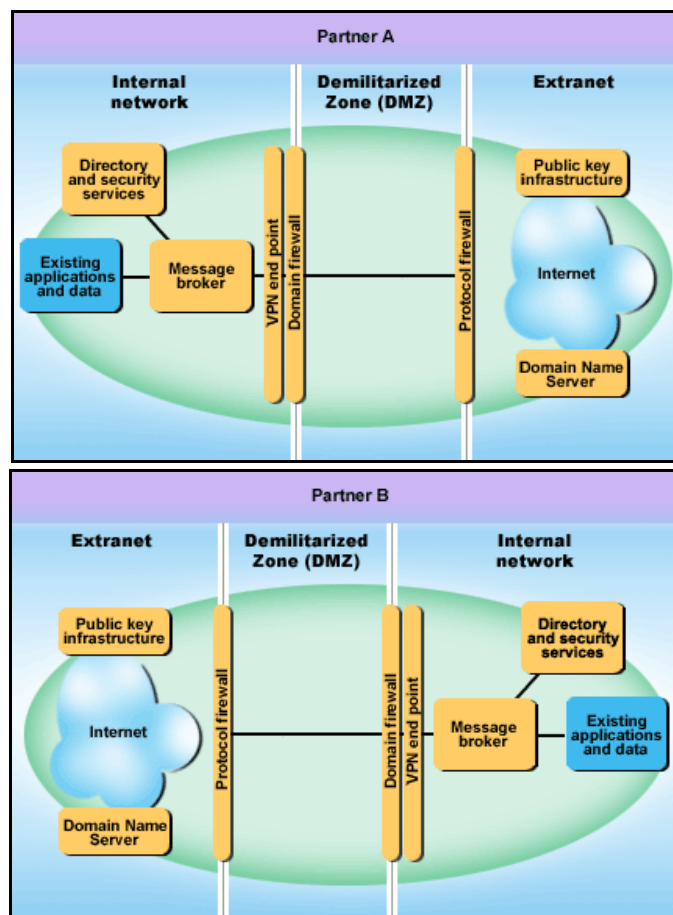


Figure 20. Application Topology 3: Runtime topology

#### 4.4.1 Illustrative Example 1- Shared middleware

This base runtime topology uses a message broker to integrate various legacy applications by providing message queuing, translation, and intelligent routing. The message broker transforms the mutually agreed upon messages exchanged across organizational boundaries into the native messages of the applications.

##### 4.4.1.1 Structure

This topology is almost identical to the shared middleware variant of topology 2. This also assumes a server-to-server configuration and VPN-based security. The only differences are that the MQ server is now been enhanced with Message Broker functionality, and multiple back-end applications are taken into consideration. Figure 21 shows topology 3 using shared middleware.

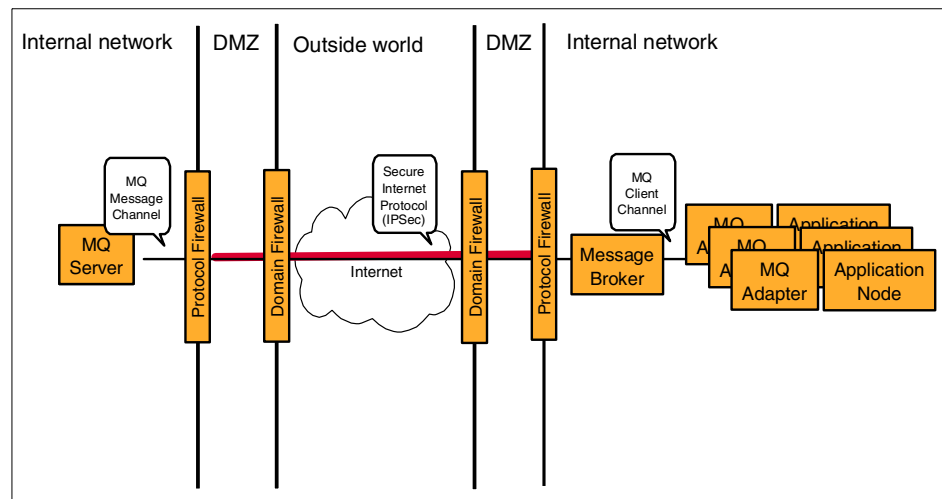


Figure 21. Illustrative example of Topology 3 using shared middleware

#### 4.4.2 Illustrative Example 2 - Open standards

The motive for a variation based on open standards is identical to the previous topology: Seamless traversal of the Internet infrastructure formed by firewall and router configurations.

##### 4.4.2.1 Structure

This topology is almost identical to the open standards variant of topology 2. The messaging middleware is being front-ended with an open standards adapter. As with the shared middleware variation of this topology, the

message queuing middleware is enhanced by a message broker, and multiple back-end applications are taken into consideration. Figure 22 shows topology 3 using open standards.

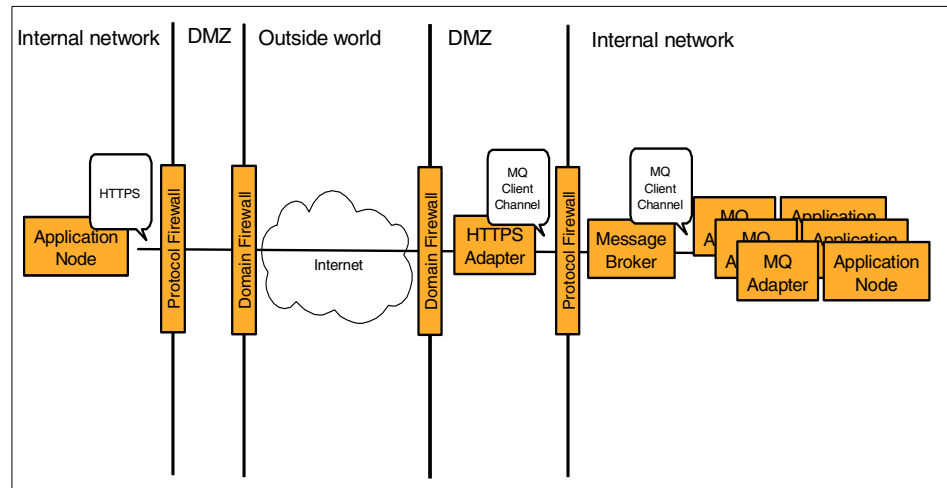


Figure 22. Illustrative example of topology 3 using open standards

#### 4.4.3 Summary

The message broker topology leverages EAI investments made in middleware technology. Legacy applications are adapted to this middleware using the shared middleware adapters from topology 2. The message broker itself can also be adapted to support standard protocols, which are often required in a B2B scenario, by using an open standards adapter, or Internet gateway.

### 4.5 An introduction to the node types

The runtime topologies will be shown in graphical form in the following sections. Each topology will consist of several nodes describing the function represented on that node. Most topologies will consist of a core set of common nodes with the addition of one or more nodes unique to that topology. To understand the runtime topologies, you will need to review the following node definitions.

#### 4.5.1 Extranet

An extranet is a private network that uses the Internet protocols and the public telecommunication system to securely share part of a business'

information or operations with suppliers, vendors, partners, customers, or other businesses. An extranet can be viewed as part of a company's intranet that is extended to users outside the company. It has also been described as a "state of mind" in which the Internet is perceived as a way of doing business with other companies as well as selling products to customers. The same benefits that HTML, HTTP, SMTP, and other Internet technologies have brought to the Internet and corporate intranets now seem designed to accelerate business-to-business integration.

#### **4.5.2 Intranet**

An intranet is a private network that is contained within an enterprise. It may consist of many interlinked local area networks and also use leased lines in the wide area network. Typically, an intranet includes connections through one or more gateway computers to the outside Internet. The main purpose of an intranet is to share company information and computing resources among employees. An intranet can also be used to facilitate working in groups and for teleconferences.

#### **4.5.3 DMZ**

In computer networks, a DMZ (demilitarized zone) is a computer host or small network inserted as a "neutral zone" between a company's private network and the outside public network. It prevents outside users from getting direct access to a server that has company data. A DMZ is an optional and more secure approach to a firewall and effectively acts as a proxy server as well.

#### **4.5.4 EDI translation package**

In its simplest form, an Electronic Data Interchange (EDI) translator converts EDI transaction sets (EDI messages) to and from flat files into a usable format for an enterprise's applications. The translator can read batches of messages from a VAN mail box and process them. More sophisticated translation packages convert the message to a request to a transaction processing system.

The format of this message must correspond to a standard implemented by the VAN provider. the most popular standards are ANSI X.12 defined by the American National Institute of Standards or ANSI sponsored by the United States government (see [www.ansi.org](http://www.ansi.org)) and EDI for Administration, Commerce and Transport (EDIFACT) sponsored by the United Nation ([www.edifact.org](http://www.edifact.org)).

An EDI transaction involves the transmission of a business document in the form of a transaction set that is prepared in accordance with an ANSI X12 or EDIFACT standard for that document. In other words, a transaction set is the

electronic equivalent of a document, such as a Purchase Order or Request for Quotation, enclosed in an "electronic envelope". Just as you can enclose several paper letters in one envelope, you can send several transaction sets (or "Functional Group") enclosed in one electronic envelope. There are currently almost two hundred transaction sets supporting the business areas of communications and controls, product data, finance, government, materials management, transportation, purchasing, industry standards transition, distribution and warehousing, and insurance.

Each transaction set contains Data Segments and Data Elements. Data Elements are basic information units, such as price, product code, or attribute (size and color, for example). A Data Segment is the electronic equivalent of a line label or line name on a business form.

#### **4.5.5 VAN**

A VAN is a networking service that leases communication lines to subscribers and adds extra services or capabilities, such as security, error detection, guaranteed message delivery, and a message buffer.

Until recently, the VAN was the only way to take advantage of EDI. A VAN provider typically leased lines from local telecommunications providers, often enhancing them with elements, such as error detection. These lines would then be used to connect trading partners. In addition to carrying the cost of the connection, companies would have to install the VAN's proprietary software to translate and transmit their traditional business documents in EDI formats.

#### **4.5.6 VAN access point**

The Value Added Network (VAN) access point is a company's connection to the VAN and represents the networking endpoint of a VAN.

The VAN access point resolves connection and document transmission between enterprise applications and the VAN provider's applications.

#### **4.5.7 VAN mailbox**

Users of a VAN send messages to and retrieve messages from a Mailbox. This service of the VAN holds messages until the receiver requests them.

#### **4.5.8 EDI/MIME translator**

This node provides the same services given by the traditional EDI translator package (see the EDI translation package definition in Section 4.5.4, "EDI

translation package” on page 54), that is, it maps the existing application data format to the EDI Format.

However, in this node, the transport layer is changed by the MIME format replacing the transport layer associated to the VAN.

The Multi-Purpose Internet Mail Extensions (MIME) are an extension of the original Internet e-mail protocol that lets people use the protocol to exchange different kinds of data files on the Internet, such as audio, video, images, application programs, and others as well as the ASCII handled in the original protocol, the Simple Mail Transport Protocol (SMTP).

Using MIME provides a generic mechanism for sending any EDI object explicitly agreed to by the trading partners. The Internet Engineering Task Force or IETF (see the Web site, [www.ietf.org](http://www.ietf.org)) defines a draft with the specification for the “MIME Encapsulation of EDI Objects” or RFC1767 Document.

Typically, EDI transactions include sensitive data; so, transmission often raises concerns about authentication, data integrity, privacy, access control, and non-repudiation. Currently, the IETF is working in a project to define a standard for the “MIME-based Secure EDI” or RFC2026 Document.

#### **4.5.9 SMTP server**

This sever implements the most popular and traditional electronic mail protocol, the Simple Mail Transport Protocol (SMTP). This must include (as the majority of the actual SMTP server) support for MIME message formats to be used for EDI document encapsulation.

#### **4.5.10 Internet gateway**

This node has a task similar to that of the EDI translator node in Variation 1 (see Section 4.5.4, “EDI translation package” on page 54), but, for this node, the information is stored in an SMTP Server from which the Internet Gateway has to retrieve messages and transform them to the specific VAN format. Then, it uses the VAN access point to deliver the messages to the correspond VAN mailbox.

#### **4.5.11 Directory and security services**

This node supplies information on the location, capabilities, and various attributes (including user ID/password pairs and certificates) of resources and users known to this Web application system. The node may supply information for various security services (authentication and authorization)



and may also perform the actual security processing, for example, verifying certificates. The authentication in most current designs validates the access to the Web application server part of the Web server, but it can also authenticate for access to the database server.

#### **4.5.11.1 LDAP**

Lightweight Directory Access Protocol (LDAP) refers to the protocol that is used to communicate from a calling program and a Directory node. Information is kept on the LDAP-based directory node about such topics as people or services or both.

For example, the directory can store information needed to identify registered partners (referred to as authentication). It can also be used to store information about which functions partners are allowed to perform after being identified (this is referred to as authorization).

#### **4.5.11.2 Security**

This node is a logical representation of the functions needed to manage the security of a system. It works in conjunction with the Directory Node. Think of the directory as the repository that holds the following:

- Data about security, such as user IDs and associated passwords or digital certificates (used to authenticate a user).
- Lists of services that a user is authorized to perform (authorization or access control)

Think of the security node as holding the set of components that define the decisions to be made. The node might perform the actual security processing. For example, verify certificates or return a list of the roles an authenticated user is allowed to perform. The authentication in most current designs validates the access to the Web Application Server, but it can also authenticate access to the Database Server.

The security runtime for Web Application Servers typically consists of two core components:

- A security plug-in attached to the Web server that issues “401” challenges for user ID and password back to a Web browser and subsequently makes security decisions when a Web address for a protected resource (HTML file, Servlet) is entered.
- A security collaborator attached to the Web application server that makes security decisions on method calls on resources hosted by the application server.

These runtime components collaborate with the security node to make decisions about authentication, authorization, and delegation.

The components that implement security are distributed throughout the network. It is unlikely that a node in the system does not include some components implementing an aspect of security. The Security Node represents the centralized services that support security on other nodes and to which security decisions are typically delegated.

The treatment of security combines network design for security with a particular emphasis on achieving a secure implementation of Internet and intranet network access. Security is built using these security services:

- Confidentiality provides privacy by protecting sensitive information from unauthorized access.
- Identification and authentication identifies entities, verifying their identities and assuring individual accountability.
- Access control provides mechanisms for granting access to authorized and authenticated users only.
- Data integrity provides detection of the unauthorized modification of data.
- Nonrepudiation assures that you can prove any transaction that takes place (also called accountability).
- Isolation provides protection by isolating a resource and, therefore, restricting potential access to it.
- Audit monitors and reviews security-relevant events.

Together, these services provide end-to-end security by integrating security facilities across heterogeneous environments.

#### **4.5.12 Queue manager**

Messages are sent to and received from queues that are managed by a queue manager. A queue manager provides a persistent message store and additional services including transaction support and routing of messages to the proper queue. The receiver of a message can be an adapter that transforms the message data into parameters to use on method or procedure calls to a non-queue-based application. Similarly, application adapters can convert information returned from a procedure or method call into a message that is then sent back to the originator of the request message.

Three key facts about Messaging and Queuing differentiate it from other communication styles:

- Communicating programs can run at different times.
- There are no constraints on application structure.

- Programs are insulated from network complexities.

MQSeries from IBM is currently the most popular queue manager.

#### **4.5.13 Virtual Private Network (VPN)**

A Virtual Private Network (VPN) is an extension of an enterprise's private intranet across the Internet or other public network. It creates a secure private "tunnel" through the Internet to the other partner. It can be placed behind the domain firewall, although you can also create configurations that access the VPN from within the DMZ.

#### **4.5.14 Protocol domain firewall nodes**

A firewall is a hardware/software system that manages the flow of information between the Internet and an organization's private network. Firewalls can prevent unauthorized Internet users from accessing private networks connected to the Internet, especially intranets, and can block some virus attacks – as long as those viruses are coming from the Internet. A firewall can separate two or more parts of a local network to control data exchange between departments. Components of firewalls include filters or screens, each of which controls transmission of certain classes of traffic. Firewalls provide the first line of defense for protecting private information, but comprehensive security systems combine firewalls with encryption and other complementary services, such as content filtering and intrusion detection. Firewalls control access from a less trusted network to a more trusted network. Traditional implementations of firewall services include:

- Screening routers (the Protocol Firewall)
- Application gateways (The Domain Firewall)

A pair of Firewall Nodes provides increasing levels of protection at the expense of increasing computing resource requirements. The Domain Firewall is typically implemented as a dedicated server Node. The Protocol Firewall is typically implemented as an IP Router.

#### **4.5.15 Public Key Infrastructure (PKI)**

PKI is a collection of standards-based technologies and commercial services supporting the secure interaction of two unrelated entities (for example, a public user and a corporation) over the Internet. In the context of the topologies defined in this redbook, PKI supports the authentication of the server to the browser client, using the SSL protocol.

#### **4.5.16 Domain Name Service (DNS) node**

The domain name server (DNS) node assists in determining the physical network address associated with the symbolic address (Web address) of the requested information. The DNS on the node diagram is that of the Internet service provider (ISP), although DNS is implemented on the accessed site also.

#### **4.5.17 Existing applications and data node**

Existing applications are run and maintained on nodes that are installed in the internal network. These applications provide for business logic that uses data maintained in the internal network. The number and topology of these existing applications and data nodes is dependent on the particular configuration used by these legacy systems.

#### **4.5.18 Message broker**

A message broker is built on a queue manager and routes messages to applications. A message broker can provide real-time, intelligent, rules-based message routing and dynamic message-content transformation and formatting. In this runtime, the message broker allows multiple applications to implement a published service with the broker providing application integration.

A Message Broker acts as a way station, or a hub, for messages passing between MQ applications. Once messages have reached the Message Broker, they can then be processed, depending on the configuration of the Message Broker and the contents of the message. Within the Message Broker, the individual functions are assigned to a collection of interconnected Nodes (message flow) where the processing and transformation activities can take place as required.

Another key component of a Message Broker is the provision of a framework to allow vendors and other partners and customers to write their own processing nodes. Other components include an extended Publish/Subscribe facility, message dictionaries, and message warehousing.

#### **4.5.19 HTTPS adapter for MQ middleware**

When integrating an internal MQ infrastructure implemented using packaged software, such as MQSeries, an HTTP adapter or Internet gateway, such as the *MQSeries Internet Gateway* for integration with MQSeries, will often be available.

#### 4.5.20 MQ server

The MQ server node contains the queue manager that is responsible for implementing the message queuing middleware. It manages the information flow between the adapters (the MQ clients), itself, and other servers.

#### 4.5.21 MQ adapter

The adapter is a piece of software that moves data between a message on a queue and an application or environment. Adapters handle data inbound to and outbound from the application or environment. Adapters are also known as bridges, links, and connectors.

The adapter connects the application to the MQ middleware. This connection needs to be very robust, particularly if we are going to provide updates to the application and need to ensure that they occur. Ideally, the adapter should allow for transactional updates of the application where the request passed by the MQ middleware is consumed as part of the transaction. If we get a failure, the request message is not consumed, and we can retry.

To a great extent, the adapter depends on the application that is being integrated. When integrating with applications implemented in environments, such as CICS or SAP R/3, and using common middleware software, such as MQSeries, adapters will be commonly available. For example, *MQSeries for OS/390* supports integration with a CICS system or the *MQSeries link for R/3* for integration with SAP.

For custom applications not implemented using packaged software, the adapter will require custom development. Products, such as *MQSeries Adapter Offering*, can assist in this process.

#### 4.5.22 Web application server

A Web application server node is an application server that includes an HTTP server (also known as a Web server) and is typically designed for access by HTTP clients and to host both presentation and business logic.

The Web application server node is a functional extension of the informational (publishing-based) Web server. It provides the technology platform and contains the components to support access to both public and user-specific information by users employing Web browser technology. For the latter, the node provides robust services to allow users to communicate with shared applications and databases. In this way, it acts as an interface to business functions, such as banking, lending, and HR systems.

This node would be provided by the company, on company premises, or hosted inside the enterprise network and inside a Demilitarized Zone (DMZ) for security reasons. In most cases, access to this server would be in secure mode, using services, such as SSL or IPSEC.

In the simplest design, this node can provide the management of hypermedia documents and diverse application functions. For more complex applications or those demanding stronger security, it is recommended that the application be deployed on a separate Web application server node inside the internal network.

Data that may be contained on the node includes the following:

- HTML text pages, images, and multimedia content to be downloaded to the client browser
- Java Server Pages
- Application program libraries, for example, Java applets for dynamic downloading to client workstations

#### **4.5.23 Database server node**

This node's function is to provide a persistent data storage and retrieval service in support of the user-to-business transactional interaction. The data stored is relevant to the specific business interaction, for example, bank balance, insurance information, current purchase by user, and so on.

It is important to note that the mode of database access is perhaps the most important factor determining the performance of this Web application in all but the simplest cases. The recommended approach is to collapse the database accesses into a single call or very few calls. This can be achieved via coding and invoking stored procedure calls on the database.

#### **4.5.24 Load balancer node**

The load balancer provides horizontal scalability by dispatching HTTP requests among several, identically-configured Web servers.

#### **4.5.25 Web server redirector node**

In order to separate the Web server from the application server, a so-called *Web server redirector node* (or just *redirector*) is introduced. The Web server redirector is used in conjunction with a Web server. The Web server serves HTTP pages, and the redirector forwards servlet and JSP requests to the application servers. The advantage of using a redirector is that you can move the application server behind the domain firewall into the secure network

where it is more protected than within the DMZ. Static pages can be served from the DMZ by this node.

The redirector can be implemented, for example, by either a reverse proxy server or by a Web server plug-in, such as the servlet redirector function of IBM WebSphere Application Server Advanced Edition.

#### **4.5.26 Application server node**

This node provides the infrastructure for application logic and may be part of a Web application server. It is capable of running both presentation and business logic but, generally, does not serve HTTP requests. When used with a Web server redirector, the application server node will run both presentation and business logic. In other situations, it may be used for business logic only.

#### **4.5.27 Other open standards adapters**

Another possibility is to integrate the legacy applications directly to the Internet without the MQ middleware. When integrating with applications implemented in environments, such as CICS or SAP R/3, Internet adapter software is usually available off-the-shelf. For example the *CICS Transaction Gateway* for integration with a CICS system, or the *SAP Internet Business Framework* for integration with a SAP R/3 system. For custom applications not implemented in such environments, the adapted will usually require custom development as well.

#### **4.5.28 Communication Interface**

The communication interface node represents the technology concepts that make possible communication between the parties systems. To establish communication between the parties, they must agree on the protocol used for it. In this context, is recommended to use a no-property protocol, such as HTTP (or HTTPS), SMTP, or EDIVAN. In some case, we can use proprietary protocols, such as the ones implemented with IBM MQSeries product (message channels).

Specifically, for this runtime topology, we are going to use the HTTP protocol to implement communication between the parties. This protocol is extended worldwide and is supported by a great number of software companies.

There is a specific technology for this protocol, called the Web application server. This is an application server that includes an HTTP server (also known as a Web server) and is typically designed for access by HTTP clients and to host both presentation and business logic.

The Web application server node is a functional extension of the informational (publishing-based) Web server. It provides the technology platform and contains the components to support access to both public and user-specific information by users employing Web browser technology. For the latter, the node provides robust services to allow users to communicate with shared applications and databases. In this way, it acts as an interface to business functions, such as banking, lending, and HR systems.

This node would be provided by the company, on company premises, or hosted inside the enterprise network and inside a demilitarized zone for security reasons. In most cases, access to this server would be in secure mode, using services, such as SSL or IPSEC.

In the simplest design, this node can provide the management of hypermedia documents and diverse application functions. For more complex applications or those demanding stronger security, it is recommended that the application be deployed on a separate Web application server node inside the internal network.

Data that may be contained on the node includes:

- HTML text pages, images, multimedia content to be downloaded to the client browser
- Java Server Pages
- Application program libraries, such as Java applets for dynamic downloading to client workstations.

#### **4.5.29 Adapter node**

This node is the nexus between the legacy application and queue manager node. It must implement a gateway between the specific communication mode of the application and the queue manager systems. The communication mode depends on the specific characteristics of the application.

For more detail about this node, see Section 3.4 “Topology 2: Direct with adapter/bridge” .

---

## **4.6 Summary**

The runtime topologies covered here as well as the application topologies described in the previous chapter are made up of atomic elements and represent the 80 percent of applications that are made up of a common set of elements. Over time, this set will evolve and new patterns may emerge while



others may fall away. The atomic elements however may repeat themselves in the new patterns.

The rest of this book will focus on patterns two and three. Patterns four and five build on patterns two and three and add workflow and long running conversations. These patterns are not discussed further in this book.



---

## Chapter 5. Technology options

This chapter classifies the technologies according to the IBM Application Framework for e-business. For each category, there is a descriptive section followed by a table identifying some representative products and the technologies they support.

In any category, the particular selections and details should be treated as being for illustrative purposes only. Since this is a rapidly-changing area, details get out-of-date very quickly. The reader should consult the related Web-site or product literature from the relevant vendors.

---

### 5.1 Introduction

Now that you have decided which application topology and runtime topology to use to put your business processes on-line, we will discuss the technology options for the IBM Application Framework for e-business platform. An introduction can be found at the following Web site:

[http://www-4.ibm.com/software/ebusiness/arch\\_overview.html](http://www-4.ibm.com/software/ebusiness/arch_overview.html)

The Application Framework for e-business addresses many important issues in the development of e-business solutions. The Framework is prescriptive; it maximizes the use of Internet and other open standards and protocols versus proprietary technologies. However, development in the context of the Framework still leaves a choice of technologies, such as the Internet and open systems, which can dramatically affect the performance, robustness, and usability of a solution. After reading the following sections, you should be able to better understand the many choices of client and server technologies. At the end of each subsection, you will find recommendations regarding specific integration technologies and their use in relationship to the IBM Application Framework for e-business.

---

### 5.2 Classifying technologies

We have chosen to use the IBM Application Framework for e-business as the framework for this book. This section describes the framework categories and the key influencing factors. Figure 23 on page 68 is a building block diagram that represents the elements of a B2BI solution.

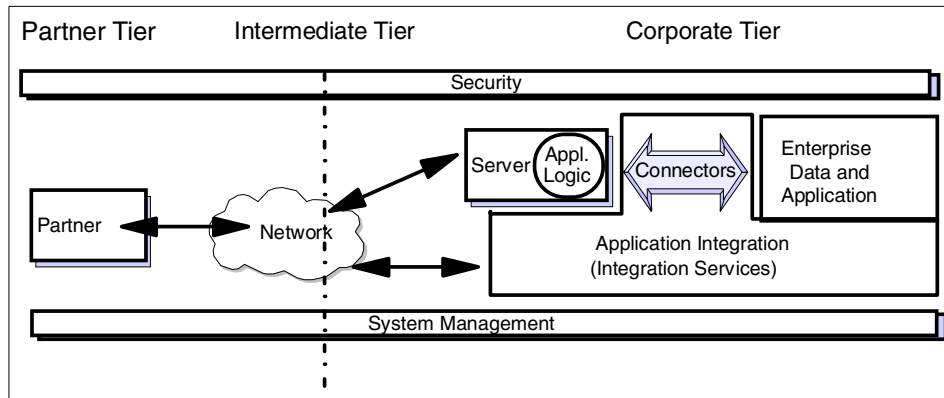


Figure 23. Elements of a B2BI architecture

Admittedly, this is a simple diagram. Real B2BI solutions involve far more detail than is implied here. However, the value of using such a diagram remains. By using the building blocks in each section, we keep your focus on the basic issues of the solution design.

The building block elements map to the elements described in the previous chapters. The mapping is not exact since the internal communications, external communication, process coordination, and systems management functional elements are contained in a number of physical building block elements.

In our diagram above, we describe the application integration layer as possibly containing the connectors between the Web Application Server and the Enterprise Data and Applications. There are two points of view: One is that this layer should provide a tier of communication. That is, all applications should communicate with this tier, which, in turn, communicates with the next tier. A solution that directly connects one application to another without using the Application Integration tier is viewed as a point-to-point solution.

In Figure 23 on page 68, there are two options in the enterprise: One is through the Web server or some other open standard, and the other is a direct connection into the enterprises application integration layer. The second option implies a closer relationship and, possibly, the use of some shared middleware standard.

The diagram is also broadly divided into three parts: The partner tier, the intermediate tier, and the corporate tier.

The partner tier, addresses the technology choices for the partner. The Corporate Tier contains all the legacy connectivity. The Intermediate Tier addresses the B2B and integration technologies as well as the Web application server components of an e-business application.

Usually, one of the three tiers dominates and dictates the technology choices made on the other tiers.

### 5.2.1 Framework categories

The IBM Application Framework for e-business provides the following set of categories within which the technologies can be classified:

**Partner (Clients)** A standard e-business environment is based on the Web browser model to enable universal access to Framework applications and on-demand delivery of application components. The B2Bi model differs only in the sense that there is no user; instead, there are applications that require universal access. The supported standards include HTML, Dynamic HTML, XML, and Java applets.

**Network Infrastructure** provides a platform for the entire e-business environment and includes TCP/IP and network services, security services, directory services, and file and print services. The supported standards include TCP/IP, CDSA, SSL, IPsec, x.509v3 certificates, LDAP, AFS/DFS, and IPP.

**Application Server Software** provides the core function for developing and supporting the e-business application logic. This includes HTTP servers, mail and community services, groupware services, database services, transaction services, and messaging services. The supported standards are SMTP, POP3, IMAP4, IRC, NNTP, FTP, iCalendar, ODBC, DRDA, and CORBA OTS/IIOP.

**Application Integration** Web-enables existing data and applications allowing e-businesses to leverage existing IT investments. Application integration also allows disparate applications (potentially written in different programming languages and built on different architectures) to communicate with each other within an enterprise and across enterprises.

**The Web Application Programming Environment** is based on Java servlets, Enterprise Java services, and Enterprise JavaBean components and provides an environment for writing dynamic, transactional, secure, business applications on Web application servers.

**e-business Application Services** e-business Application Services are higher-level application-oriented components, such as e-commerce services, that conform to the Application Framework for e-business programming model. These services allow e-business solutions to be developed faster with higher quality.

**Systems Management Services** support the end-to-end management across networks, systems, middleware, and applications.

**Development Tools** enable the creation, deployment, and management of e-business applications. They also support integrating third party tools into the development process.

### 5.2.2 Identifying key technology selection influences

In the development of any specific architecture, there will be a number of key factors that have an overall effect on the selection of technologies. These must be identified and the combined impact assessed before the selection process begins.

Here are three relevant examples:

- **Non-functional requirements:** For example, a requirement for platform independence and standards-based infrastructure services.
- **The existing technical environment:** How this is leveraged is particularly important in the context of a migration.
- **Technical architecture goals for the target architecture:** A key decision that must be made early is the choice of the component model, for example, Sun's Enterprise Java Beans, OMG CORBA, Microsoft COM, and so on. In some circumstances, there may be more than one component model involved. In this case, it may also be necessary to select bridging technologies.

Such factors act as global filters on the selection of candidate technologies. This filtering, in turn, flows on to the choices of products that implement these technologies, the application development model, the choice of development tools, and so on.

---

### 5.3 Partner (Client)

At this point in time, the most common protocol for communicating in an e-business environment has been HTTP or HTTPS, in addition to this SMTP and FTP. This was mostly due to the overwhelming number of Web-browser, and mail clients.

If the partners have a shared middleware layer, the partner could support any protocol. The limitations would then be what the network is able to transport. For example, if the partners share middleware (as in the case with MQSeries) the partner could choose to support MQI.

In the B2Bi context, the process of choosing e-business client technologies should involve the consideration of partner application requirements, the supporting network infrastructure, the legacy infrastructure, and so on.

With many existing implementations, EDI VANs will continue to exist. In these cases, the client software will be a requirement.

### **5.3.1 Choosing the partner technologies**

The final partner technology decisions will be based on a set of interacting factors. These factors are summarized in the following list:

- The network selection
  - EDI VAN
  - Internet
  - Other (SNA and so on)
- Partner Application Interface
- Supporting Software Requirements
  - Standard platform software
  - Pre-installed third party
  - Dynamically downloaded components/software
- Application client side persistence requirements

### **5.3.2 XML and the partner**

XML provides a clear separation of presentation and data based on standard definitions. For this reason, it is well suited to both U2B and B2BI. So, XML is the most appropriate format to select for data exchange, at least for now.

For those partners transitioning from legacy application, EDI formats and SMTP may be formats that need to be supported for some time to come. Some EDI vendors have XML transform services available.

---

## 5.4 Web application server

There is a remarkable similarity between emerging models of Web Application servers between the various vendors. These models usually involve five main architectural elements.

- Internet/Web Services
- Application Services
- Integration Services
- Management Services
- Application Development Services

In the IBM Application Framework for e-business, the Web Application Server is generally described as including Internet/Web Services and Application Services. The other elements are shown as separate architectural layers. For consistency with the framework, these other elements are covered in their own sections.

### 5.4.1 Internet/Web services

These services allow any form of client software to make requests for application services over the Internet. This form of client does not require additional middleware, software, and so on other than that normally associated with Internet usage. The main protocol that must be supported is HTTP.

Some application servers provide built-in Web services, that is, they work in a standalone manner. Many also provide Web server connectors that allow integration with existing Web Server environments.

There are pros and cons to this approach. It probably keeps more options open to keep the two as separate servers. This allows for separate tuning and independent evolution, recognizes that there are a large number of existing Web-server products in use, and allows non-Internet clients to be handled directly by the application server.

#### 5.4.1.1 Standard Internet services

Standard Internet services include support for the basic Internet protocols, such as HTTP, FTP, e-mail, SSL.

#### 5.4.1.2 Web Server Connectors

Provision of components that allow seamless integration with common Web Servers, such as Microsoft, Netscape, and Apache. The product may also



come bundled with its own Web server and/or support multiple proprietary Web Server APIs, such as Microsoft's ISAPI or Netscape's NSAPI.

#### **5.4.1.3 Transactive content support**

Transactive content is a term coined by the Forrester Group. It describes a Web application involving transaction processing, interactivity, and rich content linked by a high level of personalization based on profile information kept about each user.

Such an application requires support for features, such as personalization, collaboration, search engines, media streaming, content push, and so on.

#### **5.4.1.4 Additional features**

The additional features include vendor-specific features, such as Content Management, Log File Analysis, and so on.

### **5.4.2 Application services**

Application services can be considered to be a set of cooperating software processes that act as an execution or execution management environment for the server side elements of application systems. The focus is on performance, robustness, scalability, and so on at mainframe-quality service levels.

#### **5.4.2.1 Component model**

Most application servers will be based on component models, such as CORBA, Enterprise JavaBeans, or COM. Servers based on a component model are more valuable than those that merely support such a model.

#### **5.4.2.2 Standards-based infrastructure services**

The application server commonly requires the existence of standards-based infrastructure services to support features, such as transactions and so on. See Section 5.5, "Network-based infrastructure services" on page 76.

#### **5.4.2.3 Scalability and reliability features**

Features, such as multithreaded operation, support for the clustering of servers, pooling of connections and threads, load balancing and fail-over, caching, state management, and so on are vital to scalability, reliability, and performance.

### 5.4.3 Illustrative examples

Table 2 shows examples of Web application server technology.

*Table 2. Web application server technology examples*

Product	Technologies Supported	Notes
IBM - HTTP Server	HTTP, HTTPS, SSL, Servlets, MIMS	This product is included as a base product. This is an instance of a standard Web server. Such a server represents the bare minimum requirements for an e-business server. This product is based on Apache, a collaborative open source project.
Microsoft - Internet Information Server	HTTP, HTTPS, SSL, ASP, ISAPI, MIME	Microsoft IIS and Netscape Enterprise represent typical enterprise-level commercial Web servers. This class of server usually comes bundled with many additional features over and above those provided in a basic Web Server. They support an extensive but vendor-specific server API for dynamic HTML generation.
iPlanet, Netscape - Enterprise Server	HTTP, HTTPS, SSL, LDAP, Servlets, MIME	These servers are usually part of a family of Internet server products. Additional supporting servers, such as Proxy servers and Index servers, are some examples.

Product	Technologies Supported	Notes
IBM - WebSphere	HTTP, HTTPS, SSL, HTML, XML, Servlets, JSP, Corba, Java Devt., EJB, XA Support, Tivoli Support, MQSeries	<p>The four products from IBM, Sun, Inprise, and BEA represent the current state of the art in Web Application Servers. Each of these products is based on one or more of the standards-based Component models (Sun EJB, OMG Corba). These products provide (at least) enterprise application execution support in addition to highly-tuned Web and Internet services. Each product is usually released in three flavors, such as Standard, Advanced, and Enterprise. Most of these products also provide management, application development, and integration services.</p>
iPlanet, Sun - NetDynamics	HTTP, HTTPS, SSL, HTML, Servlets, CORBA, Java Devt, EJB	
Inprise - Application Server	HTTP, HTTPS, SSL, HTML, XML, Servlets, JSP, CORBA, Java Devt, EJB, JTS/OTS	
BEA - WebLogic	HTTP, HTTPS, SSL, HTML, Java 2 Enterprise, Tuxedo	

---

## 5.5 Network-based infrastructure services

Most application servers adopt or support one or more Component Models. For example, Enterprise Java Beans, CORBA, or COM.

Each of the Component models defines, supports, or assumes that a specific set of standard services is available from the network. As part of the Application Framework for e-business, IBM has defined a complete set of network infrastructure capabilities to satisfy these requirements. The network infrastructure provides customers with a secure and scalable distributed networking software platform that enables e-business applications to be accessed on-demand from any client, any network, and any location. It includes the following services, all based on open standards:

- Network services, such as DHCP, which dynamically assigns IP addresses as devices enter and leave the network, and WAP, which delivers information and telephony services to mobile phones and other wireless devices
- Security services based on public key technology that support user identification and authentication, access control, confidentiality, data integrity, and non-repudiation of transactions
- Directory services based on the Lightweight Directory Access Protocol (LDAP) that locate users, services, and resources in the network
- Host integration, for seamless communications with SNA-based networks

The application server makes use of these services to provide an enterprise-class execution environment

As with the Component Model, application services built on the standard services of a component model are more valuable than those that merely support such services.

### 5.5.1 Illustrative examples

Table 3 shows some network-based infrastructure technology examples.

*Table 3. Network-based infrastructure technology examples*

Product	Technologies Supported	Notes
IBM - e-Network Software	LDAP, DMTF CIM Extension Schema, Browser-based Terminal Emulation, Certificates and Public Key Infrastructure, VPN	IBM Networking and Communications e-Network software products provide networking infrastructure that can be used to build corporate intranets and extranets and provide secure access to the Internet. - Host integration - Mobile and wireless solutions - Security services - Personalization and service access administration These services heavily leverage the IBM SecureWay suite of products.
Visigenic - VisiBroker	CORBA Orb, OTS, Naming, Events, Trader, SSL, GateKeeper, COM-CORBA Bridge	Visigenic (now part of Inprise) and IONA are two major Object Request Broker (ORB) software vendors. Both companies provide CORBA 2-compliant ORBS and implementations of many of the standard services defined by OMG. IBM Component Broker is an equivalent product, now considered part of the WebSphere Family of products.
Iona - Orbix	CORBA Orb, OTM, Naming, Events, Notification, Trader, Security, WonderWall, COM-CORBA Bridge	
IBM - Component Broker	CORBA ORB, LifeCycle, Naming, Events, Security, Transactions, Persistence, Connectors	

---

## 5.6 Integration services

For B2Bi, the single most significant component of the architecture is the integration services. It is the component that has the most “responsibility” for the flow of information to and from a partner.

There are two parts to the integration services: First, they have to integrate with legacy systems, and, second, they have to make an interface available for communication with the partner. This interface could be done via the Web/application server or by directly using shared middleware. This component is sometimes called the integration server node.

The purpose of this node is to interface between any front end access channel, such as the Web, a call center, or a client/server (“fat client”) PC, and whatever back-end application system is needed (including applications from other companies). It will perform the following kinds of services:

- Convert protocols from the front end to match whatever the back-end systems understands
- Decompose a single message from the front end, such as a Web server, into several back-end messages (or transactions) and then recompose the replies
- Navigate from the front end to whatever back-end system needs to be accessed
- In more complex cases, control the process or unit of work for a number of back-end interactions based on a request from the front end

The intent is to relieve each front end from having to handle the complexity of interfacing with potentially multiple back-end systems, which may be in different companies. The front end (for example, a Web server) should just need to send a message to the integration server and have it look after the interface.

A second purpose for locating these interface services on the Integration server concerns security. There is a firewall between the Web server and the integration server and the Web server does not need to have any knowledge of all the back-end addresses. Many locations do not want a server located in the DMZ to have access directly to sensitive data and systems. In this case, the Web server can only send messages to the integration server and nowhere else.

### **5.6.1 Database connectivity**

Direct database access is important. There should be support for at least one of ODBC, JDBC and native database drivers.

### **5.6.2 Packaged application API integration**

Packaged applications, such as SAP, JD Edwards, and PeopleSoft, provide a programming API. Some application servers have already built connectors to the most common of these applications, such as Pre-built SAP. PeopleSoft adapter components can be purchased for NetDynamics and Sapphire/Web as part of these products.

### **5.6.3 Middleware integration**

Products, such as Tuxedo, MQSeries, CICS, Notes, Mail, and so on, provide support for connectivity.

### **5.6.4 Component model integration**

Component models, such as Enterprise JavaBeans, CORBA, and COM, provide support for communication with alternative models.

### **5.6.5 Custom integration service development kit**

Most products provide a program development kit/environment that allows the development of custom integration services. Custom integration connectors could be any of the types described previously or some hybrid form.

### **5.6.6 Application integration approaches**

The previous section described the kinds of integration technologies that are commonly provided in a Web Application server. In selecting an appropriate technology, a number of key questions must be addressed:

- At what level is integration of the e-business application with existing external data and applications to take place? This might be dictated by the architecture or technologies of the existing system and/or by the requirements of the e-business application.
- How will the external data and applications be represented within the e-business application programming model?
- How much additional development effort is required to bridge the representation introduced by the integration technology and the representation required by the e-business application?

For example, from a technical perspective, the use of messaging may be the only practical solution to a specific integration requirement. However, if the e-business server developer is expected to be working with Enterprise Java Beans representing business objects, the level of abstraction involved in marshalling and unmarshalling message data is very low. In this case, it is highly likely that some form of abstraction layer will need to be developed in order to hide this low-level activity. This could be a significant development effort in its own right.

The IBM Application Framework for e-business has identified a set of categories that present the alternative approaches. These are presented in order of increasing isolation from the underlying integration technologies and their ability to support higher levels of abstraction.

Figure 24 shows application integration approaches.

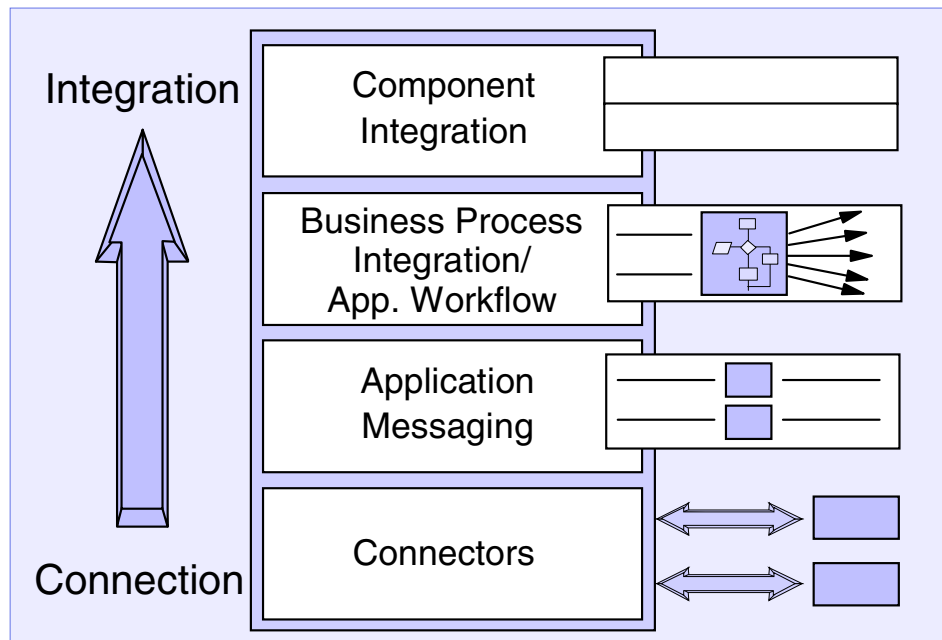


Figure 24. Application integration approaches

- *Connectors* are gateway software elements that provide linkage between the Web application server and services that are reached through the use of application-specific protocols.
- *Application messaging services* provide message-based communication between applications with assured delivery of messages.



- *Business process integration and workflow services* extend the base messaging services with message brokering, intelligent message routing, and message translation.
- *Component integration services* enable object wrapping of existing application logic written in any language. As a result, existing application logic is extended to object-oriented environments.

### 5.6.7 Illustrative examples

Table 4 shows integration technology examples.

Table 4. Integration technology examples

Product	Technologies Supported	Notes
IBM - Web Sphere Enterprise	RDBMS, CICS, Encina, CORBA, IMS	The major Web Application server vendors provide a suite of integration capabilities. IBM - WebSphere calls these Connectors, Sun - NetDynamics calls them Platform Adapter Components (PAC), while BlueStone - Sapphire/Web calls them System Integration Modules (SIM)
Sun - NetDynamics	RDBMS, PeopleSoft, SAP, PAC SDK	
BlueStone - Sapphire/Web	Native, ODBC RDBMS, SAP, Tuxedo, MQSeries	
IBM - MQSeries	JMS, XML	While these two products are completely different, they both aim to present a unifying infrastructure layer that targets integration across multiple kinds of data sources or systems. MQSeries uses a generic messaging model. OleDB provides a series of COM interfaces representing a generic data source.
Microsoft - OleDB	COM, ADO	

## 5.7 Web application programming model

The application programming model describes the conceptual frameworks, classes of objects, services, and features that form the environment within which the developer works during the course of application development.

### **5.7.1 Influence of the component model**

The choice of a Component Model has a major impact on the application programming model.

In order to manage execution, the Application Server may also require that application objects be developed within a framework specific to that server. This server framework is built upon features provided by the component model.

The combination can create a significant initial learning curve for a developer and a dependency on a vendor-specific framework. The trade-offs between learning curve, vendor dependency, and productivity gains have to be considered.

The decisions made here directly influence the choice of development tools.

### **5.7.2 Influence of architectural design patterns**

A specific technical architecture usually establishes patterns of assembly of the elements in the Component Model and any vendor-supplied frameworks. While these decisions are more about architecture design than selecting technologies, they do further shape the application programming model as the developer sees it.

For example, a design might utilize a Model-View-Controller pattern implemented by particular combinations of the Servlet (Controller) and the JSP (View) elements on the Web Server. These elements communicate with business objects implemented using Enterprise Java Beans via an abstraction layer based on the Command design pattern implemented as JavaBeans.

### 5.7.3 Illustrative examples

Table 5 shows some programming model technology examples.

Table 5. Programming model technology examples

Product	Supported technologies	Notes
OMG - Corba	ORB 2.0, IIOP, OTS, Naming, Security, Events	Each of these retransmissions has produced a major technology software platform. These platforms will shape the very core of software development, particularly e-business applications, for many years to come. It is not possible or appropriate to attempt to describe the scope of these platforms in this brief summary table. OMG is a standards-based effort with its roots in the distributed object programming model. Microsoft is a component-based approach with its roots in Windows-based clients. The SUN Java 2 Enterprise is a platform-independent component model explicitly developed for network-based applications.
Microsoft - COM	COM, Automation, ActiveX Controls, OleDocuments, Active Server Pages, OleDB, MTS	
Sun - Java 2 Enterprise	Java 1.2 SDK, Enterprise Java Beans 1.0, JNDI, JMS, JTA, JDBC, RMI/IIOP	

---

## 5.8 e-business application services

These services provide application domain-specific functionality that facilitates the development of e-business applications. Examples of such services range from IBM - San Francisco, where these are implemented as layers within the architectural framework, to standards proposed by industry-specific consortiums, such as Open Applications Group, where the goal is to facilitate integration between organizations.

### 5.8.1 Illustrative examples

Table 6 shows e-business application service technology examples.

Table 6. e-business application service technology examples

Product	Supported technologies	Notes
IBM - San Francisco	Java Frameworks	San Francisco is a heavily layered software product. The upper two layers of this product are aimed at application domains. The Common Business Objects layer introduces extensible objects, such as Company, Business Partner, and Address, which are common across business domains. The Core Business Process Layer provides base implementations of behavior for specific business domains, such as General Ledger processes. The primary goal of these layers is developer productivity through reuse.
Open Applications Group - OAGIS	OAGIS, XML	The Open Applications Group is a non-profit consortium that aims at cost-effective integration of key business application software components for enterprise and supply chain functions for end-user organizations. To this end, the group has developed a model of application integration points for common business processes. It has also defined the protocol and data definitions for participants in such a process. XML is the basis for one of the formats supported. The goal here is to facilitate integration between applications.

---

## 5.9 Systems management

Systems Management services have three main objectives:

- To reduce operations costs
- To increase availability and improve performance and quality of service
- To manage risks resulting from operational failures

The e-business approach has addressed a key problem in client/server solutions: Client application software management. However, it has a significantly-increased dependency on the network. As a consequence, system management for e-business requires an extra focus on the following areas:

- Managing the increased security risk
- Increased demands with respect to availability and scalability
- Ensuring adequate application performance
- Greatly reduced costs. Compared with a typical enterprise client/server application, an e-business application must, potentially, cater to a large Internet audience. Most Internet users expect applications to cost nothing.

### 5.9.1 System management model

The Frameworks systems management model defines a set of management functions to be applied at each layer of the application stack. As shown in Figure 25, the application stack consists of four layers, each of which needs to support a broad set of management functions shown on the right of the figure.

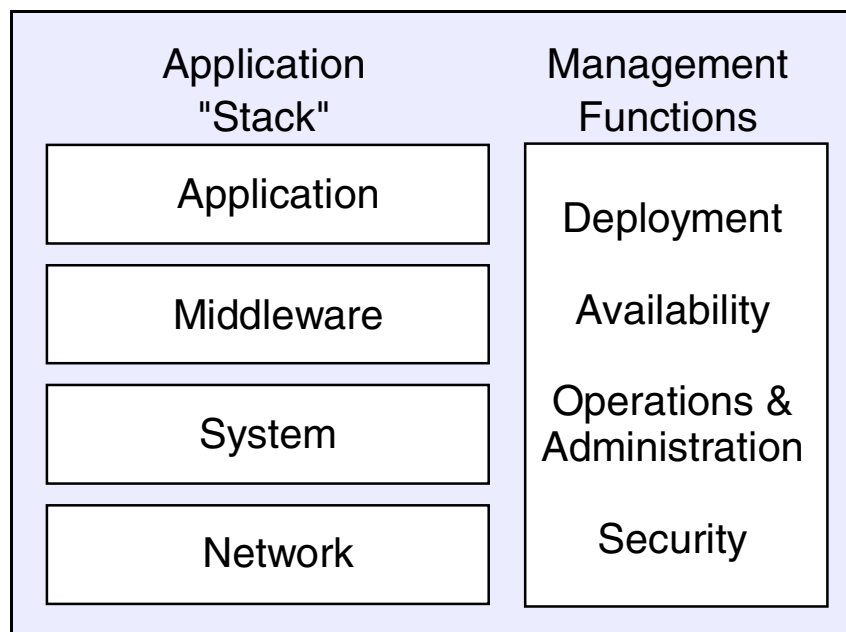


Figure 25. System management model

Although these management requirements existed for client/server systems, e-business applications add a new dimension to many of the problems. As mentioned previously, most of these management functions are due to the fact that e-business puts much more focus on the network and the server. This is a result of the need to ensure application availability anywhere as well as the interdependence between organizations.

For example, when deploying an Internet e-business application, pieces of the application are distributed to multiple locations; components may be deployed not only at the hosting site but also on customers' and suppliers' sites. Due to this distribution of function, availability is hard to manage since the connection to partners and customers is through the Internet, performance problems due to bandwidth bottlenecks or availability problems due to improper server configuration or node failure can bring business to a standstill.

Operations and administration requires a collaborative effort between disparate organizations that may or may not have any formal relationships. The security model changes the most between client/server and the e-business model. It is not only critical to safeguard information assets but also the privacy of all parties.

### **5.9.2 Cross-enterprise systems management**

As described in the preceding paragraphs, there is a definite need to manage resources between organizations. Companies are forced to rely on systems, networks, and people that they do not control and that do not necessarily function together. The application framework extends the integrated enterprise systems management environment to span company boundaries across the Internet. This is termed Cross-Enterprise Systems Management.

### 5.9.3 Illustrative examples

Table 7 shows examples of systems management technology.

Table 7. Systems management technology examples

Product	Supported technologies	Notes
Tivoli Systems-Tivoli	WBEM standards, CIM, DMI, AMS, ARM, SNMP and others	<p>The Distributed Management Task Force (DMTF) is the industry organization that is leading the development, adoption, and unification of management standards and initiatives for desktop, enterprise, and Internet environments.</p> <p>Over the last year, the DMTF has taken on enterprise-focused industry initiatives and standards, such as the Web Based Enterprise Management (WBEM) initiative and the Directory Enabled Networks (DEN) initiative, and pioneered the use of extensible markup language (XML) as the transport encoding for WBEM. The basis for this collaboration and integration is, in large part, due to the DMTF's CIM standard, which facilitates the common understanding of management data across different management systems. CIM is an implementation-neutral schema for describing overall management information.</p> <p>See <a href="http://www.dtmf.org">http://www.dtmf.org</a> for more information.</p> <p>The Computer Measurement Group, commonly called CMG, is a non-profit, worldwide organization of data processing professionals committed to the measurement and management of computer systems. CMG members are primarily concerned with performance evaluations of existing systems to maximize performance (for example, response time, throughput, and so on.) and capacity management, where planned enhancements to existing systems or the design of new systems are evaluated to find the resources necessary to provide adequate performance at a reasonable cost.</p>
Computer Associates - Unicenter	SNMP, DMI, Some DMTF and OMG standards	

---

## 5.10 The development environment

This section identifies development tools and associated software components. There are a number of non-technical factors that influence the selection of these technologies.

For any organization contemplating migration to a new architecture, the following non-technology factors must be considered:

- The current development environment, the available resources, and the associated skill sets
- An understanding of the development roles associated with e-business application development
- The ratio of internal development to out-sourced development.

### 5.10.1 e-business application development team roles

The roles being described in this section take an active part in the actual implementation process. Roles associated with organizational Web strategies, product decision-making, project management, and so on are not included.

It is important to decide whether new skill sets must be introduced into the organization or whether various parts of development will be outsourced. If such an approach is taken, the different tools must be able to conveniently support the lossless two-way interchange of information necessary for iterative development.

For many of these roles, the implied skills and associated tools may not be found in the current development environment.



---

## Chapter 6. B2B integration protocols and standards

This chapter describes at a high level some of the protocols and standards that are relevant to B2B. Associated technologies, such as those that apply to security, are not covered here.

An excellent redpiece, *Exploring Open Software Standards for Enterprise e-business Computing*, REDP0043, describes all the relevant protocols in detail and is available at the following URL:

<http://www.redbooks.ibm.com/redpapers/abstracts/redp0043.html>

The first section provides a context for understanding where a particular protocol or standard fits. The sections that follow provide a little more detail for some of the standards and protocols.

---

### 6.1 Overview

At a very simple level, B2BI describes the passing of messages from one organization to another and for the second organization to respond to those messages. Again, this can be broken down into three distinct layers:

- The mechanism used to send messages from one organization to another
- The content of the message
- The business processes that consume the messages

All protocols don't neatly map to this breakdown. Many protocols impose other standards or extend beyond the domains described above.

#### 6.1.1 Transporting the messages

There are three approaches to the problem of transporting messages from one organization to another:

- Traditional EDI VANs
- Distributed object systems, such as CORBA, COM/DCOM, Java RMI, and so on
- Internet messaging: email, HTTP, and messaging middleware.

Electronic Data Interchange (EDI) have successfully provided electronic document interchange between companies and their suppliers for a number of years. EDI has been widely used in industries, such as finance and manufacturing since the 1970s. In the USA, ANSI defined X.12, a messaging standard for various industries. In Europe, EDIFACT is the standard for EDI. In its long history, EDI has greatly contributed to automating

business-to-business transactions. However, EDI's high cost and inflexible structure has always proven a barrier to adoption by all but the largest enterprises. While EDI will continue to evolve utilizing pervasive networks, such as the Internet, to reduce costs, complementary technologies that are able to provide some of the key capabilities necessary to enable dynamic business process integration are emerging.

The use of distributed systems between organizations is not popular. Distributed systems work better for internal systems rather than systems that span enterprises.

The use of different forms of Internet messaging is probably the most prevalent form of transportation. HTTP is by far the most popular protocol.

Messaging middleware is one of those components that spans the boundary of both the content and the transportation. The MQSeries family spans all three layers providing transport, content, and business process support.

### **6.1.2 Content**

The content of the message defines what the message is. Is it a purchase order, a request for a quote, or a new product description?

cXML and xCBL from Ariba and Commerce One are some of the proposed content describers. cXML and xCBL define a schema for message content that covers fairly simple transactional business documents, such as purchase orders and order status requests. More information about this is provided in the following sections.

### **6.1.3 Business processes**

The newest initiative in the "conversation" between organizations is the integration of the process into the individual business processes of each enterprise. This component is the "business logic" of the interaction. It defines how the messages should be handled, what processes should be started, and so on.

RosettaNet has a component called Partner Interface Processes (PIPS), and this attempts to model the business process within an enterprise.

---

## **6.2 B2B Frameworks**

A generic B2B framework is based on the following:

- A common language that can be employed by existing or potential trading partners to specify how they will interact (for example, an EDI-specific language)
- An electronic contract — a Trading Partner Agreement (TPA) — that employs this common language in order to define and enforce the interaction protocols with which they will do business.

XML B2B frameworks are B2B frameworks that are based on the standard XML technology. In recent months, these facilities have been receiving the most public attention since they offer the most potential for business integration in the inter-company context. In particular, XML B2B frameworks use TPA protocols expressed in XML. These are registered to a specialized software component, usually known as a B2B server, along with the internal business processes for setting up such B2B interactions. Currently, many XML B2B framework specification initiatives have been started and are in various stages of development or still in the planning stage. These initiatives are usually sponsored by independent organizations (the path followed by IBM), such as the Organization for the Advancement of Structured Information Standards (OASIS) Consortium (<http://www.oasis-open.org>) through the XML.org, the XML Industry Portal initiative (<http://www.xml.org>), and jointly with the United Nations body for Trade Facilitation and Electronic Business (UN/CEFACT) through the ebXML

project (<http://www.ebxml.org>). The eCo framework specification produced by the CommerceNet's eCo Working group ([eco.commerce.net](http://eco.commerce.net)). In other cases, XML B2B framework standardization activities are promoted by specific companies, for example, the Microsoft BizTalk (<http://www.biztalk.org>), with the aim to accelerate the rapid adoption of XML in the B2B market. Table 17 provides a list of the main XML B2B framework standardization initiatives.

Table 8. XML B2B framework initiatives

XML B2B framework	Description
XML.org <a href="http://www.xml.org">http://www.xml.org</a>	XML.ORG is an industry Web portal operated by the Organization for the Advancement of Structured Information Standards (OASIS). Funded by a group of companies committed to establishing an open, distributed system for enabling the use of XML in electronic commerce and other industrial applications, XML.ORG is designed to provide a credible source of accurate, timely information about the application of XML in industrial and commercial settings and to serve as a reference repository for XML specifications such as vocabularies, DTDs, schemas, and namespaces.

XML B2B framework	Description
<p>ebXML project</p> <p>Web Address:  <a href="http://www.ebxml.org">http://www.ebxml.org</a></p>	<p>The United Nations body for Trade Facilitation and Electronic Business (UN/CEFACT) and the Organization for the Advancement of Structured Information Standards (OASIS) have joined forces to initiate a worldwide project to standardize XML business specifications. UN/CEFACT and OASIS have established the Electronic Business XML initiative to develop a technical framework that will enable XML to be utilized in a consistent manner for the exchange of all electronic business data. Industry groups currently working on XML specifications have been invited to participate in the 18-month project.</p>
<p>Commerce XML (cXML)</p> <p>Web Address:  <a href="http://www.cxml.org">http://www.cxml.org</a></p>	<p>cXML is an open XML-based standard created to facilitate e-commerce within trading communities. The Commerce XML initiative was started by Ariba Technologies. cXML is a suite of lightweight XML Document Type Definitions (DTDs) and their associated processes that define the exchange of catalog content and transaction information between buyers and suppliers.</p>
<p>Distributed Management Task Force</p> <p>Web Address:  <a href="http://www.dmtf.org">http://www.dmtf.org</a></p>	<p>The DMTF is the industry organization that is leading the development, adoption, and unification of management standards and initiatives for desktop, enterprise, and Internet environments. DMTF has taken on enterprise focused industry initiatives and standards, such as the Web Based Enterprise Management (WBEM) initiative and the Directory Enabled Networks (DEN) initiative, and has pioneered the use of XML as the transport encoding for WBEM.</p>
<p>eCo Specification</p> <p>Web Address:  <a href="http://eco.commerce.net">eco.commerce.net</a></p>	<p>The eCo Specification is an architectural framework that enables businesses to discover each other on the World Wide Web and determine how they can do business. The eCo framework is a product of CommerceNet's eCo Working Group, an industry-neutral group consisting of experts from over 35 companies and organizations throughout the world, and is mainly based on XML.</p>
<p>Open Applications Group</p> <p>Web Address:  <a href="http://www.openapplications.org">http://www.openapplications.org</a></p>	<p>Open Applications Group (OAG) is a non-profit consortium focusing on easier business software interoperability and is the largest publisher of XML content for business software interoperability in the world. Among other things, OAG defined the Integration Specification (OAGIS), which is a model for business software application component interoperability.</p>

XML B2B framework	Description
<p><b>BizTalk</b></p> <p>Web Address:  <a href="http://www.biztalk.org">http://www.biztalk.org</a></p>	<p>Introduced by Microsoft in March 1999, the BizTalk Framework makes it easy for businesses to exchange information between software applications and conduct business over the Internet with trading partners and customers. The BizTalk Framework includes a design framework for implementing an XML schema and a set of XML tags used in messages sent between applications. Microsoft, other software companies, and industry standards bodies will use the BizTalk Framework to produce XML schemas in a consistent manner to enable integration across industries and between business systems, regardless of platform, operating system, or underlying technology.</p>
<p><b>RosettaNet</b></p> <p>Web Address:  <a href="http://www.rosettanet.org">http://www.rosettanet.org</a></p>	<p>RosettaNet is an independent, self-funded, non-profit consortium dedicated to the development and deployment of standard electronic business interfaces to align the processes between supply chain partners on a global basis. Launched in June, 1998, RosettaNet is currently in the pilot phase of its implementation cycle.</p>
<p><b>XML/EDI Group</b></p> <p>Web Address:  <a href="http://www.xmledi.org">http://www.xmledi.org</a></p>	<p>The XML/EDI Group was founded in July, 1997 in response to President Clinton's call for industry support in dealing with Internet-based commerce issues and the emergence in time and space of pivotal technologies that allowed this to be realized through the fusion of XML and EDI.</p> <p>XML/EDI Group is an ad hoc group of professionals and volunteers in various industries dedicated to promoting and guiding the future of XML/EDI standards and products by donating their time and energy.</p>

In parallel with the general XML B2B Framework initiatives, the development of a large set of “vertical” market vocabularies (connected to a specific industry sector or cross-industry) has been started in recent years. For a list of organizations known to be producing industry-specific or cross-industry XML Specifications see, for example, the XML.org XML Catalog:  
[http://www.xml.org/xmlorg\\_registry/index.shtml](http://www.xml.org/xmlorg_registry/index.shtml).

These vertical protocols cover particular industrial business transactions that should be used within B2B frameworks. At times, they have difficulty defining overlapping or conflicting standards, especially when different working groups are active in the same space. Some industry groups are attempting to resolve these differences by promoting protocol-merging initiatives. For example, in the travel industry, the Open Travel Alliance (OTA) decided to incorporate

much of the Hotel Electronic Distribution Network Association specification in the first version of its standards.

## 6.3 More on protocols

The following sections provide more detail on specific protocols. Most of this information was extracted and is available at the Web sites listed in Table 8 on page 91.

### 6.3.1 OBI

Open Buying on the Internet (OBI) is an e-commerce standard that has been specified by the OBI Consortium. OBI is "an open, flexible framework for business-to-business Internet commerce solutions. It is intended for the high volume, low-dollar transactions that account for 80 percent of most organizations' purchasing activity". It is expected to streamline the non-mission-critical procurement processes of organizations (for example, MRO materials) by specifying a standard set of roles that OBI-compliant selling and buying parties must conform to. Furthermore, the standard is supposed to make it easier to achieve compliance by requiring the use of widely-accepted, standards-based technologies, such as HTTP, digital certificates (X509), secure sockets layer (SSL), and EDI.

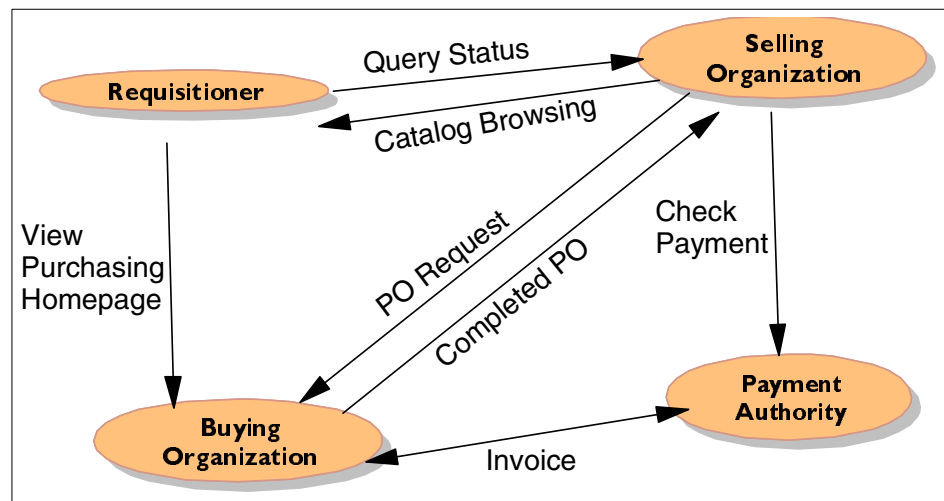


Figure 26. OBI information flow

There are four essential entities involved in an OBI system. The Buying Organization procures items as part of its daily business operations. The

Requisitioner, a member of the buying organization, is interested in procuring certain items as part of the non-mission-critical process of the organization within his/her command. The Selling Organization supplies goods and services to other businesses. Finally, there is the Payment Organization (which may not exist in all OBI scenarios), which acts as a clearing-house for all payment and settlement activities between the selling and buying organizations. All the aforementioned entities are assumed to have digital certificates that uniquely and securely establish their identities. Of course, these entities are all assumed to be connecting to the Internet.

### **6.3.2 Rosettanet**

RosettaNet is a "global business consortium creating the electronic commerce framework to align processes in the IT supply chain. Founded in 1998, RosettaNet is an independent, self-funded, non-profit consortium dedicated to the development and deployment of standard electronic commerce interfaces to align the processes between IT supply chain partners on a global basis.

Rosettanet use the analogy of a human-to-human business exchange to a server-to-server electronic business exchange. In order to communicate in a human-to-human business exchange, humans must be able to produce and hear sound. Further, they must then agree on a common alphabet and use it to create individual words. Grammatical rules are then applied to the words to create a dialog. That dialog forms the business process, which is conducted (or transmitted) through an instrument, such as a telephone.

The fundamental system of exchanging sounds in a human-to-human business exchange can be compared to the Internet, which enables two servers to exchange information during a server-to-server electronic business exchange. HTML/XML functions as the "alphabet" of this electronic exchange, and, presently ECOM applications serve as the instrument by which an electronic business process is transmitted.

In order to scale eBusiness, dictionaries are needed as well as the framework, the Partner Interface Processes (PIPs), and the eBusiness processes. RosettaNet fills this existing gap by focusing on building a master dictionary to define properties for products, partners, and business transactions. This master dictionary, coupled with an established implementation framework (exchange protocols), is used to support the eBusiness dialog known as the Partner Interface Process or PIP. RosettaNet PIPs create new areas of alignment within the overall EC and IT supply-chains eBusiness processes, allowing EC and IT supply-chain

partners to scale eBusiness and fully leverage Ecom applications and the Internet as a business-to-business commerce tool.

### **6.3.3 cXML**

Commerce XML (cXML) is a new proposed standard being developed by "more than 40 leading companies" for business-to-business electronic commerce. Several companies have publicly announced support for cXML including Ariba, Sterling Commerce, Ironside Technologies, SAQQARA Systems, POET, and Extricity Software. According to the Ariba announcement, "cXML is a set of lightweight XML DTDs -- based on the World Wide Web Consortium's XML standard -- with their associated request/response processes.

cXML defines a request/response process for the exchange of transaction information. These business processes include purchase orders, change orders, acknowledgments, status updates, ship notifications, and payment transactions. The contributors to the cXML initiative are focused on achieving reference implementations through the creation and rapid iteration of cXML. The cXML specification, including reference production implementations and associated implementation knowledge, will be submitted to the appropriate standards organizations.

The cXML initiative is, therefore, complementary to existing XML initiatives led by CommerceNet, RosettaNet, Information & Content Exchange (ICE), and Open Buying on the Internet (OBI). The cXML specification will be made publicly available in March 1999. cXML was created in a unique collaboration between buyers, suppliers, and Internet technology companies.

### **6.3.4 Simple Object Access Protocol (SOAP)**

SOAP provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. SOAP does not itself define any application semantics, such as a programming model or implementation-specific semantics; rather, it defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules. This allows SOAP to be used in a large variety of systems ranging from messaging systems to RPC.

SOAP consists of three parts:



- The SOAP envelope construct defines an overall framework for expressing what is in a message, who should deal with it, and whether it is optional or mandatory.
- The SOAP encoding rules define a serialization mechanism that can be used to exchange instances of application-defined datatypes.
- The SOAP RPC representation defines a convention that can be used to represent remote procedure calls and responses.

Although these parts are described together as part of SOAP, they are functionally orthogonal. In particular, the envelope and the encoding rules are defined in different namespaces in order to promote simplicity through modularity.

In addition to the SOAP envelope, the SOAP encoding rules, and the SOAP RPC conventions, this specification defines two protocol bindings that describe how a SOAP message can be carried in HTTP messages either with or without the HTTP Extension Framework.

For more information see the following Web site:

<http://msdn.microsoft.com/xml/general/soapspec.asp>

### **6.3.5 SET**

Secure Electronic Transaction (SET) is an open standard, multiparty protocol for conducting secure bank card payments over the Internet. Interoperability is ensured by design through specific protocols and message formats. SET provides message integrity, authentication of all financial data, and encryption of sensitive data. The IBM Payment Server is designed to support the SET protocol and is enrolled in the SET compliance testing process awaiting formal designation as compliant with the SET specification.

### **6.3.6 Commerce Business Library (CBL)**

Commerce Business Library (CBL) is a set of common semantics, common syntax, and message packaging for information held by and exchanged among Internet commerce transaction partners and market participants. CBL is under development by CommerceNet and VEO Systems Inc.

Commerce One's Common Business Library (xCBL, <http://www.commerceone.com/xml/cbl/>) is a set of common semantics, common syntax, and message packaging for information held by and exchanged among Internet commerce transaction partners and market participants. Business documents include product descriptions, purchase orders, invoices, and shipping schedules.

### **6.3.7 Product Information Exchange (PIX)**

Product Information Exchange (PIX) is a set of protocols that support catalog interoperability on the Internet through defined guidelines for content, communication, format, and presentation of product data. PIX is under development by CommerceNet members.

### **6.3.8 Information and Content Exchange (ICE)**

Information and Content Exchange (ICE) is a protocol to facilitate the controlled exchange and management of electronic assets between network partners and affiliates. The ICE authoring group is led by Vignette Corp. and Firefly Network Inc.

### **6.3.9 Internet Open Trading Protocol (IOTP)**

Internet Open Trading Protocol (IOTP) is an emerging standard for specifying offers for sale, agreements to purchase, payment (by using existing payment protocols, such as SET Secure Electronic Transaction™), the transfer of goods and services, delivery, receipts for purchases, multiple methods of payment, and support for problem resolution. IOTP is focused on interchange between consumers, merchants, and support services. IOTP is being developed by the "Trade" working group of the Internet Engineering Task Force (IETF).

### **6.3.10 Open Financial Exchange (OFX)**

Open Financial Exchange (OFX) is a specification for the electronic exchange of financial data between financial institutions, business, and consumers via the Internet; it is focused on the PC desktop. OFX is developed by CheckFree, Intuit, and Microsoft under the Transpoint brand name.

### **6.3.11 Platform for Privacy Preferences Project (P3P)**

The Platform for Privacy Preferences Project (P3P) provides a framework for informed on-line interactions. The goal of P3P is to enable users to exercise preferences over Web sites' privacy practices. P3P applications will allow users to be informed about Web site practices, delegate decisions to their computer agent when they wish, and tailor relationships with specific sites. P3P is a project under W3C.

### **6.3.12 Open Trading Protocol (OTP)**

The Open Trading Protocol (OTP) at <http://www.otp.org>, is a specification of banking, payment, and technology companies. It specifies offers for sale, agreements to purchase, payment (by using existing payment protocols, such

as SET Secure Electronic Transaction), the transfer of goods and services, delivery, receipts for purchases, multiple methods of payment, and support for problem resolution. OTP is focused on interchange between consumers, merchants, and support services. OTP is being developed by the "Trade" working group of the Internet Engineering Task Force (IETF).

#### **6.3.13 XML/EDI**

XML/EDI (at <http://www.xmledi.com>) is a specification of a group of CommerceNet, ANSI X12, and the Graphics Communication Association. It defines how traditional X12 EDI business data elements can be represented using XML.



## Chapter 7. IBM product guide

This chapter provides a high-level, non-technical overview of most IBM software products.

IBM provides one software platform with many solution levels as shown in Figure 27. The IBM WebSphere software platform for e-business consists of three solution levels designed to deliver, grow, and differentiate a client's e-business. The foundation of the software platform, Application Servers and Integration, delivers back-end data and applications to the Web. Application Accelerators and Customer and Partner Applications allow e-businesses to lead in the marketplace by providing customized e-business solutions and the partners to implement them. With its full range of offerings, the WebSphere software platform for e-business provides a base for addressing e-business and business-to-business (B2B) challenges.

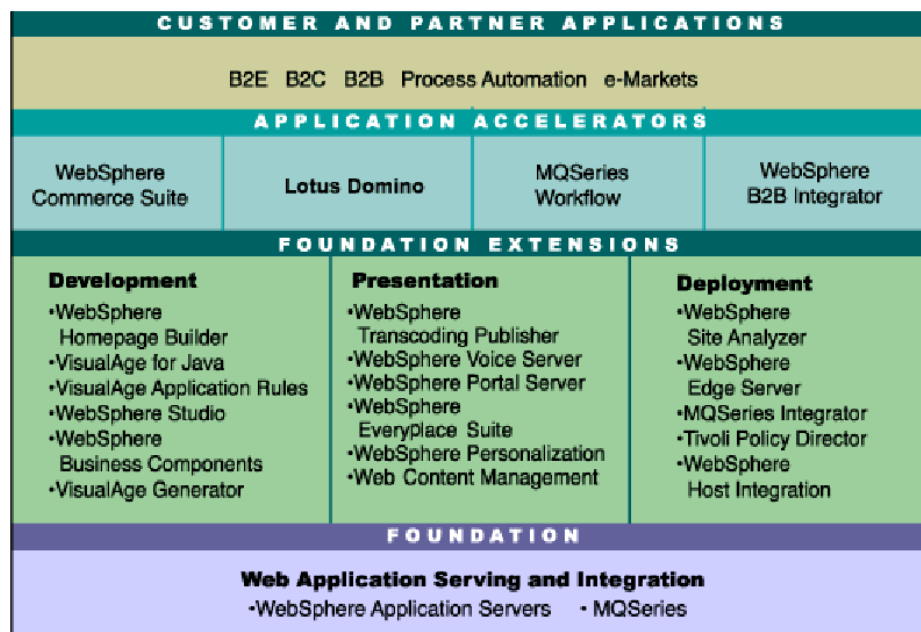


Figure 27. IBM product suite

---

## 7.1 Foundation

The WebSphere software platform for e-business starts with a Foundation formed from the WebSphere Application Servers and MQSeries business integration software.

### 7.1.1 WebSphere Application Server

IBM WebSphere Application Server is an e-business application deployment environment built on open standards-based technology. It is the cornerstone of WebSphere application offerings and services. The Standard Edition lets you use Java servlets, JavaServer Pages, and XML to quickly transform static Web sites into vital sources of dynamic Web content. The Advanced Edition is a high-performance EJB server for implementing EJB components that incorporate business logic. The Enterprise Edition integrates EJB and CORBA components to build high-transaction, high-volume e-business applications.

### 7.1.2 MQSeries

IBM's innovative, award-winning MQSeries is the market leader in commercial messaging, provides a key element of enterprise systems, and enjoys rich support from business partners. It also provides flexible, rapid application integration offering unparalleled business flexibility.

MQSeries is integral to the patterns being discussed in this redbook. Detailed information is provided in Chapter 8, "MQSeries and MQSeries integrator" on page 109.

---

## 7.2 Foundation extensions

In addition to the Foundation, WebSphere provides Foundation Extensions, which are the integrated services and tools that companies need to develop complete e-business solutions. This solutions level not only helps leverage and extend existing IT assets and systems today, it positions businesses to adapt to future growth. Foundation Extensions facilitate rapid, component-based development of next generation e-business applications.

By using powerful, innovative IBM Web technologies based on open industry standards, new opportunities are created with customers, suppliers, and business partners. Development Extensions, Presentation Extensions, and Deployment Extensions are the modular building blocks that extend from the Foundation with Web publishing allowing growth all the way to enterprise-scale transaction processing.

### **7.2.1 VisualAge for Java**

IBM VisualAge for Java makes it easier than ever to create scalable, hard-working e-business applications. Advanced, easy-to-use functions slash development time, enable a high-level of code reuse, and allow for integrated team development. The industry's favorite visual Java IDE, VisualAge for Java, provides the industry's most advanced support for building, testing, and deploying 100 percent Pure Java applications, JavaBeans components, servlets, and applets - and scaling from Windows NT to OS/390.

### **7.2.2 VisualAge Application Rules**

IBM VisualAge Application Rules gives developers (even those with little or no Java programming experience) the ability to quickly build and deploy business rules-based applications for e-business. Coupled with IBM tools and middleware, VisualAge Application Rules provides a complete, flexible solution for building Web-based transactional applications, something especially important to the exploding B2B market.

### **7.2.3 VisualAge Generator**

The IBM VisualAge Generator combines the power of VisualAge for Java and WebSphere Rapid Application Development technology to deliver bulletproof e-business applications. Tailored to IT or line-of-business departments that need robust, fault-tolerant, scalable applications, VisualAge Generator gives non-object-oriented skilled programmers a higher level of abstraction for writing new e-business applications quickly and efficiently. Specifically designed to deliver high-volume transaction processing in multi-tier, multi-platform client/server environments, VisualAge Generator masks the complexity of data and communications connections and is optimized for DB2, CICS, IMS, and WebSphere servers.

### **7.2.4 WebSphere Studio**

The IBM WebSphere Studio provides a comprehensive set of tools that reduces the time and effort required to build dynamic and exciting Web applications. With the industry's first visual layout tool for dynamic Web pages using Java Server Pages technology (JSPs), full HTML, JavaScript, and DHTML layout capability, WebSphere Studio allows Web site producers and Web application developers who need dynamic Web site content and logic to quickly and easily develop and maintain e-business applications. Tight integration between WebSphere Studio, VisualAge for Java, and WebSphereApplication Servers makes it easy for development teams to communicate and work together to develop Web-based e-business applications.

### **7.2.5 WebSphere Homepage Builder**

Whether you are a beginner or an expert, IBM WebSphere Home Page Builder gives you the capability to build lively pages with state-of-the-art Web technology. Home Page Builder brings together everything you need to build pages and publish your site in one package.

### **7.2.6 WebSphere Business Components**

IBM WebSphere Business Components provide an easy-to-use coherent package of software implementation that can be developed and delivered and composed from other components and has explicit and well-specified interfaces for the services it provides and expects from others. IBM has developed WebSphere Business Components to help developers reduce their development time, run across diverse infrastructures, and deliver value to your company faster with a complete project solution.

### **7.2.7 WebSphere Transcoding Publisher**

Extend the reach of your data with the IBM Websphere Transcoding Publisher, an easy-to-use, flexible solution for bridging existing data and applications to new environments. Transcoding Publisher is server-side software that adapts, reformats, and filters data so that it is optimally formatted for the destination environment, whether it is a business-to-business transaction or an emerging pervasive computing device. Transcoding Publisher can be extended to deliver custom transforms in response to new breeds of devices, emerging XML variants, and e-business trends.

### **7.2.8 WebSphere Voice Server**

The IBM WebSphere Voice Server with ViaVoice Technology, DirectTalk, CallPath, Message Center, and ViaVoice Embedded Technology are new members of the WebSphere family that extend the power of the human voice to applications across Web and call-center environments. Built on open architecture, such as VoiceXML and Java, WebSphere voice solutions enable the mobile Internet.

### **7.2.9 WebSphere Portal Server**

The IBM WebSphere Portal enables companies to build their own custom portal Web site that serves the needs of employees, business partners, and customers. Users can sign-on to the portal and receive a personalized Web page providing access to the information and Web applications they need.



### **7.2.10 WebSphere Everyplace Suite**

The IBM WebSphere Everyplace Suite is an integrated software package that combines, in one box, all the necessary "technology ingredients" that businesses, software developers, and Web integrators will need to create new applications and easily connect Web and enterprise data to a wide range of non-PC devices including wireless handsets, PDAs, and other Internet appliances.

### **7.2.11 WebSphere Personalization**

IBM WebSphere Personalization provides users of WebSphere Application Server and WebSphere Studio with the capabilities to build a Web site, intranet, or extranet that delivers Web pages customized to the interests and needs of each site visitor.

### **7.2.12 MQSeries Integrator**

IBM MQSeries Integrator is a powerful message-brokering software that automatically distributes information to those who need it. Using MQSeries for transporting messages across different computing platforms, it routes information according to enterprise-defined rules, transforming and reformatting it to suit the receiving application. MQSeries Integrator seamlessly integrates applications, databases, and networks.

MQSeries Integrator is integral to the patterns being discussed in this redbook. Detailed information is provided in Chapter 8, "MQSeries and MQSeries integrator" on page 109.

### **7.2.13 WebSphere Edge Server**

IBM WebSphere Edge Server provides an integrated solution for local and wide-area load balancing, content-based quality of service routing, and Web content filtering and caching for multi-vendor Web server environments.

### **7.2.14 WebSphere Site Analyzer**

IBM WebSphere Site Analyzer provides analysis for enterprise Web site visitor trends, usage and content, and WebSphere Commerce Suite reporting. WebSphere Site Analyzer helps you make factual e-business decisions based on enterprise Web site visitor behavior. Use WebSphere Site Analyzer to describe visitor trends and preferences, manage Web site content and structure, and improve the overall effectiveness of Web initiatives and campaigns. WebSphere Site Analyzer is the only Web analytic software available that is designed to be tightly-integrated across the entire WebSphere platform.

### **7.2.15 WebSphere Host Integration Solution**

With the WebSphere Host Integration Solution, you can start simple and grow your e-business fast, leveraging your legacy data with new e-business applications to maximize your total return on investment. The Host Integration development components coupled with the Websphere Studio and WebSphere Application Server in a single offering simplifies creation and deployment of e-business solutions. The WebSphere Host Integration Solution is the most comprehensive and flexible Web-to-host offering in the industry for a single price per user and integrates with WebSphere Transcoding Publisher for delivery of host data to any pervasive device.

### **7.2.16 Tivoli Policy Director**

Tivoli Policy Director is a robust and secure policy management tool for e-business and distributed applications. It uniquely addresses the challenges of e-business security-escalating costs, growing complexity, and the inability to implement security policies across platforms. Policy Director is designed to unite core security technologies around common security policies. This helps reduce implementation time and management complexity, thereby, lowering the total cost of secure computing.

---

## **7.3 Application Accelerators**

WebSphere Application Accelerators are modular and extensible business services that enable partners to quickly implement tailored solutions in response to market demands and customer needs. This solutions level positions a business to take advantage of emerging technology for competitive advantage in your industry with solution offerings for e-commerce, collaboration, and business-to-business (B2B) integration.

To become a better competitor on the Web, companies must redesign their business model and reduce time and cost related to transforming business processes. New e-business technology must be adopted. At the same time, existing applications and data must be preserved, integrated, and extended to the Web. This becomes complicated without the right tools. WebSphere Application Accelerators simplify this process and enable businesses to achieve specific business goals by offering readily-customized, off-the-shelf software and service provider solutions.

### **7.3.1 WebSphere Commerce Suite**

IBM WebSphere Commerce Suite, a front-end integrated e-commerce solution, helps you capture the e-commerce opportunity by providing the tools

you need to quickly establish and grow your e-commerce site as your needs dictate. WebSphere Commerce Suite can help you create exciting, personalized Internet experiences for your customers' experiences that drive top line revenues while your bottom line remains protected.

### **7.3.2 Lotus Domino**

The Domino Server Family is an integrated messaging and Web application software platform for growing companies that need to improve customer responsiveness and streamline business processes. Domino helps you leverage your existing investments in people, skills, tools, and back-end systems. Now, you don't have to worry about tying together multiple software products for messaging, security, systems management and data distribution, and replication; Domino integrates it all.

### **7.3.3 MQSeries WorkFlow**

IBM MQSeries Workflow aligns and integrates your organization's resources and capabilities with our business strategies, accelerating process flow, cutting costs, eliminating errors, and improving workgroup productivity. With the ability to capture and use knowledge about your business processes, MQSeries Workflow helps organizations define, document, test, control, execute, improve, and integrate their business processes. The ease of change enables your organization to react quickly to new market requirements.

### **7.3.4 WebSphere Business-to-Business Integrator**

Transform your infrastructure to embrace emerging technologies with IBM WebSphere Business-to-Business (BtoB) Integrator software. WebSphere BtoB Integrator enables you to seamlessly span-the-gap between your own enterprise computing systems and those of your customers, suppliers, business partners, and e-marketplaces simultaneously. This software supports every facet of your core businesses, combines technologies from the extensive IBM catalog of integration and transaction software, is based on open XML technology, and is built to run on the award-winning WebSphere Application Server platform.

### **7.3.5 Universal Description, Discovery, and Integration (UDDI)**

Web services are Internet-based modular applications that perform a specific business task and conform to a specific technical format. IBM develops and implements leading e-business services for B2B, B2C, and eMarketplace solutions so that you can use this technology to deliver real business results.

---

## 7.4 Customer and partner applications

All levels of the WebSphere software platform for e-business extend into the Customer and Partner Applications solution level to create further opportunities for differentiation. More than 200 applications and services are available for the Foundation alone. In addition, IBM has over 20,000 business partners to help implement e-business strategy with applications built on the WebSphere software platform for e-business.

IBM business partners include:

- **ISVs:** Integral Systems, Lucent, and i2
- **Web integrators:** Agency.com, Concrete Media, Razorfish, and USWeb/CKS
- **System integrators:** Andersen Consulting, Deloitte Touche Tohmatsu, Ernst & Young, SPL World Group, Perficient, Inc., and IBM Global Services

## Chapter 8. MQSeries and MQSeries integrator

The IBM family of business integration solutions are based around MQSeries. Having the strengths of a business integration solution, the product suite naturally lends itself to B2BI. Some of the benefits that it brings are:

- MQSeries runs on more than 30 different platforms so that virtually any platforms found in the commercial and industrial worlds can be connected.
- MQSeries is the de facto messaging standard, with more than 60 percent of the messaging market and more than 5000 customer sites.
- MQSeries is developed by IBM, which can draw on more experience of enterprise-strength transactional computing than anyone else in the business.
- MQSeries enjoys global service support and broad support from many technology and service partners.

Figure 28 shows topology 2. Topology 3 builds on topology 2 in that it adds a message broker function where the queue manager is depicted. Both topologies, 2 and 3, can gain significantly from IBM MQSeries, although it is possible to populate the runtime topologies with alternative messaging products or other middleware choices.

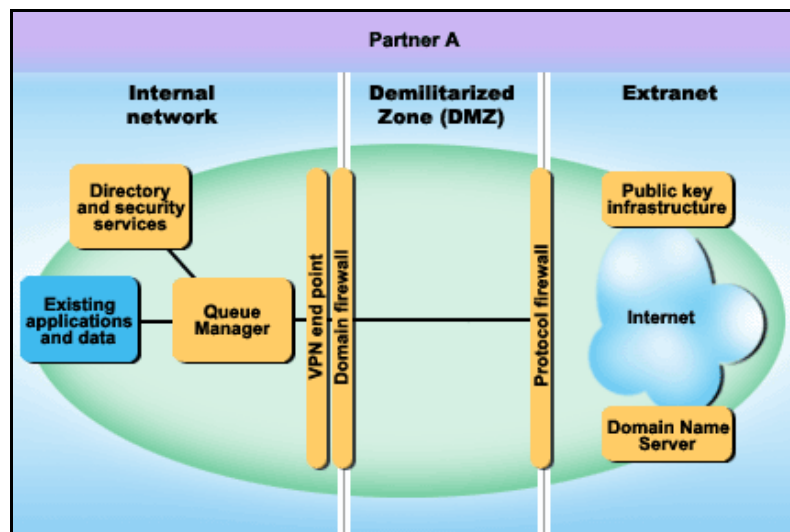


Figure 28. Runtime topology 2

This chapter describes both the messaging product and the message brokering product that would fit into topologies 2 and 3. The workflow product is not discussed in detail here since it does not fit into the latter topologies.

---

## 8.1 Business integration and the MQSeries Family

At the core of the MQSeries family is MQSeries itself. It provides assured once-only delivery of messages across more than 30 industry platforms using a variety of communications protocols.

MQSeries provides support for applications with a number of application programming interfaces (MQI, AMI, JMS) in several programming languages and for a number of communication models (including point-to-point and publish/subscribe). MQSeries also provides a number of connectors and gateways to other products, such as Lotus Domino, Microsoft Exchange, SAP/R3, CICS, and IMS.

MQSeries Integrator extends the messaging capabilities of MQSeries by adding message broker functionality driven by business rules. It provides the intelligence to route and transform messages, the possibility to filter messages (topic-based or content-based) and database capabilities for enrichment of the messages or for warehousing the messages. Also, it provides a framework for extending the functionality with plug-ins to user-written or third-party solutions for specific requirements.

MQSeries Workflow is the third component of the IBM family of products for business integration. It is also based on the messaging technology offered by base MQSeries, but it adds a further dimension to the integration of applications. It's not only about applications, but about integrating all resources in a business process. It ensures that the right information gets to the right target, either a person or application, at the right moment in the process flow.

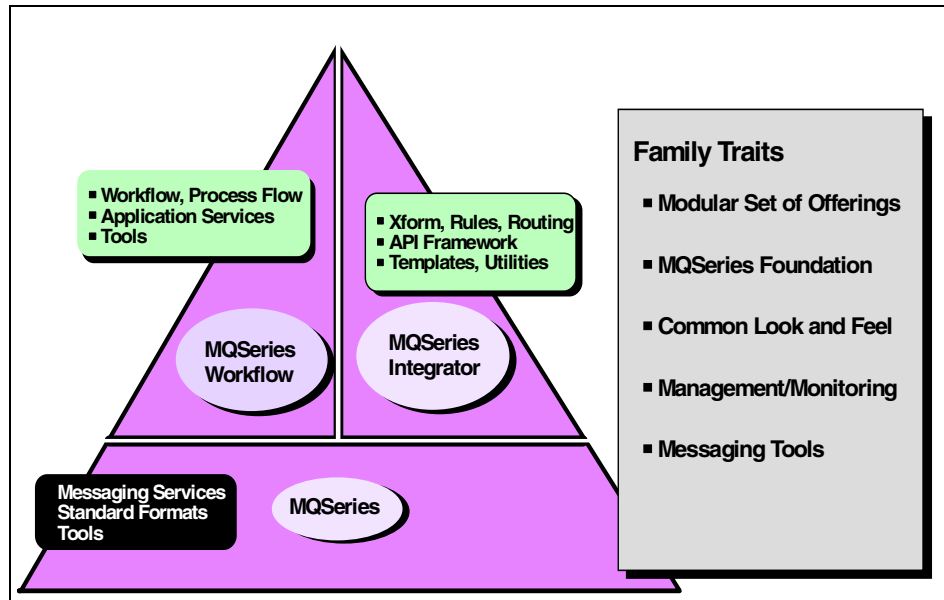


Figure 29. The MQSeries family for business integration

## 8.2 MQSeries primer

MQSeries is the award winning IBM middleware for commercial messaging and queuing. It is used by thousands of customers in every major industry in many countries around the world. MQSeries speeds implementation of distributed applications by simplifying application development and testing.

MQSeries runs on a variety of platforms. The MQSeries products enable programs to communicate with each other across a network of unlike components, such as processors, subsystems, operating systems, and communication protocols. MQSeries programs use a consistent application program interface (API) across all platforms. Figure 30 on page 112 shows the main parts of an MQSeries application at run time.

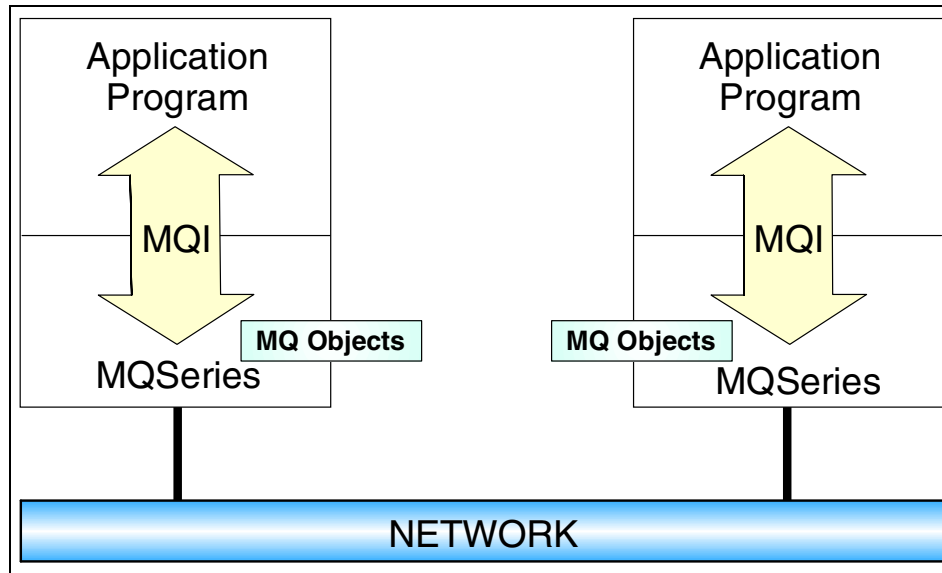


Figure 30. MQSeries at run time

Programs use MQSeries API calls, that is the Message Queue Interface (MQI), to communicate with a queue manager (MQM), the run-time program of MQSeries. For the queue manager to do its work, it refers to objects, such as queues and channels. The queue manager itself is an object as well.

The following section provides a brief overview of MQSeries including clients and servers.

### 8.3 What is Messaging and Queuing?

Message queuing is a method of program-to-program communication. Programs within an application communicate by writing and retrieving application-specific data (messages) to and from queues without having a private, dedicated, logical connection to link them.

*Messaging* means that programs communicate with each other by sending data in messages and not by calling each other directly.

*Queuing* means that programs communicate through queues. Programs communicating through queues need not be executed concurrently.

With *asynchronous messaging*, the sending program proceeds with its own processing without waiting for a reply to its message. In contrast,



*synchronous messaging* waits for the reply before it resumes processing. For the user, the underlying protocol is transparent. The user is concerned with conversational or data-entry type applications.

MQSeries is used in a client/server or distributed environment. Programs belonging to an application can run in one workstation or in different machines on different platforms. Applications can easily be moved from one system or platform to another. The programs can be written in various programming languages, including Java. The same queuing mechanism is valid for all platforms, and so are the currently 13 APIs.

Since MQSeries communicates via queues, it can be referred to as using indirect program-to-program communication. The programmer cannot specify the name of the target application to which a message is sent. However, he or she can specify a target queue name, and each queue is associated with a program. An application can have one or more “input” queues and may have several “output” queues containing information for other servers to be processed or for responses for the client that initiated the transaction.

The programmer does not have to worry about the target program being busy or not available. He or she isn’t even concerned about the server being down or having no connection to it. The programmer sends messages to a queue that is associated with an application, and the application may or may not be available at the time of the request. MQSeries takes care of the transport to the target application and even starts it, if necessary.

If the target program is not available, the messages stay in a queue and are processed later. The queue is either in the sending machine or in the target machine, depending on whether the connection between the two systems can be established or not. Applications can be running all day long or they can be triggered, that is, automatically started when a message arrives or after a specified number of messages have arrived. Figure 31 on page 114 shows how two programs, A and B, communicate with each other.

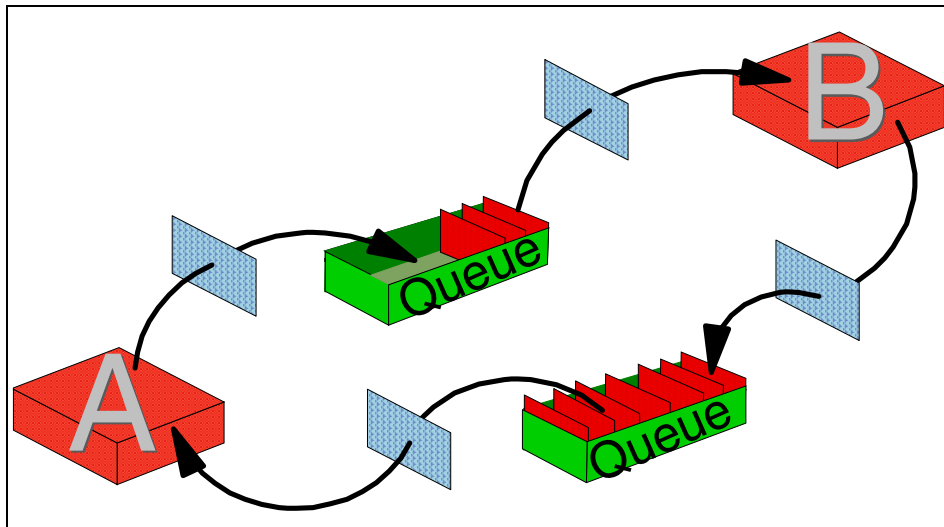


Figure 31. Messages and Queues

We see two queues: One is the “output” queue for A and, at the same time, the “input” queue for B while the second queue is used for replies flowing from B to A.

The squares between the queues and the programs represent the Message Queuing Interface (API) the program uses to communicate with MQSeries’ run-time program, the queue manager. As we said before, the API is a simple multiplatform API consisting of 13 calls. The API will be discussed later.

---

## 8.4 About messages

A message consists of two parts:

- Data that is sent from one program to another
- The message descriptor or a message header

The message descriptor identifies the message (message ID) and contains control information, also called attributes, such as message type, expiration time, correlation ID, priority, and the name of the queue for the reply.

A message can be up to 4 MB or 100 MB long, depending on the MQSeries version you use. MQSeries Version 5 (for distributed platforms) supports a maximum message length of 100 MB.

### 8.4.1 Message segmenting and grouping

In MQSeries Version 5, messages can be *segmented* or *grouped*. Message segmenting can be transparent to the application programmer. If permitted, the queue manager segments a large message when it does not fit in a queue. On the receiving end, the application has the option to either receive the entire message in one piece or each segment separately. This may depend on the buffer size available for the application.

A second method of segmenting leaves the programmer in control so that he or she can split a message according to logical boundaries or buffer size available for the program. The programmer puts each segment as a separate physical message; thus, several physical messages build one logical message. The queue manager ensures that the order of the segments is maintained.

To reduce traffic over the network, you can also group several small messages together and build one larger physical message. This message is then sent to the destination and is disassembled there. Message grouping also guarantees that the order in which the messages are sent is preserved.

### 8.4.2 Distribution lists

Using MQSeries Version 5, you can send a message to more than one destination queue with one MQPUT call. This is done with a dynamic *distribution list*. A distribution list can be a file that is read at the time an application starts. It can be modified any time. It contains a list of queue names and the queue managers that own them. A message sent to multiple queues belonging to the same queue manager is sent over the network only once and, thus, reduces network traffic. The receiving queue manager replicates the messages and puts them into the destination queues. This function is called *late fan-out*.

### 8.4.3 Message types

MQSeries knows four types of messages:

<b>Datagram</b>	A message containing information for which no response is expected
<b>Request</b>	A message for which a reply is requested
<b>Reply</b>	A reply to a request message
<b>Report</b>	A message that describes an event such as the occurrence of an error or a confirmation on arrival or delivery

#### 8.4.4 Persistent and non-persistent messages

Application design determines whether a message must reach its destination under any circumstances or if it can be discarded when it cannot get there in time. MQSeries differentiates between *persistent* and *non-persistent* messages. *Delivery of persistent messages is assured*; they are written to logs to survive system failures. In an AS/400 these logs are Journal Receivers. Non-persistent messages cannot be recovered after a system restart.

#### 8.4.5 The message descriptor

The table below contains some interesting attributes of the message descriptor. We mention them here because they explain some of the functions the queue manager provides for you.

Table 9. Attributes of the message descriptor

Version	Return address
Message ID/Correlation ID	Format
Persistent/non-persistent	Sender application and type
Priority	Report options/Feedback (COA, COD)
Date and time	Backout counter
Lifetime of a message	Segmenting/grouping information

The following are some attributes of the message descriptor:

- The *version* of the message descriptor depends on the MQSeries version and platform you use. For the functions introduced with Version 5, additional fields were needed to keep information about segments and their order and distribution list information, to name a few. This enlarged structure carries the version number, 2. Other queue managers who do not support these functions ("Version 1 queue managers") treat the additional information as data.
- *Message* and/or *correlation ID* are used to identify a specific request or reply message. The programmer can move a value in one or both fields or have MQSeries create a unique ID for him or her. Before the programmer puts the request message in the queue, he or she can save the ID(s) and use them in a subsequent get operation for the reply message. The program that receives the request message copies this information into the reply message. This allows the originating program (the one that gets the reply) to instruct MQSeries to look for a specific message in the queue instead of getting the first one in the queue.

- We discussed *persistent* and *non-persistent* messages earlier. Persistent messages always arrive at their destination, even when the system fails. They are “hardened”, that is, saved on disk. You can make a specific message persistent or all messages on a particular queue.
- You can assign a *priority* to a message and, thus, control the order in which it is processed.
- The queue manager stores *time* and *date* when the MQPUT occurred in the message header. The time is in GMT, and the year has four digits and, thus, is Y2K compliant.
- You can also specify an *expiration date*. When this date is reached and an MQGET is issued, the message will be discarded. There is no “daemon” that checks queues for expired messages. Expired messages can stay in a queue for weeks until a program attempts to read it.
- The return address is very important for request/reply messages. You have to tell the server program where to send the reply message. Clients and servers have a one-to-many relationship, and, usually, the server program cannot find out from the user data where the request message came from. Therefore, the client provides the *reply-to queue* and *reply-to queue manager* in the message header. The server uses this information when it performs the MQPUT API call.
- In the *format* field, the sender can specify a value that the receiver can use to decide whether data conversion can be done. It is also used to indicate that there is an additional header (extension) present.
- The message also carries information about the sending application (program name and path) and the platform it is running on.
- *Report options* and *feedback* code are used to request information, such as confirmation on arrival or delivery, from the receiving queue manager. For example, the queue manager can send a report message to the sending application when it puts the message in the target queue or when the application gets it off the queue.
- Each time a message is backed out, the *backout counter* is increased. An application can check this counter and act on it, for example, send the message to a different queue where the reason for the backout is analyzed by an administrator.
- Message segmenting and grouping has been mentioned earlier. The queue manager uses the message header to store information about the physical message (for example, if it is a message group, the first or last segment, or which one in between).

## 8.5 About the Queue Manager

The heart of MQSeries is the message queue manager (MQM), MQSeries' run-time program. Its job is to manage queues and messages for applications. It provides the Message Queuing Interface (MQI) for communication with applications. Application programs invoke functions of the queue manager by issuing API calls. For example, the MQPUT API call puts a message on a queue to be read by another program using the MQGET API call. This scenario is shown in Figure 32.

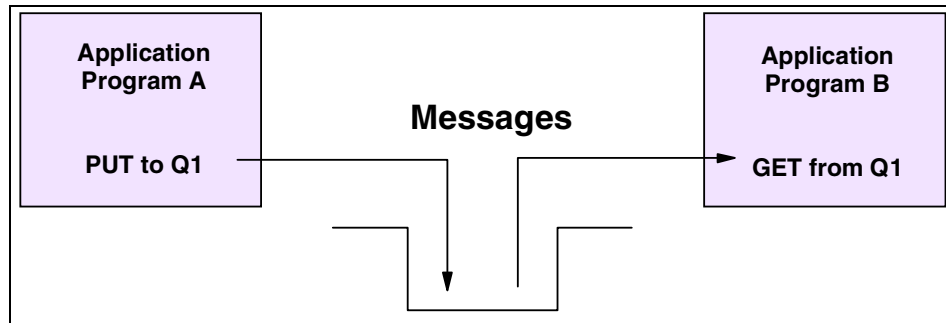


Figure 32. Program-to-program communication - One system

A program may send messages to another program that runs in the same machine as the queue manager (shown above) or to a program that runs in a remote system, such as a server or a host. The remote system has its own queue manager with its own queues. This scenario is shown in Figure 33.

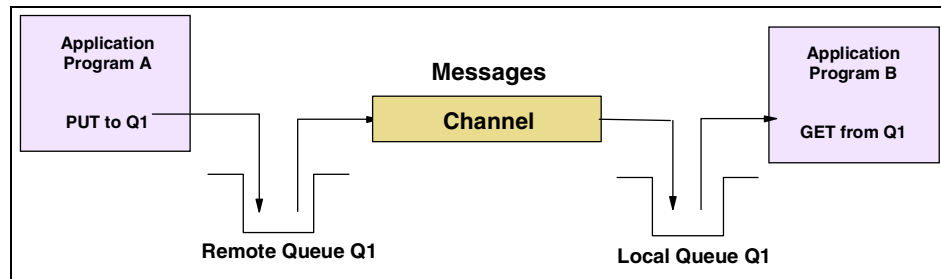


Figure 33. Program-to-program communication - Two systems

The queue manager transfers messages to other queue managers via *channels* using existing network facilities, such as TCP/IP, SNA, or SPX. Multiple queue managers can reside in the same machine. They also need channels to communicate.

Application programmers do not need to know where the program to which they are sending messages runs. They put their messages in a queue and let the queue manager worry about the destination machine and how to get the messages there. MQSeries knows what to do when the remote system is not available or the target program is not running or busy.

For the queue manager to do its work, it refers to objects that are defined by an administrator, usually, when the queue manager is created or when a new application is added. The objects are described in Section 8.7, “About Queue Manager objects” on page 122. The functions of a queue manager can be summarized as follows:

- It manages queues of messages for application programs.
- It provides an application programming interface, the Message Queue Interface (MQI).
- It uses existing networking facilities to transfer messages to other queue managers when necessary.
- It coordinates updates to databases and queues using a two-phase commit. Gets and puts from/to queues are committed together with SQL updates or backed out if necessary.
- It segments messages (if necessary) and assembles them. It can also group messages and send them as one physical message to their destination where they are automatically disassembled.
- It can send one message to more than one destination with one API call using a user-defined dynamic distribution list, thus, reducing network traffic.
- It provides additional functions that allow administrators to create and delete queues, alter the properties of existing queues, and control the operation of the queue manager. MQSeries for Windows NT Version 5.1 provides graphical user interfaces; other platforms use the command line interface or panels.

MQSeries clients do not have a queue manager in their machines. Client machines connect to a queue manager in a server. The queue manager manages the queues for all clients attached to it.

In contrast to MQSeries clients, each workstation that runs MQSeries for Windows (Version 2) has its own queue manager and queues. MQSeries for Windows is a single-user queue manager and is not intended to function as a queue manager for other MQSeries clients. This product is designed for a mobile environment.

**Note**

MQSeries for Windows and MQSeries for Windows NT are two different products.

## 8.6 About Queue Manager clusters

With MQSeries for MVS/ESA and Version 5.1 for distributed platforms, you can join queue managers together in clusters. Queue managers that form a cluster can run in the same machine or in different machines on different platforms. Usually, two of those “cluster queue managers” maintain a repository that contains information about all queue managers and queues in the cluster. This is called a full repository. The other queue managers maintain only a repository of objects in which they are interested, a partial repository. The repository allows any queue manager in the cluster to find out about any cluster queue and who owns it. The queue managers use special cluster channels to exchange information.

Clustering also permits multiple instances of a queue (with the same name) on different queue managers. This allows for workload distribution, that is, the queue manager can send messages to different instances of an application.

In normal distributed processing, we send messages to a specific queue owned by a specific queue manager. All messages destined for that queue manager are placed in a transmission queue on the sender's side. This transmission queue has the same name as the destination queue manager. The message channel agents move the messages across the network and place them into the destination queues. Figure 34 shows the relationship between a transmission (Xmit) queue and the target queue manager.

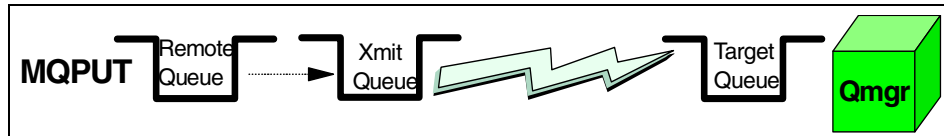


Figure 34. MQPUT to a remote queue

With clustering, you send a message to a queue with a specific name somewhere in the cluster; in Figure 35 on page 121, it is represented by a cloud. You specify the name of a target queue, not the name of a remote queue definition. Clustering does not require remote queue definitions. They are only useful when you send a message to a queue manager that is not a



member of the cluster. You can also specify a queue manager and direct the message to a specific queue, but, very often, it is left to the queue manager to determine where the queue is (or where the queues are) and to which one to send the message.

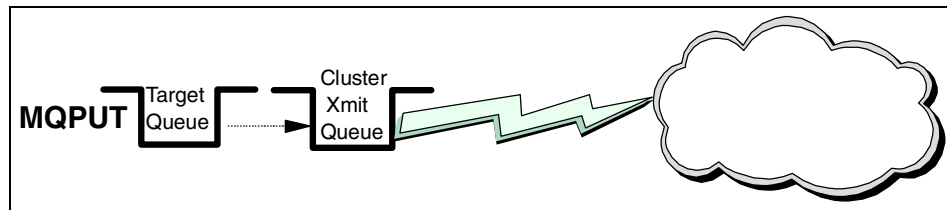


Figure 35. MQPUT to a cluster queue

Think of an MQSeries cluster as the place where multiple instances of a queue can exist. They come and go as required by an administrator in order to satisfy changing availability and throughput requirements. This has to be achieved completely dynamically and without placing the administrator under a great burden to configure and control. In addition, the programmer does not have to think about multiple queues; he or she just treats them as if writing to a single queue.

This is not to say that there is no burden on the programmer or administrator. Enhanced levels of availability and exploitation of parallelism require some planning. The administrator or system designer must ensure that there is enough redundancy in the configuration to meet their needs. The application designer must ensure that messages are capable of being processed in multiple places.

You create multiple instances of a queue by defining a queue with the same name on multiple queue managers that belong to the cluster. You must also name the cluster when you define the queue. Without this attribute, the queue would only be known locally. When the application specifies only the queue name, where is the message sent? Figure 36 gives you an idea.

MQSeries distributes the messages round-robin. You can, however, change this default action by writing your own workload balancing exit routine.

Figure 36 on page 122 shows messages put in one of the three cluster queues named A.

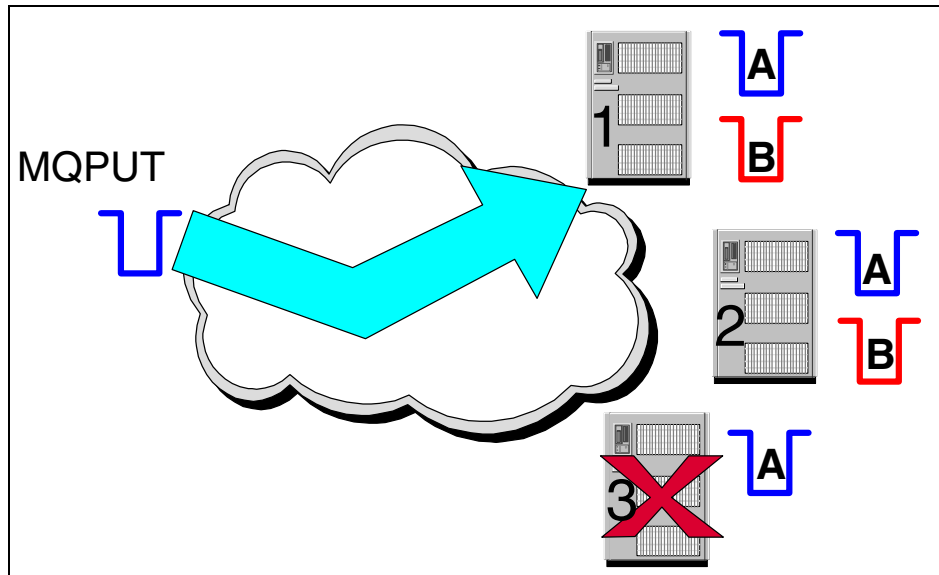


Figure 36. Accessing cluster queues

Each of the three queue managers on the right owns a queue with this name. By default, the first message is placed in queue A on queue manager 1, the next in queue A on queue manager 2, the third goes to queue manager 3, and the fourth message to the queue on queue manager 1 again.

In another scenario involving queue B, we notice that the third queue manager is down and the third instance of queue B is not available. The sending queue manager becomes aware of this problem because it subscribed to information about all queue managers and queues it is interested in, that is, where it sends messages. As soon as it finds out that there is a problem with the third instance of B, it distributes messages to the first two instances only. Special messages about changes of the status of cluster objects are instantly published to all queue managers that subscribed to that object.

---

## 8.7 About Queue Manager objects

This section introduces you to queue manager objects, such as queues and channels. The queue manager itself is also an object. Usually, an administrator creates one or more queue managers and their objects. A queue manager can use objects of the following types:

- Queues

- Process definitions
- Channels

The objects are common across different MQSeries platforms. There are other objects that apply to MVS systems only, such as the buffer pool, PSID, and storage class. AS/400 MQ objects are known to the OS/400 operating system as object type \*USRSPC (user space) in the QMQMDATA library.

### 8.7.1 Queues

Message queues are used to store messages sent by programs. There are local queues that are owned by the local queue manager and remote queues that belong to a different queue manager.

Queues are described in more detail in Section 8.8, “About message queues” on page 124.

### 8.7.2 Channels

A channel is a logical communication link. In MQSeries, there are two different kinds of channels:

#### 1. *Message channels*

A message channel connects two queue managers via message channel agents (MCAs). Such a channel is unidirectional. It comprises two message channel agents, a sender and a receiver, and a communication protocol. An MCA is a program that transfers messages from a transmission queue to a communication link and from a communication link into the target queue. For bidirectional communication, you have to define two channel pairs consisting of a sender and a receiver. Message channel agents are also referred to as movers.

#### 2. *MQI channels*

A Message Queue Interface (MQI) channel connects an MQSeries client to a queue manager in its server machine. Clients do not have a queue manager of their own. An MQI channel is bidirectional.

Figure 37 on page 124 shows both channels types. You see four machines: Two clients connected to their server machine via MQI channels and the server connected to another server or a host via two unidirectional message channels. Some channels can be defined automatically by MQSeries. There are different types of message channels depending on how the session between the queue managers is initiated and for what purpose they are used.

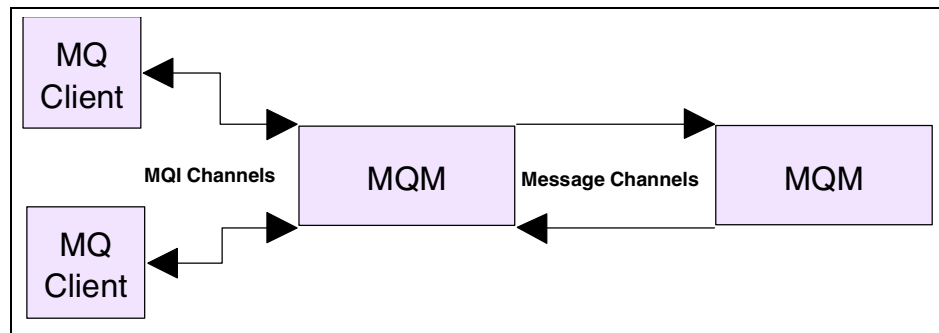


Figure 37. MQSeries channels

To transmit non-persistent messages, a message channel can run at *two speeds*: Fast and normal. Fast channels improve performance, but (non-persistent) messages can be lost in case of a channel failure.

A channel can use the following transport types: SNA LU 6.2, TCP/IP, NetBIOS, SPX, and DEC Net. Not all are supported on all platforms.

*MQSeries for Windows Version 2* uses message channels to connect to other machines. Since this product is designed as a single user system, it does not support MQI channels. This product supports only TCP/IP.

#### **Process definitions**

A process definition object defines an application to a queue manager. For example, it contains the name of the program (and its path) to be triggered when a message arrives for it.

## **8.8 About message queues**

Queues are defined as objects belonging to a queue manager. MQSeries knows a number of different queue types, each with a specific purpose. The queues you use are located either in your machine and belong to the queue manager to which you are connected, or in your server (if you are a client). Table 10 lists different queue types and their purposes. More detailed information follows.

Table 10. Queue types and their purposes

Queue type	Purpose
Local queue	A real queue
Remote queue	Structure describing a queue

Queue type	Purpose
Transmission queue (xmitq)	Local queue with a special purpose
Initiation queue	Local queue with a special purpose
Dynamic queue	Local queue created “on the fly”
Alias queue	If you don’t like the name
Dead-letter queue	One for each queue manager
Reply-to queue	Specified in a request message
Model queue	Model for local queues
Repository queue	Holds for cluster information

### 8.8.1 Local queue

A queue is local if it is owned by the queue manager to which the application program is connected. It is used to store messages for programs that use the same queue manager. For example, program A and program B each has a queue for incoming messages and another queue for outgoing messages. Since the queue manager serves both programs, all four queues are local.

#### Note

Both programs do not have to run in the same workstation. Client workstations usually use a queue manager in a server machine.

### 8.8.2 Cluster queue

A cluster queue is a local queue that is known throughout a cluster of queue managers, that is, any queue manager that belongs to the cluster can send messages to it without the need of a remote definition or defining channels to the queue manager that owns it.

### 8.8.3 Remote queue

A queue is “remote” if it is owned by a different queue manager. A remote queue definition is the local definition of a remote queue. A remote queue is not a real queue. It is a structure that contains some of the characteristics of a queue hosted by a different queue manager.

The application programmer can use the name of a remote queue just as he or she can use the name of a local queue. The MQSeries administrator

defines where the queue actually is. Remote queues are associated with a transmission queue.

**Note**

A program cannot read messages from a remote queue. You do not need a remote queue definition for a cluster queue.

#### **8.8.4 Transmission queue**

This is a local queue with a special purpose. A remote queue is associated with a transmission queue. Transmission queues are used as an intermediate step when sending messages to queues that are owned by a different queue manager.

Typically, there is only one transmission queue for each remote queue manager (or machine). All messages written to queues owned by a remote queue manager are actually written to the transmission queue for this remote queue manager. The messages will then be read from the transmission queue and sent to the remote queue manager.

Using MQSeries clusters, there is only one transmission queue for all messages sent to all other queue managers in the cluster.

Transmission queues are transparent to the application. They are used internally by the queue manager. When a program opens a remote queue, the attributes of the queue are obtained from the transmission queue. Therefore, the results of a program writing messages to a queue will be affected by the transmission queue characteristics.

#### **8.8.5 Dynamic queue**

Such a queue is defined "on the fly" when the application needs it. Dynamic queues may be retained by the queue manager or automatically deleted when the application program ends. Dynamic queues are local queues. They are often used in conversational applications to store intermediate results. Dynamic queues can be:

- Temporary queues that do not survive queue manager restarts
- Permanent queues that do survive queue manager restarts

#### **8.8.6 Alias queue**

Alias queues are not real queues but definitions. They are used to assign different names to the same physical queue. This allows multiple programs to

work with the same queue, accessing it under different names and with different attributes.

### 8.8.7 Model queue

A model queue is not a real queue. It is a collection of attributes that are used when a dynamic queue is created.

### 8.8.8 Initiation queue

An initiation queue is a local queue to which the queue manager writes a trigger message when certain conditions are met on another local queue, for example, when a message is put into an empty message queue or in a transmission queue. Such a trigger message is transparent to the programmer. Two MQSeries applications monitor initiation queues and read trigger messages, the trigger monitor that starts applications and the channel initiator that starts the transmission between queue managers.

#### Note

Applications do not need to be aware of initiation queues, but the triggering mechanism implemented through them is a powerful tool to design and write asynchronous applications.

### 8.8.9 Reply-to-queue

A request message must contain the name of the queue into which the responding program must put the reply message. This can be considered the “return address”. The name of this queue and the name of the queue manager that owns it are stored in the message header. This is the responsibility of the application program.

### 8.8.10 Dead-letter queue

A queue manager must be able to handle situations when it cannot deliver a message. Here are some examples:

- The destination queue is full.
- The destination queue does not exist.
- Message puts have been inhibited on the destination queue.
- The sender is not authorized to use the destination queue.
- The message is too large.
- The message contains a duplicate message sequence number.

When the above conditions are met, the messages are written to the dead-letter queue. Such a queue is defined when the queue manager is created. It will be used as a repository for all messages that cannot be delivered.

### 8.8.11 Repository queue

Repository queues have existed since Version 5.1 and Version 2.1 for OS/390. They are used in conjunction with clustering and hold either a full or partial repository of queue managers and queue manager objects in a cluster (or group) of queue managers.

### 8.8.12 Creating a Queue Manager

You may create as many queue managers as you like and have them running at the same time. You create a queue manager with the `crtmqm` command; to make it the default, specify the `/q` parameter.

The following command creates the default queue manager, MYQMGR (in a Windows NT environment):

```
crtmqm /q MYQMGR
```

#### Note

Queue manager names are case-sensitive.

There are default definitions for objects every queue manager needs, such as model queues. These objects are created automatically. Most certainly, you will have to create other objects that pertain to the applications you run. Usually, those application-specific objects are kept in a script file, such as `mydefs.in`. You apply them to a newly-created queue manager with the following command:

```
runmqsc < mydefs.in
```

MQSeries for Windows NT Version 5.1 provides a graphical user interface to create and manipulate queue managers and their objects.

A dead-letter queue is not automatically created. To create one when you create the queue manager, specify it as shown in the following example:

```
crtmqm /q /u system.dead.letter.queue MYQMGR
```

To start the queue manager, issue the following command:

```
strmqm
```



---

## 8.9 Manipulating Queue Manager objects

MQSeries for distributed platforms provides the utility, RUNMQSC, to create and delete queue manager objects and manipulate them. The queue manager must be running when you use the utility. RUNMQSC works in two ways:

- You can type the commands.
- You can create a file containing a list of commands and use this file as input.

The commands in the next screen start the default queue manager (which is already running, as the response indicates) and create the local queue, QUEUE1, for it. Another command alters the queue manager properties to define a dead-letter queue.

To start the utility in an interactive mode, type `runmqsc`. To end it, type `end`. Another way to create MQSeries objects is by using an input file instead of typing the commands; for example:

```
runmqsc < mydefs.in > a.a
```

where `mydefs.in` is the script file that contains the commands and `a.a` is the file that will contain the responses from the RUNMQSC utility so that you can check if any error occurred. The output can either appear in the window or can be redirected to a file.

The following screen describes how to use Control Commands to manipulate objects.

```

C:\strmqm
MQSeries queue manager running.

runmqsc
84H2001,6539-B42 (C) Copyright IBM Corp. 1994, 1997. ALL RIGHTS RESERVED
Starting MQSeries Commands.

define qlocal('QUEUE1') replace descr ('test queue')
  1 : define qlocal('QUEUE1') replace descr ('test queue')
AMQ8006: MQSeries queue created.
alter qmgr deadq(system.dead.letter.queue)
  2 : alter qmgr deadq(system.dead.letter.queue)
AMQ8005: MQSeries queue manager changed.
end
  3 : end
2 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.

C:\

```

## 8.10 Clients and servers

MQSeries distinguishes clients and servers. Before you install MQSeries on a distributed platform, you have to decide if the workstation will be an MQSeries client, an MQSeries server, or both. With MQSeries for Windows, a new term was introduced, the leaf node (described later). There are two kinds of clients:

- Slim client or MQSeries client
- Fat client

Fat clients have a local queue manager; slim clients do not.

When a slim client cannot connect to its server, it cannot work because the queue manager and queues for a slim client reside in the server. Usually, an MQSeries client is a slim client. Several of these clients share MQSeries objects (the queue manager is one of them) in the server to which they are attached.

### Note

The MQSeries Client for Java is a slim client.

In some cases, it may be advantageous to have queues in the end user's workstation, especially in a mobile environment. This allows you to run your application when a connection between client and server does not (temporarily) exist.

You may install client and server software in the same system and use it as an end user's workstation. If your operating system is Windows NT, you can install MQSeries for Windows NT V5.1 or MQSeries for Windows V2.1 (also called MQWin). If your operating system is Windows 95, use MQWin V2.1. This product has been designed for end users and uses fewer resources.

The difference between an end user's workstation that is a client and one that has a queue manager is the way messages are sent. The queues reside either in the end user's workstation or in the server. Figure 38 again shows the use of MQI and message channels.

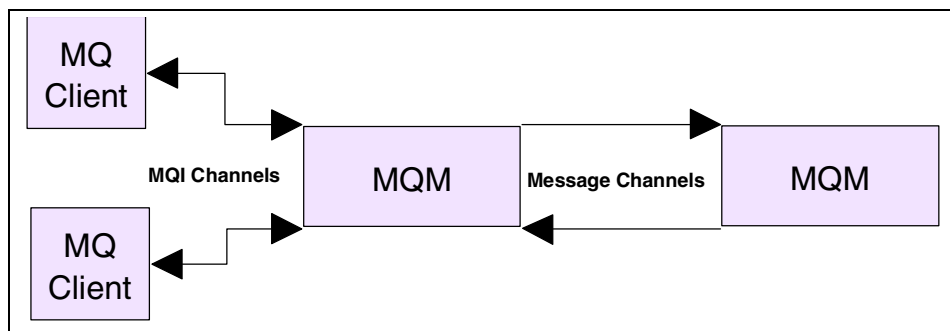


Figure 38. MQI and message channels

- MQI channels connect clients to a queue manager in a server machine. All MQSeries objects for the client reside in the server. MQI channels are faster than message channels.
- A message channel connects a queue manager to another queue manager. The queue manager can reside in the same or in a different machine.

The following sections summarize the three workstation types.

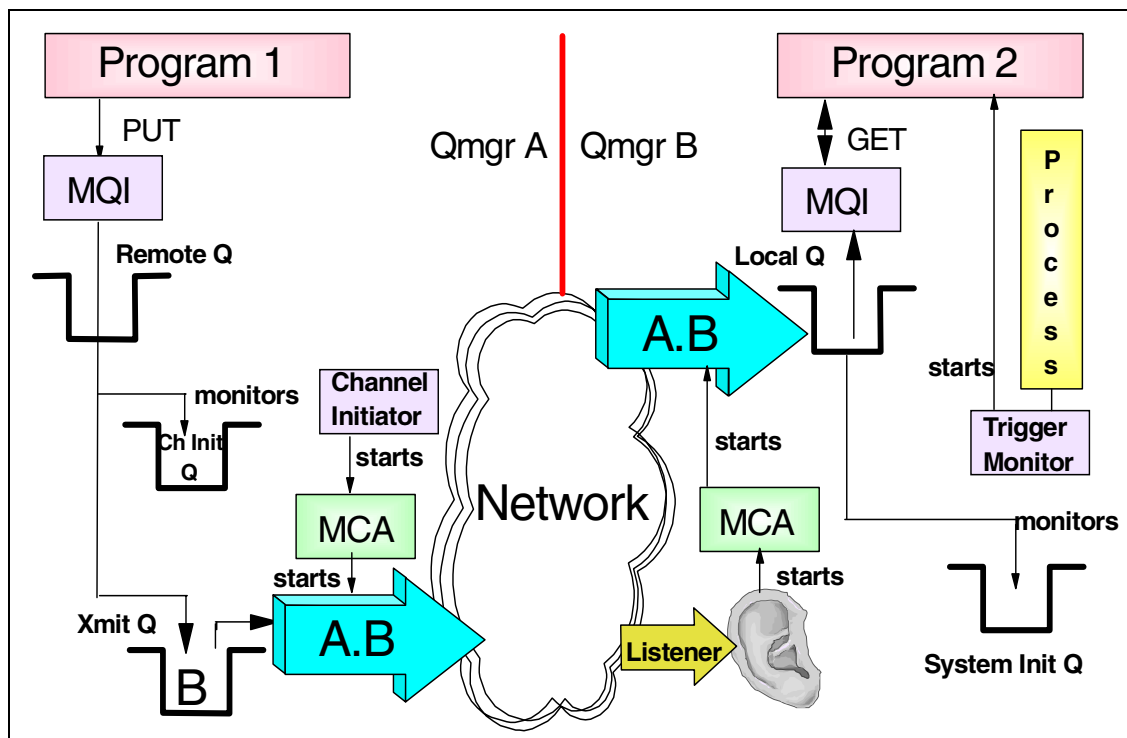
### ***MQSeries client***

A client workstation does not have a queue manager of its own. It shares a queue manager in a server with other clients. All MQSeries objects, such as queues, are in the server. Since the connection between client and server is synchronous, the application cannot work when the communication is broken. You could refer to such workstations as "slim" clients.

A workstation can be a client and a server. A server is an intermediate node between other nodes. It serves clients that have no queue manager and manages the message flow between its clients, itself, and other servers. In addition to the server software, you may also install the client software. This configuration is used in an application development environment.

MQSeries for Windows was designed for use by a single user. It has its own "small footprint" queue manager with its own objects. However, it is not an intermediate node between other nodes. It is called a leaf node. You could also refer to it as a "fat" client. This product is able to queue outbound messages when connection to a server or host is not available and inbound messages when the appropriate application is not active.

Figure 39 shows the parts and architecture of MQSeries.



132 Business-to-Business Integration Using MQSeries and MQSI

The application program uses the Message Queue Interface (MQI) to communicate with the queue manager. The MQI is described in more detail later. The queuing system consists of the following parts:

- Queue Manager (MQM)
- Listener
- Trigger Monitor
- Channel Initiator
- Message Channel Agent (MCA) or mover

When the application program wants to put a message on a queue, it issues an MQPUT API call. This invokes the MQI. The queue manager checks whether the queue referenced in the MQPUT is local or remote. If it is a remote queue, the message is placed into the transmission (xmit) queue. The queue manager adds a header that contains information from the remote queue definition, such as destination queue manager name and destination queue name.

**Note**

Each remote queue must be associated with an xmit queue. Usually, all messages destined for one remote machine use the same xmit queue.

Transmission is done via channels. Channels can be started manually or automatically. To start a channel automatically, the xmit queue must be associated with a channel initiation queue. Figure 39 on page 132 shows that the queue manager puts a message into the xmit queue and another message into the channel initiation queue. This queue is monitored by the *channel initiator*.

The channel initiator is an MQSeries program that must be running in order to monitor initiation queues. When the channel initiator detects a message in the initiation queue, it starts the message channel agent (MCA) for the particular channel. This program moves the message over the network to the other machine, using the sender part of the unidirectional message channel pair.

On the receiving end, a *listener* program must have been started. The listener, also supplied with MQSeries, monitors a specified port, by default, the port dedicated to MQSeries, 1414. When a message arrives, it starts the *message channel agent*. The MCA moves the message into the specified local queue using the receiver part of the message channel pair.

**Note**

Both channel definitions, sender and receiver, must have the same name. For the reply, you need another message channel pair.

The program that processes the incoming message can be started manually or automatically. To start the program automatically, an initiation queue and a process must be associated with the local queue, and the *trigger monitor* must be running.

When the program starts automatically, the MCA puts the incoming message into the local queue and a trigger message into the initiation queue. This queue is monitored by the trigger monitor. This program invokes the application program specified in the process definition. The application issues an MQGET API call to retrieve the message from the local queue.

---

## 8.12 Communication between queue managers

In this section, we discuss what you have to define to send messages to a queue manager that resides in another system. We use message channels for communication between queue managers as shown in Figure 38 on page 131.

The logic is illustrated in Figure 40 on page 136, and the necessary MQSeries definitions are shown in Table 11 on page 136.

Each machine has a queue manager installed, and each queue manager manages several local queues. Messages destined for a remote queue manager are put into a *remote queue*. A remote queue is not a real queue; it is the definition of a local queue in the remote machine. A remote queue is associated with a transmission (xmit) queue, which is a local queue. Usually, there is one xmit queue for each remote queue manager.

A transmission queue is associated with a message channel. Message channels are unidirectional, meaning that you have to define two channels for a conversational type of communication. Also, you have to define each channel twice, once in the system that sends the message (sender channel) and once in the system that receives the message (receiver channel). Each channel pair (sender and receiver) must have the same name. This scenario is elucidated in Figure 40 on page 136. Next, let us find out how we get this to work.

### 8.12.1 How to define a connection between two systems

Figure 40 on page 136 shows the required MQSeries objects for connecting two queue managers.

In each system we need the following:

- A remote queue definition that mirrors the local queue in the receiver machine and links to a transmission queue (Q1 in system A and Q2 in system B).
- A transmission queue that holds all messages destined for the remote system until the channel transmits them (QMB in system A and QMA in system B).
- A sender channel that gets messages from the xmit queue and transmits them to the other system using the existing network (QMA.QMB in system A and QMB.QM.A in system B).
- A receiver channel that receives messages and puts them into a local queue (QMB.QMA in system A and QMA.QMB in system B); receiver channels can be started automatically by the queue manager when Channel Auto Definition (CHAD) is enabled.
- A local queue from which the program gets its messages (Q2 in system A and Q1 in system B).

In each system, you must define the appropriate queue manager objects. The objects are defined in the two script files shown in Table 11 on page 136.

When you use clustering, you don't have to define transmission queues. There is only one transmission queue per queue manager, and that is created automatically when the queue manager is created.

You also don't have to define channels, neither sender nor receiver channels; they are automatically created when needed.

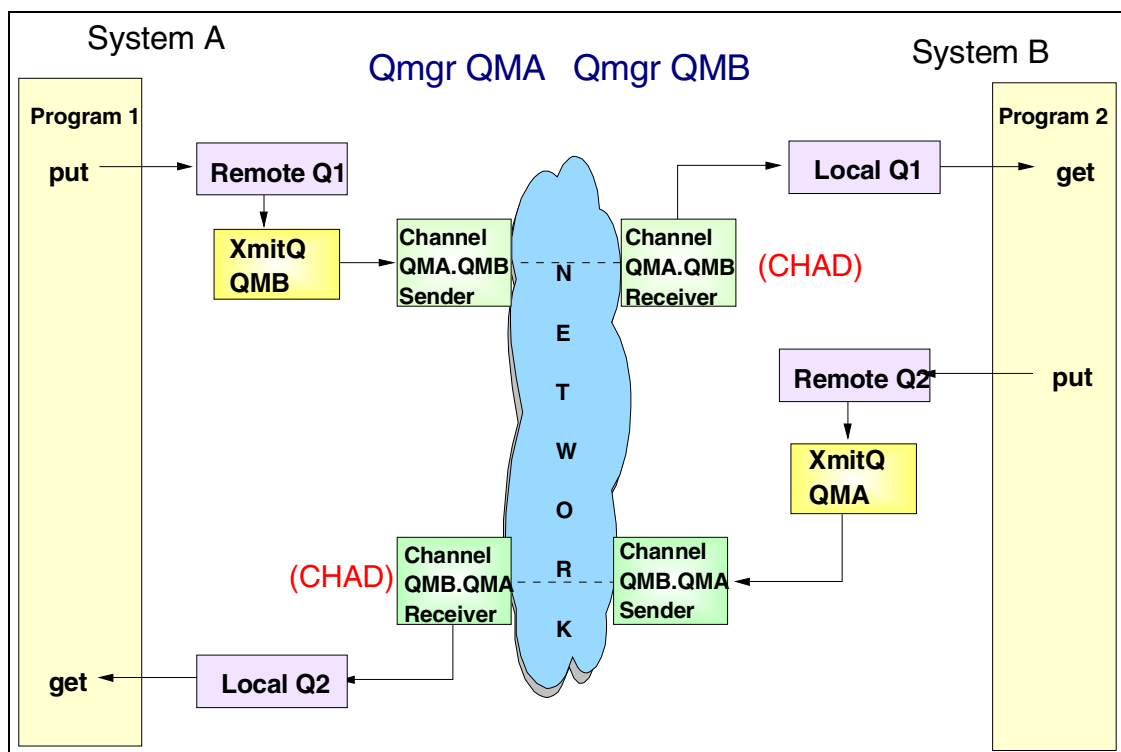


Figure 40. Communication between two queue managers

Table 11 lists the MQ Series objects defining a connection between two queue managers.

Table 11. MQSeries Objects defining connection between two queue managers

System A (QMA)	System B (QMB)
DEFINE QREMOTE(Q1) + RNAME(Q1) RQMNAME(QMB) + XMITQ(QMB)	DEFINE QLOCAL(Q1)
DEFINE QLOCAL(QMB) + USAGE(xmitq)	
DEFINE CHANNEL(QMA.QMB) + CHLTYPE(sdr) + XMITQ(QMB) + TRPTYPE(tcp) + CONNAME(9.24.104.123)	DEFINE CHANNEL(QMA.QMB) + CHLTYPE(rcvr) + TRPTYPE(tcp)



System A (QMA)	System B (QMB)
DEFINE QLOCAL(Q2)	DEFINE QREMOTE(Q2) + RNAME(Q2) RQMNAME(QMA) + XMITQ(QMA)
	DEFINE QLOCAL(QMA) + USAGE(xmitq)
DEFINE CHANNEL(QMB.QMA) + CHLTYPE(rcvr) + TRPTYPE(tcp)	DEFINE CHANNEL(QMB.QMA) + CHLTYPE(sdr) + XMITQ(QMA) + TRPTYPE(tcp) CONNAME(ABC1)

### 8.12.2 How to start communication manually

First, the objects have to be known to the queue managers. You use RUNMQSC to create the objects. Make sure that the queue manager is running. Next, start the listeners and the channels. You need to start only the sender channel in each system. MQSeries starts the receiver channel. The commands to start listener and channel for queue manager QMA are as follows:

```
strmqm QMA

start runmqlsr -t tcp -m QMA -p 1414

runmqsc

start channel (QMA.QMB)

end
```

With the first command, you start queue manager QMA. The next command starts the listener. It listens on behalf of QMA on port 1414 if TCP/IP is used. The third command starts runmqsc in interactive mode. The channel, QMA.QMB, is started under control of runmqsc. For the other queue manager, you issue equivalent commands. You also have to start the applications in both systems.

#### 8.12.2.1 How to start communication automatically

You can use the channel initiator to start channels. Instead of the commands shown above, enter the following commands (for Windows NT, UNIX, and OS/2):

```
start runmqlsr -t tcp -m QMA -p 1414
start runmqchi
```

With the first command, you start the listener, and, with the second, you start the channel initiator program.

The channel initiator monitors a channel initiation queue and starts the proper channel to read in the message. The default initiation queue is SYSTEM.CHANNEL.INITQ.

You may also start the channel initiator from RUNMQSC (Windows NT, UNIX, and OS/2). The command is:

```
start chinit
```

OR

```
start chinit initq(SYSTEM.CHANNEL.INITQ)
```

To have the transmission queue triggered, add the following parameters:

```
DEFINE QLOCAL(A.TO.B) REPLACE +  
    USAGE(xmitq) +  
    TRIGGER  
    TRIGTYPE(every) +  
    INITQ(SYSTEM.CHANNEL.INITQ) +  
    DESCR('Xmit Queue')
```

The queue manager can trigger the process that starts the channel program in three ways:

- When the first message is put into the transmission queue
- Every time a message is put into the xmit queue
- When the queue contains a specified number of messages

Figure 41 on page 139 shows the logic behind triggering.

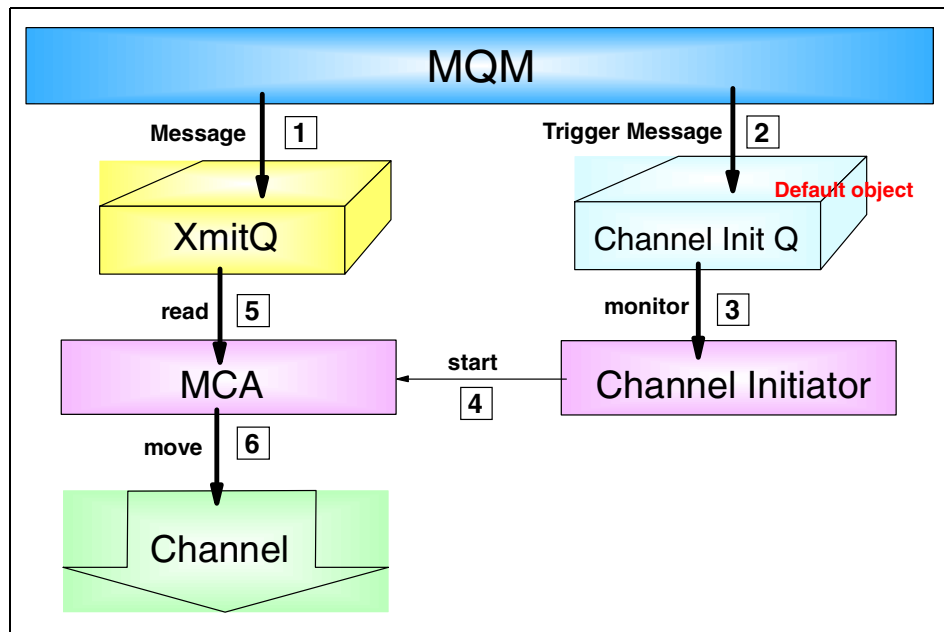


Figure 41. Triggering channels

The steps are as follows:

1. The program issues an MQPUT to a remote queue, and a message is placed into the transmission queue.
2. When the queue manager puts a message into the transmission queue, it checks the trigger type specified in the queue definition. Depending on that definition and on how many messages are in the queue, it may put an additional message in the channel initiation queue. This “trigger message” is transparent to the user.
3. Since the channel initiator was started earlier, for example, at boot time, it monitors the channel initiation queue and removes the trigger message.
4. The channel initiator starts the message channel agent (also called mover).
5. The channel program gets the message off the transmission queue and invokes any channel exit routines, if specified.
6. The message is then moved over the network to its destination.

### 8.13 How to trigger applications

This section describes how to trigger an application program that runs in the server machine. Both triggering and triggered applications can run under the same or different queue managers.

**Note**

MQSeries for Windows V2.1 does not support triggering.

Figure 42 shows the logic of triggering.

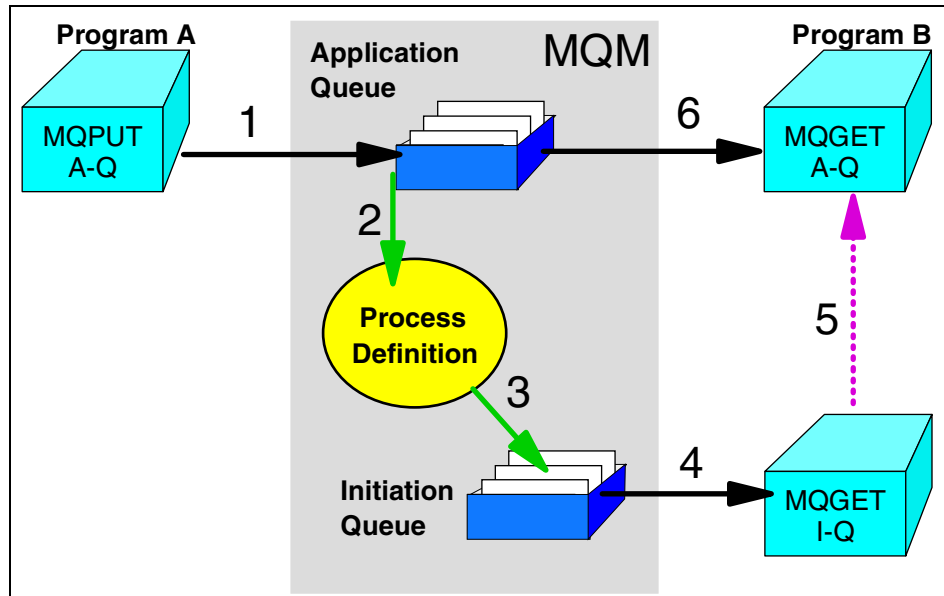


Figure 42. Triggering an application

Here, Program A sends a message to A-Q to be processed by Program B. The MQSeries triggering mechanism is as follows:

1. Program A issues an MQPUT and puts a message into A-Q for Program B.
2. The queue manager processes this API call and puts the message into the application queue.
3. It also finds out that the queue is triggered. It creates a trigger message and looks in the Process Definition to find the name of the application and puts it in the trigger message. The trigger message is put into the initiation queue.

4. The trigger monitor gets the trigger message from the initiation queue and starts the program specified.
5. The application program starts running and issues an MQGET to retrieve the message from the application queue.

The definitions necessary to trigger an application are as follows:

- The target queue must have “triggering” specified as shown in bold below:

```
DEFINE QLOCAL(A-Q) REPLACE +
    TRIGGER
    TRIGTYPE(first) +
    INITQ(SYSTEM.DEFAULT.INITIATION.QUEUE ) +
    PROCESS(proc1)
    DESCR('This is a triggered queue')
```

- The process definition associated with the target queue can be as follows:

```
DEFINE PROCESS(proc1) REPLACE +
    DESCR('Process to start server program') +
    APPLTYPE(WINDOWSNT) +
    APPPLCID('c:\test\myprog.exe')
```

What trigger type to use depends on how the application is written. You have three choices as shown in Figure 43.

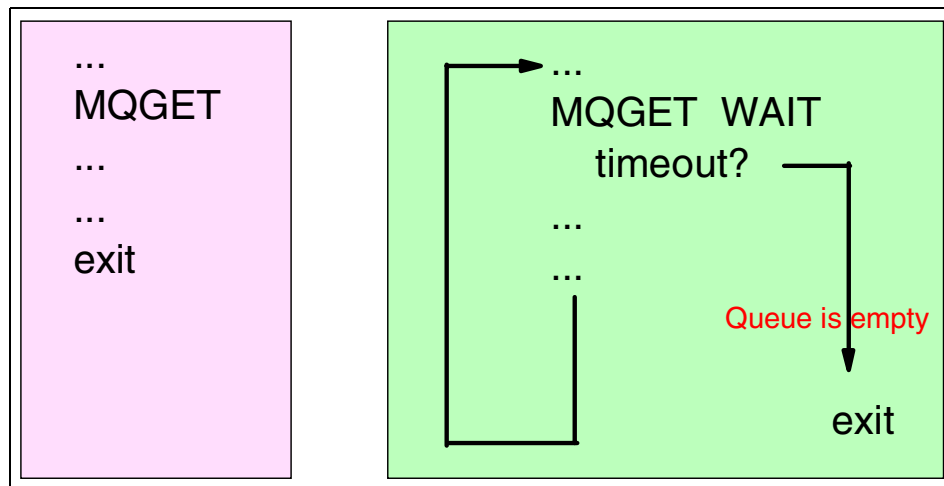


Figure 43. MQSeries application trigger choices

The three choices are:

**EVERY** Every time a message is put in the target queue, a trigger message is also put in the initiation queue. Use this when your

program exits after processing one message or transaction, as shown in the left part of Figure 43 on page 141.

**FIRST**

A trigger message is put in the initiation queue only when the target queue has been empty. Use this when the program exits only then when there are no more messages in the queue as shown in the right part of Figure 43 on page 141.

**n messages**

A trigger message is put in the initiation queue when there are n messages in the target queue. For example, you can start a batch program when the queue holds 1000 messages.

---

## 8.14 Communication between client and server

In the following section, we discuss what is necessary to define and test the connection between an MQ client and its MQ server. A more detailed description is provided in the publication *MQSeries Clients*, GC33-1632.

### 8.14.1 How to define a client/server connection

Figure 44 shows that the MQSeries Client product is installed in the client machine.

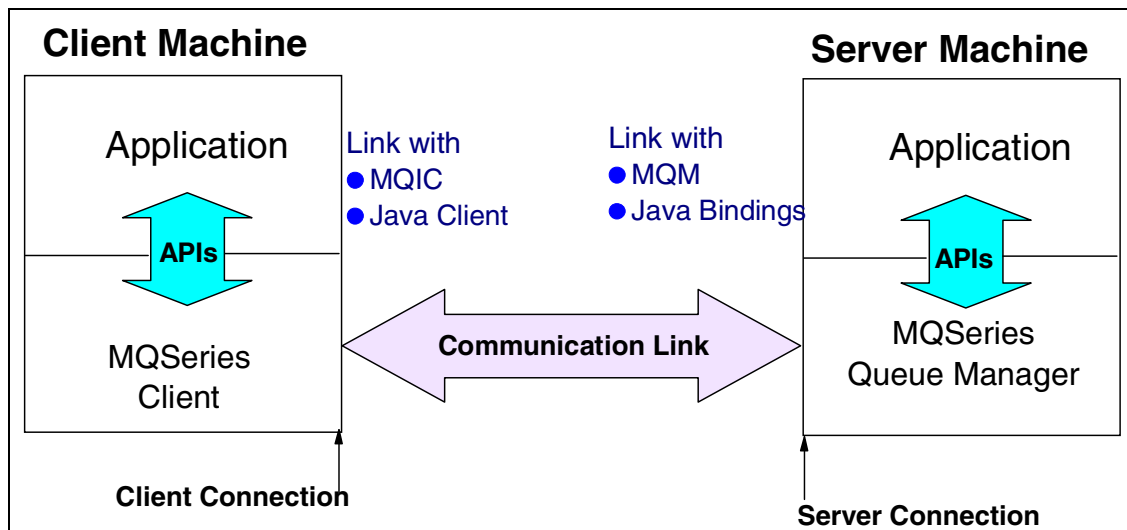


Figure 44. Client/Server connection

We said earlier that clients and servers are connected with MQI channels. An MQI channel consists of a sender/receiver pair, called the Client Connection (CLNTCONN) and Server Connection (SVCONN) channels.

You have to know what transmission protocol is used (for example, TCP/IP), the port the listener listens to (1414 is the default), and the address of the systems to which you want to connect. For an address, you can specify an LU name, a host name or machine name, or a TCP/IP address.

The client connection channel is defined as an environment variable, such as:

```
set MQSERVER=CHAN1/TCP/9.24.104.206(1414)
```

where:

- `MQSERVER` is the name of the environment variable.
- `CHAN1` is the name of the channel to be used for communication between client and server. This channel is defined in the server. MQSeries will automatically create it if it does not exist.
- `TCP` denotes that TCP/IP is to be used to connect to the machine with the address following the parameter.
- `1414` is the default port number for MQSeries. You may omit this parameter if the listener on the server side also uses this default.

The definition of the server is as follows:

```
DEFINE CHANNEL('CHAN1') CHLTYPE(SVRCONN) REPLACE +  
          TRPTYPE(TCP) MCAUSER(' ')
```

For the MQSeries Client for Java, the environment variables are set in the applet code. An applet can run in any machine, such as a network station, and it has no access to environment variables. The following example shows what statements to include in your Java program:

```
import com.ibm.mq.*;  
  
MQEnvironment.hostname = "9.24.104.456";  
  
MQEnvironment.channel  = "CHAN1";  
  
MQEnvironment.port     = 1414;
```

### 8.14.2 How a Client/Server connection works

Now, we will describe how to trigger an application program that runs in the server machine. Since there are MQI channels of the type server connection between clients and server, all clients use the queue manager in the server machine. When a client puts a message on a queue, it has to be read and processed by a program. This program can be started when the server starts,

or the queue manager can start it when needed by using the MQSeries triggering mechanism.

Figure 45 on page 145 shows two clients connected to a server. Both clients request services from the same program (Appl S1). Since that application runs in the same system as the queue manager, we have only local queues. Some queues are specifically for a particular client; for example, QA1 is the reply queue for client A, and QA2 is the reply queue for client B. Other queues are used by both clients and server. For example, QS1 is used as the output queue for both clients and as the input queue for the server program.

Next, we describe the MQSeries objects and API call sequences in both client and server.

### **8.14.3 How a Client sends a request**

The client starts a program that puts a message on a queue. For this function, five MQSeries API calls are executed:

- MQCONN to connect to the queue manager in the server
- MQOPEN to open the message queue QS1 for output
- MQPUT to put a message in the queue
- MQCLOSE to close the queue QS1
- MQDISC to disconnect from the queue manager

Of course, the program can put many messages in the queue before it closes it and disconnects. Closing the queue and disconnecting from the queue manager can be done when the application ends because there are no more messages to process.

The MQSeries client code that runs in the client machine processes the API calls and routes them to the machine defined in the environment variable.



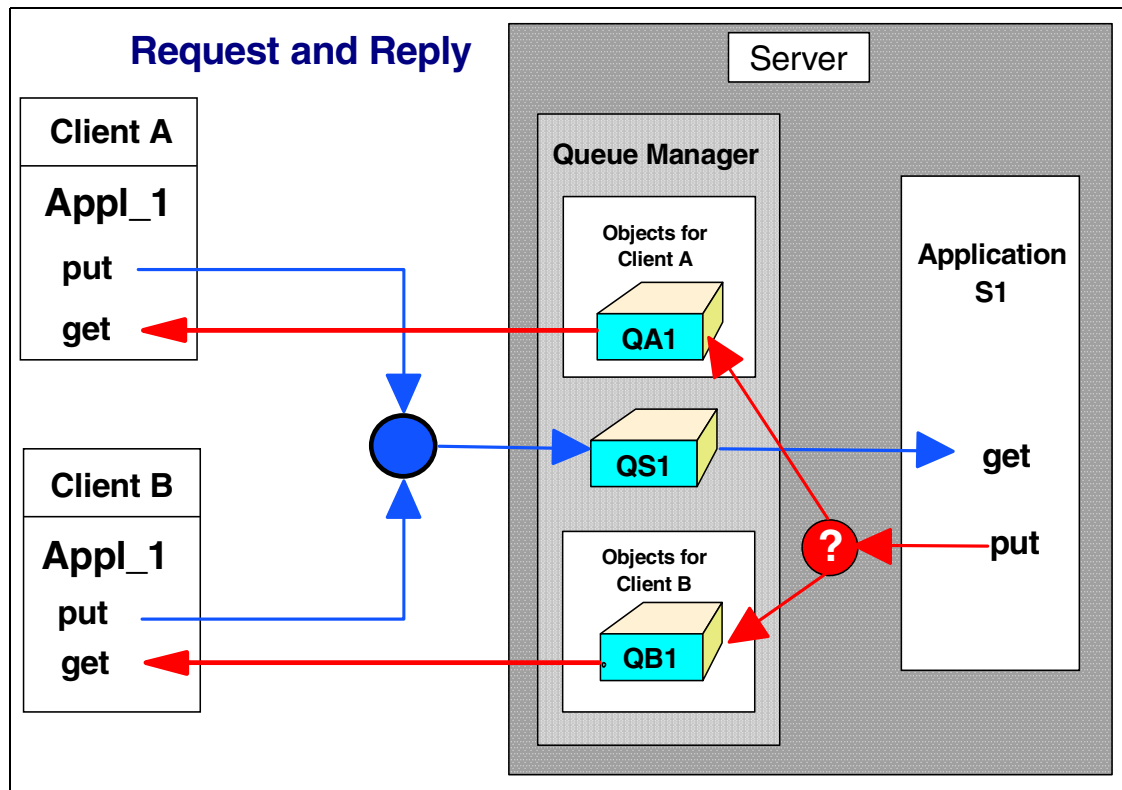


Figure 45. Clients and server communicating

#### 8.14.4 How the server receives a request

The following queue manager objects are needed in the server machine:

- A channel of the type server connection.
- A local queue, QS1, into which the clients put their messages.
- An initiation queue into which the queue manager puts a trigger message when a request for queue QS1 arrives. You can use the default initiation queue.
- A process definition that contains the name of the program to be started when the trigger event occurs (S1).
- One or more queues in which the program puts the reply messages, QA1 and QB1.

In the server machine, two programs have to be started: The listener and the trigger monitor. The listener listens for messages on the channel and puts them on the queue, QS1. Since QS1 is triggered, the MQM puts a trigger

message on the trigger queue each time a message is put on QS1. When a message is placed on the trigger queue, the trigger monitor starts the program defined in the process.

The server program, S1, connects to the queue manager, opens the queue, QS1, and issues an MQGET to read the message.

#### **8.14.5 How the server sends a reply**

After processing a request, the server puts the reply in the reply queue for the client. To do this, it has to open the output queue (QA1 or QB1) and issue an MQPUT.

Since several clients use the same server application, it is advisable to give the server a *return address*, that is, the names of the queue and the queue manager that will receive the reply message. These fields are in the header of the request message containing the reply-to-queue manager and reply-to-queue (here, QA1 or QB1). It is the responsibility of the client program to specify these values.

Usually, the server program stays active and waits for more messages, at least for a certain time. For how long can be specified in the wait option of the MQGET API.

#### **8.14.6 How the client receives a reply**

The client program knows the name of its input queue; here, it is QA1 or QB1. The application can use two modes of communication:

- *Conversational*

If the application uses this mode of communication with the server program, it waits for the message to arrive before it continues processing. This means that the reply queue is open and an MQGET with the wait option has been issued. The client application must be able to deal with two possibilities:

- The message arrives in time.
- The timer expires, and no message is there.

- *True asynchronous*

When using this mode, the client does not care when the request message arrives. Usually, the user clicks a push button in a menu window to activate a program that checks the reply queue for messages. If a message is present, this or another program can process the reply.

---

## 8.15 The Message Queuing Interface (MQI)

A program talks directly to its local queue manager. It resides in the same processor or domain (for clients) as the program itself. The program uses the Message Queuing Interface (MQI). The MQI is a set of API calls that request services from the queue manager.

### Note

When the connection between a client and its server is broken, no API calls can be executed because all objects reside in the server.

The 13 APIs are shown in Table 12.

Table 12. MQSeries APIs and their purposes

API	Purpose
MQCONN	Connect to a queue manager
MQDISC	Disconnect from a queue manager
MQOPEN	Open a specific queue
MQCLOSE	Close a queue
MQPUT	Put a message on a queue
MQGET	Get a message from a queue
MQPUT1	MQOPEN + MQPUT + MQCLOSE
MQINQ	Inquire properties of an object
MQSET	Set properties of an object
MQCONNX	Standard or fastpath bindings
MQBEGIN	Begin a unit of work (database coordination)
MQCMIT	Commit a unit of work
MQBACK	Back out

The most important ones are MQPUT and MQGET. The other calls are used less frequently. Comments regarding several APIs follow:

*MQCONN* establishes a connection with a queue manager using the standard bindings.

*MQCONN* establishes a connection with a queue manager using fastpath bindings. Fastpath puts and gets are faster, but the application must be well behaved, that is, well tested. Application and queue manager run in the same process. When the application crashes, it takes the queue manager down with it. This API call is new in MQSeries Version 5.

*MQBEGIN* begins a unit of work that is coordinated by the queue manager and may involve external XA-compliant resource managers. This API has been introduced with MQSeries Version 5. It is used to coordinate transactions that use queues (*MQPUT* and *MQGET* under syncpoint) and database updates (SQL commands).

*MQPUT1* opens a queue, puts a message on it and closes the queue. This is a combination of *MQOPEN*, *MQPUT*, and *MQCLOSE*.

*MQINQ* requests information about the queue manager or one of its objects, such as the number of messages in a queue.

*MQSET* changes some attributes of an object.

*MQCMIT* specifies that a syncpoint has been reached. Messages put as part of a unit of work are made available to other applications. Messages retrieved as part of a unit of work are deleted.

*MQBACK* tells the queue manager to back out all message puts and gets that have occurred since the last syncpoint. Messages put as part of a unit of work are deleted. Messages retrieved as part of a unit of work are reinstated on the queue.

**Note**

*MQDISC* implies the commit of a unit of work. Ending the program without disconnecting from the queue manager causes a rollback (*MQBACK*).

MQSeries for AS/400 does not use *MQBEGIN*, *MQCMIT*, or *MQBACK*. The commit control operation codes of the AS/400 language are used.

---

## 8.16 A code fragment

The code fragment below shows the APIs to put a message on one queue and get the reply from another queue.

**Note**

The fields, CompCode and Reason, will contain completion codes for the APIs. You can find them in the Application Programming Reference.

**Comments:**

1. This statement connects the application to the queue manager with the name MYQMGR. If the parameter QMName does not contain a name, then the default queue manager is used. MQ stores the handle of the queue manager in the variable, HCon. This handle must be used in all subsequent APIs.
2. To open a queue, the queue name must be moved into the object descriptor that will be used for that queue. This statement opens QUEUE1 for output only (open the option, MQOO\_OUTPUT). The handle to the queue and values in the object descriptor are returned. The handle, Hobj1, must be specified in the MQPUT.
3. MQPUT places the message assembled in a buffer on a queue.  
Parameters for MQPUT are:
  - The handle of the queue manager (from MQCONN)
  - The handle of the queue (from MQOPEN)
  - The message descriptor
  - A structure containing options for the put (refer to the Application Programming Reference)
  - The message length
  - The buffer containing the data
4. This statement closes the output queue. Since the queue is predefined, no close processing takes place (MQOC\_NONE).
5. This statement opens QUEUE2 for input only using the queue-defined defaults. You could also open a queue for browsing, meaning that the message will not be removed.

Figure 46 on page 150 shows a code fragment.

```

MQHCONN HCon;                // Connection handle
MQHOBJ  HObj1;               // Object handle for queue 1
MQHOBJ  HObj2;               // Object handle for queue 2
MQLONG  CompCode, Reason;    // Return codes
MQLONG  options;
MQOD    od1 = {MQOD_DEFAULT}; // Object descriptor for queue 1
MQOD    od2 = {MQOD_DEFAULT}; // Object descriptor for queue 2
MQMD    md = {MQMD_DEFAULT};  // Message descriptor
MQPMO   pmo = {MQPMO_DEFAULT}; // Put message options
MQGMO   gmo = {MQGMO_DEFAULT}; // Get message options
:
// 1 Connect application to a queue manager.
strcpy (QMName, "MYQMGR");
MQCONN (QMName, &HCon, &CompCode, &Reason);

// 2 Open a queue for output
strcpy (od1.ObjectName, "QUEUE1");
MQOPEN (HCon, &od1, MQOO_OUTPUT, &HObj1, &CompCode, &Reason);

// 3 Put a message on the queue
MQPUT (HCon, HObj1, &md, &pmo, 100, &buffer, &CompCode, &Reason);

// 4 Close the output queue
MQCLOSE (HCon, &HObj1, MQCO_NONE, &CompCode, &Reason);

// 5 Open input queue
options = MQOO_INPUT_AS_Q_DEF;
strcpy (od2.ObjectName, "QUEUE2");
MQOPEN (HCon, &od2, options, &HObj2, &CompCode, &Reason);

// 6 Get message
gmo.Options = MQGMO_NO_WAIT;
buflen = sizeof(buffer - 1);
memcpy (md.MsgId, MQMI_NONE, sizeof(md.MsgId));
memset (md.CorrelId, 0x00, sizeof(MQBYTE24));
MQGET (HCon, HObj2, &md, &gmo, buflen, buffer, 100, &CompCode, &Reason);

// 7 Close the input queue
options = 0;
MQCLOSE (HCon, &HObj2, options, &CompCode, &Reason);

// 8 Disconnect from queue manager
MQDISC (HCon, &CompCode, &Reason);

```

Figure 46. A code fragment

6. For the get, the nowait option is used. The MQGET needs the length of the buffer as an input parameter. Since there is no message ID or correlation ID specified, the first message from the queue is read. You may specify a wait interval (in milliseconds) here. You can check the return code to find out if the time has expired and no message arrived.

7. This statement closes the input queue.
8. The application disconnects from the queue manager.

---

### 8.17 MQSeries integrator components

The key components in an MQSeries Integrator environment include:

- Configuration Manager and configuration repository
- One or more brokers
- Control Center
- User Name Server
- MQSeries Queue Manager(s)
- Message Repository Manager (MRM)

Using the Control Center, the MQSeries Integrator administrator defines messages in the message repository and message flows. A message flow is a series of operations and rules that a broker executes for a retrieved message. Message flows are grouped in an execution group. Execution groups are assigned to a broker.

When a client application puts a message on an input queue of the broker, the broker will retrieve the message and parse it. From that point on, a message is stored in the common message interface, which is, basically, a tree structure representing all fields in that message. The broker then executes the actions that are configured in the message flow, and, eventually, the message arrives at an output queue ready to be retrieved by another client application.

Figure 47 on page 152 shows the overview of MQSeries integrator.

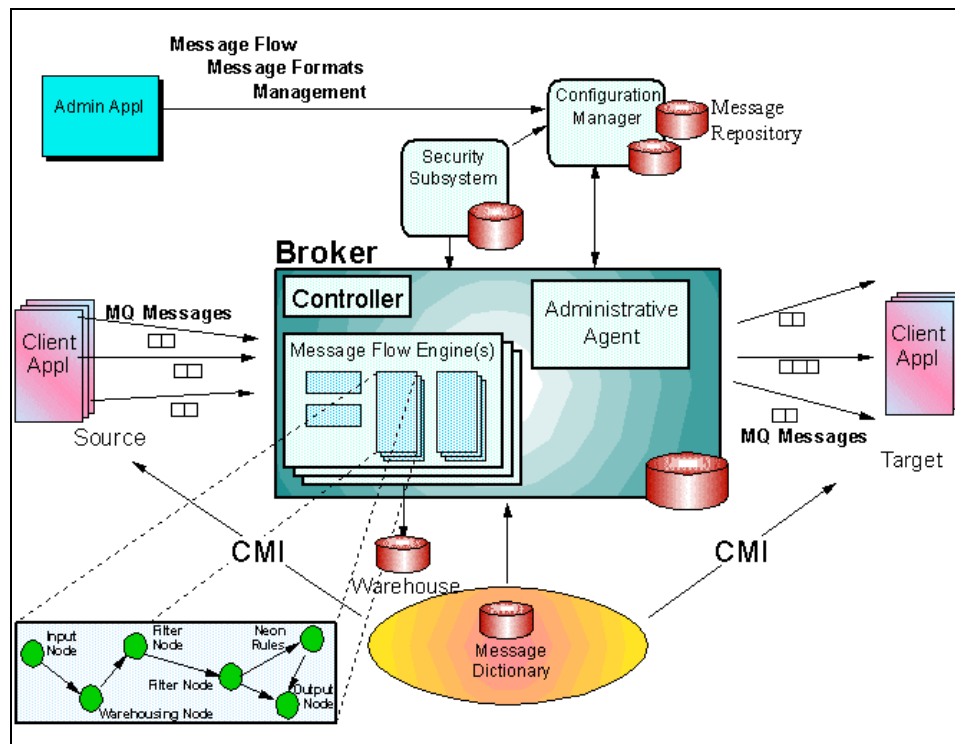


Figure 47. Overview of MQSeries integrator

A message flow consists of a number of nodes, usually starting with an MQInput node and ending with one or more MQOutput nodes. An MQInput node is the encapsulation of a normal MQGET operation, while an MQOutput node is the encapsulation of a normal MQPUT operation. Between these two nodes, a message flow can contain other nodes that transform the message. A Filter node allows you to specify a Boolean expression for routing the message. In a Compute node you can specify a script to alter message fields. The script is written in the programming language ESQL, which is an extension of standard SQL. Another important type of node is a Database node. These nodes allow you to interact with an external database. You can use information from a database to enrich a message or route a message. You can use the message to make updates in the table. It is also possible to store the message using the Warehouse node.

Besides the standard nodes, called IBMPrimitives, you can extend the palette of nodes with user-written or third-party nodes that perform a specific function. Within such a user-written node, you have access to the message



that flows in and you can create a new transformed message, just like you can with a Compute node.

### 8.17.1 The Configuration Manager

An MQSeries Integrator system is controlled by the Configuration Manager; the components and resources managed by the Configuration Manager make up the broker domain. The Configuration Manager maintains the broker domain configuration in the configuration repository. The Control Center is used to create and modify this configuration.

The Configuration Manager is the main component of the MQSeries Integrator runtime environment and serves three main functions:

- It maintains configuration details in the configuration repository. This is a set of database tables that provide a central record of the broker domain components.
- It manages both the initialization and deployment of the broker and message processing operations in response to functions performed using the Control Center.
- It checks the authorities of defined user IDs for the authority to initiate those actions.

The Configuration Manager provides services to the other broker domain components, providing configuration updates in response to Control Center actions.

The Configuration Manager requires:

- A set of tables in a database, known as the configuration repository. This database must be created using DB2 Universal Database for Windows NT. The Configuration Manager uses a Java Database Connectivity (JDBC) connection to this database.
- A set of tables in a database known as the *message repository*. This database must be created using DB2 Universal Database for Windows NT. The Configuration Manager uses an Open Database Connectivity (ODBC) connection to this database.
- A set of fixed name queues defined on the MQSeries Queue Manager that hosts the Configuration Manager. This MQSeries Queue Manager must exist on the same physical system as the Configuration Manager and is identified at the time the Configuration Manager is created.

These queues are created when the `mqsicreateconfigmgr` (including associated variables) command is executed through either the GUI or the

command prompt. No actions are required by the administrator to add the required definitions.

- A server connection defined to the MQSeries Queue Manager that hosts the Configuration Manager. This connection is used by all instances of the Control Center that communicate with the Configuration Manager.
- Sender and receiver channels to each broker in the broker domain except for a broker that would share its queue manager with the Configuration Manager.

### 8.17.2 The Control Center

The Control Center is used to configure and control the broker domain. This works in association with the Configuration Manager, passing messages back and forth as information is requested and making updates to the components.

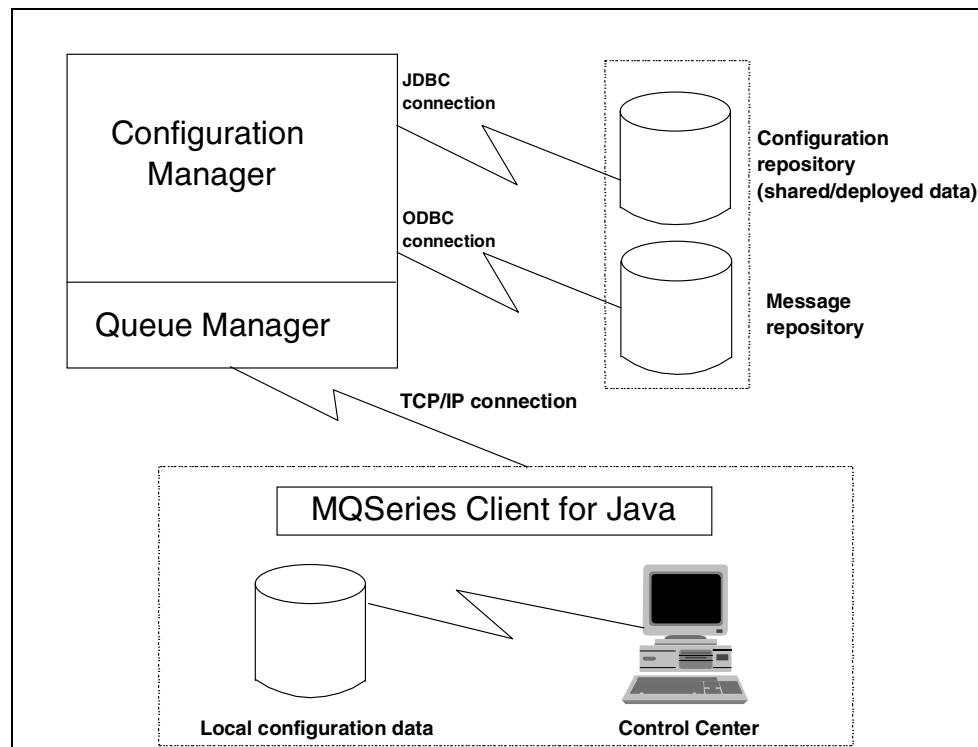


Figure 48. Relationship between the Control Center and Configuration Manager

Any number of Control Center instances can be installed and invoked. The Control Center depends on MQSeries for Java for its connection with the

Configuration Manager. The Control Center may be installed on the same physical system as the Configuration Manager or on any system that can connect to the Configuration Manager. The Control Center dynamically creates a client connection to connect to the Configuration Manager's queue manager using the information provided at invocation.

The Control Center is structured to provide a series of views on the configuration and message repositories. Views are selected by choosing one of five roles. All Roles shows every view. Other roles include message flow and message set developer, message flow and message set assigner, operational domain controller, and topic security administrator.

The Control Center is the business administration tool used to define message flows for applications and to access the databases and resources used by MQSeries Integrator. This interface allows the manipulation of the following resources:

- Message flows
- Message sets
- Brokers
- Collectives
- Principals

These resources are stored in the configuration repository database and in the message repository database.

The Control Center can be used to do the following:

- Develop, modify, assign, and deploy message flows
- Develop, modify, assign, and deploy message sets
- Define the broker domain topology and create collectives
- Create and modify access control lists (ACLs) to control publish/subscribe security
- View status information

The Control Center manages the configuration data for a particular workspace. There are three different versions of this data. When the Control Center is started, the *local version* of the configuration data is presented. The three presentations are:

**Local**      A copy of configuration data on which a user is working. To obtain a local copy of configuration data, you need to *check out* the resources from the shared copy. While *checked out*, other users are prevented from updating the resource.

Any changes made to a local version will not be visible to other users until the status of the resource is changed by *checking in* the resource.

**Shared** A version of the configuration data that is shared by all the users of the Control Center. Once resources are checked out to the local workspace; they can be modified and, once modified, checked in.

**Deployed** This is the active version of configuration data that is operational at the broker.

### 8.17.3 The Message Broker

The Message Broker consists of a number of processes. Figure 49 shows an overview of a broker.

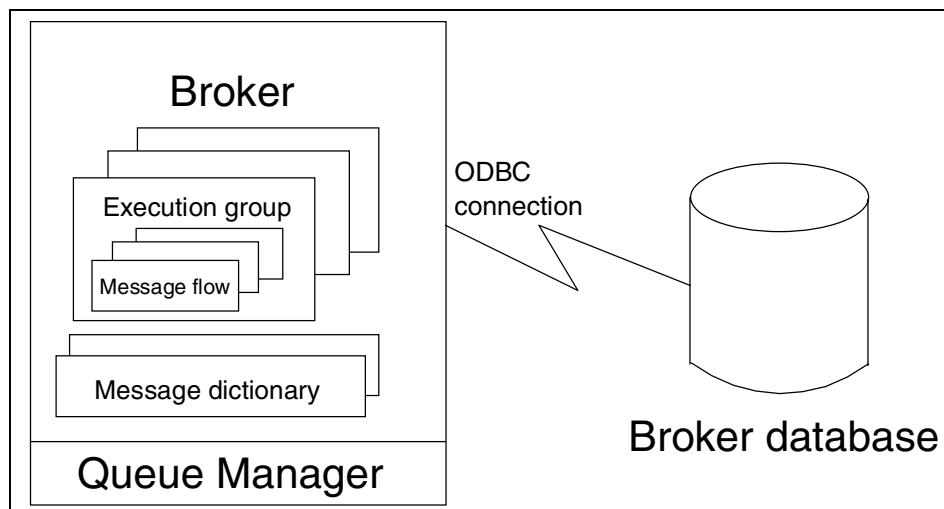


Figure 49. Overview of a broker

For each execution group assigned to a broker, the broker creates an additional data flow engine. As explained earlier, an execution group is a way of grouping message flows.

Any number of brokers can be added within a broker domain. More than one broker may reside on a physical system. Each broker defined will require a unique MQSeries Queue Manager. A single broker can share the MQSeries Queue Manager of either or both the Configuration Manager or User Name Server.

The broker is made up of a number of other components:

- Controller
- Message Format Services. These support definitions of metadata that describe the data format within messages.
- Persistent State Datastore. The broker has a mandatory database to hold all persistent state data needed by the broker. This may include:
  - The deployed message flow definitions
  - Persistent subscriptions
  - Publish/subscribe neighbors

Each broker requires:

- A set of tables in a database to hold the broker's local data. This is accessed using the ODBC connection. These tables are also referred to as the broker's *local persistent store*.
- A set of named queues on the queue manager associated with the broker domain.
  - Again, no actions are required by the administrator to add these definitions. They are added during execution of the `mqsicreatebroker` command.
- Each broker needs its own queue manager. It can share the queue manager hosting either or both the Configuration Manager and optional User Name Server.
  - Since the broker uses a set of predetermined queue names, this creates the dependency of an MQSeries Queue Manager per broker.

Creating the broker on the system on which the broker component is installed is not automatically recorded in the configuration repository. The Control Center must be used to create the reference. Creating a reference does the following:

- Stores the broker information in the configuration repository.
- Defines a default execution group on this broker. Further execution groups can then be defined.
- After creating a broker reference, it needs to be deployed to take effect. The deploy action:
  - Initiates communication between the Configuration Manager and this broker.
  - Initializes the broker so that it is ready to execute message flows.

- For operation, a broker requires both configuration and initialization data. *Configuration* and *initialization data* are logically separate and are stored in different physical repositories.

Configuration data defines the broker's operational setup. The *master copy* is stored in the Configuration Manager, which is centrally-managed. It is also cached to facilitate fast restart and make sure the broker continues to operate if a connection to the Configuration Manager is lost.

Initialization data defines the bootstrap parameters needed by the broker and is physically stored in the registry of Windows NT. Broker initialization data is made up of the following:

- **Filepath** - The root of the file system where MQSeries Integrator is installed
- **WorkPath** - The location of a work directory for MQSeries Integrator
- **DataSourceName** - The datasource name (ODBC) for the broker's local persistent storage
- **DataSourceUserID** - The user name used for accessing the broker database
- **DataSourcePassword** - The password used for accessing the broker database
- **QueueManagerName** - The name of the queue manager associated with this broker
- **InternalQueueName** - An internal queue used for configuration purposes
- **Language** - An identifier for help and message texts
- **Version** - A version number

The data is initialized during installation of the broker and may be modified as a function of broker administration.

Each broker instance needs an assigned, permanent, and fixed name. This is the *Broker Instance* name. This name, which is similar to the static identifier assigned to databases before they are created, is used to distinguish tables pertaining to one broker from another where multiple brokers have been set up using the same database.

#### **8.17.3.1 Persistent store**

The broker (as opposed to the Configuration Manager) has a mandatory database that holds all persistent state data needed by the broker. The state data includes the following:

- A local cache of broker configurations stored in an optimized format to facilitate quick restart
- The operational state of the broker, which message flows are enabled, and so on
- A table of *durable subscriptions*
- A table of *retained publications*

In order to manage this state, the broker requires a persistent store. The persistent store is a relational database that the broker accesses using standard JDBC/ODBC interfaces.

#### 8.17.4 Controller

The *controller* is the main control process for the broker. Its purpose is to monitor a table of broker *definitions* to ensure that the defined broker processes (and only those processes) are actually running. To do this, the controller needs to notice changes to both the table and process state and make the process state reflect the table.

When the controller starts, it will create an internal *cached* broker definition table by loading information from the broker definition table, which is from the persistent store. This allows the controller to determine which child processes it needs to spawn, one for each *Execution Group required* and one for the *administration agent*, which is always created and kept active.

If any one child process dies, the controller process will check the internal definition table for this process and perform the action indicated in that entry, usually to restart the process. In addition, the controller will start a thread that is responsible for waiting for the broker definition table to change. When this happens, the controller process reads the whole broker definition table and compares it to the internal broker definition table. Any differences will then be resolved by starting or stopping the child processes.

There will be only one controller process in a correctly-installed system, and only that controller process ever starts other broker processes.

If the controller process fails and is restarted, it will look for orphaned *administrative agent* and *MessageFlowEngine Processes*. If any are found, the controller will attempt to kill them before starting its own child processes. If it is unable to kill them, the controller will start up its own children and report on those it could not kill.

On Windows NT, the controller runs as a Windows NT Service.

#### 8.17.4.1 Administrative agent

The *administrative agent* manages the updates to the broker definition table. It will monitor the broker's configuration and administrative queues and process the corresponding messages it receives to create, delete, or modify an entry in the table and to start and stop the broker.

It can run as a separate process from the controller. This is so the reliability of the monitor will not be compromised. It is permissible to run as one process, but the robustness of the system will be weaker.

Configuration messages are routed to either the controller to start or stop execution engines or to a specific execution engine.

#### 8.17.4.2 Message flow execution engine

A *message flow execution engine* is an environment supporting execution of business message flows. The execution engines are started and managed by the controller, with each typically running its own process.

The broker installation process stores the code that implements the message flow nodes as *loadable implementation libraries* (\*.lil) in the broker's executable directory.

When the message flow execution engine starts up, it will initialize itself and then load all the lils it can find in the bin directory. This implies that all message flows on a machine can have the same capability regarding node and message classes.

When such a message flow execution engine is started for the first time, it will have a default configuration that is sufficient for it to be able to respond to configuration messages. When the *administrative agent* receives a configuration request for a particular execution engine, it passes the request to the engine causing it to update its configuration as per the instructions in the message. The configuration updates have the following properties:

- All configuration changes are made transactionally; either all changes in one configuration message are made or none of them are made.
- All configuration changes are made immediately and safely; message processing ceases just before changes are made and resumes with changed behavior immediately afterwards.
- All configuration changes are made persistently; once changes are made, restarting a message flow (or machine) will automatically configure itself to its last known state.



- The configuration messages may themselves be processed under transactional control so that either several brokers get a corresponding set of changes or none of them get any changes.

### 8.17.5 The User Name Server

Figure 50 shows the role and place of the User Name Server in the MQSeries Integrator security hierarchy.

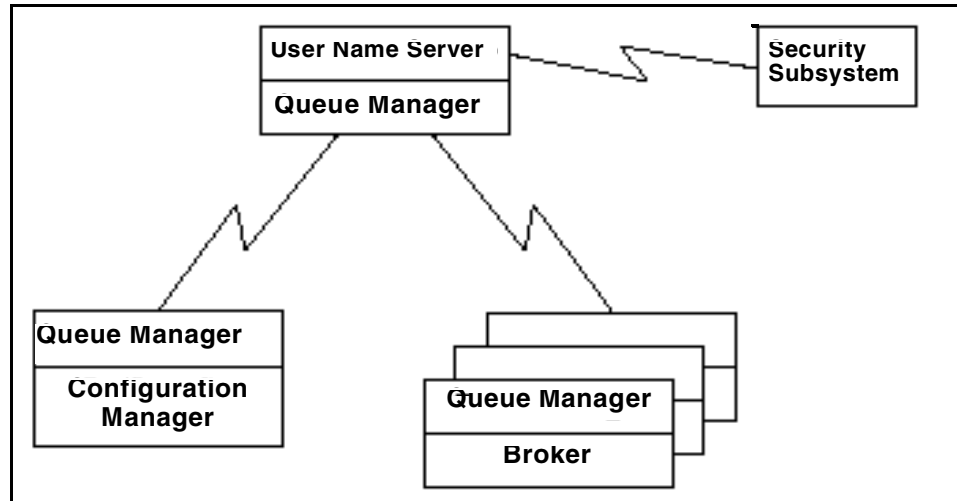


Figure 50. Role of the User Name Server

This is an optional resource. If no use of publish/subscribe is planned in the broker(s), no topic security is required, and neither is the User Name Server. It is much easier to include in the broker domain when it is first designed rather than add at some later date when requirements change.

The User Name Server monitors the underlying security subsystem, accessed via the Windows NT User Manager, and provides information about users and groups that it shares with the Configuration Manager and brokers.

Message flows that give publish/subscribe service to applications might require topic security. This gives the ability to control the authority of the application, based on the user ID they are running under, to do the following:

- Publish on topics
- Subscribe to topics
- Request persistent delivery of messages

Creation of the User Name Server defines the following resources:

- A set of fixed name queues defined on the MQSeries Queue Manager hosting the User Name Server and identified during creation. This MQSeries Queue Manager may be shared with the Configuration Manager, broker, or both.

### 8.17.6 Security subsystem

The Windows NT User Manager is populated with five new MQSeries Integrator groups to which connection is required to enable users to use various features and functions of MQSeries Integrator. If authorized users select their required role in *user preferences* while using the Control Center, they will be able to see and select the tabs required to perform those roles.

Further security is possible for the publish/subscribe function. For this further security to be available, the optional User Name Server must be created.

### 8.17.7 Databases

The MQSI components use databases to store configuration and operational information. These databases are used to provide a persistent store.

The Configuration Manager requires two sets of tables to support both the message repository and the configuration repository. These are created and initialized when the Configuration Manager is created and can be in a single database or across two separate databases. Access can be by a remote or local connection.

Normally, three databases will be created, one each for Configuration Manager, message repository, and broker.

The broker needs a set of database tables to manage its operation. These are created within the same database as the configuration repository or message repository or both. Tables for each broker can be in the same database as tables belonging to another broker or a separate database. Access can be by either a remote or local connection.

### 8.17.8 Dependencies

MQSeries Integrator has several dependencies. MQSeries Integrator is heavily dependent on the facilities of MQSeries messaging to provide connectivity, message integrity, and some transactional support. A summary follows to help show the demands that MQSeries Integrator will place on the system:

- **Queue managers** - A single MQSeries Queue Manager can host only one broker. The Configuration Manager and User Name Server both depend on an MQSeries Queue Manager but can share with each other and the broker, or both.
- **Communications** - A network of MQSeries Queue Managers is required to support MQSeries Integrator, and the connectivity must be defined. Any supported communication protocol may be used. Windows NT supports:
  - SNA LU6.2
  - SPX
  - NetBIOS
  - TCP/IP

The client/server connection between the Control Center and the Configuration Manager is limited to using TCP/IP.

- **Configuration Manager** - The Configuration Manager depends on an MQSeries Queue Manager with a set of fixed named queues and a server connection channel. The queues and channel resources are defined during creation of the Configuration Manager.

A sender and receiver channel pair are also required to be able to communicate with each broker in the broker domain, except for the broker (if so defined) created on the same host MQSeries Queue Manager.

- **Broker** - Each broker depends on a dedicated MQSeries Queue Manager. A broker may not share it with another broker. A broker may, however, share the MQSeries Queue Manager with either the Configuration Manager or User Name Server or both.

A set of fixed name queues that are defined at broker creation are required, as are sender and receiver channels to allow communication with the Configuration Manager. Further sender and receiver channels are also required to allow communication with the User Name Server.

A sender and receiver channel pair are also required to communicate with all brokers in the same collective, or to which it is identified as a neighbor in the topology.

- **Application**. Each application using MQSeries Integrator must be able to connect to an MQSeries Queue Manager in the MQSeries network to allow it to place messages on the queue serviced by the message flow required.

Each application retrieving messages from an application from a queue written to by message flow must be able to connect to the queue manager that owns the queue.

If the application retrieving messages is a subscriber to a publish/subscribe service, the messages it receives are propagated to the broker to which it has subscribed, regardless of the proximity of the broker and associated queue manager hosting the publish/subscribe service.

- **User Name Server** - The User Name Server depends on an MQSeries Queue Manager with a set of fixed named queues. Sender and receiver channel pairs are also required for communication with the Configuration Manager and every broker in the broker domain to which it provides principal definitions. The queues are defined during the creation of the User Name Server.

- **Databases** - MQSeries Integrator components use databases to store configuration and operational data. A summary follows:

- The Configuration Manager needs two independent sets of tables to support the message repository and configuration repository.

The tables are created and initialized during Configuration Manager creation. The two repositories can share the same database, which must be DB2. The configuration must have a remote or local connection to the database or databases.

- Each broker needs access to a set of tables to support its operation.

These tables are created and initialized during broker creation. They can be created in and can share the same DB2 database as the configuration repository and/or message repository.

Subsequently-defined brokers can share the same tables since each entry on each table row identifies an individual broker. If preferred, separate databases, and, thus, separate tables, can be set up for each broker.

The broker requires either a local or remote connection to the database.

### 8.17.9 Message domains, message sets, message types

MQSeries Integrator makes a first distinction between self-defined messages and predefined messages. Self-defined messages use the XML standard to structure their content. We will also refer to these types of messages as *generic XML messages*. To use self-defined messages, you do not have to do anything specific.

MQSeries Integrator considers two views of a message: The logical view and the physical view. The logical view of a message is the structure of a

message: Its fields, the order, and the relationship between the fields. A message might have four fields:

1. Employee number
2. Request type
3. Order number
4. Work department

Applications receiving and sending these kinds of messages know the meaning of each field and its characteristics. They know that, for example, employee number, request type, and order number are six-character fields and that work department is a 3-byte character field.

The physical view of the message, also called the wire format, is the sequence of bits that build up the message. For the above message, the physical view is as follows:

```
000110NEW 999999AAA
```

For predefined messages, you need to make MQSeries Integrator aware of the logical view and the physical view of the message. For this purpose, you use the Control Center. If you have already defined some messages using the MQSeries Integrator Version 1 product, you can use the NEONFormatter to maintain or add predefined messages.

Knowing all this, we can specify what the message domain is. It identifies, for the broker, where to look for the actual definition of a message:

1. XML for self-defining messages
2. MRM for messages defined using the Control Center
3. NEON for messages defined using the NEONFormatter
4. BLOB for messages that have no definition

A message type is the definition of the logical view of a message. Message types are grouped in message sets. You can think of a message set as the collection of message types used in a single application or project.

The last characterization of a message is the message format. This describes the physical view of a message, that is, the sequence of bits that make up the message. The message format can have three values: XML, PDF, and CWF. Note that PDF does not stand for Portable Document Format. It is a message format that is used in some financial applications. Custom Wire Format

(CWF) is a format using data types in common programming languages. The XML format is for messages that comply with a DTD.

---

## 8.18 XML and MQSeries

When we think about applications, the major points to consider are how they handle the following:

1. Data storage
2. Data sharing
3. Data transformation

XML can be used to store data. The availability of standard parsers makes it easy to read and write data in an XML format, which speeds up development time. Another advantage is that the format to store the data is an open format instead of a proprietary format. Given that your data storage is now based on open standards, the same format can be used to share the data with different systems. This allows for easier integration. You no longer need proprietary import/export functions to allow your data to be shared with other applications. Even if the application continues to store its data in a proprietary format (think about database servers), XML is still appealing as the standard format for exporting and importing data to and from other sources. These other sources can be computer systems within an enterprise or the systems for your business partners. The ability to link computer systems together is a key focus in many industries, and, given its open nature, XML can help here. Thus, XML is also the language for data transmission.

### 8.18.1 Importance for the MQSeries family

Given the above examples of possible uses of XML, the importance of XML to MQSeries is immediately clear. The sharing and transmission of data between different computer systems is exactly the key mission of the base MQSeries product. MQSeries allows you to distribute data across more than 35 different platforms with assured delivery. The publish/subscribe extension makes it possible to share data with as many applications as you like. The publisher (source of information) has no link with the subscriber (the consumer) of that information. MQSeries makes sure that the right data arrives at the right place independently of platform, network protocol, or the availability of systems. The nice thing is that nothing needs to be done to have MQSeries work with XML messages. For MQSeries, an XML-formatted message is just another message.

But, that is not the end for the MQSeries family. Although the acceptance of XML is growing fast, you will still have much data in proprietary formats for a long time. Nobody will change a key application in their enterprise just to make use of XML, and, even if you had the resources, there are situations where you are not in control of the behavior of an application. Think about all the packaged applications with their private import/export facilities. For all these types of data, you can work with MQSeries Integrator to transform the data into XML messages before delivering the data to XML-enabled applications, or vice-versa, transforming the XML messages into the proprietary formats.

Even between XML-enabled applications, MQSeries Integrator can play a key role. Like any technology, XML will evolve. Industry-specific XML languages will exist that are not compatible, and, over time, a specific industry might have a number of different versions of its XML standard. Again, MQSeries Integrator can help you transform the data into the correct format.

Finally, you can easily imagine an application that generates huge XML streams while the consumer of the data is only interested in part of the data. MQSeries Integrator can work here to filter out the necessary elements.

### **8.18.2 Use of XML within MQSeries Integrator**

Given all the advantages of XML that we showed earlier, it comes as no surprise that MQSeries Integrator itself makes heavy use of XML technology. We already saw that the Control Center of MQSeries Integrator is storing and sharing its data in an XML format. When you export or import message sets from your Configuration Manager, it is again stored in an XML format.

When externalizing data for system management products, MQSeries Integrator generates event messages in an XML format. One can imagine that, in the near future, this style of event messages will be used for the base MQSeries product instead of the current, proprietary PCF format.

Another form of data externalization is the user trace. The user trace is a very helpful component of MQSeries Integrator for a developer and tester of message flows. Because the data is internally managed in an XML format, it was the clear choice to use this format again for tracing purposes. Note, however, that a tool is available in MQSeries Integrator to reformat the XML-formatted user trace in a conventional text file. Thus, if you have no immediate access to an XML-capable viewer, such as Microsoft's Internet Explorer, you're still able to make use of the tracing facility.

We have already mentioned that a user can build nodes themselves for specific purposes. The way to configure MQSeries Integrator to use these customer-built nodes is again using XML-formatted configuration files.

While the above list is absolutely not intended to be a complete list, it is already clear that XML technology is a core part of MQSeries Integrator and that its use can only grow over time in the MQSeries Family.



---

## Part 2. B2B integration guidelines

The intent of this part of the book is to provide information or a pointer to information about development, deployment, systems management, and deployment guidelines.

Designing an e-business solution is not an exact science; so, no step-by-step checklists are provided, nor does this section attempt to be a comprehensive source of technical and product information. However, it does present a broadly-defined approach and pointers to more information.

Chapter 9, “Application design guidelines” on page 171, introduces consideration of the functional components of the application within the context of the runtime topologies.

Chapter 10, “Application development guidelines” on page 213, provides guidelines for application development, considering the roles, processes, and tools that are required.

Chapter 11, “Performance guidelines” on page 229, introduces performance guidelines by considering the components of the topologies under discussion that are particularly relevant to performance.

Chapter 12, “Systems management” on page 235, looks at the management of MQSeries and related components.



---

## Chapter 9. Application design guidelines

Business-to-Business Integration Patterns address the interaction of business processes between organizations. As mentioned before, it can be viewed as taking enterprise application integration one step further – the integration of applications between businesses.

There could be a number of approaches to any given EAI solution within an Enterprise. Integrating across enterprises adds additional requirements to any solution.

With topologies two and three, an enterprise could choose to provide a shared middleware interface to partners or a more generalized/open interface. See Section 4.3, “Topology 2 - Direct with adapter/bridge” on page 46, and Section 4.4, “Topology 3 - Message broker” on page 51, for more details.

Both solutions have their advantages as well as disadvantages. A shared middleware approach is easier to design and would probably be quicker to implement. A generalized interface may need to support multiple protocols and interfaces and would, therefore, have a longer development cycle but would provide greater flexibility. There are many more considerations. The selected solution would depend on the relationship between the partners, the integration requirements, and so on.

This chapter describes the design guidelines for implementing runtime topologies 2 and 3.

---

### 9.1 Application elements

The design guidelines outlined here primarily focus on Business-to-Business Integration applications. Before exploring these guidelines, it is important to understand the overall structure of these types of Web applications. Chapter 4, “Choosing the runtime topology” on page 41, presents the overall topology of such an application and explains the responsibilities of various nodes in the topology. Chapter 5, “Technology options” on page 67, presents various technology options available for implementing Business-to-Business Integration as well as some recommendations and considerations.

As mentioned previously, B2Bi can be viewed as doing EAI between businesses. This chapter will not focus specifically on the EAI elements, but on the elements that cross the business boundaries.

This chapter assumes the use of MQ and MQSI to be the products to be mapped to the message queue and message broker components within topologies 2 and 3. Figure 51 represents the runtime topology for application topology three.

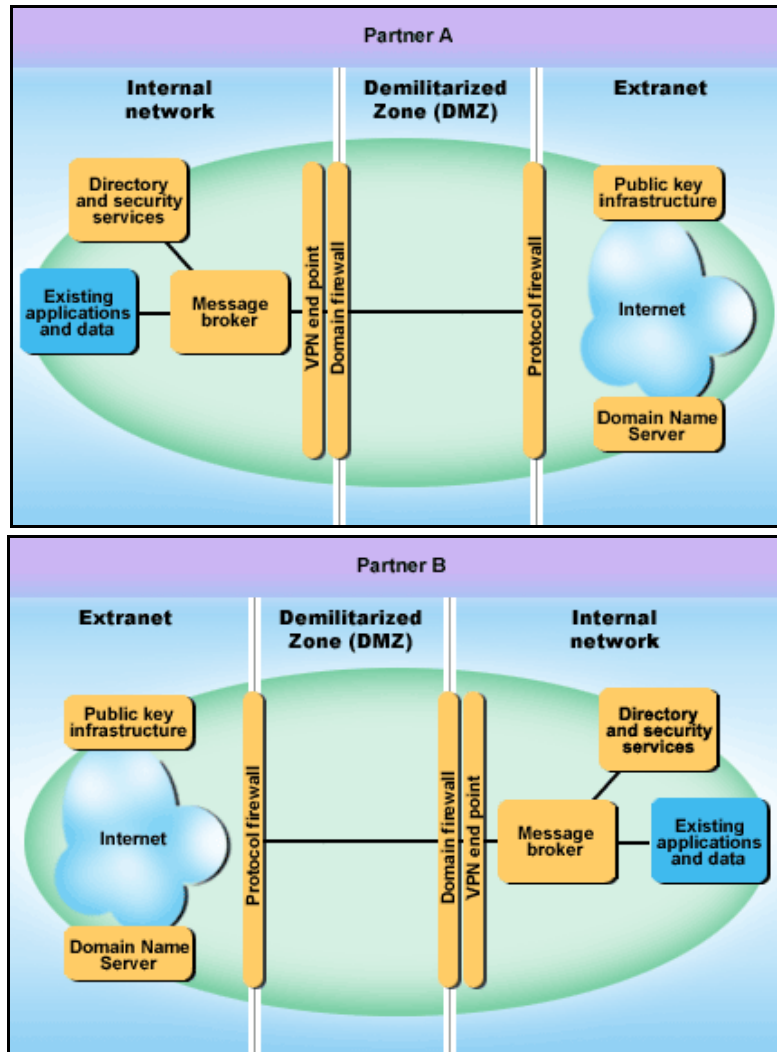


Figure 51. Application topology 3: Runtime topology

The symmetrical diagram represents one possible solution and assumes, in this case, that the partner uses some shared middleware. It is possible to make a more generic interface available in which case the partner could select alternate software.

Figure 52 focuses on the elements of communication between applications.

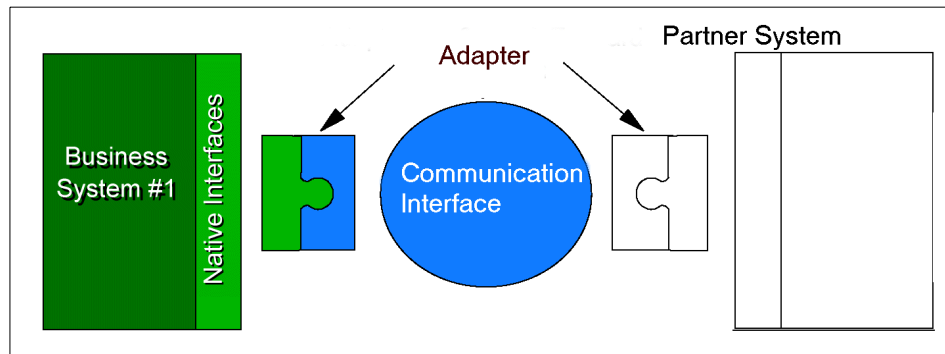


Figure 52. Application to application communication

In Figure 52, the partner could select to communicate using the same adapter as Business System #1, or the partner could select to use another medium of communication.

In the sections that follow, we will describe the following key elements of the interaction:

- Using synchronous and asynchronous communications
- General principles for distributed systems
- Connecting to the communication interface
- Resolving multiple backend systems using the hub and spoke architecture

The final sections of this chapter describe how MQSeries and MQSeries Integrator may be used for topology 2 and 3. See Section 4.3, “Topology 2 - Direct with adapter/bridge” on page 46, and Section 4.4, “Topology 3 - Message broker” on page 51, for more details.

---

## 9.2 Communicating between applications

Irrespective of whether the shared middleware or a generalized approach is taken, communication would be between the application servers in each enterprise.

Application servers can be connected using either synchronous or asynchronous communication methods.

### 9.2.1 Synchronous communication

Synchronous communication is like two people having a telephone conversation; both sides are available to communicate (talk) and do so until a satisfactory conclusion is reached.

Systems that communicate synchronously have two modes for the communication. The first is like the telephone conversation: One side will call the other and wait for a reply; then, each side takes it in turn to communicate (in communications, this is called half-duplex). The other mode, called full-duplex, is where both sides communicate simultaneously, something not possible for human phone calls! From the perspective of a single application process it is also not possible to communicate in full-duplex. Systems that achieve this have a pair of application processes functioning in half-duplex mode. So, synchronous communication is actually made up of one to many *request and wait for reply* pairs.

### 9.2.2 Asynchronous communication

Asynchronous communication is like the case of calling someone on the telephone and getting their voice mail system. In this case, you have to leave a message if you want to communicate, and you make an assumption that the owner of the voice mail will actually check that they have a message and then answer it. There is also the assumption that we have provided enough information to answer it!

While the analogy of asynchronous communication describes a model of asynchronous communication, it is not quite good enough for robust communication. The reason is that in order to leave a message, you have to be able to talk to the voice mail system. But, what if that system is unavailable?

Ideally, asynchronous communication should provide a way to leave the message in the sending system and have that message forwarded when the other system is available. This capability is called Store and Forward without having to wait for a reply.

### 9.2.3 Synchronous and asynchronous communication

In connecting application servers together the Store and Forward capability is ALWAYS required unless the systems being integrated have identical availability characteristics. So, in the phone example, store and forward (voice mail) is required unless the two parties are always available to take calls with one another at exactly the same times of the day. In practice, this is not possible unless unrealistic restrictive arrangements are made.

So, for practical purposes, store and forward is always required when connecting application servers of different business systems. For application servers in the same business system, where there is one tightly-controlling organization, it may be possible (although unlikely) to do without store and forward.

Therefore, synchronous communication is request and wait for a reply, and asynchronous communication is store and forward without having to wait for a reply.

Suppose that in the store and forward case, the next thing is to actually wait for a reply from the opposite system's store and forward mechanism. What is the difference between that and synchronous communication? From a programming style point of view, there is no difference since this provides the request and wait for a reply model.

So, why do you need synchronous communication at all?

There is one fundamental difference between the two models and that difference has everything to do with transactional integrity during the update of data. You should, therefore, note that in applications that query information with no updates, there is no difference between synchronous and asynchronous for request and wait for reply requirements.

So, what about update of information?

#### **9.2.4 Comparison in a two-system update**

We will use an example of communicating between two systems where the objective is to update a database in each of the systems. We will add the further requirement that both systems be updated or that both systems not be updated.

To achieve the goal, the first system must receive a request of some sort. It will interpret the request and update a local database as well as send a request to the second system asking for it to update its database. You need some way to make sure that both updates have occurred and then respond to the requester that we have finished.

In the case of pure synchronous communication, the most robust way of achieving the goal is to use a distributed transaction with a two phase commit of the two databases. This will produce a flow of communication that would be as shown in Figure 53 on page 176 (assuming that you are successful in your updates):

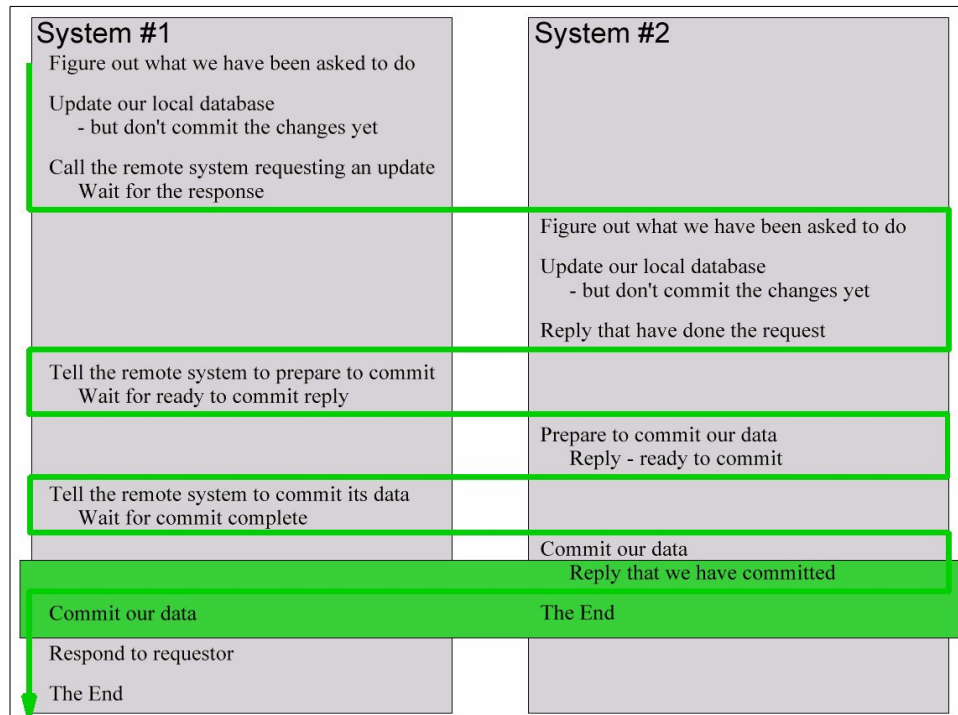


Figure 53. Synch communication:Distributed transaction with two phase commit

The key thing about this flow is that it is all one transaction, and the actual commitment of the data occurs right at the end. Up to the point where processing enters the shaded box, if a failure of some kind occurred, all of the updates can be backed out. The shaded box represents the window of uncertainty that always exists when you update two separate systems. This approach really minimizes the duration of this window, but, due to the fact that communication has to obey the laws of physics, in particular, that communication is not instantaneous due to the maximum speed being that of light. Therefore, there must be a delay between the actual commitment of the update in system #2 and the acknowledgment of that commitment reaching system #1. In that delay period, a failure in the network may mean that system #2 has actually committed the data, but system #1 has not seen the reply. System #1 now has a problem in how to decide whether to commit or rollback its own updates.

In an ideal system, system #1 would wait, holding all the pending updates, for the network to come back and then decide what to do. However, there is no way for system #1 to know when this might happen, and it is usually not



practical to hold database update locks indefinitely because other applications will need the data. As a result, system #1 is usually designed to take either the commit or the rollback option when the failure is detected. The most common choice is to roll back the updates; this is referred to as a presumed abort commitment strategy. Of course, system #2 may have committed, in which case, we have inconsistent data, namely, system #2 updated, system #1 not updated. A very unlikely outcome, but possible.

So, with synchronous communication, you ensure that both systems are updated by doing all the update commitment as part of one transaction. In the event of failure, our transaction manager (provided by our application server) can issue a rollback request to all the datastores involved.

So, what about the asynchronous alternative? In Figure 54, the asynchronous flow is shown, and you have an additional system to think about, namely, the store and forward system.

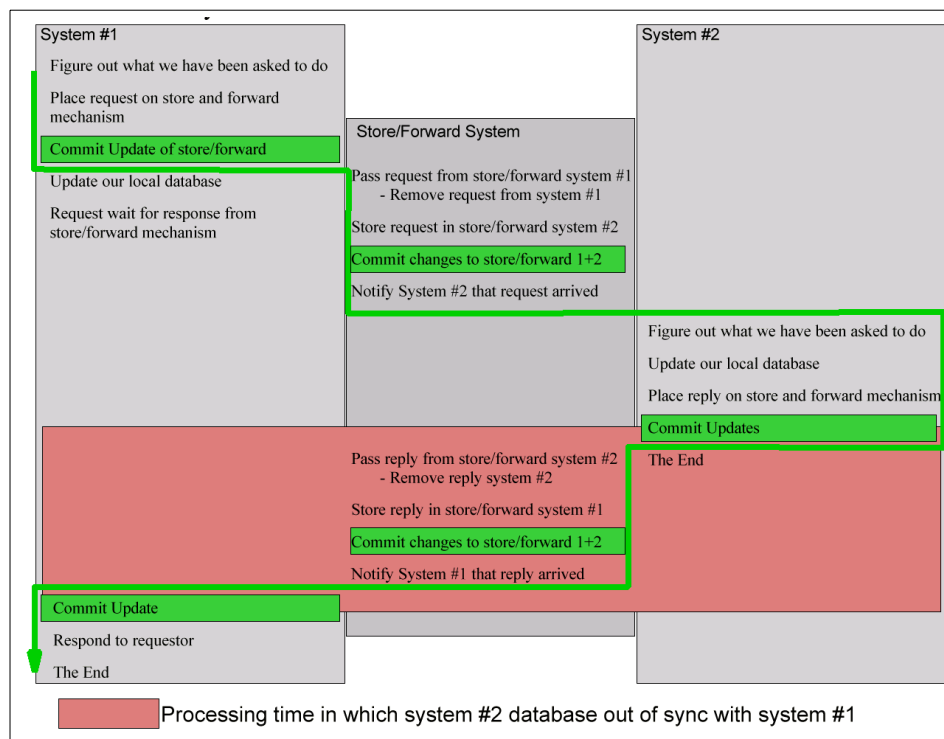


Figure 54. Asynch communication:Distributed transaction with two phase commit

While the end result of the flow is the same as in the synchronous case, the state of the data along the way is very different.

In order to ensure that an update is performed on the second system, the store and forward mechanism must make sure that the information is passed from system #1 to system #2 and that it is passed only once. You do not want to do two updates! To do this, the store and forward system must have a transactional capability for passing the request from one system to the other, so that it is either on the first system or on the second system. What cannot be allowed is for the request to end up on both systems or neither of the systems.

The flow shown makes an asynchronous request of system #2 before updating its own database. This allows you to hold an update lock on system #1 while the system #2 update is being performed. This is to minimize the time that the two systems will be out of synchronization. The update could be done before making the request. This could be done where the design of the application is such that the time between the updates of the two systems is not critical - as long as both eventually happen!

It should also be noted that five transactions are executed in this process versus the one transaction in the purely synchronous case. Therefore, one might conclude that it will take more processing time with this asynchronous alternative. This is true when you are trying to limit the time difference in updating the two systems. If you are not concerned about that, from System #1's point of view, the asynchronous approach is actually faster because you don't have to wait for System #2 to complete its update.

So, the trade-off in asynchronous versus synchronous update is as follows:

#### Asynchronous

- Complete separation of the availability profiles of the two systems.
- Faster update of first system if you are not concerned about the time difference in the two updates.
- Systems will be out of synchronization for a small period of time.

#### Synchronous

- Requires identical availability profiles of the two systems
- Faster update of both systems where an application is time-sensitive to synchronization
- Systems not out of synchronization at any time, but possibly in doubt

---

### 9.3 General principles

Figure 55 shows the principle in the design of distributed systems.

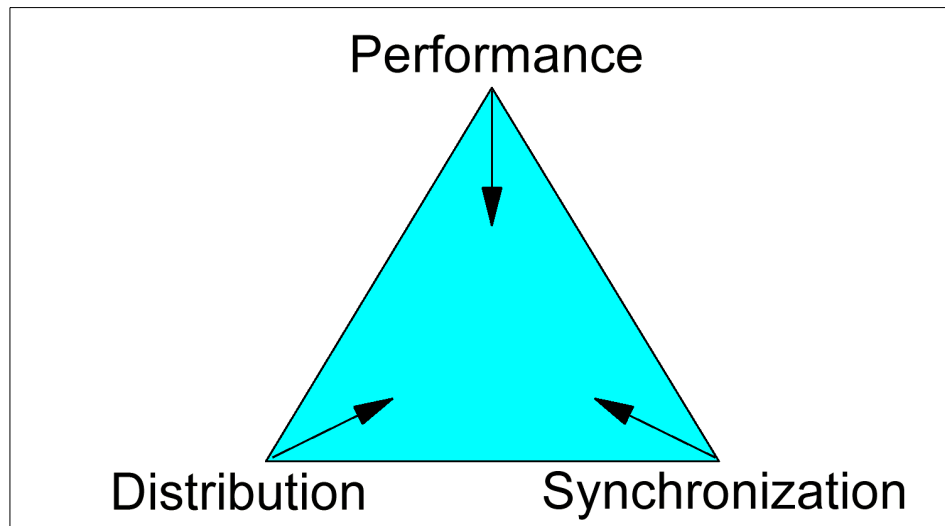


Figure 55. *Distributed Systems: General principles*

You can optimize your system anywhere within the triangle of Performance (how fast does your system run), Distribution (how far apart are you systems), and Synchronization (how close together in time are updates). The closer you get to each apex, the more optimal each aspect. In general, you can optimize for any two of the three, but not all three. No system optimizes for poor performance; so, you get to choose distribution versus synchronization. Business to Business communication is inherently distributed; so, there goes synchronization!

So, if you really want to design your applications so that the updates of the two systems do not have to be time synchronized, there is absolutely no requirement for synchronous update. The time synchronization requirement must come from the business requirements of the application. There are very few (if any) business processes that have an absolute time synchronization requirement!

The conclusion is that asynchronous intercommunication should always be used for integrating business systems (application servers). This applies a greater design burden on the application developers since they have to consider the impact of time synchronization in the asynchronous case. However, the run time benefits are considerable because you can now

operate your systems completely independently if required. This independence is almost always mandatory in business-to-business integration between separate businesses.

So, to integrate business systems, we need a store and forward capability to pass requests between the systems to be integrated. However, we have the question of how to connect the business system to the store and forward mechanism. This is particularly important where we want to integrate a system that we have purchased and do not have any source code for it!

---

## **9.4 Connecting to a store and forward mechanism**

There are two ways of connecting an application server, with its associated applications, to the store and forward mechanism: Invasive insertion of the chosen mechanism into the application and passive adaptation of an existing application interface.

Topologies two and three do not make any assumptions as to exactly which one would be used. Instead, it would depend largely on the application being integrated.

### **9.4.1 Invasive insertion**

This approach requires that you have access to the source code of the application; so, is generally not applicable to cases where a packaged application is purchased. It is also time consuming because you must retest the application to ensure that the changes made do not introduce errors into the application.

In general, this is used in cases where there is an established integration mechanism and a new application is being built that wants to participate in the integration.

### **9.4.2 Passive adaptation**

This is achieved through the use of an adapter of some type that makes use of a existing interface to the application and then bridges that interface to the chosen communication interface.

If the application already supports the chosen communication interface, the adapter is very straightforward and, really, just an extension of the application. However, adapters can also be very complex, particularly if the only way to access the application is through some sort of screen interface where we might have to build a “screen scraping” adapter.

In general, this approach is faster to build than the invasive one because you only have to test the adapter; as a consequence, it is the preferred approach. In fact, it can be argued that the invasive approach is really the creation of an adapter that is being inserted into the application code. The trade-off being that passive technique is usually simpler to build, whereas the invasive approach may well provide for more efficient performance. So, the conclusion is that adapters, either passive or invasive, are required to connect your applications to the chosen integration communication mechanism. This can be represented as shown in Figure 56.

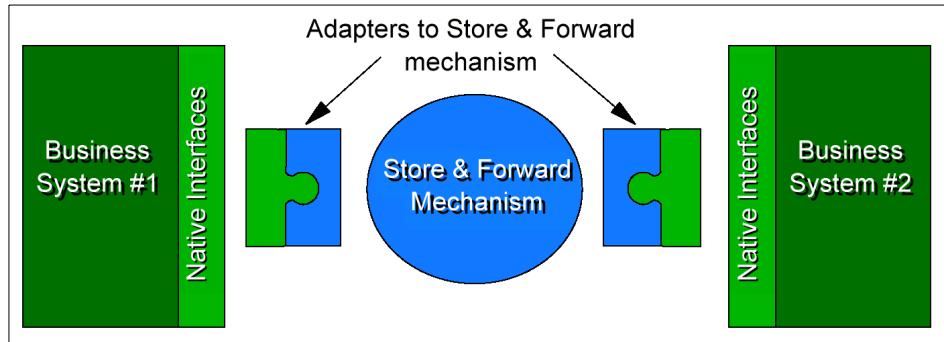


Figure 56. Placing the adapter

### 9.4.3 Placing the adapter

After you have your adapter, the question of where to place it arises.

You could, in theory, place the adapter on a system that is remote to the application if you have a native interface that can operate remotely. The benefit of this is that our store and forward mechanism can be restricted in its platform choice, and you can concentrate the skills required to build adapters on a small number of platforms. This apparent benefit is far outweighed by the drawbacks associated with the implied requirement to run multiple communication protocols to make the adapters work. This makes the operations side of the system complex and hard to monitor. In addition, there is the base assumption that we have an interface that is remotely accessible – in general the only remote interface that you can count on is the one that an end-user would use (and not always then!), which implies screen scraping – a relatively difficult approach to integration.

In practice, the better choice is to place the adapter on the same system as the application. This means you may be able to use one protocol that you can then monitor and manage much more easily. In the case of packaged applications, the application vendor may well provide an adapter for standard

communication mechanisms. These supplied adapters will always be on the application server side.

Of course, this means that your chosen store and forward mechanism must be available on all of the platforms you want to integrate. Ideally, your selection should be for the most pervasive store and forward mechanism. The industry's most pervasive store and forward mechanism is MQSeries.

#### 9.4.4 The role of the adapter

Therefore, the conclusion is that you want an adapter to connect the application to the store and forward mechanism. This connection needs to be very robust, particularly if you are going to provide updates to the application and need to ensure that they occur. Ideally, the adapter should allow for transactional update of the application where the request passed by the store and forward mechanism is consumed as part of the transaction. If you get a failure, the request is not consumed, and you can retry. This presents a major problem for application integration since the vast majority of passive adapters cannot participate within a transaction in the target application. If they can, there is no problem. If they cannot, a compromise design can be deployed in developing the adapter.

The required processing flow is shown below. The assumptions are that your store and forward mechanism has a transactional capability that the adapter can use and that our target system does not have an externalized transaction capability.



Figure 57. Process for a passive adapter

The processing occurs under the transactional scope of the Store and Forward mechanism ensuring that you cannot lose the request in the event of a system failure. The placement of the response on the store and forward mechanism represents successful completion of your request. It should be

noted that there is a window of opportunity between the completion of our *perform required processing* and the placement of a response on the store and forward mechanism where, in the event of a failure, recovery of the store and forward transaction will not allow you to determine if you completed the update on the business system. This window is extremely small but should be considered in the design of the system.

Therefore, the primary role of an adapter can be considered to be to make sure that it does what it's been asked to do and receive a response.

What else could the adapter do?

Clearly, under the mantra of *a simple matter of programming*, the adapter could do just about anything. However, the two areas most often considered are routing of the request in the store and forward mechanism and transforming the information gathered from the application into a form the requesting system requires. In the case of routing, the store and forward adapter clearly needs to be able to locate at least one other system or you are not going to get very far with your request! However, you could enable the adapter to locate all possible systems and allow it to communicate directly. This point-to-point connection allows for communication with the least network latency. In order to enable this, you would have to provide a directory for the adapter's use that contained the entire system topology that all system adapters could share. This approach has drawbacks, such as finding a directory accessible from all systems, the performance impact of such directory queries for routing, and the currency of the information if you cache to improve performance. These drawbacks can be solved by using robust distributed systems services, such as those found in DCE or lightweight Web-oriented services, such as LDAP-based directories.

However, there is one large question that cannot be resolved, and that is the number of connections produced by a point-to-point integration strategy.

Figure 58 on page 184 shows the number of connections for three, four, and ten directly-connected systems.

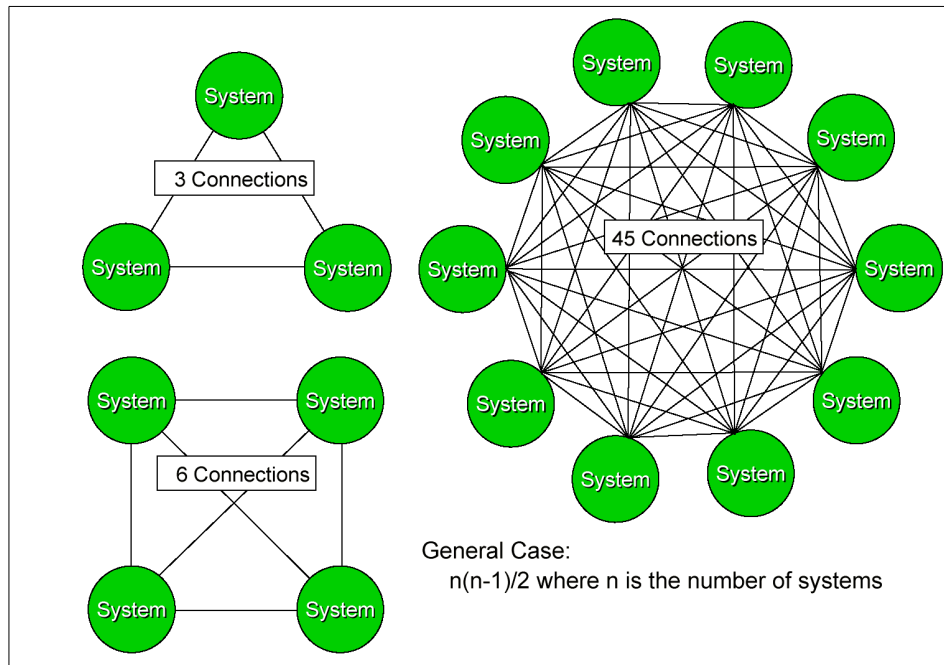


Figure 58. Number of connections for 3, 4 and 10 integrated systems

As the number of systems grows, the number of connections quickly becomes a management problem – one mainly associated with monitoring whether all connections are operational.

## 9.5 Hub and Spoke integration architecture

A better approach is to adopt a spoke and hub connection architecture where the systems connect to a central hub, which then makes the routing decision. The systems now only need to know where the central hub is (note that this information still has to be distributed to the adapters!) and have one connection each. The hub is in charge of making the routing decision.

Note that this is the same principle used in TCP/IP routing where TCP/IP clients know a router or gateway that then routes all traffic onwards.

Figure 59 on page 185 shows the number of connections in the spoke and hub architecture for the same number of systems shown in the directly-connected example.



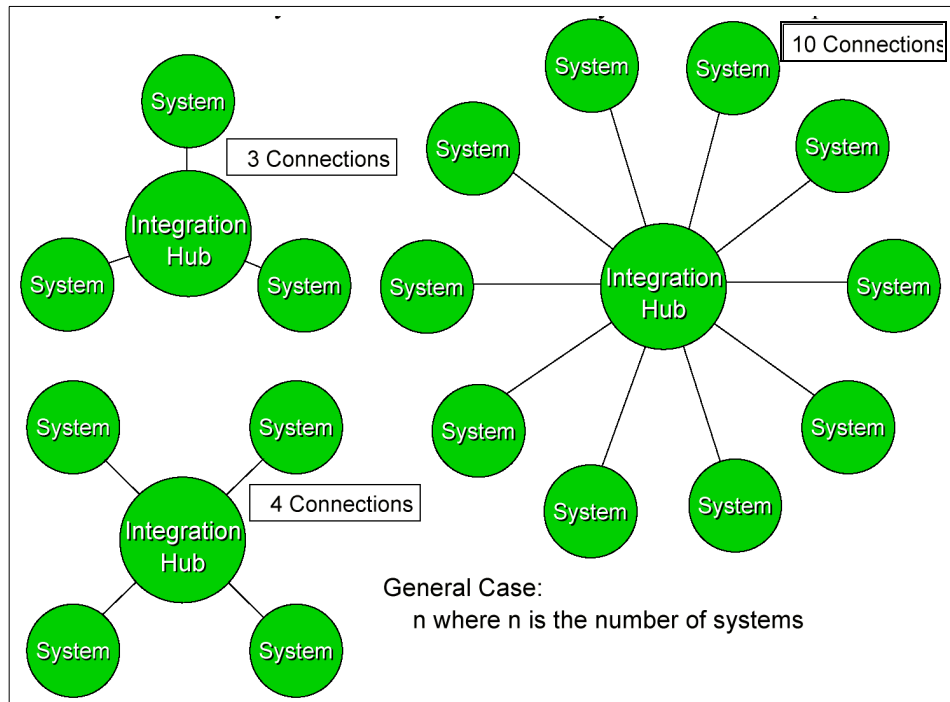


Figure 59. Number of Hub systems for 3, 4, and 10 integrated systems

A graphical comparison of the increasing complexity of the two approaches is shown in Figure 60 on page 186.

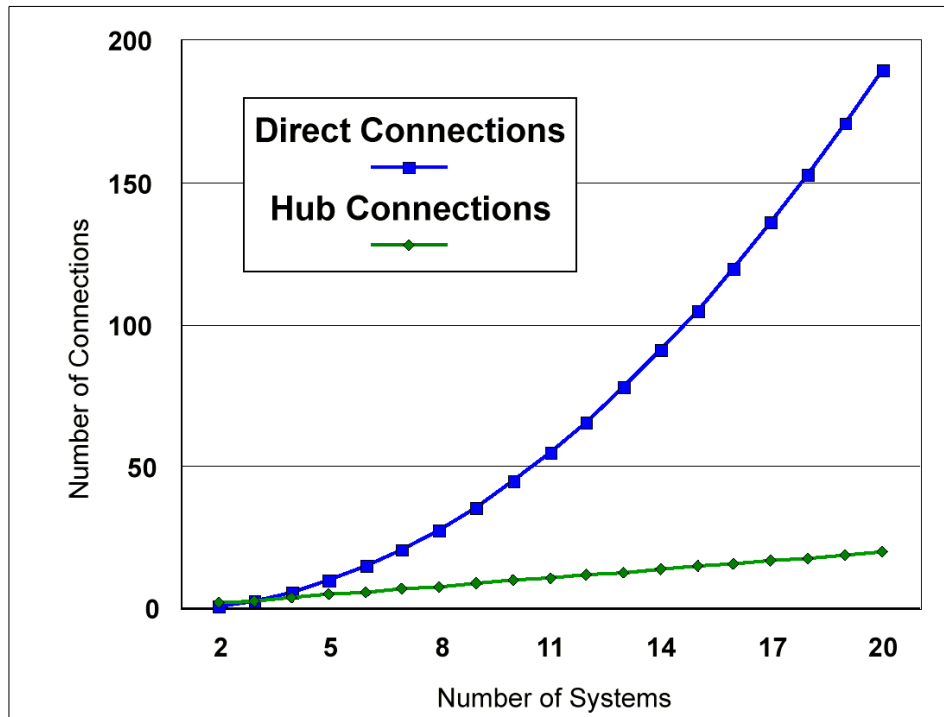


Figure 60. Complexity of Direct Connected Systems vs. Hub Connected Systems

Clearly the hub approach becomes increasingly simpler to manage as you increase the systems. The question is at what point should we consider the hub approach? After all, for two systems, you actually have more connections with the hub!

The answer to this question is FOUR systems.

Why four? Well, because you must look at the problem from the point of view of minimizing the number of points of failure. In any set of integrated systems, you have to maintain the number of systems plus the number of connections. In the direct connection approach, the number of failure points (FP) is the number of systems plus  $n(n-1)/2$  number of connections, represented as

$$FP = n + n(n-1)/2$$

where  $n$  is the number of systems.

In the hub connection approach, the number of failure points is the number of systems, plus one for the hub and the same number of connections as there are systems, represented as

$$FP = 2n + 1$$

where n is the number of systems.

The graphical representation of these is shown in Figure 61, and the cut over point is between 3 and 4 systems, thus, the number four.

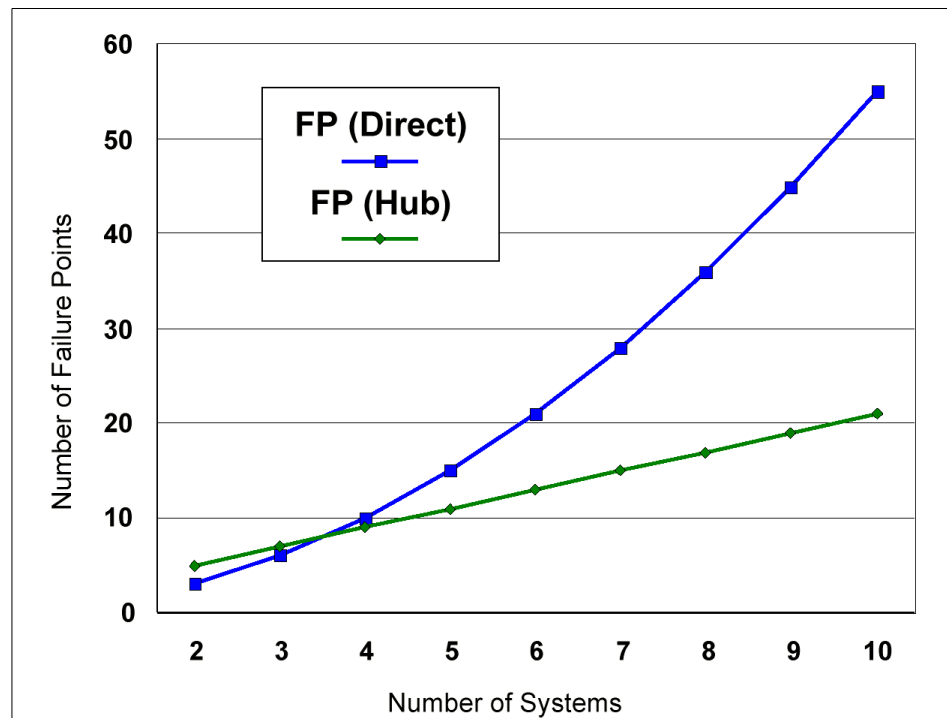


Figure 61. Comparison of direct connected systems vs hub connected systems

Now, many may think, “that is a much too simplistic approach to making that decision. What about the cost of the building the hub versus building the adapters?” Clearly, the economics of building the initial system do come into play, but the longer the system is in use the more money will be spent on monitoring and maintaining the system. Minimizing the complexity and associated failure points is the best way to control this expense and if you run the system long enough the cost benefit of the simpler system outweighs the potential build cost. The point at which this happens is very dependent on

how much money is spent in the building. However, if you have four systems, it could be argued that given enough time, it always happens; this is another reason for picking four!

Also, if you are integrating two systems and intend to add at least two more at some point in the future, you should also consider the hub approach to simplify your future development!

Once we have decided that routing belongs at the center, you need to consider the transformation of the information that one application will send to another one. In the ideal case, we would use a common form of information; however, just as in real life, everyone does not speak the same language. Systems tend not to have a common form. In addition, even if we agree on a common form, and XML appears to be a candidate, we still have to agree on the specific meaning of the phases. It is likely that transformation will be required for many years to come.

### 9.5.1 Where to do the transformation

We believe that transformation also belongs in the central hub because this allows for the simplest adapter strategy, that is, *connect me to the hub*. If you put transform in the hub, you either have to make available all possible transformations or you need to adopt a common format for moving information.

The first of these two presents a significant systems management problem and is not usually considered a viable option.

The transform to common format could be considered. From the point of view of building the system, doing the common format in the adapter transform seems attractive because this allows the business system provider to do all the work on their local platform rather than having to provide a transformation at the central hub to accompany the adapter. However, if you look at the runtime aspects of the systems, particularly if you wish to change your transformation rules, it is much simpler to do this at the central hub than to change all of the adapters.

This is the same principle as the management of client/server systems where code needs to be distributed to workstations vs. a centrally-managed system. For any reasonable number of systems (say, four again to be consistent!) the centrally-managed one is the more long term cost effective approach.

We conclude that transformation belongs at the hub, assuming you needed a hub for the routing capability.

One would argue that all functionality associated with the management of information flow between integrated systems belongs in the central hub. Other examples of management include services to publish information from business systems that can be subscribed to by other systems and the control of which business systems should be invoked to achieve an entire business process or workflow.

---

## 9.6 Application design summary

Business systems are best integrated using a pervasive asynchronous store and forward mechanism with a spoke and hub architecture. The only case where this is not true is where there is a business requirement to have point in time synchronization of updates in two or more systems where a synchronous, distributed transaction approach would be needed. In this case, the architecture would be point to point. Very few, if any, business processes have this requirement. Connection of the business system to the store and forward mechanism should be achieved by use of an adapter where the adapter's role is to provide a transactional (or as close to transactional as possible) connection to the store and forward system. All management of information flow should be provided by a central hub.

---

## 9.7 Using MQSeries

MQSeries is the most predominant business integration software. Although used mostly for intra business application integration or EAI, it has all of the attributes that were discussed in the previous sections making it suitable for B2Bi. Figure 62 on page 190 is a simple diagram showing a business system communicating with a partner. In the figure, three interfaces are marked out.

A is the interface or connector between the store and forward mechanism to the back end application. B is the store and forward mechanism, and C is the interface presented to the partner from the store and forward mechanism.

If we assume that B represents MQSeries and MQSI, in this section, we will discuss some of the considerations or options available for implementing the components A and C. A being the connector and C being the interface you expose to the partner.

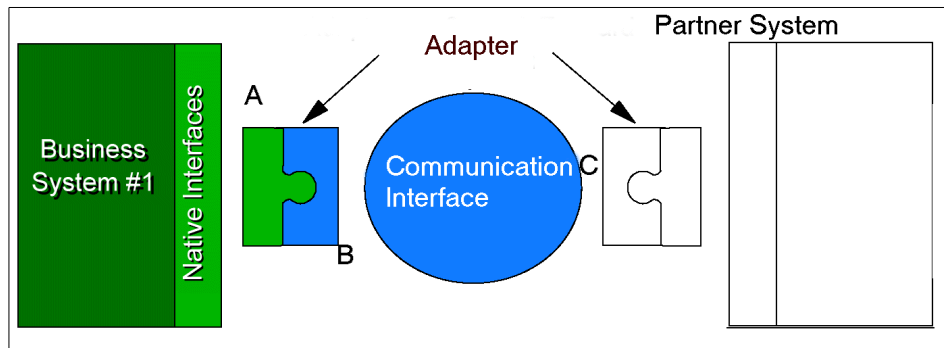


Figure 62. Application to application communication

## 9.8 Connecting to the business application using MQSeries or MQSI

This section describes the options available to connect applications into the integration infrastructure (the store and forward mechanism or the message broker).

In this discussion, connecting to an application is also termed an adapter. It has been used in a broader sense to this point. For the rest of this section, the term, *adapter*, will refer to the component that is built to connect into the business application.

### 9.8.1 Application types

The type of adapter you select or build depends on the application into which you are integrating.

Figure 63 on page 191 describes a method of classifying the types of adapter based on the type of applications into which they are required to connect.

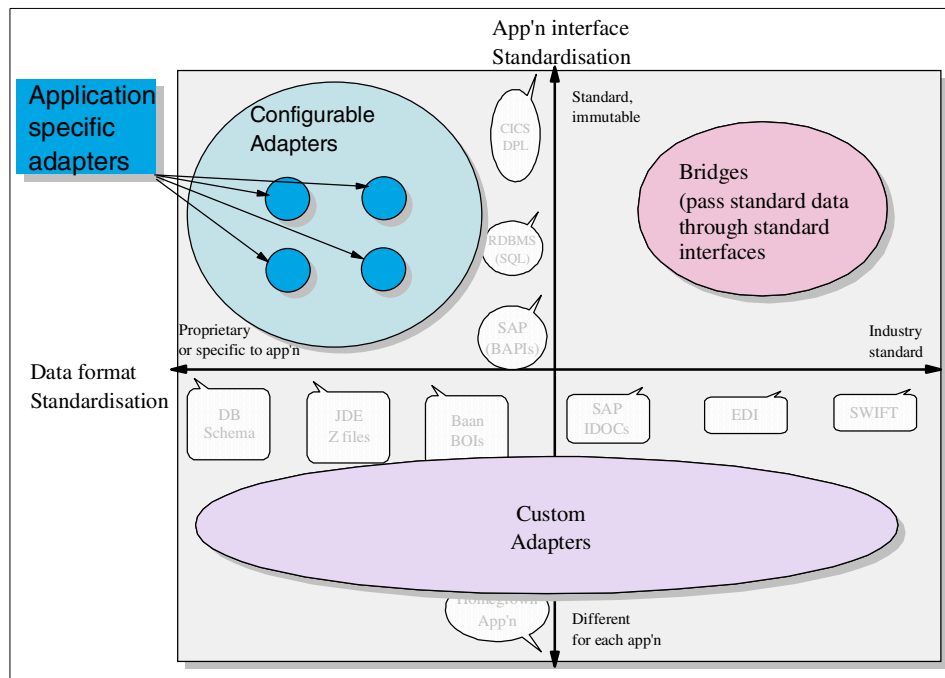


Figure 63. Adapter Interface classification and differentiation

The vertical axis represents the degree of standardization of the interface that is required for the application type. It ranges from completely different to standard. This refers to the application interface as it would appear in different enterprises. For example, a CICS provides a standard interface irrespective of the enterprise in which it is deployed. Therefore, a standard adapter can be developed for it, possibly, with some configuration parameters.

The horizontal axis represents the data format being passed to the target application. This could range from being proprietary to a specific application to an industry standard. If the data is in a standard format, it is much easier to create a standard adapter since you are aware of what inbound and outbound formats are required.

These differences give rise to three different adapter types:

- **Custom Adapters** – These adapters are required to connect to one-of-a kind applications. The behavior of these adapters are usually complex.

- **Configurable Adapters** – These adapters will usually be required to be connected to standard applications. The adapter is not necessarily programmed but configured.
- **Bridges** – These adapters are more like bridges, allowing data to flow from one system to another. They require very little configuration and setup.

Adapters are available from a number of vendors that fit into each of the above categories. Some of these vendors include the following:

- **NEON** – Have configurable adapters for industry formats (EDI, SWIFT) and packaged applications (SAP R/3, Siebel, PeopleSoft). Neon also has an Adapter Development Kit (ADK) that includes helper classes.
- **Extricity** – Have configurable adapters for packaged applications (SAP R/3, Clarify and so on) as well as an Adapter Development Environment. Adapters are, however, not separable from their Alliance Integration Server.
- **CrossWorlds** – Have configurable adapters for packaged applications (SAP R/3, Vanitive, Clarify and so on). Adapters are not separable from the Interchange Server.

For more information on the above vendors, please refer to their respective Web sites.

## 9.8.2 The MQSeries Adapter Offering

The MQSeries Adapter Offering consists of two product components:

- MQSeries Adapter Builder
- MQSeries Adapter Kernel

and two supportpac components:

- MQSeries Adapter Sets
- MQSeries Integrator Library

### 9.8.2.1 The MQSeries Adapter Builder

The adapter builder is a tool that allows a developer to import an application's interface into a repository by processing C header files containing function prototypes and structure definitions. It also allows a message format definition to be imported into the repository by processing XML Data Type Documents (DTDs). The OAG provide DTDs for all of their BODs on their Web site. The repository used by the tool is based on the same technology used by MQSeries Integrator v2, and, therefore, the message format definitions can be shared between the adapters and the message broker.



A developer with knowledge of the application can visually map the data from the message structure into the data structures expected by the application's interfaces and vice versa, that is, map the data from the application's interface into the desired message structure. The mapping definitions include basic data manipulation, such as padding, truncation, alignment, and so on. If more complex behavior is required, custom code can be written and stored in the repository. The tool then allows the developer to sequence the calls to the applications that are necessary to process all the data within a message.

Once the logic required in the adapter to transform data and interact with the application has been defined, the tool generates C code to implement the adapter, together with make files to build the code on a number of different platforms. Any custom written code is included in the generated code, and the generated code includes comments and tracing calls to aid debugging.

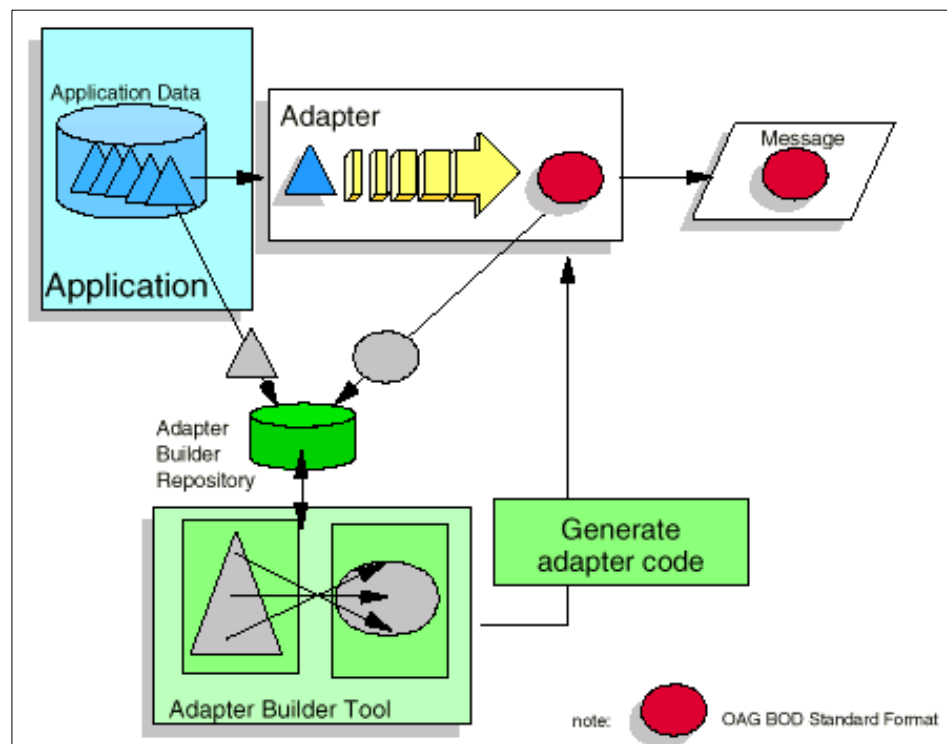


Figure 64. Adapter builder tool

### 9.8.2.2 The MQSeries adapter kernel

The MQSeries adapter kernel provides the standard runtime functionality required by any MQSeries adapter created by the adapter builder tool

including interaction with MQSeries queuing, configuration, and handling log and debug information. The adapter kernel is the same for all adapters, and the same code is, therefore, built for each of the supported platforms.

#### ***MQSeries adapter sets***

Initially, there will be sets of standard adapters for three ERP applications: SAP R/3, Baan IVb, and JD Edwards OneWorld. The adapter sets will be available to download from the Web as “patterns” to be loaded into the adapter builder tool and customized to meet each customer’s specific requirements. The patterns will define predetermined mapping between the fields in the OAG BOD messages and the application’s data, which can be augmented with extra non-standard data defined in the customer’s installation of the application.

#### ***MQSeries integrator libraries***

For those customers who use the NEON components of MQSeries Integrator, IBM will provide format libraries for the BOD messages handled by the adapter sets. These BOD definitions will be distributed in the form of import files; each of these will contain all the BOD definitions used by one of the functional areas, such as MES, corresponding to an adapter set. The format libraries will also provide useful definitions for customers who want to define their own XML messages in the NEON formatter.

---

### **9.9 General MQSeries guidelines**

The main points to consider when designing an application using message queuing are:

- Application considerations
- Program considerations
- Queue considerations
- Message considerations

These items are covered in detail in the redbook, *An Early Look at Application Considerations Involved with MQSeries*, GG24-4469. Although it does not use the latest version of MQSeries, it does cover the overall concepts for using messaging middleware.

The process of building an adapter for an EAI application differs significantly from the process of building a B2Bi application. That is the interface presented to the partner. When communicating outside of your own enterprise, the interface you present to your partner is of significant

importance. It will determine how easy or difficult it is to integrate with your enterprise.

The details on planning or building an application using MQ Series are covered in the MQ product documentation and a number of the redbooks listed at the end of this chapter. The MQ manuals are available on-line at the following Web site:

<http://www-4.ibm.com/software/ts/mqseries/library/manualsa/>

---

## 9.10 Application style

MQSeries and the associated message brokering architecture support the *store and forward*, *request/reply* and *publish and subscribe* programming models using either MQSeries persistent or non-persistent messages. In sequence of preference, use the following rules of thumb:

- *Store and forward* with non-persistent messages.

Because no response from the called Service is required and the messages do not need to be recoverable, this model has the best scalability and performance characteristics. There is no network traffic and, therefore, no network delays.

- *Store and forward* with persistent message.

Now MQSeries has to log the messages to ensure they are never lost. This involves disk IO, and, therefore, this model performs/scales less. There is no network traffic and, therefore, no network delays.

- *Request/reply* with non-persistent messages.

Use the *request/reply* model only when the client needs a response from the Service to be able to continue processing (for example, inquiry). In most cases, MQSeries non-persistent messages can be used in this programming model. The processing time of the called Service as well as network delays add to the response time.

- *Request/reply* with persistent messages.

Because of the additional MQSeries logging, the Service processing time, and the network delays, this model has the worst performance and scalability characteristics.

*Publish and subscribe* has the same characteristics as the *store and forward* model.

Compared to the synchronous programming model, the MQSeries programming models allows for the building of applications that scale and

perform better than could be achieved with synchronous communication models.

- When using *store and forward*, the client can continue processing as soon as MQSeries has accepted the message. The destination Service is not involved at all and does not even need to be available. No network traffic takes place.
- *Request/reply* is, basically, the only programming model that is supported when systems communicate synchronously. The requesting system waits for the answer of the called system before continuing. Network delays add to the wait time. MQSeries supports this programming model as well but offers opportunities to reduce the wait times and improve scalability.
- If the application allows, the replies could be sent to and handled by another application that may even run on another physical machine. In the synchronous model, this cannot easily be done. The requester has to handle the replies.
- When sending an MQSeries *request/reply* message, the requestor does not have to wait for the response. It can continue processing and process the response later. This allows the requestor to fire multiple messages and continue processing and collecting the responses later. The messages are processed by the Services in parallel. In the synchronous model, unless one develops multi-threaded programs, all request/response pairs are processed serially, and the individual wait times add together.

---

## 9.11 Application Programming Interface options

With MQSeries, you receive a family of three APIs designed to make programming straightforward for any messaging task, from the simple to the most advanced. All three APIs support the *request/reply*, *fire-and-forget*, and *publish and subscribe* application programming models.

### 9.11.1 MQI

The MQSeries Queuing Interface provides the lowest level of interfacing with MQSeries. It is available on all platforms on which MQSeries runs including the MQSeries client platforms. There are bindings for almost all language environments including C(++), Cobol, COM, Java. It is a rich interface through which all functionality MQSeries offers can be accessed. It is possible that the many options available through this interface may lead to inefficient, inflexible, or even conflicting MQSeries implementations. It is, therefore, advised to provide “corporate” guidelines for using this interface.

The MQI should be used when the other available interfaces are not appropriate.

### 9.11.2 AMI

Unlike the MQI, the Application Messaging Interface is an open standardized interface accepted by the Open Application Group for accessing MQSeries. It is very easy to use and is implemented on top of the MQI. The AMI interface, however, will be only available on certain MQSeries platforms and is not available for the MQSeries clients.

### 9.11.3 Java-based APIs

MQSeries provides support for developing MQSeries applications in Java through a number of Java-based APIs.

- MQSeries classes for Java
- MQSeries classes for Java Message Service (JMS)

#### 9.11.3.1 MQSeries classes for Java

The MQSeries classes for Java allow a program written in the Java programming language to connect to MQSeries as an MQSeries client using TCP/IP or directly to an MQSeries server using the Java Native Interface (JNI). They allow Java applets, applications, and servlets access to the messaging and queuing services of MQSeries. If the client-style connection is used, no additional MQSeries code is required on the client machine. The MQSeries classes for Java enable a message-based approach to application integration using Java.

MQSeries Java follows a traditional MQSeries object model approach. It allows greater flexibility and more control. It is relatively straightforward to use and requires an understanding of the MQ API (MQI).

#### ***Some considerations***

Maintain code levels as a general rule of thumb. on MQSeries servers, reinstall the supportpac(MA88) after applying the CSD.

JDK codepage support can be varied. For example, on HP, codepage 1051 (platform default) is not supported by the JDK.

When using the Java bindings servlets within WebSphere, use of JNI dependent code requires that the application server classpath (as opposed to the Web application classpath) be set. Failure to do this typically results in an `UnsatisfiedLinkError`.

### 9.11.3.2 MQSeries classes for Java Message Service (JMS)

MQSeries classes for Java Message Service is a set of Java classes that implement Sun Microsystems Java Message Service specification. A JMS application can use the classes to send MQSeries messages to either existing MQSeries or new JMS applications.

Use of the MQSeries classes for Java Message Service offers benefits associated with using an open standard to write MQSeries applications, such as the protection of investment both in skills and application code. In addition, the JMS classes provide some additional features not present in the MQSeries classes for Java. These extra features include:

- Explicit support for publish and subscribe
- Asynchronous Message Delivery
- Message Selectors
- Structured message classes

It also provides an administration tool for defining administration objects and storing them in an enterprise directory service (JNDI namespace).

Java Message Service (JMS) is a specification of a portable API for asynchronous messaging. JMS has been developed by Sun Microsystems in collaboration with IBM and other vendors interested in promoting industry-wide standard frameworks.

Many aspects of the Java API implementation are the responsibility of the vendor developing the product, and, naturally, IBM will provide a robust JMS service using MQSeries core technology.

JMS is an object-oriented Java API with a set of generic messaging objects for programmers to write event-based messaging applications. JMS supports both request /reply and publish/subscribe models as separate object models.

MQSeries JMS follows a simpler programming model since there is a higher level of abstraction. This abstraction layer does have an associated performance overhead. With MQSeries JMS, an application programmer may not need MQ-specific skills. It does, however, have reduced control and flexibility from a traditional perspective. It has extra functionality, described previously, that is not available in MQSeries Java.

#### ***Some considerations***

Within Multithreaded applications, connections are thread safe but sessions and other things are not. Sessions and message consumers should not be

accessed concurrently from different threads. One session per thread is a good rule to follow.

Ensure appropriate cleanup by calling the appropriate `close()` methods.

When doing Exception handling, upon catching a `JMSEException`, it is good practice to call `getLinkedException`, retrieving the associated `MQException`.

#### **9.11.3.3 Client vs. Bindings transport**

Programmable options allow MQ Java to connect to MQSeries in either of the following ways:

- As an MQSeries client using TCP/IP
- In bindings mode connecting directly to MQSeries

##### ***Client connection***

If you are using MQ Java as an MQSeries client, it can be installed either on the MQSeries server machine, which may also contain a Web server, or on a separate machine. Installation on the same machine as a Web server has the advantage of allowing you to download and run MQSeries client applications on machines that do not have MQ Java installed locally.

Wherever you choose to install the client, it can be run in three different modes:

- **From within any Java-enabled Web browser** – When running in this mode, the locations of the MQSeries queue managers that can be accessed may be constrained by the security restrictions of the browser being used.
- **Using an applet viewer** – To use this method, you must have the Java Developer's Kit (JDK) or Java Runtime Environment (JRE) installed on the client machine.
- **As a stand-alone Java program or in a Web application server** – To use this method, you must have the Java Developer's Kit (JDK) or Java Runtime Environment (JRE) installed on the client machine.

##### ***Bindings connection***

When used in bindings mode, MQ Java uses the JNI to call directly into the existing queue manager API rather than communicating through a network. This provides better performance for MQSeries applications than using network connections. Unlike the client mode, applications written using the bindings mode cannot be downloaded as applets.

To use the bindings connection, MQ Java must be installed on the MQSeries server.

### ***Some considerations***

An MQ Client implementation connects to a queue manager over TCP/IP via the MQ listner. It offers a pure Java solution and is useful for connecting to remote queue managers. It may be used for local queue managers.

Bindings is used in an MQ Server application. The queue manager must be local to the application. It utilizes JNI to call the C MQI. There are performance benefits over the client. Also, the application and queue manager are more closely coupled.

### **9.11.3.4 Applets vs. servlets vs. applications**

Each of the above methods has its drawbacks.

Applets tend to be somewhat heavyweight. They are often prone to browser JVM differences and may have security restrictions.

Servlets are the favored option for Web-based solutions. They have better performance and provide equivalent access to applications.

Java applications provide no run-time restrictions. They also allow access to native code. Performance optimization may also be performed using native compilation.

---

## **9.12 Considerations for the Partner Interface using MQSeries**

There are a number of ways of enabling Internet access to the Enterprise using MQSeries. This section describes the interfaces that could be made available and lists some of their characteristics. An older but relevant publication, *Connecting the Enterprise to the Internet with MQSeries and VisualAge for Java*, SG24-2144, has more information about the sections that follow. A number of additional resources are also available in the related publications section.

### **9.12.1 MQ Queue to MQ Queue: Intercommunication**

The information in this section was taken from the *MQSeries Intercommunication manual*, SC33-1872, available online at <ftp://ftp.software.ibm.com/software/ts/mqseries/library/books/csqzae03.pdf>

The manual contains a significant amount of detail that has not been included in the summary that follows.



One method of communicating with your partner when you have a symmetrical topology (when your partner also had his or her application or an application you provided) talk through an MQ Queue.

In MQSeries, intercommunication means sending messages from one queue manager to another. The receiving queue manager could be on the same machine or another machine, nearby, or on the other side of the world. It could be running on the same platform as the local queue manager, or it could be on any of the platforms supported by MQSeries. This is called a distributed environment. MQSeries handles communication in a distributed environment, such as this, using Distributed Queue Management (DQM).

The local queue manager is sometimes called the source queue manager, and the remote queue manager is sometimes called the target queue manager or the partner queue manager.

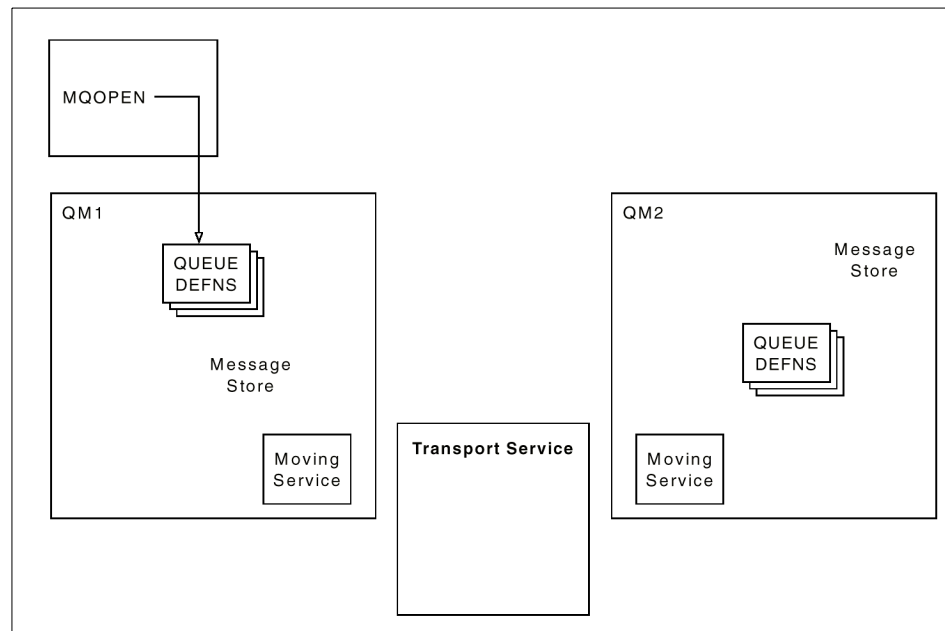


Figure 65. Overview of the components of distributed queuing

Distributed queuing works in the following way:

1. An application uses the MQOPEN call to open a queue so that it can put messages on it.

2. A queue manager has a definition for each of its queues, specifying information such as the maximum number of messages allowed on the queue.
3. If the messages are destined for a queue on a remote system, the local queue manager holds them in a message store until it is ready to forward them to the remote queue manager. This can be transparent to the application.
4. Each queue manager contains communications software called the moving service component; through this, the queue manager can communicate with other queue managers.
5. The transport service is independent of the queue manager and can be any one of the following (depending on the platform):
  - Systems Network Architecture Advanced Program-to Program Communication (SNA APPC)
  - Transmission Control Protocol/Internet Protocol (TCP/IP)
  - Network Basic Input/Output System (NetBIOS)
  - Sequenced Packet Exchange (SPX)
  - User-Datagram Protocol (UDP)

#### ***What do we call the components?***

1. MQSeries applications put messages onto a local queue, that is, a queue on the same queue manager.
2. A queue manager has a definition for each of its queues. It may also have definitions for queues that are owned by other queue managers. These are called *remote queue definitions*.
3. If the messages are destined for a remote queue manager, the local queue manager stores them on a *transmission queue* until it is ready to send them to the remote queue manager. A transmission queue is a special type of local queue on which messages are stored until they can be successfully transmitted and stored at the remote queue manager.
4. The software that handles the sending and receiving of messages is called the *Message Channel Agent (MCA)*.
5. Messages are transmitted between queue managers on a *channel*. A channel is a one-way communication link between two queue managers. It can carry messages destined for any number of queues at the remote queue manager.

#### ***Components needed to send a message***

If a message is to be sent to a remote queue manager, the local queue manager needs definitions for a transmission queue and a channel. Each end of a channel has a separate definition defining it, for example, as the sending end or the receiving end. A simple channel consists of a sender channel definition at the local queue manager and a receiver channel definition at the

remote queue manager. These two definitions must have the same name, and, together, they constitute one channel.

There is also a message channel agent (MCA) at each end of a channel. Each queue manager should have a dead-letter queue. Messages are put on this queue if, for some reason, they cannot be delivered to their destination.

Figure 66 shows the relationship between queue managers, transmission queues, channels, and MCAs.

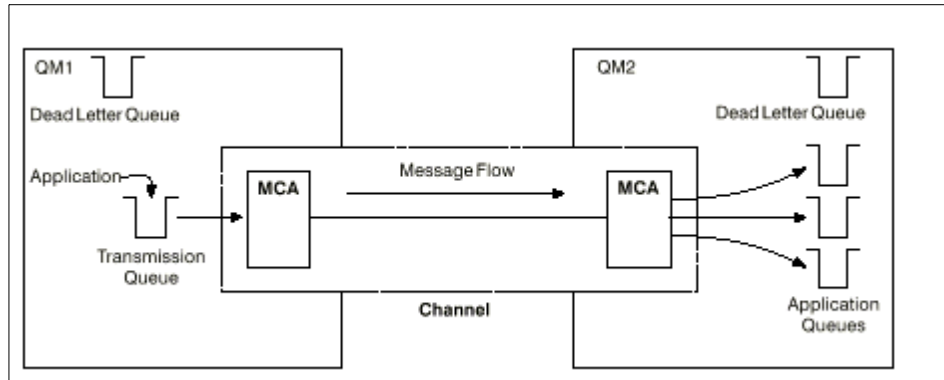


Figure 66. Sending messages

### **Components needed to return a message**

If your application requires messages to be returned from the remote queue manager, you need to define another channel, to run in the opposite direction between the queue managers, as shown in Figure 67 on page 204.

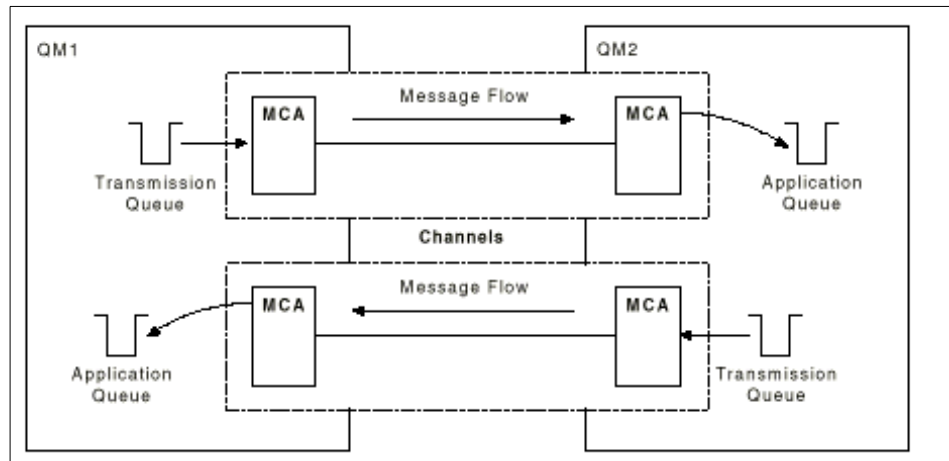


Figure 67. Sending messages in both directions

### Considerations

This option is probably the most efficient of all the interfaces described in this section. It does, however, require that the partner agree and purchase a shared middleware platform, MQSeries in this instance. This interface would probably be chosen by close partners. For example, two companies that are interdependent or have common shareholders.

Communication over an open network would probably also present problems although an Internet gateway is available and is described in Appendix B, "MQSeries Internet pass-thru" on page 307.

### MQ Queue to MQ Queue intercommunication over HTTP

This was released very recently. It does not require the Internet pass-thru option since it communicates over HTTP. However, there are some limitations.

#### 9.12.1.1 MQ classes for Java

These classes were described in Section 9.11.3.1, "MQSeries classes for Java" on page 197. The classes implement an object-oriented model for accessing MQSeries resources.

It supports various connection or transport options. In an MQ client implementation, it connects to the queue manager over TCP/IP via the MQ listener. This is a pure Java solution and is useful for connecting to remote queue managers.

A bindings connection would be used by an MQ server application. The queue manager must be local to the application. It utilizes JNI to call the C MQI. It has some performance benefits over the client. The application and the queue manager are more closely coupled.

#### **9.12.1.2 MQSeries connector**

The IBM Common Connector Framework defines a common wrapper for IBM e-business connectors, such as CICS, IMS, HOD, and MQSeries. It enables a common tooling methodology using Visualage for Java. This helps developers create applications/servlets in a consistent and common way.

CCF defines a common set of classes for all connectors as well as a runtime infrastructure. The MQSeries Connector is the MQSeries implementation of the IBM e-business connectors Common Connector Framework.

This framework makes it easier to build applications visually. It facilitates the deployment of pre-canned “service” beans to the Visual Application developer who simply wires methods and properties into application concepts. The IBM Visualage for Java allows easy construction of CCF type beans into Commands/Navigators.

The MQSeries connector utilizes the MQSeries Classes for Java v5.1 to access MQSeries. It supports client transport(TCP/IP) only. Support for bindings transport is in the works. It ships with Visualage Java Enterprise Edition V3.0.

#### **9.12.1.3 MQ classes for JMS**

These classes were described in Section 9.11.3.1, “MQSeries classes for Java” on page 197.

#### **9.12.1.4 Other technologies**

This section describes other methods of connecting into MQSeries.

##### ***MQSeries LotusScript extension***

This extension allows Lotus Script programs executing in a Domino Server to connect into an MQSeries Queue Manager. The Domino Server can be accessed by either a Notes client or a Web Browser. Domino’s support for Java agents allows Java programs using the MQSeries classes for Java to also be used.

##### ***MQSeries automation classes for ActiveX***

This option gives access to MQSeries via an ActiveX Class. The server side would be developed using Active Server Pages (ASP).

#### **9.12.1.5 MQSeries Internet gateway**

The Internet Gateway is a program that interfaces with the Web server and provides information or services on requests from a client program. The MQSeries Internet Gateway provides a bridge between the synchronous WWW and asynchronous MQSeries applications. The gateway, Web browser, and MQSeries together provide an Internet-connected Web browser with access to MQSeries applications.

The gateway acts as a mediator between the HTTP server and MQSeries. It receives data from the HTTP server and packages it into an MQSeries message. It then waits for the MQSeries reply message and passes it back to the HTTP server. The Common Gateway Interface (CGI) is a standard supported by almost all Web servers that defines how information is exchanged between a Web server and an external program.

More information is available in the redbook, *Connecting the Enterprise to the Internet with MQSeries and VisualAge for Java*, SG24-2144.

#### **9.12.2 Summary of interface options**

Each of the interfaces have its own characteristics and limitations. Choosing an interface will be dependent upon, among other things, the application requirements and the current infrastructure.

---

### **9.13 Building the hub and spoke architecture using MQSI**

In topology three, the application to application diagram could be described using Figure 68.

As discussed in Section 9.5, “Hub and Spoke integration architecture” on page 184, the hub approach becomes increasingly simpler to manage as we increase the systems.

MQSeries Integrator provides many of the functions required by such an architecture. This section describes some of the functions provided by MQSI and how they may be factored into your application design.

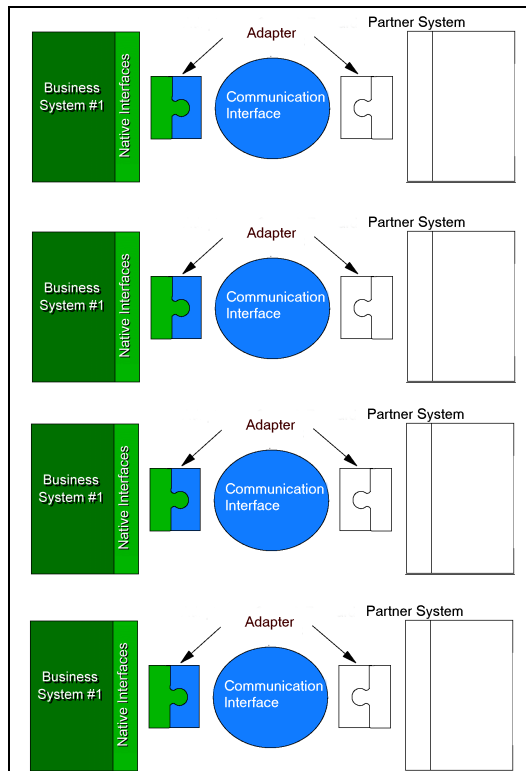


Figure 68. Multiple application with multiple partners

To build a complete MQSeries Integrator Version 2.0 application, you need to consider the following activities:

- Define the information space and model.
- Build the business message flows.
- Develop or modify applications that feed messages into the message flows, and consume the messages they produce.

The components that make up the first two activities are described in Chapter 8, “MQSeries and MQSeries integrator” on page 109. This summary was taken from the redbook, *Business Integration Solutions with MQSeries Integrator*, SG24-6154.

The product manuals, *MQSeries Integrator Introduction and Planning* and *MQSeries Integrator Using the Control Center*, contain further detailed information on the first two activities.

The next section summarizes the third activity: Developing applications that work with MQSeries Integrator Version 2.0.

### 9.13.1 MQSeries Integrator applications

MQSeries Integrator supports two communication models for applications. In the point-to-point model, an application is putting a message on a queue serviced by a broker. When the message has passed through the message flow, the broker puts it on a queue serviced by another client application. The second model is known as publish/subscribe. Applications wishing to receive messages will create a subscription on a topic. Applications wishing to send messages will create a publication for a topic. Publisher and subscriber are unaware of each other. The broker makes sure that publications are sent to the correct subscribers. Subscriptions can be topic based and/or content based.

When sending messages to a queue serviced by a broker, MQSeries Integrator needs to know something about the message. There are two ways to provide this information:

1. Your application prefixes the user data of the message with the MQSeries Integrator header, MQRFH or MQRFH2.
2. The MQSeries Integrator administrator has provided the necessary information in the configuration of the MQInput node. Basically, this instructs the broker to handle all messages on this queue as if they belong to the configured message domain, message set, and message type.

Applications that can use the MQRFH or MQRFH2 header provide the same information. If messages have an MQRFH or MQRFH2 header and, at the same time, the MQInput node has values for message domain, set, and type, the broker will check the values. When a message has conflicting information in the MQRFH or MQRFH2 header, the broker will handle this as a failure. Note that the MQRFH header was used in earlier publish/subscribe applications and in MQSeries Integrator Version 1 applications. The new version supports the older MQRFH header.

If publish/subscribe messages have no MQRFH or MQRFH2 header, the broker will rely on the topic information in the properties of the MQInput node, but existing publish/subscribe applications will, normally, already have the MQRFH header. For full use of the broker, it might be necessary to change these applications to use the MQRFH2 header.



### 9.13.2 Multiple hubs?

When designing a message brokering architecture on a large scale, it is pertinent to ask whether it is appropriate to deploy all instances of the broker components in one physical location or to split out some of this function across more than one location.

Clearly, it is simpler in the first instance to locate all major broker components in one location (perhaps on several physical processors). It is simpler in terms of systems management, operationally, deployment, database maintenance, etc. The main reasons for deploying across multiple sites, located possibly in different parts of the world, are:

- **Resilience** – Protection against a major outage of the data center.

This item is critical and, even on its own, probably justifies the design of an architecture built to be capable of being deployed in multiple locations. The key thing to incorporate into the design is to make sure that applications located on a spoke of one of the hubs can actually connect directly to a hub other than their local hub.

- **Capacity** – Extra processing hardware may be available in different locations.

This item could be discounted since it is always possible to install additional hardware in any location. The physical location should be irrelevant to overall processing capacity.

- **Performance** – Applications in one global region may prefer to communicate via their local hub

It is intuitively "obvious" that two applications that are attached to one hub are more likely to perform more quickly when sending data locally than if they have to send data to a processing hub on the other side of the world. However, reality may turn out to be different, for example, if the network that connects the principal data centers is very high bandwidth, then that consequent network performance between these principal sites may turn out to be a very small contributor to overall performance metrics.

The major contributor to performance and response time is most likely to be attributed to the local network connecting the hub to the application residing on the "spoke".

### 9.13.3 Database resilience

Within a given location, each MQSI broker will need access to an underlying database. This database will be used exclusively by MQSI to contain details of incoming message formats and rules. In MQSI V2, this is known as the

Message Repository manager (MRM) and, in MQSI V1, is represented by the Rules and Formats database. Each cloned instance of MQSI can use the same instance of the database; so, this database is a key component of the broker implementation. The MQSI instance uses standard database technology to connect to the database, for example, DB2 Client, accessed through an ODBC connection; so, in theory, the physical location of the database is actually not relevant. However, in practical terms, it would probably not be appropriate to place this database in a different physical location. Failure to gain access to this database would mean that the brokers would not be able to operate.

Resilience of this database is, therefore, key, and it should, therefore, be placed on devices that are equipped with hardware technology for failover. The database itself should be placed on shared disks across these machines. Failure of one of these processors will simply result in the backup processor taking over from the failed processor using the same network addresses so that the clients (that is, The MQSI instances) will not have to make any configuration changes.

#### **9.13.4 Message routing - Basis**

A key component of the broker architecture will be the basis on which routing decisions are made. Clearly, there are many places in the end-to-end flow at which decisions can be made, and the logic and data that drives these decisions will execute in a particular environment and use data held in a certain repository to make these decisions.

Some examples of where these decisions can be are:

- In the originating application (assuming that there is the capability to include exit-style routing logic)
- In the adapter connecting the source application to MQSeries
- In a preprocessing component, such as an adapter, that intercepts an MQSeries message immediately prior to being passed into the MQSI broker.
- Within the MQSI broker using information that is held within the message and logic held in the MQSI process flow
- Within the MQSI broker using information that is held within the message and logic held in the MQSI process flow combined with routing logic held in a relational database
- In a post-processing adapter that intercepts a message immediately after exiting MQSI using some form of routing algorithms and/or data repository

This type of function is best done in as central a place as possible, and whatever mechanism is chosen to hold the rules for such routing, it should be capable of being easily created, updated, disseminated, and managed.

In particular, if a database approach is taken, it is likely to be required to be accessed from more than one broker, possibly, in more than one location. Assuming that brokers in more than one location require this access, although this would be possible over a distributed network, the criticality of this information would lead to the conclusion that multiple copies of this information would be required to be located locally to the instance of the MQSI broker. This, in turn, leads to the conclusion that this information would be "read-only" from the broker and the requirement to introduce careful update and data replication procedures. It also introduces the requirement to be able to control use of the broker environment, for instance, to be able to quiesce some or all of the system (for example, Stop a broker, PutInhibit an MQSeries queue, stop a queue manager, or QM channel, and so on) while new routing information was fully disseminated.



---

## Chapter 10. Application development guidelines

The development of an e-business application does not differ very much from the development of any object-oriented, client/server application. However, there are some special considerations, which we will outline in this chapter. We will summarize the development process used to build an e-business application from the start of the development project until its deployment in a production system.

B2BI applications use a significant amount of middleware to achieve the end goal.

---

### 10.1 The development process

Today, it is quite common in the industry to develop object-oriented software with an iterative and incremental process. This approach has different roots. For more information, refer to the books, *Object-Oriented Analysis and Design with Applications*, ISBN 0-8053-5340-2, by Booch and Grady, *Object-Oriented Software Engineering: A Use Case Driven Approach*, ISBN 0-2015-4435-0, by Jacobson and Ivar, and *Object-Oriented Modeling and Design*, ISBN 0-1362-9841-9, by Rumbaugh, James, et al.

There is no defined standard process for development that everyone uses. Different teams typically adopt a recognized process using a vendor methodology or using their services team methodology. IBM Global Services has its own methodology used in customer engagements that covers the development process. The development process described throughout this chapter is simplified and more generic than the IBM Global Services methodology but is similar to it.

This process is described in detail in the redbook, *Patterns for e-business: User-to-Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*, SG24-5864. For completeness, a summary of the process is presented in the following sections.

The process we will discuss is divided into different phases. Each phase is done in a sequential manner and is subdivided into further smaller phases. Some phases are only run through once. Others are done over and over again, forming the iterative and incremental part of the development process. The actual process and which phases you use might differ slightly depending on the development team or organization that uses the process.

We can divide the whole process into the following phases:

- Solution outline
- Macro design
- Micro design
- Build cycle
- Deployment

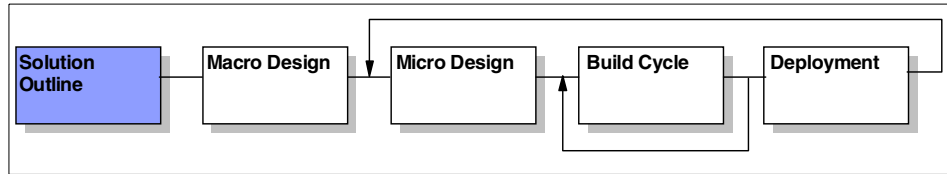


Figure 69. Development process overview

In the solution outline phase you decide the scope of the project, explore what the essential business needs are, come up with an idea of the base architecture, and get the commitment from the project sponsor to start.

Then you start with the macro design which concentrates on the detailed requirements gathering, business process modelling, high level analysis and design, the base architecture, and a plan for the following development phases, including a development release plan. These two phases are usually done once in a project.

Now the iterative and incremental part of the development starts. For each release of the developed e-business application, the micro design, build cycle, and deployment phases are completed. Usually, a certain set of use cases that have to be developed to meet a part of the system requirements make up a release. The releases are defined in the project plan produced in the previous phase. A release can be an internal one that is not deployed to any users. This is quite common for early stages of big projects. Others, like alpha or beta releases, might be deployed to a certain number of test users. It might take several iterations until a first official release of the application is deployed to the users. In turn, there are often several releases to the users until all requirements are met, plus maintenance releases to fix errors and other defects.

---

## 10.2 The scope of this chapter

In this chapter, we will explain the development process used for e-business applications and will focus on:

- The results
- The process to get those results
- The tools to produce those results

We chose this structure since some process models, such as the IBM Global Services methodology, suggest this approach. The idea is to drive the whole process from the results, called work products. The process is divided into phases as explained before. Each phase is divided into activities. In turn, each activity might include several tasks. Each phase, activity, or even task produces particular work products as output and needs others as input. The IBM Global Services methodology also provides technical papers (techniques) that describe how to produce the different work products.

---

### **10.3 The application and architecture domains**

As we mentioned before, we do not explain a specific methodology in this chapter. We will describe the process used to create the different work products in a general way, but we will not go into the work breakdown structure that the IBM Global Services methodology provides. However, the IBM Global Services methodology provides a domain concept that will help to position the contents of this chapter.

A domain is a logical grouping of related work products. Different roles and skills fit into a domain through enabling specialization. Domains build the basis for method tailoring which is an important part of the IBM Global Services methodology.

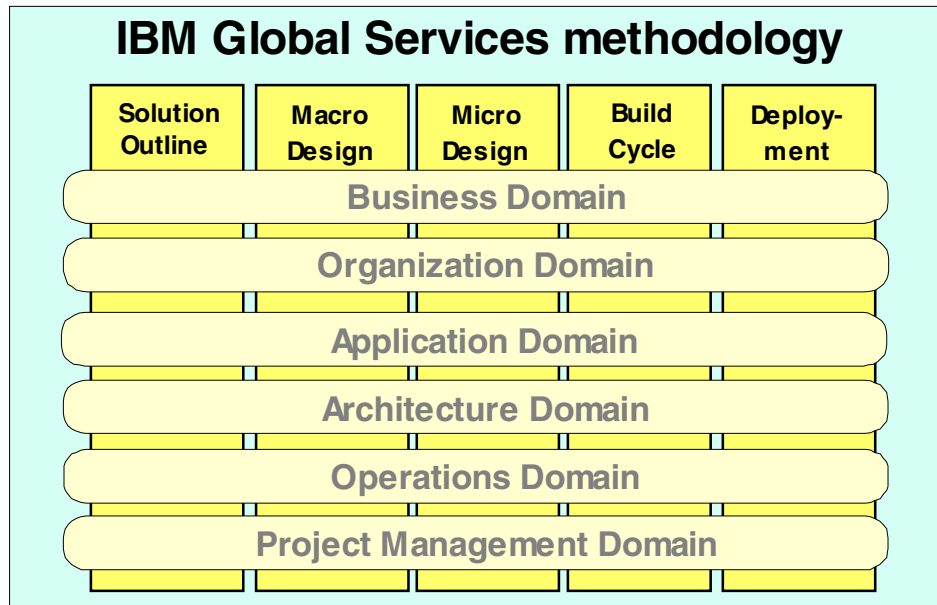


Figure 70. Domain concept in IBM Global Services methodology

Domains span the whole development process. There are six top level domains. We will concentrate on the application and architecture domains in this chapter. This does not mean that you should not be concerned with the other domains; for example, the use case model work product in the application domain is dependent on the business process model work product from the business domain.

## 10.4 Solution outline

The first phase of a development project is the startup. This phase normally begins with a small team of domain experts, analysts, and IT architects exploring the requirements for the new solution. Beyond the pure business requirements, it is important to explore the existing environment to find out how the new application can fit. The target audience for the solution has to be named, and their experience has to be determined.

Based on all this initial information, an architecture for the application has to be determined. The team has to decide on the overall strategy of the solution that will drive the whole project based on the business impact the solution has on the organization.



In the process of making these base architectural decisions, the team can use the assets of the patterns for e-business:

- First, choose a business pattern that best fits your business problem. The Patterns for e-business home page at <http://www.ibm.com/software/developer/web/patterns/> will help you make this decision. For the purpose of this discussion, the chosen business pattern is user-to-business.
- Next, review the application topologies for the chosen pattern.

The architectural decisions made are a separate work product and should be well documented. They are used as input for the macro design where the architecture is driven from the chosen logical application topology.

---

## 10.5 Macro design

In the macro design phase, the project team is usually extended from the few domain experts, analysts, and IT architects working in the solution outline phase to a broader skill set. This team works on the refinement of the results from the solution outline phase. Their task is to do the following:

- Refine the requirements to come up with the business process model by identifying and describing key business use cases.
- Evaluate various technology options available for the implementation. At this point, you need to choose a set of technologies for application development. Chapter 5, “Technology options” on page 67, helps you make this decision.
- At this stage, it is also important to plan the deployment model. For that reason, it is important to finalize the operational model. Chapter 4, “Choosing the runtime topology” on page 41, helps you choose a logical operational architecture. Since such decisions have an implication on the long-term system management of the deployed application, we urge you to consider the guidelines provided in Chapter 12, “Systems management” on page 235.
- Subsequently, the logical operational architecture needs to be mapped to a physical instantiation. Chapter 4, “Choosing the runtime topology” on page 41, provides guidance in achieving this.

As in the solution outline phase, you should document your architectural decisions as work products because they will serve as input for the following release cycles.

Other tasks that have to be done in this phase include:

- Design and plan the tests.
- Set up the development environment.
- Create a development plan for the following release cycles.

The release cycles start after all architectural decisions have been made and documented.

---

## 10.6 Micro design

The development plan outlines several release cycles to implement the requirements iteratively and incrementally. After the macro design phase, the requirements of a project are captured in different work products, such as the business process model, domain use cases, domain class models, and interaction diagrams for those use cases.

Each release cycle starts with the micro design that focuses on transforming the business model into a design model by taking the selected use cases and running them through a typical object-oriented development phase.

Transforming means that we use the business model to bring it to such a technically detailed level that it can be implemented. This is done by adding all the architecture and implementation-specific classes and components to the existing business model.

The design patterns explained in Chapter 9, “Application design guidelines” on page 171, are used for the transition of the business model to the actual design model. These design recommendations should lead you from your high-level application design model to a micro design model that is ready to be implemented.

The work products we produce in the micro design phase of the development process are:

- Use cases
- Class models
- Interaction diagrams
- State diagrams
- Component model
- Deployment model

The work products are described in detail in the redbook, *Patterns for e-business: User-to-Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*, SG24-5864.

Most of the work products are Unified Model Language (UML) artifacts. The UML specifications are defined by the Object Management Group (OMG, <http://www.omg.org>). Refer to the UML specification under <http://www.omg.org/uml>, and to *UML Distilled: Applying the Standard Object Modeling Language*, ISBN 0-2013-2563-2, by Fowler, Martin, Scott, Kendall, and Jacobson, for further information on UML. We use UML to express the work products and show the examples in UML notation. We work with Rational Rose 2000 Professional J Edition as our modelling tool. Use the documentation and tutorials that come with the Rational Rose product to find more details about its use. Refer to the Rational Rose home page at <http://www.rational.com/products/rose/>, to get the latest information about the product.

---

## 10.7 Build cycle

In each release cycle, the micro design is followed by the build cycle. The designed system is actually coded and tested in several build cycles. As in the micro design, each build cycle focuses only on the requirements valid for that particular release. So, with every build cycle, the developed system is growing in the functionality implemented.

In the build cycle, the results of the micro design are turned into code:

- Write and unit test the source code.
- Build the executable code if necessary, for example, all Java code.
- Perform various tests on the executable code.
- Test the application in a runtime environment.
- Prepare for deployment.

The incremental approach used to run the release cycles is also used for the different activities of the build cycle. It is run in several iterations for one release, with each iteration transforming more of the design into tested executable code that is ready to be deployed.

In this section, we will focus on the process of building and testing. Be sure to refer to Chapter 11, “Performance guidelines” on page 229, while implementing the source code and when executing your stress and performance tests.

For more detailed information on the following topics including using WebSphere Studio, VisualAge for Java, and source code management, refer to the redbook, *Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*, SG24-5755.

---

## 10.8 Deployment

Depending on the size of the developed components in the build cycle, the development team might decide to build it in several iterations. Even though each build cycle produces executable code, only the final result is usually deployed. The whole project is also run iteratively. It is up to the development team to decide which release, built in a development cycle, is going to be deployed. There might be alpha or beta releases that are deployed only to a certain number of test users.

### ***Deployment: Work products***

A deployment plan has to be created that encompasses not only when and how to install and set up the newly-developed application, but it must include all hardware and prerequisite software requirements.

You also have to plan for system management taking into consideration what has to be managed and how, how to establish the required security, and what has to be done for availability and recovery, before you can deploy the application into a production environment. More information on systems management can be found in Chapter 12, "Systems management" on page 235.

When preparing for deployment of a Web application, you have to plan and execute the hardware setup. In e-business applications with one of the architectures described in this book, this means server and network hardware, database and Web application server machines, network routers, and firewall machines. For an intranet application, this might also include client hardware, such as thin network computers. The software for all this hardware has to be installed and configured including any databases used, Web and application servers, firewalls, and security software. If the system uses application topology 2, it needs access to other production systems. In some cases, this means adding or changing components of a running system. This task must be addressed in the deployment plan.

You also have to plan how to hand over the operations of the production system to the staff that will be responsible for it.

---

## 10.9 Developing MQSeries applications

With MQSeries, you receive a family of three APIs designed to make programming straightforward for any messaging task, from the simple to the most advanced. Three APIs can be used for exchanging messages:

- MQSeries Message Queue Interface (MQI)
- Java and MQSeries for Java
- AMI

All three of the APIs can interoperate.

### 10.9.1 Message Queue Interface (MQI)

The Message Queue Interface (MQI) is the API that provides full access to the underlying messaging implementation and is available for all key languages and environments.

MQI is an easy-to-use, programming interface that allows applications to communicate transparently across the various platforms that make up the enterprise-wide computing environment. The MQI allows full access to MQSeries messaging support.

In-depth information on developing applications using MQI is available on-line at the following Web site:

<http://www-4.ibm.com/software/ts/mqseries/library/manualsa/index.htm>

The *MQSeries Application Programming Guide*, SC33-0807, contains all the information required to use this API.

### 10.9.2 MQSeries classes for Java and MQSeries classes for JMS

The following paragraphs will describe MQSeries classes for Java and JMS.

#### 10.9.2.1 MQSeries classes for Java?

MQSeries classes for Java (MQ base Java) allows a program written in the Java programming language to connect to MQSeries as an MQSeries client or directly to an MQSeries server. It enables Java applets, applications, and servlets to issue calls and queries to MQSeries giving access to mainframe and legacy applications, typically, over the Internet without necessarily having any other MQSeries code on the client machine. With MQ base Java, the user of an Internet terminal can become a true participant in transactions rather than just a giver and receiver of information.

### 10.9.2.2 MQSeries classes for Java Message Service (JMS)

MQSeries classes for Java Message Service (JMS) (MQ JMS) is a set of Java classes that implement Sun's Java Message Service (JMS) interfaces to enable JMS programs to access MQSeries systems. Both the point-to-point and publish-and-subscribe models of JMS are supported.

There are a number of benefits that arise from using MQ JMS as the API for writing MQSeries applications. Some advantages derive from JMS being an open standard with multiple implementations; others result from additional features that are present in MQ JMS but not in MQ base Java.

Benefits arising from the use of an open standard include:

- The protection of investment both in skills and application code
- The availability of people skilled in JMS application programming
- The ability to plug in different JMS implementations to fit different requirements

More information about the benefits of the JMS API can be found on Sun's Web site at <http://java.sun.com>.

The extra function provided over MQ base Java includes:

- Asynchronous message delivery
- Message selectors
- Support for pub/sub messaging
- Structured message classes

In-depth information about developing applications using JMS is available on-line at the following Web site:

<http://www-4.ibm.com/software/ts/mqseries/library/manualsa/index.htm>

*MQSeries Using Java*, SC34-5456, contains all the information required to develop applications using this API.

### 10.9.3 AMI

The MQSeries products enable programs to communicate with one another across a network of dissimilar components, such as processors, operating systems, subsystems, and communication protocols, using a consistent application programming interface, the MQSeries Message Queue Interface (MQI). The purpose of the Application Messaging Interface (AMI) is to provide a simple interface that application programmers can use without needing to understand all the functions available in the MQI. The functions that are

required in a particular installation are defined by a system administrator using *services* and *policies*.

#### **10.9.3.1 Main components of the AMI**

There are three main components in the AMI:

- The message, which defines *what* is sent from one program to another
- The service, which defines *where* the message is sent
- The policy, which defines *how* the message is sent

To send a message using the AMI, an application has to specify the message data together with the service and policy to be used. You can use the default services and policies provided by the system or create your own. Optionally, you can store your definitions of services and policies in a *repository*.

#### **10.9.3.2 Sending and receiving messages**

You can use the AMI to send and receive messages in a number of different ways:

- Send and forget (datagram), where no reply is needed
- Distribution list, where a message is sent to multiple destinations
- Request/response, where a sending application needs a response to the request message
- Publish/subscribe, where a broker manages the distribution of messages

#### **10.9.3.3 Interoperability**

The AMI is interoperable with other MQSeries interfaces. Using the AMI, you can exchange messages with one or more of the following:

- Another application that is using the AMI
- Any application that is using the MQI
- A message broker, such as MQSeries Publish/Subscribe or MQSeries Integrator

#### **10.9.3.4 Programming languages**

The Application Messaging Interface is available in the C, COBOL, C++, and Java programming languages. In C and COBOL, there are two interfaces: A high-level interface that is procedural in style and a lower level object-style interface. The high-level interface contains the functionality needed by the majority of applications. The two interfaces can be mixed as required.

In C++ and Java, a single object interface is provided.

In-depth information on developing applications using AMI is available on-line at:

<http://www-4.ibm.com/software/ts/mqseries/txppacs/ma0f.html>

The interface has to be downloaded from the above site and installed. An on-line guide, *Application Messaging Interface*, SC34-5604, contains all the information required to develop applications using this API.

---

## 10.10 Application development for MQSeries Integrator

In the previous chapter, we provided an overview of how to build a complete MQSeries Integrator Version 2.0 application. You should consider the following activities:

- Define the information space and model.
- Build the business message flows.
- Develop or modify applications that feed messages into the message flows, and consume the messages they produce.

In addition to writing applications to use MQSI, it is possible and would, possibly, be necessary to extend the functionality of MQSeries Integrator by developing a custom plug-in. This process is described in detail in the redbook, *Business Integration Solutions with MQSeries Integrator*, SG24-6154. The following sections are a summary of the requirements for developing a plug-in taken from the latter publication.

### 10.10.1 Terminology

In discussions about the framework that MQSeries Integrator provides for extending its functionality, a number of terms are used, such as plug-in, user-written node, user-written parser, and lil.

A *plug-in* is a generic name for any type of function that can be added to your MQSeries Integrator environment.

A *user-written node* or *node* is the first type of plug-in. It is the type of plug-in that you can add to the palette of the Control Center. The function that a user-written node can provide is very broad. It can route the message to one of its output terminals based on some specialized logic that is not easy to express in a Compute node. It can edit the message by accessing external resources and so on.



A *parser* is a second type of plug-in. You can decide to write your own parser for specialized messages that you cannot or do not want to express in the Message Repository Manager (MRM).

Finally, a *loadable implementation library* or a *lil* is the implementation module for one or more nodes or one or more parsers. A lil is implemented as a dynamic link library (DLL) on the Windows NT platform. It does not have the file extension *.dll* but *.lil*.

### 10.10.2 Overview of the requirements for a plug-in

Given that a plug-in is used at the heart of a broker, it is clear that there are some specific requirements for the implementation of a plug-in, such as API requirements, memory management, exception management, and threading issues.

#### 10.10.2.1 API requirements

There are a number of specific functions that a module must implement. We refer to these functions as implementation functions. Your lil can also use the functionality of the broker by calling utility functions, and, finally, you have the initialization function. This last function, with the name, `bipGetMessageflowNodeFactory`, is called by the broker after the lil has been loaded and initialized by the operating system. The broker calls this function to understand what your lil is able to do and how the broker should call the lil.

To express what your lil can do, or better yet, what nodes your lil supports, the `bipGetMessageflowNodeFactory` function has to call the utility function, `cniCreateNodeFactory`. This function passes back a factory name (or group name) for all the nodes that your lil supports. Next, the lil should call the utility function `cniDefineNodeClass` to pass the name of each node and a virtual function table of the addresses of the implementation functions.

To make the above more concrete, refer to the redbook, *Business Integration Solutions with MQSeries Integrator*, SG24-6154, which includes code snippets.

Because you are passing function pointers, you are free to choose the name of your implementation functions. Their signature is fixed, of course. The list below specifies what functions you have to implement:

- `cniCreateNodeContext`
- `cniGetAttributeName`
- `cniGetAttribute`
- `cniSetAttribute`

- `cniEvaluate`

When the broker has received the table of function pointers, it will call the function, `cniCreateNodeContext`, for each instantiation of the plug-in. If you have three message flows that are using your plug-in, your `cniCreateNodeContext` function is called for each of them. This function should allocate memory for that instantiation of the plug-in to hold the values for the configured attributes. Next it should call the appropriate utility functions to tell the broker about the input terminals and the output terminals it supports. Note that this must be done during `cniCreateNodeContext`. Next, the broker calls the `cniSetAttribute` function to pass the values for the configured attributes for this instantiation of the plug-in.

At this point, the instantiation of the plug-in is finished. When the broker retrieves a message from the queue and that message arrives at the input terminal of your plug-in, the broker will call the implementation function, `cniEvaluate`. This function should decide what to do with the message, propagate the message to a terminal, and, eventually, transform the message.

#### **10.10.2.2 Implementation requirements and restrictions**

Message flows can be used in multiple execution groups and in multiple threads at the same time. Therefore, the implementation of a node needs to be thread-safe and avoid the use of any global data.

To avoid creating orphaned memory, the node should make sure that all memory acquired by the node is released at some point.

All data passed between the broker and the plug-in is coded in a wide character set, even for attributes in an integer format. The implementation functions, `cniSetAttribute` and `cniGetAttribute`, should make sure that the necessary conversions are done with data received from the broker or passed back to the broker.

The plug-in should also limit the use of any operating system-specific functions. The use of external configuration parameters, such as environment variables or ini files, should be avoided because the broker cannot assist in distributing these resources. MQSeries Integrator is designed in such a way that brokers can be administered from remote locations. If a plug-in requires external data, you should make sure that this data is available before the deployment of a message flow using the plug-in.

And, finally, here are some considerations about exception management and errors. When a plug-in calls utility functions that execute in the broker, the

return code parameter will contain an indication of any error conditions. The plug-in should test the value and perform any recovery operations if needed. The utility function, `cciGetLastExceptionData`, can then be used to obtain more information about the error. Once the plug-in has done its logging and recovery, the exception should be passed back to the broker again by calling the utility function, `cciRethrowLastException`. The plug-in itself can also raise exceptions using the utility function, `cciThrowException`. It is critical for the integrity of the broker that a plug-in respect and use these exception management functions.



## Chapter 11. Performance guidelines

This chapter discusses the various aspects of performance and how the design or implementation of an e-business application can affect them. This chapter does not present exhaustive guidelines on performance but serves as a pointer to sources of more detailed information.

It's imperative to include performance considerations at the architecture and design phase for all e-business development projects. Without clear performance objectives during initial architecture definition and high level design, performance becomes an afterthought; something to be dealt with after hardware and software products have been selected and application code developed. We have enough data and experience now to know that these projects often fail when they are deployed - not because they cannot deliver the required business function, but because they cannot service the demand generated by their customer community in a satisfactory way.

A white paper, *Designing e-business Solutions for Performance*, by Maggie Archibald and Mike Schlosser, covers this topic extensively. Although the paper explores performance relative to a User-to-Business pattern, the methodology is roughly the same.

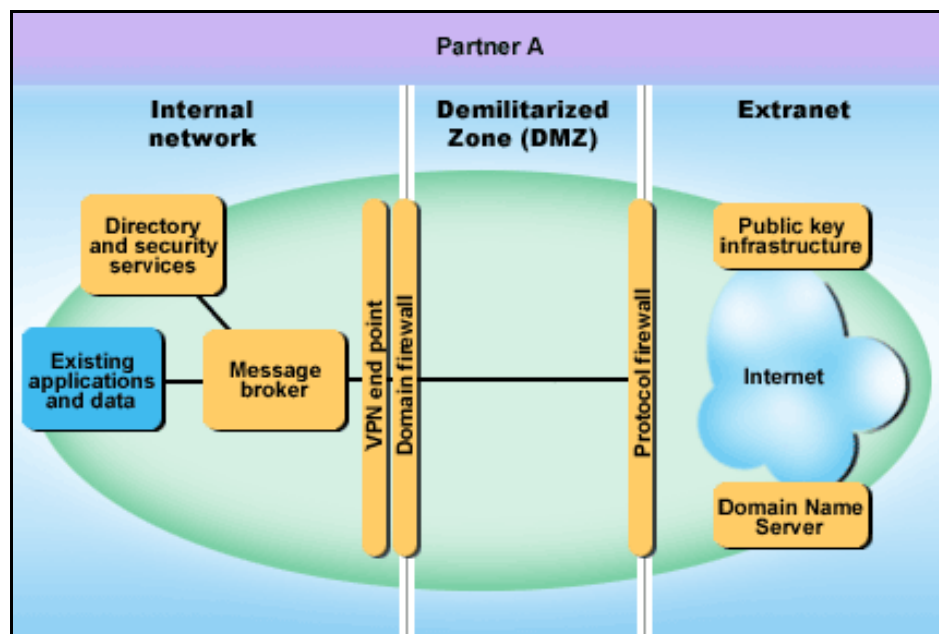


Figure 71. Application topology 3, the runtime components

With B2Bi topologies 2 and 3, performance is a function made up of three parts:

1. The implementation and deployment of the many pieces of middleware that integration solutions require.
2. The application design of the adapters and communications used to integrate the applications across the enterprises.
3. The performance of the applications being integrated.

This chapter will summarize some of those components and provide pointers to more extensive material. You will be able to use this material to reflect on your proposed architecture.

---

## **11.1 MQSeries and MQSI tuning, capacity planning, and performance**

The elements that make up the planning, tuning, and optimizing guidelines are subject to the environment within which MQSeries is deployed. The *MQSeries Planning Guide*, GC33-1349, provides detailed information for the many platforms that MQSeries supports. In addition to this guide, the following sections describe additional material available for this task.

### **11.1.1 Hardware and capacity**

MQSeries and MQSI have base requirements. These requirements are based on the platform being deployed. The processor, amount of storage and random access memory (RAM) required for typical configurations of MQSeries are listed in the planning guides. For MQSeries, the *MQSeries Planning Guide*, GC33-1349, provides the base hardware requirements for each of the platforms. The *Introduction and Planning Guide*, GC34-5599, provides this information for MQSI.

The above guides also provide capacity information for the various platforms based on a sample configuration that you could extrapolate based on your environment and experience.

### **11.1.2 MQSeries and MQSI application performance**

The following are some ideas to help you design efficient applications:

- Design your application so that processing goes on in parallel with the user's thinking time:
  - Display a panel and allow the user to start typing while the application is still initializing.

- Don't be afraid to get the data you need in parallel from different servers
- Keep connections and queues open if you are going to reuse them instead of repeatedly opening and closing, connecting, and disconnecting.
- 

**Note**

However, a server application which is putting only one message should use MQPUT1

- Keep your messages within a unit of work so that they can be committed or backed out simultaneously.
- Use the nonpersistent option for messages that do not need to be recoverable.

The *Introduction and Planning Guide*, GC34-5599, provides some guidance on performance for MQSI.

### 11.1.3 Additional performance information

Information about MQSeries performance is available on the Internet at:

<http://www.ibm.com/software/ts/mqseries/txppacs/txpml.html>

The sections below describe some of the SupportPacs and their contents that are available for various IBM hardware platforms.

#### 11.1.3.1 OS/390

The MQSeries SupportPacs can be found at the following Web sites:

<http://www-4.ibm.com/software/ts/mqseries/txppacs/mp16.html> and

<http://www-4.ibm.com/software/ts/mqseries/txppacs/mp19.html>. They provide an in-depth look at optimizing MQSeries for the OS/390. These SupportPacs provide capacity planning and tuning information for MQSeries for MVS/ESA Version 1.2 and MQSeries for OS/390 Version 2.1, most of which is common to both.

This SupportPac contains the following:

1. It describes the steps taken to optimize performance of an MQSeries for MVS/ESA V1.2 or MQSeries for OS/390 V2.1 system that uses persistent, 1000-byte messages. Starting with the MQSeries-supplied default definitions, it describes the symptoms of potential performance problems and the actions taken to resolve them.
2. It provides a variety of capacity information.
3. It provides tuning advice.

### 11.1.3.2 AS/400

The MQSeries SupportPac available at

<http://www-4.ibm.com/software/ts/mqseries/txppacs/mp43.html> provides an in-depth look at optimizing MQSeries for the OS/390.

This SupportPac aims to provide capacity planning information relevant to MQSeries for AS/400 Version 4 Release 2.1 and to highlight the performance features available in this release.

The measurements described in the SupportPac are categorized below:

- **Synchronous request/reply timings** – These measurements show the round trip times that can be achieved between Queue Managers when using synchronous messaging. They illustrate the benefit of using Fast Channels.
- **MQSeries channels** – These evaluations measure the message throughput that can be achieved using MQSeries channels. They show the effects of using Fast Channels and batch interval to maximize performance.
- **MQSeries client performance** – These figures indicate the level of performance that can be expected when connecting MQ thin clients to an AS/400. They illustrate the performance benefits of using correlation ID.

### 11.1.3.3 AIX

The MQSeries SupportPac is available at

<http://www-4.ibm.com/software/ts/mqseries/txppacs/mp67.html> and provides an in-depth look at optimizing MQSeries for AIX.

This SupportPac provides capacity planning information relevant to MQSeries for AIX V5.1. It presents the results of MQSeries client-based performance evaluations although most of the tuning activities also apply to distributed queuing. The SupportPac includes:

- Tables and charts that summarize the performance characteristics of various MQSeries client based configurations
- Interpretation of the evaluations and their implications for those designing or sizing MQSeries configurations
- Appendices containing the detailed performance data

### 11.1.3.4 Windows

The MQSeries SupportPac is available at

<http://www-4.ibm.com/software/ts/mqseries/txppacs/mp74.html> and provides an in-depth look at optimizing MQSeries for Windows NT.



This SupportPac provides capacity planning information relevant to MQSeries for Windows NT V5.1. It presents the results of MQSeries client-based performance evaluations, although most of the tuning activities also apply to distributed queuing. The SupportPac includes:

- Tables and charts that summarize the performance characteristics of various MQSeries client based configurations
- Interpretation of the evaluations and their implications for those designing or sizing MQSeries configurations
- Appendices containing the detailed performance data

#### **11.1.3.5 MQSeries Integrator Performance**

The MQSeries SupportPacs are available at the following URLs:

<http://www-4.ibm.com/software/ts/mqseries/txppacs/mpi1.html>

<http://www-4.ibm.com/software/ts/mqseries/txppacs/mpi2.html>

<http://www-4.ibm.com/software/ts/mqseries/txppacs/mpi2.html>

<http://www-4.ibm.com/software/ts/mqseries/txppacs/mpi4.html>

They do not provide a comprehensive performance test but rather a series of useful tests and conclusions designed to allow the reader to determine which factors will most affect the performance of their own particular MQSeries Integrator setup. Also provided in some of the SupportPacs is the source of the C programs used in the generation of the results.



---

## Chapter 12. Systems management

Systems Management is complex and includes the management of different environments, networks, hardware, operating systems, and applications. This section only describes security for MQSeries since MQ is the predominant component for patterns 2 and 3. It does not include any description of how this interrelates with managing other software or network hardware.

However, it should be noted that many of these problems affect each other. For example, an MQSeries problem can affect or be affected by numerous products. For example:

- If disk space runs out, this is first seen as an MQSeries problem; for example, it is not possible to issue an MQPUT to a queue, then second, as an operating system problem, and, finally, as a hardware problem.
- If there is a defect on a network adapter and this causes the network to slow down or to fall out, IP connections may close, and, finally, MQSeries channels between some systems may go in a stopped or retry state, and a transmission queue may get full.

This chapter describes the management of MQSeries and MQSI since they are the most significant components of Topology 2 and 3.

---

### 12.1 Managing MQ?

For truly effective and efficient MQSeries network management, you need to manage your MQSeries network from a central point, as well as minimize time-consuming and sometimes error-prone administrative tasks. For example, reliance on local configuration and operation at each system in a distributed, multi-platform environment can be costly. What is needed is a centralized administrative and operations management solution that accelerates your deployment of MQSeries along with improving the efficiency in managing your MQSeries network.

Typically, as you design and develop your MQSeries network, the parameters of each MQSeries node must be defined. Conversational node pairs, such as channels, require parameters that match, which demands coordination between administrators of disparate systems. This configuration management process is labor-intensive, complex, and error-prone requiring specialized technical personnel with deep knowledge about different operating platforms. Simple typographical errors and naming convention conflicts can add to the difficulties.

IBM provides very basic management services with its MQSeries package by design. IBM relies on additional functions (SupportPacs) or solution partners to fill these important application requirements for the end-user.

### **12.1.1 What should be managed in MQSeries networks?**

Systems Management is very important to ensure that you have an MQSeries network with high availability, good throughput, and fewer specially-educated staff. For MQSeries, you should look at the following:

- Platform Support is divided into two groups: MQSeries support and GUI Interface support. MQSeries support is based on areas, such as MVS, AIX, NT, OS/2, and so on. GUI support usually involves Windows NT/95, OS/2, UNIX, and 3270.
- Management Console Information areas, such as Central Point of Control, GUI, Local and Remote Platforms, and Ad Hoc Queries.
- Enterprise-Wide Management investigates areas, such as Mainframes, Workstations, Databases, and the Enterprise integration with CICS, IMS, DB2, and mid-range monitoring products.
- Open Systems Platform Connectivity connects your MQSeries management toolset to a management tool such as HP OpenView, Tivoli TME, CA Unicenter, or Cabletron Spectrum. Microsoft SMS is usually not covered by the management products.
- Security Services, such as Machine Authentication, User ID Authentication, Message Validation, and Data Encryption.
- Configuration touches several areas, such as MQSeries Queues, Channels, Automated Tasks, and Automatic Discovery.
- Software Distribution for distributing and updating MQSeries software on different platforms; this task should include the distribution and update of the MQSeries objects and may only be completely done when the queue manager or the whole MQSystem is down or when a system backup (including the contents of the queues) has been done before.
- Performance, for example, Queue Manager Status, Message Manager Performance, Buffer Manager Performance, Log Manager Performance, Channel Performance, and Dead Letter Queue Management. A Dead Letter Queue management function should be able to manipulate messages in the Dead Letter Queue by, for example, viewing, deleting, or moving them to other queues.
- Scalability of applications includes areas, such as Dynamic Routing, Load Balancing and Speed, and Application Deadlocks.

- Application Testing Tools look at areas, such as Edit Messages, Test Data Creation, Manage Queues, and System Testing.
- Consulting Services are capabilities a vendor currently offers to support in using the management tools. Application Design, Education and Training, Project Management, Application Development and Performance, and Availability of Middleware Consultants should be evaluated.

### 12.1.2 MQSeries Systems management products

Today, there are several comprehensive and robust products available from both large and small vendors. Two reports that contain a fairly in-depth description and comparison of the products available are The MQSeries Systems Management Market available at the following URL:

<http://www-4.ibm.com/software/ts/mqseries/library/whitepapers/sysman/>

and the IBM Support Pack #MS08 "Evaluation of MQSeries Systems Management Products" available at

<http://www-4.ibm.com/software/ts/mqseries/txppacs/txpm3.html#sysmng>

The preceding link includes a number of additional SupportPacs related to systems management.

The functional areas that relate to the management and administration of your current and planned MQSeries is commonly broken down as follows:

- **Configuration Management** – The ability to deploy MQSeries code and create and delete MQSeries objects including Queue Managers, Queues, Channels, and Processes from a single point of control
- **Operations Management** – The ability to start and stop resources, such as Queue Managers, Channels, Trigger Monitors, Channel Listeners, and Channel Initiators from a single point of control
- **Problem Management** – The ability to detect, track, and resolve problems with MQSeries objects from a single point of control
- **Performance Management** – The ability to determine the performance of MQSeries objects and networks including Queues and Channels from a single point of control

You should also consider strategic issues that may determine what products you will need to manage your MQSeries network. These include:

Do you need stand-alone MQSeries management, a broader application view, or enterprise-wide support? Are the mainframe(s) acting as a central application message switch or as another distributed end-node? Should the central Manager Platform be accessible from a 3270 console, a Windows/NT or UNIX GUI, or a Web browser? Which MQSeries platforms must be

managed and which vendors provide full support for those platforms? Is there an incumbent vendor managing other subsystems, such as CICS or DB2, and what is your organization's enterprise-wide management direction? Do you have adequate in-house technical talent to select and implement a systems management solution, or should you plan for outside assistance?

---

## 12.2 Security

With any operating environment, there are usually a number of components that provide various services. The most significant of these are:

- **Identification and authentication (I&A)** – This service forms the basis of many of the other services and involves the provision of a user identifier (user ID or principal) and the verification that the identifier is valid (that is, it represents the actual user and is not some intruder impersonating a valid user).
- **Authorization** – This is access control and relies upon the availability of some user identifier to compare against access control lists (ACLs). Note that the authorization service is only useful if it is used; an intruder might attempt to bypass (and neutralize) the authorization service.
- **Data confidentiality or encryption**
- **Data integrity** – Even though data might be visible (that is, not encrypted), the data integrity service ensures that data is not altered.
- **Non-repudiation** – This is the provision of some form of token, such as a digital signature, that guarantees that a particular piece of data originated from a particular user.

Note that although the Security component is positioned as one of the distribution services, it also provides services within the node (particularly authorization but all of the other services as well).

The way in which these security services are used by the various aspects of MQSeries is explained in the following sections.

### 12.2.1 MQSeries security functions

The following sections apply to all MQSeries products.

#### 12.2.1.1 MQSeries applications

Before an application connects to a queue manager, it will have undergone some form of I&A procedure. This might be the provision of a user ID and password or it might be some more elaborate process, such as a smart card identification. Alternatively, it might be that there is no requirement for the I&A

procedure and, thus, no user identifier associated with the application (for example, the building that houses the system might be physically secure, and user identifiers might not be considered necessary).

The consequence of this is that each application that issues MQI calls has an associated user identifier (which might be null) that is used to authorize the use of certain MQI functions (primarily MQCONN and MQOPEN) and options (such as PUT and GET) against particular objects (usually queues). This means that any application user identifier trying to access MQSeries resources must be suitably authorized.

Because the I&A procedure takes place before the application connects to the queue manager, it is the responsibility of components other than MQSeries to provide the I&A service. MQSeries is only responsible for capturing the user identifier for use in providing other security services, such as authorization. (The user identifier is captured when the application connects to the queue manager.)

Similarly, it is often the responsibility of some other component to provide the authorization service, with MQSeries having responsibility for calling that service. This works satisfactorily for systems, such as OS/390, where there is a standardized interface (for example, SAF) to the authorization service. For the OS/2 Warp, Windows NT, and UNIX platforms, however, there is no standard authorization service or interface provided, and so, MQSeries provides its own interface. This is called the authorization service and is documented in the MQSeries Programmable System Management manual.

#### **12.2.1.2 The Object Authority Manager**

For MQSeries for AS/400, MQSeries for Compaq (DIGITAL) OpenVMS, MQSeries for Tandem NSK, MQSeries on UNIX systems, and MQSeries for Windows NT, there is an additional component written to the SEI that provides an authorization service. This is called the object authority manager (OAM), and it restricts access to MQSeries objects based upon the access control lists (ACLs) that it manages.

The OAM is documented in the following product documentation:

- *MQSeries for AS/400 V5.1 System Administration Manual*
- *MQSeries for Digital OpenVMS System Management Guide*
- *MQSeries for Digital UNIX System Management Guide*
- *MQSeries for Tandem NonStop Kernel System Management Guide*
- *MQSeries for VSE/ESA System Management Guide*

### **12.2.2 MQSeries messages**

The basic function of MQSeries is to pass messages between applications. The message header (the message descriptor, MQMD) contains a field, `UserIdentifier`, where a user identifier can be placed allowing the application that gets the message to know from which user the message originated. The user identifier can be placed in the MQMD in one of three ways:

- The user identifier from a previous message (that is, one for which an MQGET has been performed) can be passed to a subsequent message. This is known as passing the security context.
- A suitably authorized (that is, trusted) application can place any user identifier in the field.
- If neither of the above is used, MQSeries automatically places the user identifier of the application that did the MQPUT in this field.

Therefore, this aspect of MQSeries operation provides an identification service (associating a user identifier with the message); it does not provide an authentication service. It is, currently, the responsibility of MQSeries applications both to provide any required authentication token (on the MQPUT side) and to verify that token (on the MQGET side).

Note also that, when an application does an MQGET for a message, there is no attempt to reset the application's user identifier to that contained in the message header. This function is called context management and is not supported by MQSeries.

### **12.2.3 Point-to-point security**

In addition to providing services as a local resource manager, a queue manager also provides distributed queuing, enabling messages to be distributed around a network of queue managers. This distributed messaging function is provided by means of MQSeries channels. Each channel is composed of a pair of message channel agent programs (MCAs), which provide the protocol for assured, once-only message delivery using an underlying transport mechanism to exchange messages.

When the two MCAs establish communication, it might be necessary for each to verify the identity of the other. This would be the case if one queue manager did not trust the connection or the identity of the partner queue manager (for example, if they were owned by separate enterprises). This verification can be accomplished in one of the following ways:



- Some transport mechanisms (in particular APPC) provide security features, such as session authentication. Note that this provides verification of the partner system (the partner logical unit), rather than the partner application (the MCA) but this might satisfy the security requirements of the queue manager.

- The MCAs each provide a security exit point which can be used to call user-written security exits for the exchange of user identifiers and associated authentication tokens (password, ticket, and so on). This allows each MCA to verify the identity of its partner.

Using the MCA security exit allows the channel to be independent of the underlying transport mechanism and to provide a consistent service across many transports. This is especially important when providing a service (like security) that is available only on a limited set of transports.

This aspect of MQSeries operation provides support for the I&A service. If required, the security exit can use the central security services to provide authentication tokens but this is not a requirement.

- On AIX, HP-UX, OS/2 Warp, Sun Solaris, Windows NT, and Windows 95, MQSeries provides exits relating to DCE security. Source code is also provided to help you understand the program, and to assist you in creating your own.

Note that it is possible to use any or all of the above mechanisms for point-to-point security, enabling an MCA to be assured that it is exchanging messages with the correct queue manager.

#### **12.2.4 End-to-end security**

End-to-end security refers to security services that can be provided when a message is PUT by an application and to the corresponding services that are available when the target application performs a GET. The relevant services are identification and authentication (possibly across the network), data confidentiality, data integrity, and non-repudiation.

MQSeries is not responsible for the provision of these services, but it is responsible for providing appropriate interfaces to call these services.

Today, the only aspect of end-to-end security that is (directly) supported by MQSeries is Identification, where a user identifier can be placed in a message header. MQSeries does not provide (direct) support for any of the other services.

However, these services can be implemented in either MQSeries application programs or the MCA message exit, which is a customer exit invoked each time a message is passed between two queue managers.

### **12.2.5 Where to find more information**

For information about MQSeries security on your platform, see the following chapters in the *MQSeries Planning Guide*, GC33-1349:

- Chapter 11, Security planning for MQSeries for AS/400
- Chapter 16, Security planning for MQSeries for Compaq (DIGITAL) OpenVMS
- Chapter 21, Security planning for MQSeries for OS/2 Warp
- Chapter 27, Security planning for MQSeries for OS/390
- Chapter 34, Security planning for MQSeries for Tandem NSK
- Chapter 38, Security planning for MQSeries for UNIX systems
- Chapter 41, Introduction to MQSeries for VSE/ESA
- Chapter 44, Security planning for MQSeries for Windows NT

---

### Part 3. An application example

The third part of the book looks at a solution with MQSeries, MQSeries Integrator, WebSphere, and DB2. This solution consists of a Web application that finds information about a customer on a number of separate systems. We have called this solution the *single customer view*. We are aware that the issue of getting a single customer view is far more complex than what is shown here. This solution will help customers in architecting their solutions. Basically, the solution shows how to break up a single request into multiple requests using MQSeries Integrator and how to reformat each part of the request into the format that the back-office application expects. Finally, we show how each reply is collected and used to build a single reply for the front-end application.

This part was produced as a part of another project. The material is reproduced here for illustrative purposes. The application could be described as a User-to-Business or a Business-to-Business Integration application since it has the properties of both. The distinction is made in how the partner interface is implemented. In this section, it is used to illustrate B2Bi. This application is an example of topology 3. The topology implements both the routing and transformation functions that are illustrated in topology 3.



## Chapter 13. Getting a single customer view with MQSeries

This section was developed independently of this project. It was originally part of the redbook, *Business Integration Solutions with MQSeries Integrator*, SG24-6154.

This section shows how MQSeries and MQSeries Integrator can be used to enable the gathering of information from multiple MQSeries-enabled legacy systems and then display this information on a single Web page.

Figure 72 is a high-level view of the components of the solution. The application takes a request from an application server using a servlet, queries three legacy backend systems, and consolidates a response to the requester.

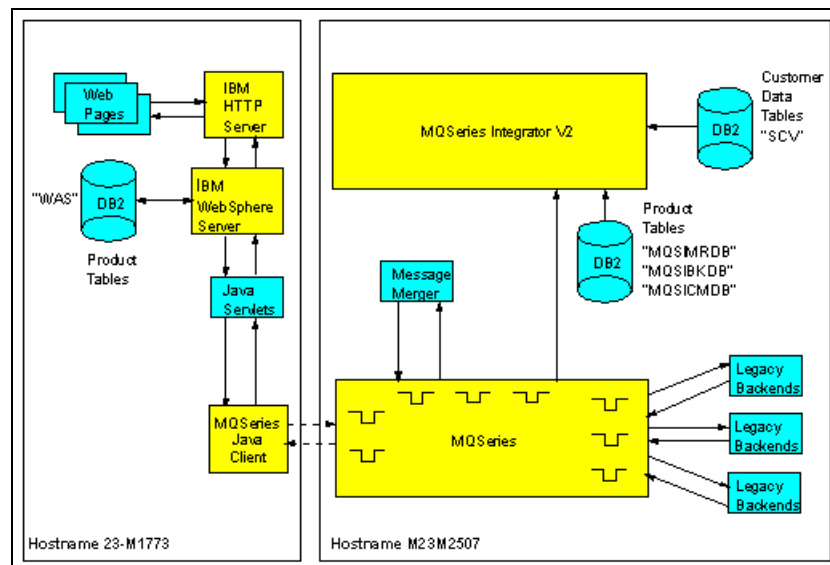


Figure 72. Component overview

The environment used for development of this application solution consisted of two Netfinity 3000 workstations running Windows NT Workstation V4.0. A list of hardware and software used for this example can be found in Appendix A, "Hardware and software specifications" on page 305.

Information on installing and implementing all the other software components in the solution is outside the scope of this redbook.

## 13.1 Outbound flow

This section describes the outbound flow of the request message from the Web page to the legacy back-end programs. There is a main flow, RB\_SCV\_1, and three sub-flows, RB\_SCV\_Request\_Endow, RB\_SCV\_Request\_House, and RB\_SCV\_Request\_Motor.

### 13.1.1 RB\_SCV\_1message flow

Figure 73 shows the RB\_SCV\_1 message flow.

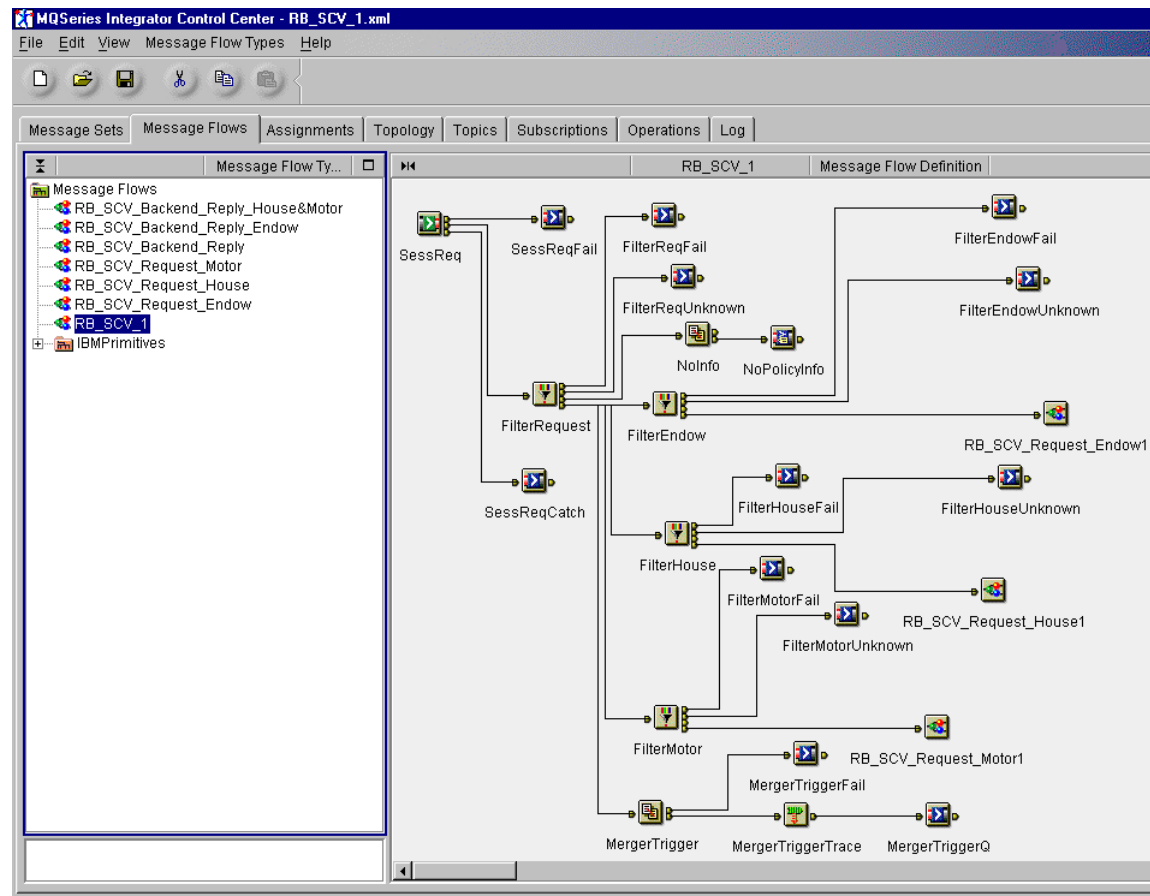


Figure 73. RB\_SCV\_1 message flow

This is the main outbound flow. Its purpose is to:

- Retrieve the request message(s).

- Determine if information exists for the identified customer
- Determine which back-end application(s) to send requests to
- Send a message to the Message Merger's trigger queue to tell it how many responses to expect and the message ID of the original request so that the response messages can be associated with the correct request message.

The flow was built by dragging the appropriate nodes from the list of IBMPrimitives and then wiring them together. The properties of each of the nodes will be discussed in the order in which they appear in the flow, top to bottom, left to right, starting with the MQInput node SessReq. Only the significant nodes, those that transform the message or in some way affect how it is processed, will be discussed in detail. The first time a type of node (for example, Compute node) is presented, a step-by-step review of configuring the node will be covered. Subsequent instances of that node in the flow will list only the configuration values.

Trace nodes and MQOutput nodes that are wired to Failure, Unknown, or Catch terminals of the Compute or Filter nodes were defined and wired to facilitate testing and debugging. You could implement this flow with none of the error type terminals wired, but then, the failed message will end up in the queue manager's dead letter queue. Since MQSeries Integrator does not put any feedback code in the dead letter header before it writes the message there, the only way to find out which node failed will be to review the message flow, turn on the trace, and review the output. This can be tedious and time consuming; so, the recommendation is to wire all possible terminals during the development process.

#### **13.1.1.1 SessReq MQInput node**

The first node in the outbound flow is SessReq, an MQInput node. Its purpose is to retrieve messages from the queue identified in the node's Basic tab, shown in Figure 74 on page 248, and pass them on to the next node in the flow.

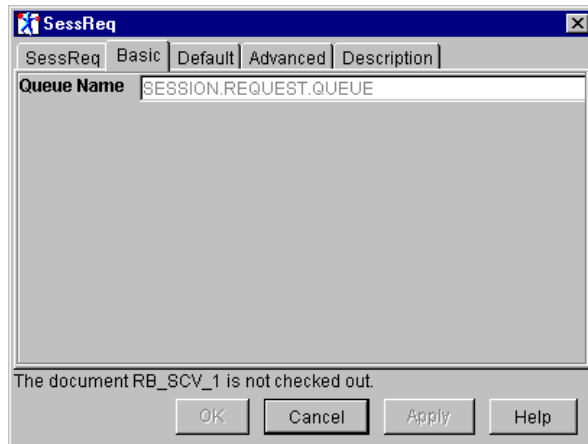


Figure 74. SessReq MQInput node: Basic tab

The Basic tab of the MQInput node identifies the name of the queue that will be monitored for messages. Figure 75 shows the Default tab.

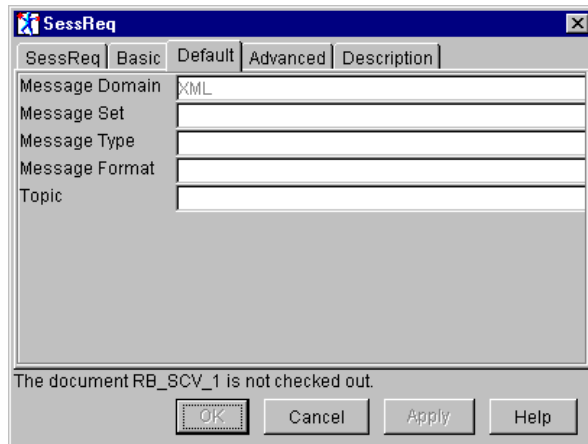


Figure 75. SessReq MQInput node: Default tab

Since the request message on the queue does not have an RFH header, the Message Domain property of the MQInput node's Default tab must be set to XML so that broker knows to use the built-in XML parser to parse the message.



### 13.1.1.2 SessReqFail MQOutput node

This node receives messages that are propagated to the Failure terminal of the SessReq MQInput node. It writes them to the queue identified by the queue manager and queue name specified on the node's Basic tab, shown in Figure 76.

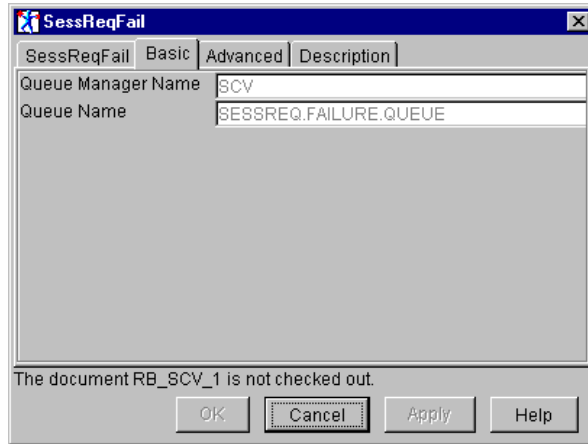


Figure 76. SessReqFail MQOutput node

### 13.1.1.3 FilterRequest Filter node

The purpose of the FilterRequest Filter node is to determine if the customer requested exists in the database. The single customer view sample application uses a simple DB2 database to determine which types of policies the customer owns so that requests for information are only sent to the back-end applications where the customer has policy information. The primary reason for doing this is so that the inbound flow knows how many responses to expect but it would also lead to increased efficiency because unnecessary messages would be eliminated and back-end programs would only be invoked if they contained information for the customer request.

The ESQL, shown in Figure 77 on page 250, is a standard SQL construct to check if the customer number passed in the request message exists on the database.

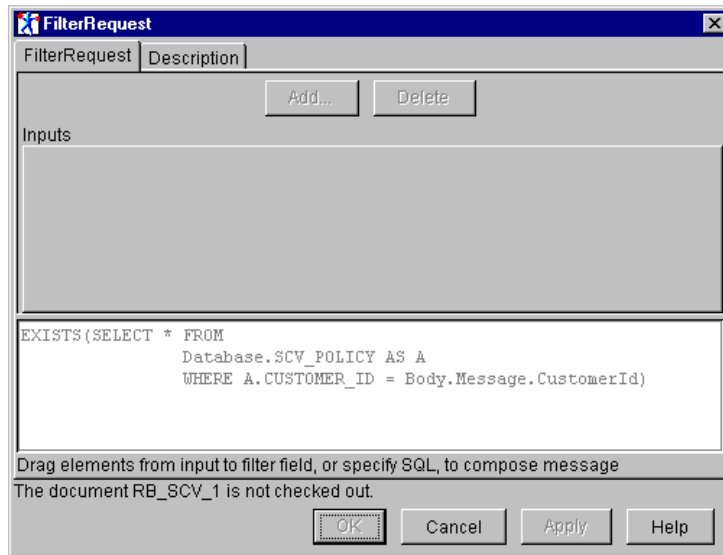


Figure 77. FilterRequest Filter node

If the customer number on the request message exists in the database, the message is propagated to the true node. In Figure 73 on page 246, you will notice that if the message is propagated to the true terminal, it is actually passed to four nodes: FilterEndow, FilterHouse, FilterMotor and MergerTrigger. If the customer number does not exist in the database, the message is propagated to the false node causing a failure message to be written to the reply-to queue identified on the request message.

#### 13.1.1.4 FilterReqFail MQOutput node

This node receives the request message if processing fails in the FilterRequest Filter node. It writes the message to the queue identified by the queue manager and queue name specified on the node's Basic tab as shown in Figure 78 on page 251.

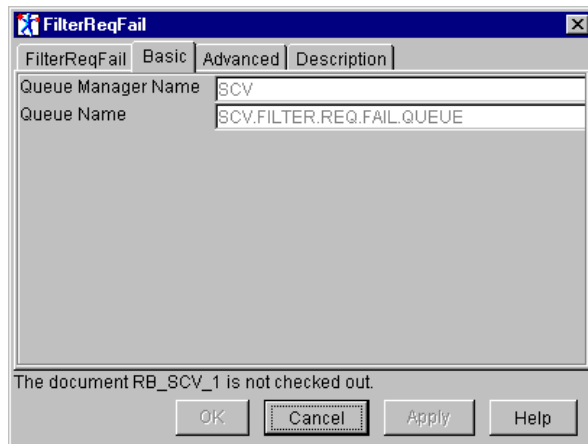


Figure 78. FilterReqFail MQOutput node

#### 13.1.1.5 FilterReqUnknown MQOutput node

This node receives the request message if it is propagated to the Unknown terminal during processing in the FilterRequest Filter node. It writes the message to the queue identified by the queue manager and queue name specified on the node's Basic tab as shown in Figure 79.

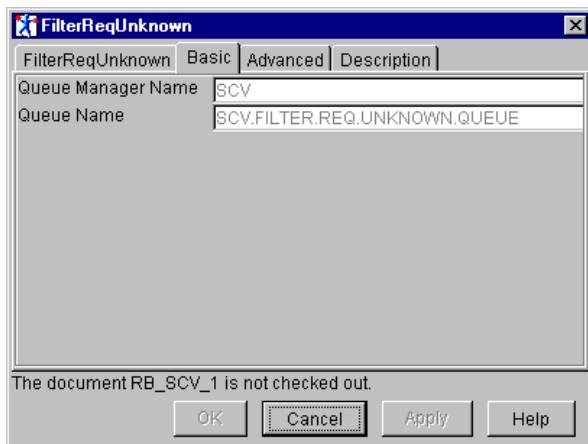


Figure 79. FilterReqUnknown MQOutput node

#### 13.1.1.6 NoInfo Compute node

The request message is passed to this node when it is propagated to the false terminal of the FilterRequest Filter node. A request would be propagated to the false terminal if the customer number on the request

message does not exist on the SCV\_Policy DB2 database table. The message is augmented with a return code and propagated to the output terminal.

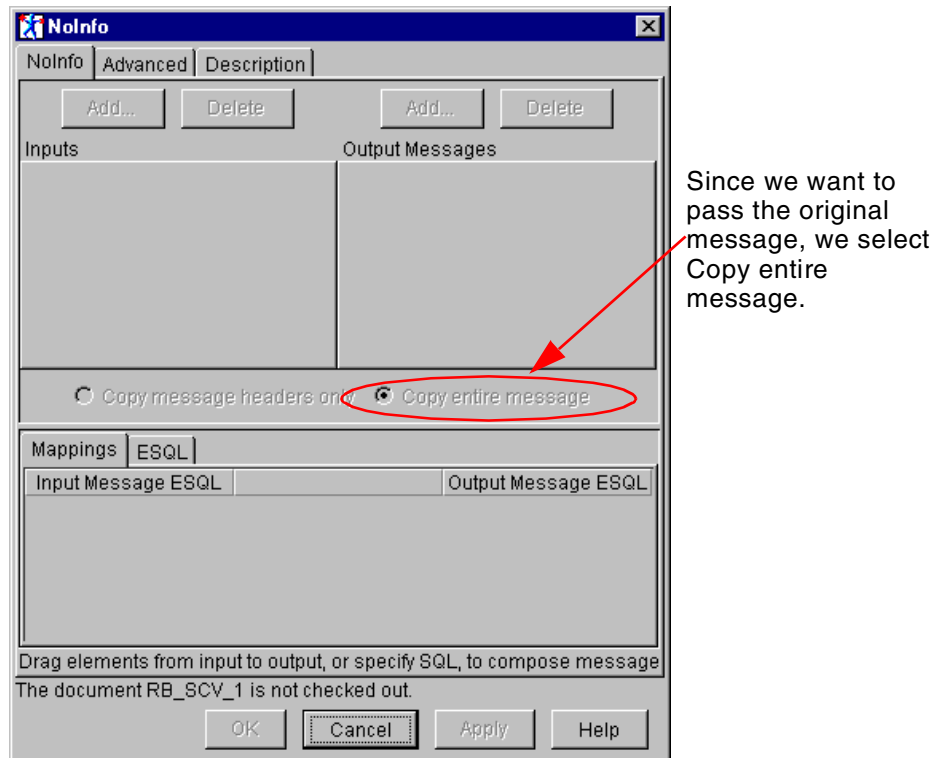
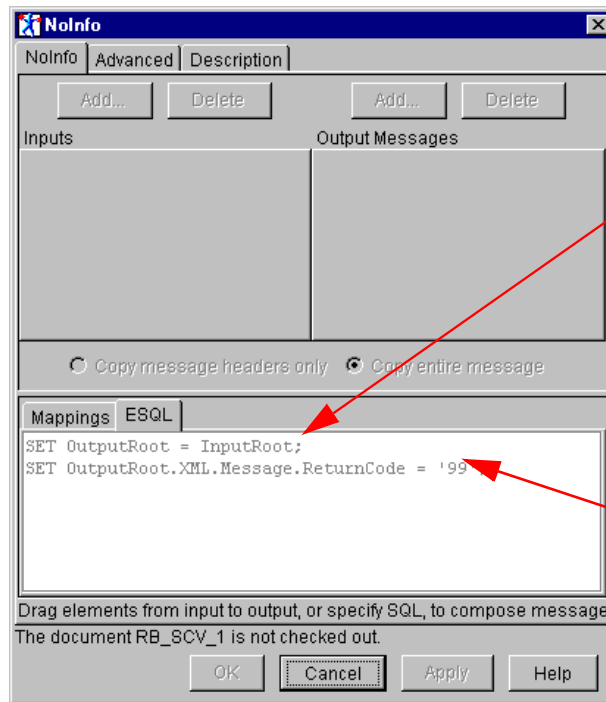


Figure 80. NoInfo Compute node

Figure 81 on page 253 shows the NoInfo Compute node ESQL.



The first line of ESQL is generated for us because Copy entire message was selected.

The second line was inserted manually to add an additional field, ReturnCode, to the message before it is propagated to the node's output terminal.

Figure 81. Noinfo Compute node ESQL

#### 13.1.1.7 NoPolicyInfo MQReply node

This node writes the message passed to it to the reply-to queue identified in the message header.

#### 13.1.1.8 FilterEndow Filter node

Figure 82 on page 254 shows the FilterEndow node.

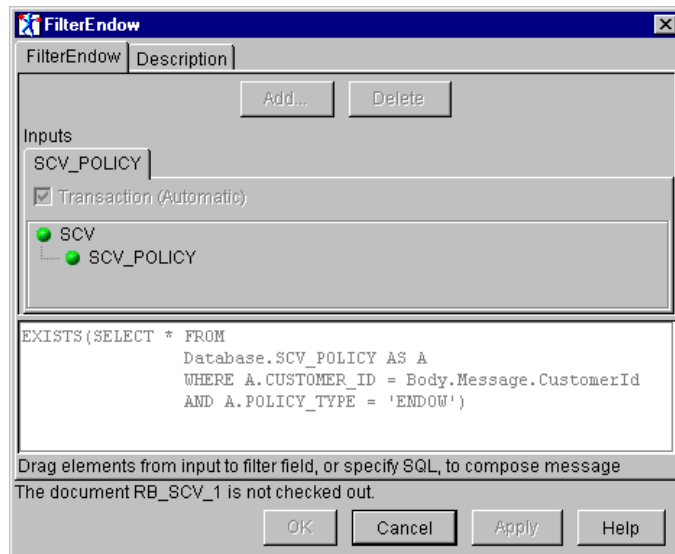


Figure 82. Node: FilterEndow

The purpose of the FilterEndow node is to determine if Endowment policy information exists for the customer. In Figure 73 on page 246, notice that the false terminal is not wired. If no Endowment policy information exists for this customer, message processing for this leg of the flow is abandoned. If information does exist on the database for this customer, the message is propagated to the true terminal, which is wired to a sub-flow called RB\_SCV\_Request\_Endow. This sub-flow is discussed in Section 13.1.2, “RB\_SCV\_Request\_Endow message flow” on page 267.

#### 13.1.1.9 FilterEndowFail MQOutput node

This node receives a request message that fails processing in the FilterEndow node and has been propagated to the Failure terminal of that node. It writes the message to the queue identified by the queue manager and queue name specified on the node’s Basic tab. Figure 83 on page 255 shows the FilterEndowFail MQOutput node.

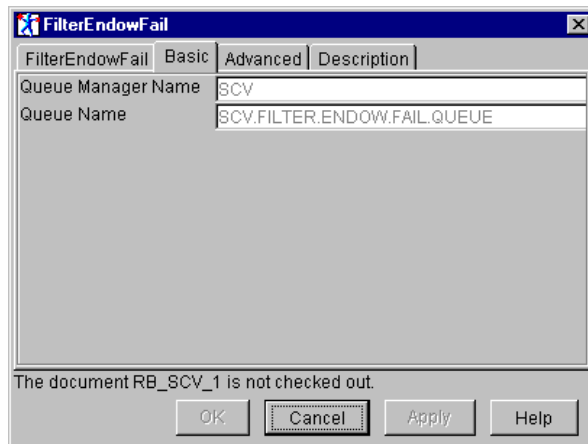


Figure 83. FilterEndowFail MQOutput node

#### 13.1.1.10 FilterEndowUnknown node

This node receives a request message that fails processing in the FilterEndow node and has been propagated to the Failure terminal of that node. It writes the message to the queue identified by the queue manager and queue name specified on the node's Basic tab. Figure 84 shows the FilterEndowUnknown MQOutput node.

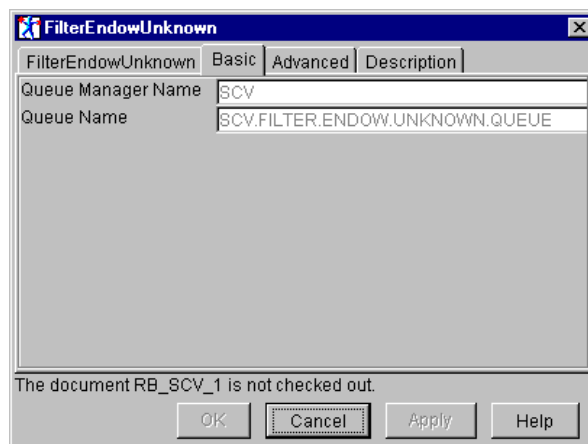


Figure 84. FilterEndowUnknown MQOutput node

#### 13.1.1.11 RB\_SCV\_Request\_Endow1

This is actually the connection to a sub-flow, RB\_SCV\_Request\_Endow. It has no properties. You add the flow to the palette by dragging onto the palette

from the list of message flows on the left hand side of the window. Because the sub-flow has been created with an Input Terminal node, there will be an input terminal on the icon that you can then wire to (an output terminal of) another node in the flow. In this sample application, the true terminal of the FilterEndow Filter node is wired to the input terminal of this sub-flow so that the request message will be passed. The nodes that comprise this sub-flow are described in Section 13.1.2, “RB\_SCV\_Request\_Endow message flow” on page 267.

#### 13.1.1.12 FilterHouse Filter node

The purpose of the FilterHouse node is to determine if House policy information exists for the customer. In Figure 73 on page 246, you will notice that the false terminal is not wired. If no House policy information exists for this customer, message processing for this leg of the flow is abandoned. If information does exist on the database for this customer, the message is propagated to the true terminal, which is wired to a sub-flow called RB\_SCV\_Request\_House.

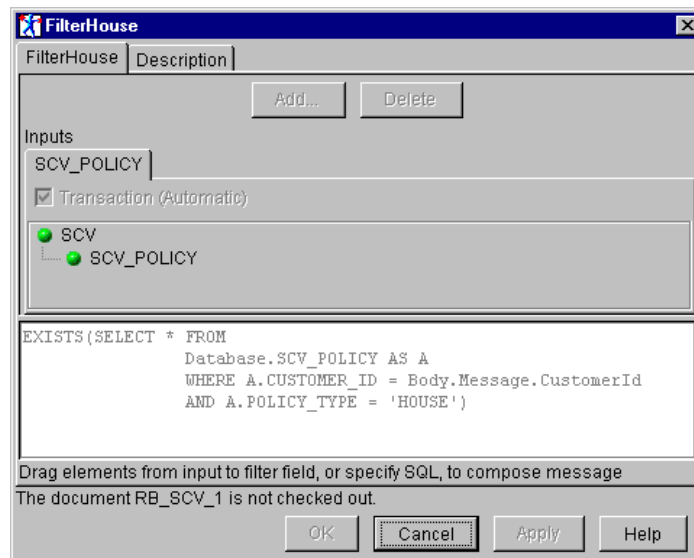


Figure 85. FilterHouse Filter node

#### 13.1.1.13 FilterHouseFail MQOutput node

This node receives a request message that fails processing in the FilterHouse node and has been propagated to the Failure terminal of that node. It writes the message to the queue identified by the queue manager and queue name specified on the node's Basic tab.



Figure 86 shows the FilterHouseFail MQOutput node.

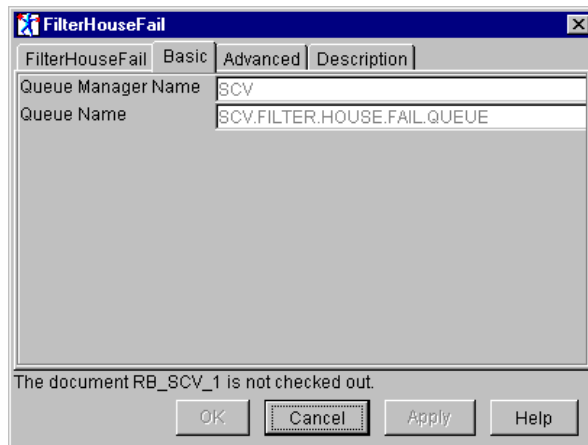


Figure 86. FilterHouseFail MQOutput node

#### 13.1.1.14 FilterHouseUnknown MQOutput node

This node, shown in Figure 87, receives a request message that fails processing in the FilterHouse node and has been propagated to the Unknown terminal of that node. It writes the message to the queue identified by the queue manager and queue name specified on the node's Basic tab.

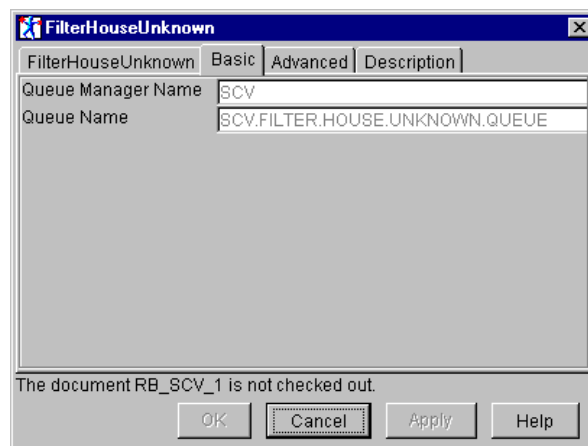


Figure 87. FilterHouseUnknown MQOutput node

#### 13.1.1.15 RB\_SCV\_Request\_House1

This is actually the connection to a sub-flow, RB\_SCV\_Request\_House. It has no properties. You add the flow to the palette by dragging onto the palette from the list of message flows on the left hand side of the window. Because the sub-flow has been created with an Input Terminal node, there will be an input terminal on the icon that you can then wire to (an output terminal of) another node in the flow. In this sample application, the true terminal if the FilterHouse Filter node is wired to the input terminal of this sub-flow so the request message will be passed. The nodes that comprise this sub-flow are described in Section 13.1.3, “RB\_SCV\_Request\_House message flow” on page 290.

#### 13.1.1.16 FilterMotor Filter node

The purpose of the FilterMotor node is to determine if Motor policy information exists for the customer. In Figure 73 on page 246, you will notice that the false terminal is not wired. If no Motor policy information exists for this customer, message processing for this leg of the flow is abandoned. If information does exist on the database for this customer, the message is propagated to the true terminal, which is wired to a sub-flow called RB\_SCV\_Request\_Motor. Figure 88 shows the FilterMotor Filter node.

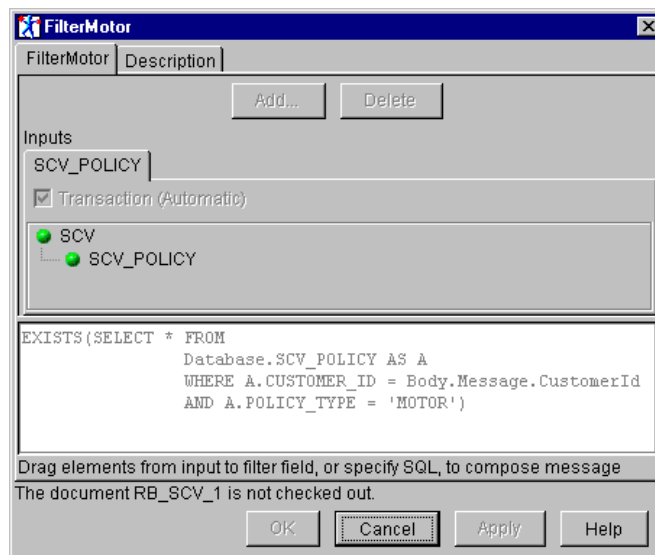


Figure 88. FilterMotor Filter node

#### 13.1.1.17 FilterMotorFail MQOutput node

This node receives a request message that fails processing in the FilterHouse node and has been propagated to the Failure terminal of that node. It writes the message to the queue identified by the queue manager and queue name specified on the node's Basic tab. Figure 89 shows the FilterMotorFail MQOutput node.

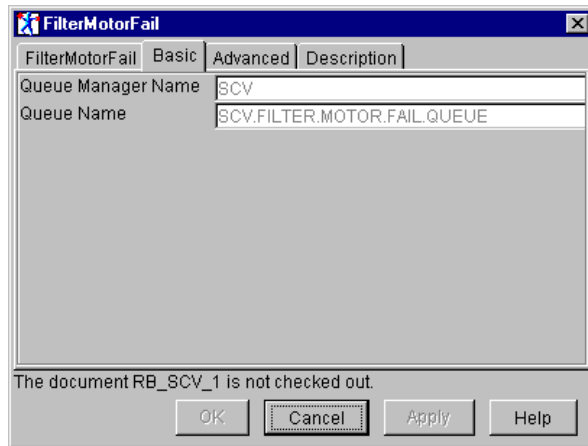


Figure 89. FilterMotorFail MQOutput node

#### 13.1.1.18 FilterMotorUnknown MQOutput node

This node receives a request message that fails processing in the FilterHouse node and has been propagated to the Unknown terminal of that node. It writes the message to the queue identified by the queue manager and queue name specified on the node's Basic tab. Figure 90 on page 260 shows the FilterMotorUnknown MQOutput node.

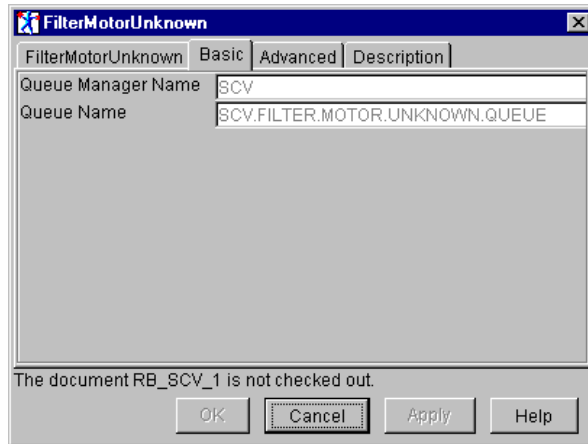


Figure 90. FilterMotorUnknown MQOutput node

#### 13.1.1.19 RB\_SCV\_Request\_Motor1

This is actually the connection to a sub-flow, RB\_SCV\_Request\_Motor. It has no properties. You add the flow to the palette by dragging onto the palette from the list of message flows on the left hand side of the window. Because the sub-flow has been created with an Input Terminal node, there will be an input terminal on the icon that you can then wire to (an output terminal of) another node in the flow. In this sample application, the true terminal of the FilterMotor Filter node is wired to the input terminal of this sub-flow so the request message will be passed. The nodes that comprise this sub-flow are described in Section 13.1.4, “RB\_SCV\_Request\_Motor message flow” on page 293.

#### 13.1.1.20 MergerTrigger Compute node

The MergerTrigger Compute node creates a “trigger” message for the Java applet MessageMerger. This trigger message tells MessageMerger how many response messages to expect and the correlation ID of those messages so that it can match up the back-end response messages to the original request message. Figure 91 on page 261 shows the MergerTrigger Compute node.

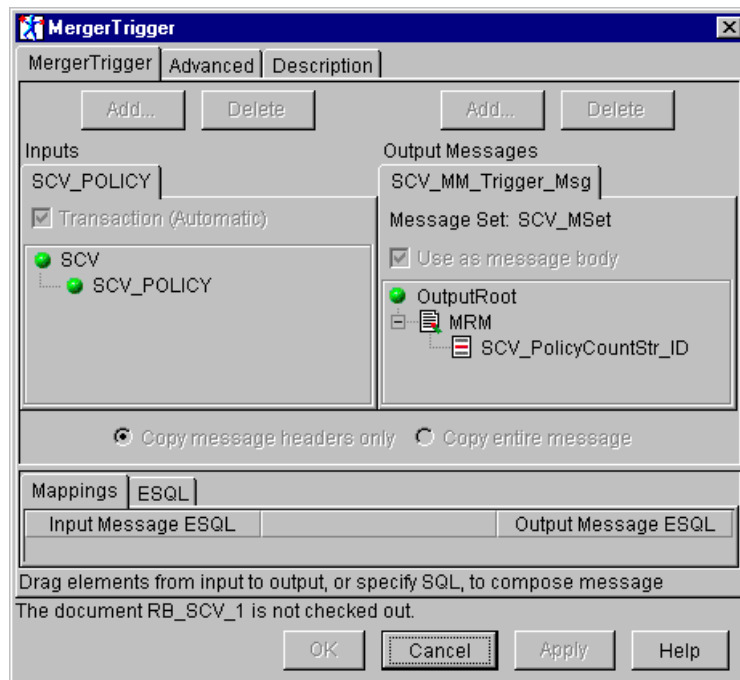
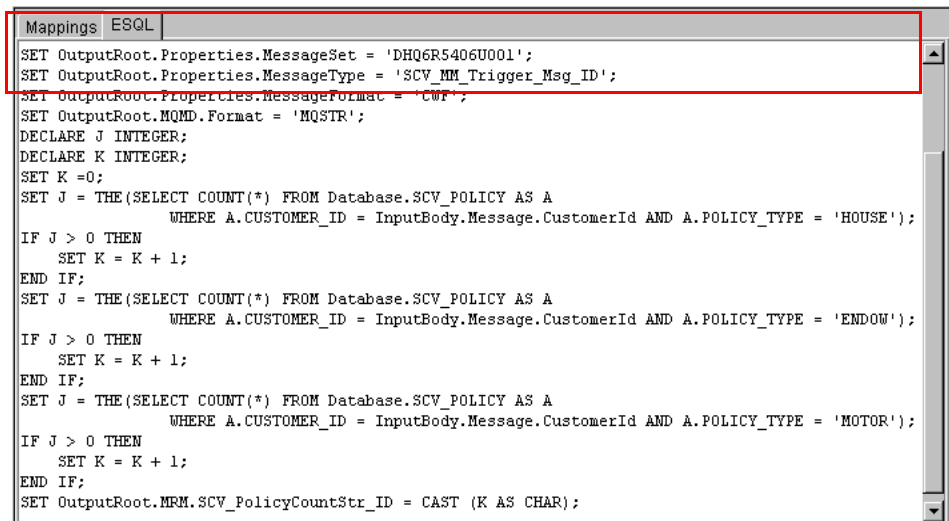


Figure 91. MergerTrigger Compute node

Input to this node is the DB2 database table SCV\_Policy. This datasource is added by clicking the **Add** button and then filling in the Data source and Table name variables. The output from this node is a pre-defined message, SCV\_MM\_Trigger\_Msg in MRM message set SCV\_MSet. Since the output message looks nothing like the input message, we select **Copy message headers only**.

Figure 92 on page 262 shows the ESQL for the MergerTrigger Compute node. Note that the ESQL generated by the Copy message headers only option are “scrolled off” the top of the window and not shown.



```
Mappings ESQL
SET OutputRoot.Properties.MessageSet = 'DHQ6R5406U001';
SET OutputRoot.Properties.MessageType = 'SCV_MM_Trigger_Msg_ID';
SET OutputRoot.Properties.MessageFormat = 'CWF';
SET OutputRoot.MQMD.Format = 'MQSTR';
DECLARE J INTEGER;
DECLARE K INTEGER;
SET K = 0;
SET J = THE(SELECT COUNT(*) FROM Database.SCV_POLICY AS A
             WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId AND A.POLICY_TYPE = 'HOUSE');
IF J > 0 THEN
    SET K = K + 1;
END IF;
SET J = THE(SELECT COUNT(*) FROM Database.SCV_POLICY AS A
             WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId AND A.POLICY_TYPE = 'ENDOW');
IF J > 0 THEN
    SET K = K + 1;
END IF;
SET J = THE(SELECT COUNT(*) FROM Database.SCV_POLICY AS A
             WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId AND A.POLICY_TYPE = 'MOTOR');
IF J > 0 THEN
    SET K = K + 1;
END IF;
SET OutputRoot.MRM.SCV_PolicyCountStr_ID = CAST (K AS CHAR);
```

Figure 92. *MergerTrigger Compute node ESQL*

The first three of the four lines marked in Figure 92 set the properties for the new message. Since the incoming request message is XML and the output message is a legacy format, we need to tell the broker how to parse the new message. The ESQL statements marked in Figure 93 on page 263 declare two integer variables for use in determining the number of policies of a given type that exist for the customer (J) and how many types of policy exist for the customer (K).

```

Mappings ESQLEditor
SET OutputRoot.Properties.MessageSet = 'DHQ6R5406U001';
SET OutputRoot.Properties.MessageType = 'SCV_MM_Trigger_Msg_ID';
SET OutputRoot.Properties.MessageFormat = 'CUF';
SET OutputRoot.MQMD.Format = 'MQSTR';
DECLARE J INTEGER;
DECLARE K INTEGER;
SET K = 0;
SET J = THE(SELECT COUNT(*) FROM Database.SCV_POLICY AS A
            WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId AND A.POLICY_TYPE = 'HOUSE');
IF J > 0 THEN
    SET K = K + 1;
END IF;
SET J = THE(SELECT COUNT(*) FROM Database.SCV_POLICY AS A
            WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId AND A.POLICY_TYPE = 'ENDOW');
IF J > 0 THEN
    SET K = K + 1;
END IF;
SET J = THE(SELECT COUNT(*) FROM Database.SCV_POLICY AS A
            WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId AND A.POLICY_TYPE = 'MOTOR');
IF J > 0 THEN
    SET K = K + 1;
END IF;
SET OutputRoot.MRM.SCV_PolicyCountStr_ID = CAST (K AS CHAR);

```

Figure 93. MergeTrigger Compute node ESQLEditor (continued)

The (K) variable is initialized to zero and will be incremented by one for each policy type that exists. The lines of ESQLEditor marked in Figure 94 interrogate the database to determine if any “HOUSE” policies exist for the customer.

```

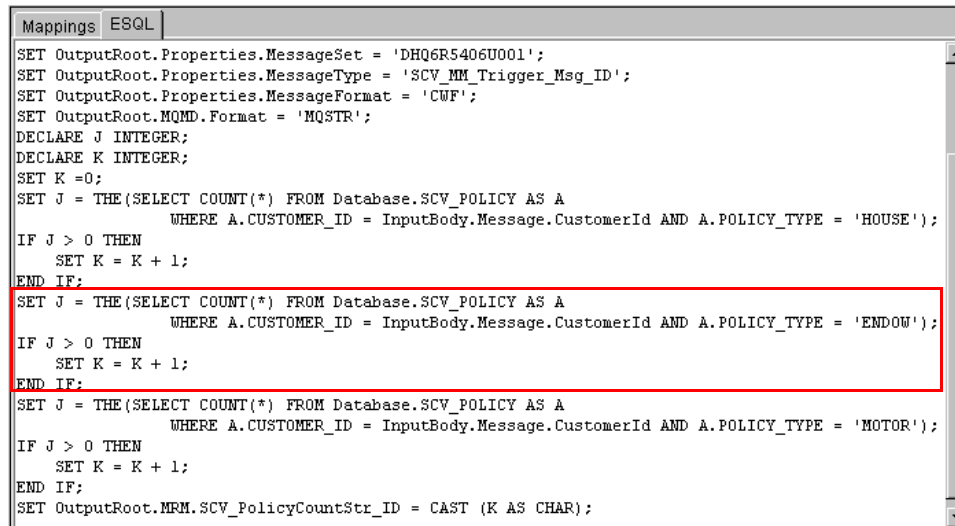
Mappings ESQLEditor
SET OutputRoot.Properties.MessageSet = 'DHQ6R5406U001';
SET OutputRoot.Properties.MessageType = 'SCV_MM_Trigger_Msg_ID';
SET OutputRoot.Properties.MessageFormat = 'CUF';
SET OutputRoot.MQMD.Format = 'MQSTR';
DECLARE J INTEGER;
DECLARE K INTEGER;
SET K = 0;
SET J = THE(SELECT COUNT(*) FROM Database.SCV_POLICY AS A
            WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId AND A.POLICY_TYPE = 'HOUSE');
IF J > 0 THEN
    SET K = K + 1;
END IF;
SET J = THE(SELECT COUNT(*) FROM Database.SCV_POLICY AS A
            WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId AND A.POLICY_TYPE = 'ENDOW');
IF J > 0 THEN
    SET K = K + 1;
END IF;
SET J = THE(SELECT COUNT(*) FROM Database.SCV_POLICY AS A
            WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId AND A.POLICY_TYPE = 'MOTOR');
IF J > 0 THEN
    SET K = K + 1;
END IF;
SET OutputRoot.MRM.SCV_PolicyCountStr_ID = CAST (K AS CHAR);

```

Figure 94. MergeTrigger Compute node ESQLEditor (continued)

If any policies exist then the policy type counter (K) is incremented by 1.

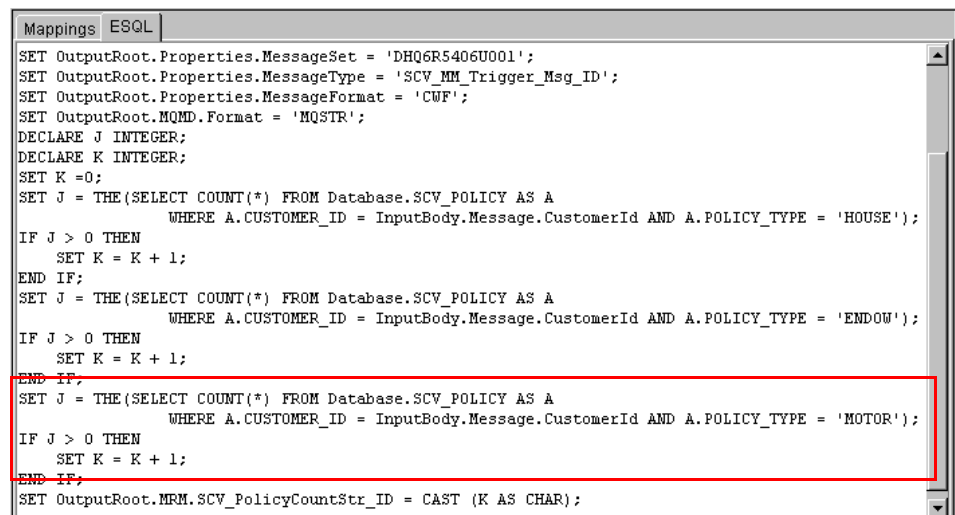
The lines of ESQL marked in Figure 95 interrogate the database to determine if any “ENDOW” policies exist for the customer.



```
Mappings ESQL
SET OutputRoot.Properties.MessageSet = 'DHQ6R5406U001';
SET OutputRoot.Properties.MessageType = 'SCV_MM_Trigger_Msg_ID';
SET OutputRoot.Properties.MessageFormat = 'CWF';
SET OutputRoot.MQMD.Format = 'MQSTR';
DECLARE J INTEGER;
DECLARE K INTEGER;
SET K = 0;
SET J = THE(SELECT COUNT(*) FROM Database.SCV_POLICY AS A
            WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId AND A.POLICY_TYPE = 'HOUSE');
IF J > 0 THEN
    SET K = K + 1;
END IF;
SET J = THE(SELECT COUNT(*) FROM Database.SCV_POLICY AS A
            WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId AND A.POLICY_TYPE = 'ENDOW');
IF J > 0 THEN
    SET K = K + 1;
END IF;
SET J = THE(SELECT COUNT(*) FROM Database.SCV_POLICY AS A
            WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId AND A.POLICY_TYPE = 'MOTOR');
IF J > 0 THEN
    SET K = K + 1;
END IF;
SET OutputRoot.MRM.SCV_PolicyCountStr_ID = CAST (K AS CHAR);
```

Figure 95. MergeTrigger Compute node ESQL (continued)

If any policies exist, the policy type counter (K) is incremented by 1.

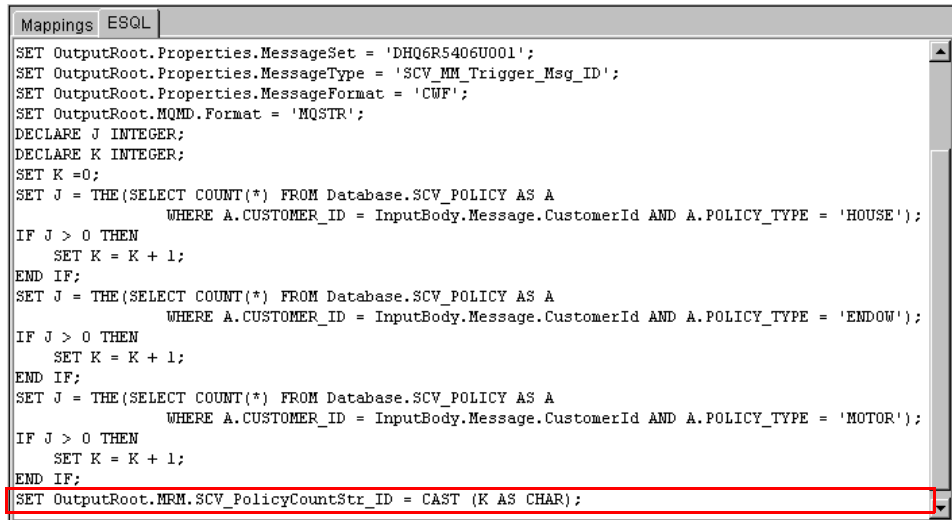


```
Mappings ESQL
SET OutputRoot.Properties.MessageSet = 'DHQ6R5406U001';
SET OutputRoot.Properties.MessageType = 'SCV_MM_Trigger_Msg_ID';
SET OutputRoot.Properties.MessageFormat = 'CWF';
SET OutputRoot.MQMD.Format = 'MQSTR';
DECLARE J INTEGER;
DECLARE K INTEGER;
SET K = 0;
SET J = THE(SELECT COUNT(*) FROM Database.SCV_POLICY AS A
            WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId AND A.POLICY_TYPE = 'HOUSE');
IF J > 0 THEN
    SET K = K + 1;
END IF;
SET J = THE(SELECT COUNT(*) FROM Database.SCV_POLICY AS A
            WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId AND A.POLICY_TYPE = 'ENDOW');
IF J > 0 THEN
    SET K = K + 1;
END IF;
SET J = THE(SELECT COUNT(*) FROM Database.SCV_POLICY AS A
            WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId AND A.POLICY_TYPE = 'MOTOR');
IF J > 0 THEN
    SET K = K + 1;
END IF;
SET OutputRoot.MRM.SCV_PolicyCountStr_ID = CAST (K AS CHAR);
```

Figure 96. MergeTrigger Compute node ESQL (continued)



The lines of ESQL marked in Figure 96 interrogate the database to determine if any “MOTOR” policies exist for the customer. If any policies exist then the policy type counter (K) is incremented by 1.



```
Mappings ESQL
SET OutputRoot.Properties.MessageSet = 'DHQ6R5406U001';
SET OutputRoot.Properties.MessageType = 'SCV_MM_Trigger_Msg_ID';
SET OutputRoot.Properties.MessageFormat = 'CUF';
SET OutputRoot.MQMD.Format = 'MQSTR';
DECLARE J INTEGER;
DECLARE K INTEGER;
SET K = 0;
SET J = THE(SELECT COUNT(*) FROM Database.SCV_POLICY AS A
            WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId AND A.POLICY_TYPE = 'HOUSE');
IF J > 0 THEN
    SET K = K + 1;
END IF;
SET J = THE(SELECT COUNT(*) FROM Database.SCV_POLICY AS A
            WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId AND A.POLICY_TYPE = 'ENDOW');
IF J > 0 THEN
    SET K = K + 1;
END IF;
SET J = THE(SELECT COUNT(*) FROM Database.SCV_POLICY AS A
            WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId AND A.POLICY_TYPE = 'MOTOR');
IF J > 0 THEN
    SET K = K + 1;
END IF;
SET OutputRoot.MRM.SCV_PolicyCountStr_ID = CAST (K AS CHAR);
```

Figure 97. MergeTrigger Compute node ESQL (continued)

The final line of ESQL in the MergeTrigger Compute node sets the SCV\_PolicyCountStr\_ID field in the output message to the value of (K). Since the preceding ESQL has been executed, (K) gets incremented for each policy type that exists for the customer. This correlates to the number of back-end requests for information that will be sent.

#### 13.1.1.21 MergeTriggerFail MQOutput node

This node receives a request message that fails processing in the MergeTrigger Compute node and has been propagated to the Failure terminal of that node. It writes the message to the queue identified by the queue manager and queue name specified on the node's Basic tab.

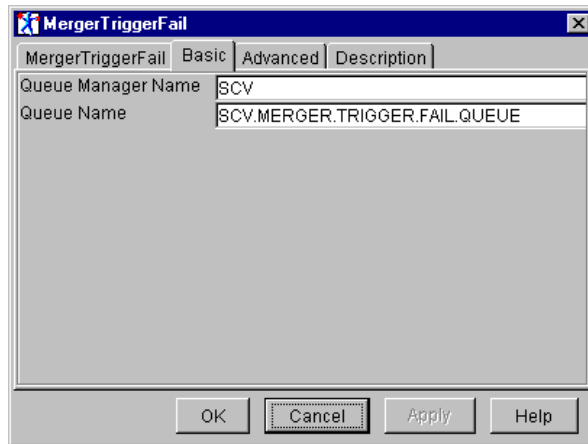


Figure 98. MergerTriggerFail MQOutput node

#### 13.1.1.22 MergerTriggerTrace Trace node

This node, shown in Figure 99, is used to show what the message being passed looks like at this point in the flow. It's used for debugging purposes. For a discussion about configuring a Trace node, please see Section 13.1.2.4, "RequestEndowTrace Trace node" on page 285.

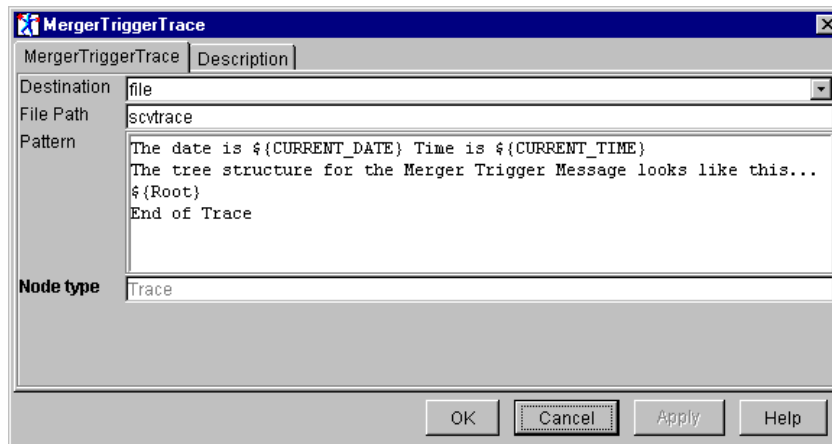


Figure 99. MergerTriggerTrace Trace node

#### 13.1.1.23 MergerTriggerQ MQOutput node

This MQOutput node, shown in Figure 100 on page 267, parses the message according to the properties set in the MergerTrigger Compute node and

writes the message to the queue identified by the queue manager and queue specified on the node's Basic tab.

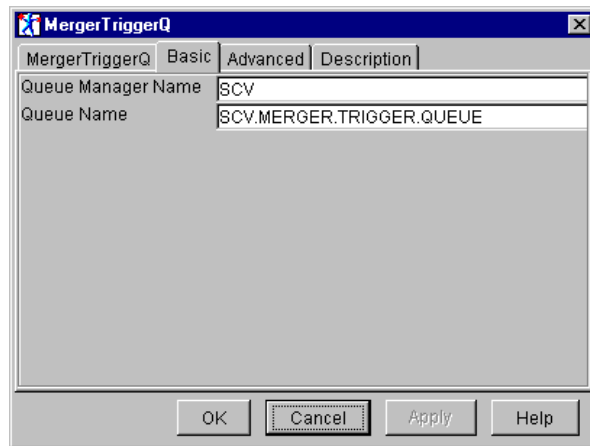


Figure 100. MergerTriggerQ MQOutput node

### 13.1.2 RB\_SCV\_Request\_Endow message flow

Figure 101 on page 268 shows the RB\_SCV\_Request\_Endow message flow.

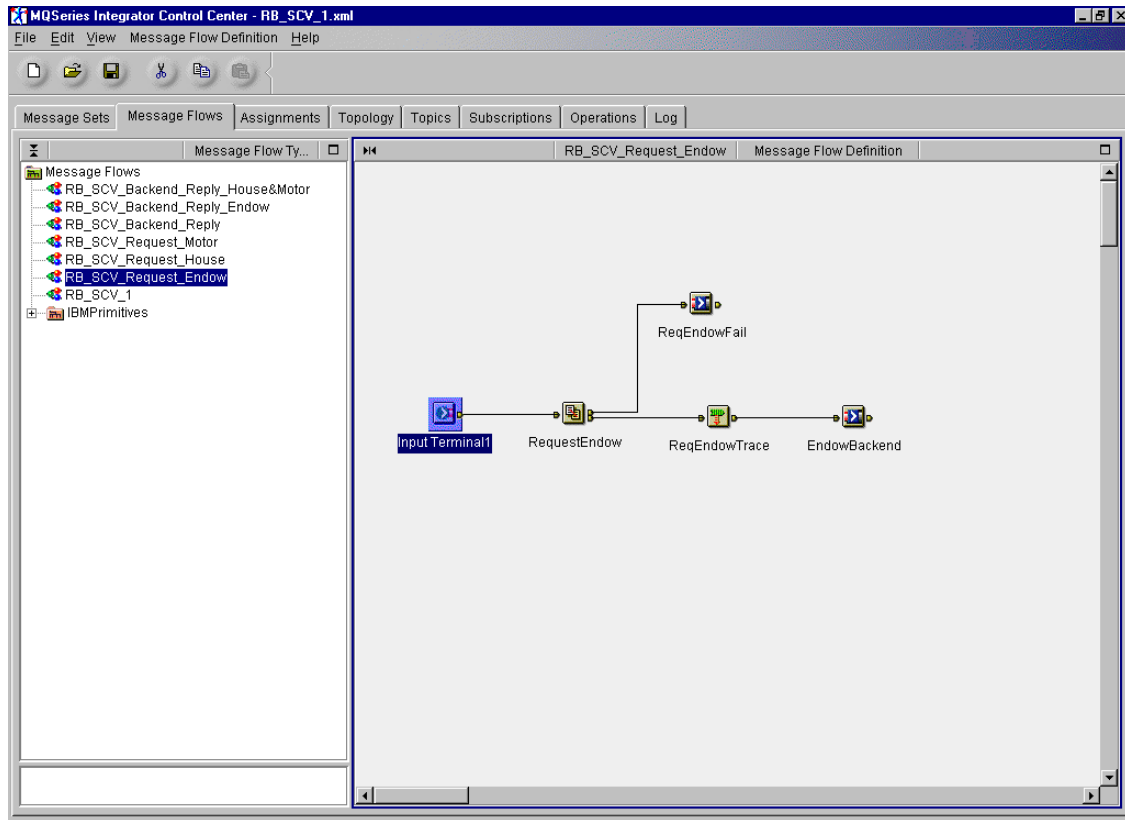


Figure 101. Flow RB\_SCV\_Request\_Endow

The purpose of this “sub-flow” is to augment the request message with information from the database and then reformat it from XML into a format that is expected by the legacy back-end program.

#### 13.1.2.1 InputTerminal1 Input Terminal node

This node has no properties. Its purpose is to provide an input terminal to the RB\_SCV\_Request\_Endow sub-flow. This is how the main flow, RB\_SCV\_1, passes the request message to the sub-flow.

#### 13.1.2.2 RequestEndow Compute node

Figure 102 on page 269 shows the RequestEndow Compute node.

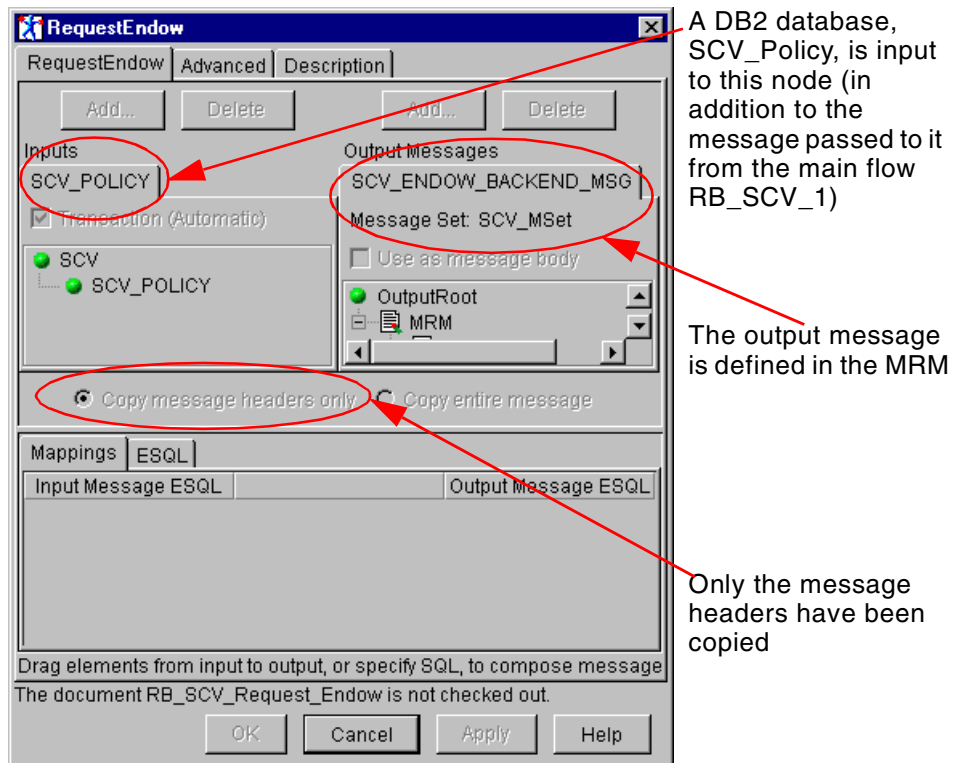


Figure 102. RequestEndow node

The node in Figure 102 was created by dragging a Compute node onto the palette and setting the appropriate properties. The following figures show the steps taken.

After dragging a Compute node onto the palette, right-click the node image and select the **Properties** pull-down as shown in Figure 103.

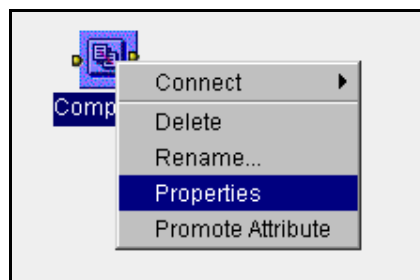


Figure 103. The Compute node Properties pull-down menu

First select **Copy message headers only** as shown in Figure 104.

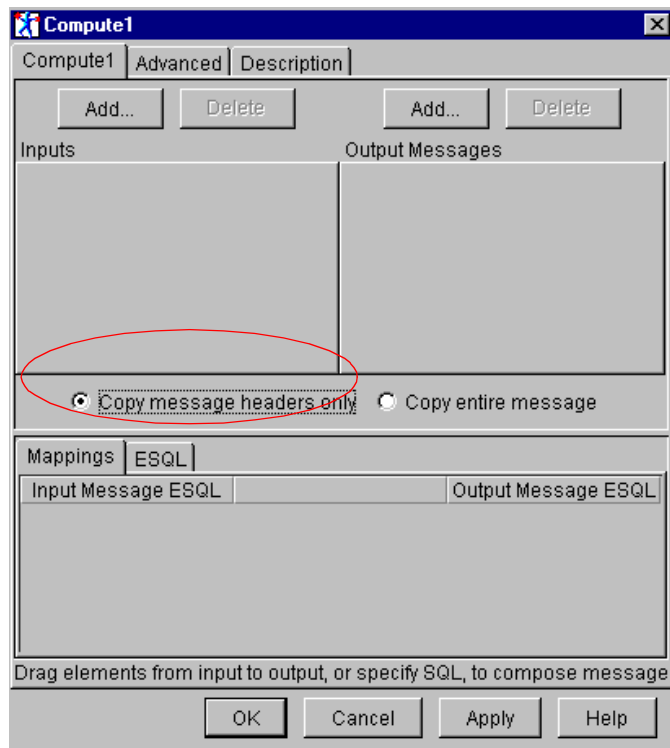


Figure 104. Copying message headers only

This causes the message header information (but no message data) to be copied from the input format to the output format. Since the output message is completely different from the input message, we only want the GUI to prefill the header information in the output message. Figure 105 on page 271 shows the Compute node add input task.

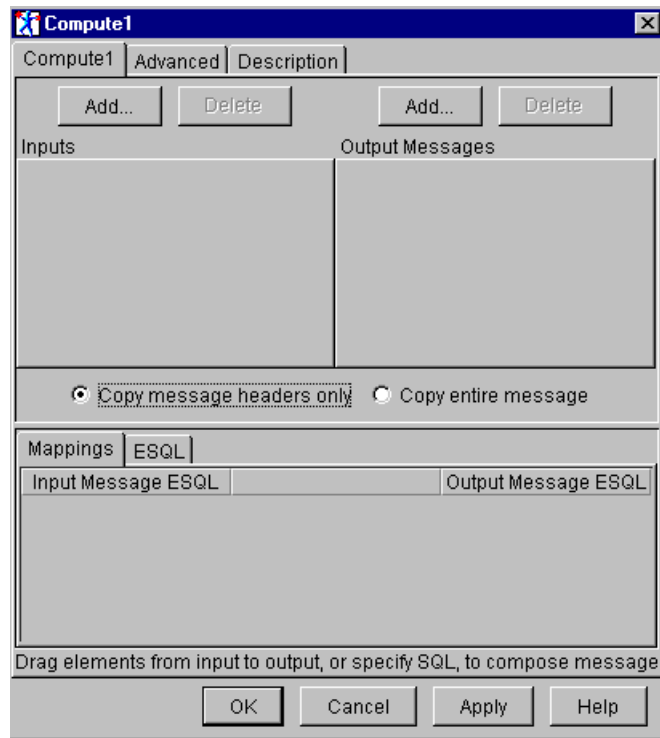


Figure 105. The Compute node add input task

The next step is to add input information to the Compute node. To do this, we click the **Add** button on the Inputs side of the box as shown in Figure 106 on page 272.

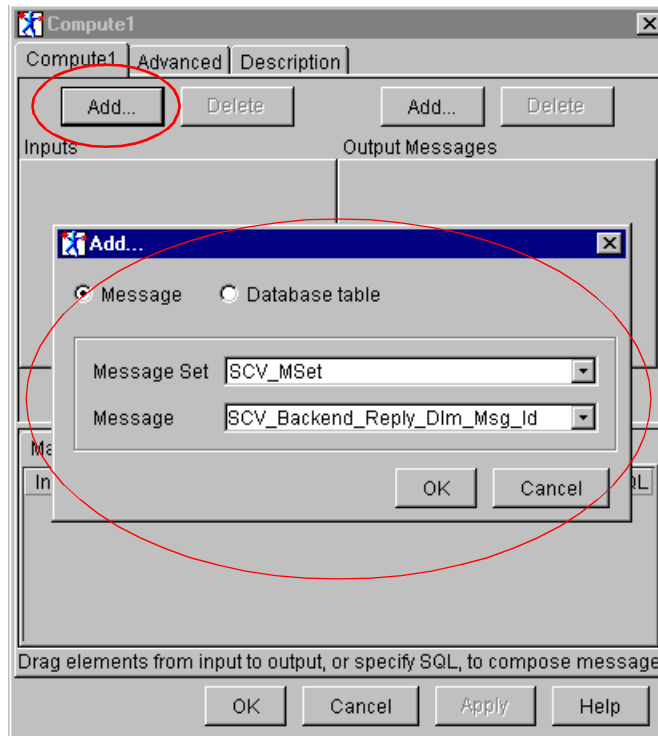


Figure 106. Adding input sources to a Compute node

Clicking the **Add** button causes the Add pop-up window to appear. It is prefilled with message set information from the Message Repository Manager (MRM). Since our input is going to be a database, we select the **Database table** option instead and click the **OK** button as shown in Figure 107 on page 273.



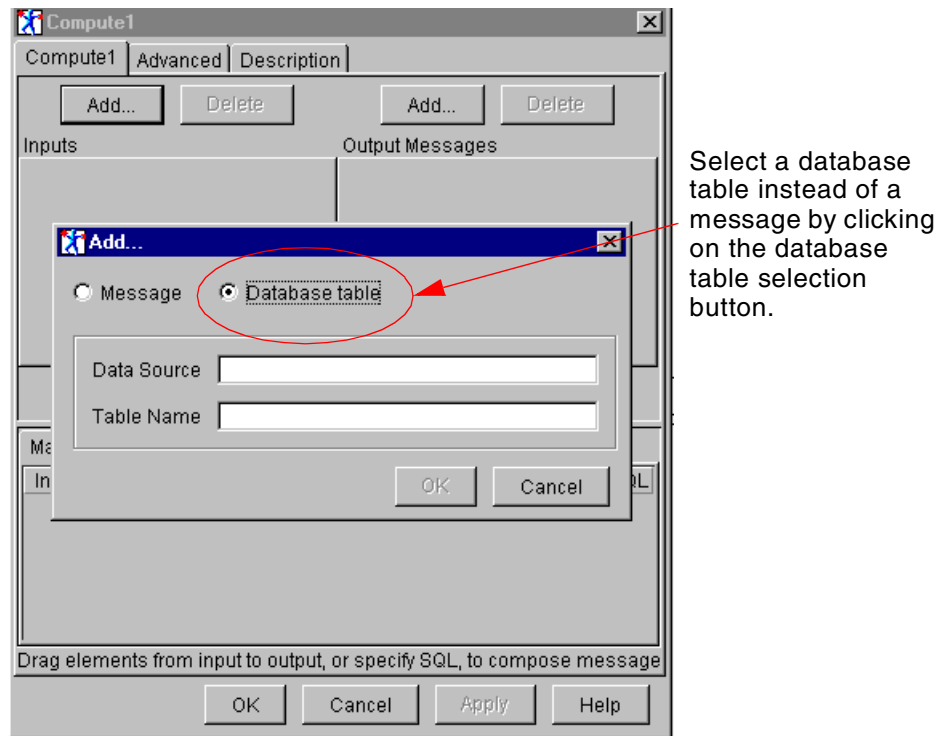
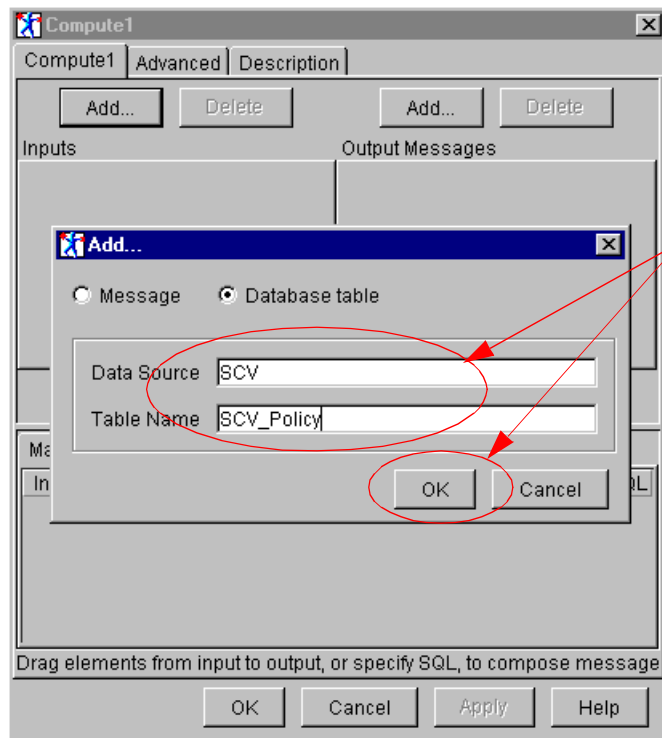


Figure 107. Adding a database table as a Compute node input source

The Data Source and Table Name fields are not prefilled by MQSeries Integrator. You must know the names of the database and tables that are going to be used as input to this Compute node.

In the case of the single customer view sample application, the database we are using is named SCV, and the table name is SCV\_Policy; so, we type those in and click OK as shown in Figure 108 on page 274.



Type in the names of the Data Source (database name) and Table name and click on the **OK** button to add this external database table as an input data source to the compute module.

Figure 108. Entering data source and table names to the Compute node input

Defining an external database as input is now complete. The output message format required by the legacy back-end program has already been defined to the Message Repository Manager so all we have to do is select it. We start the selection process by clicking the **Add** button on the Output Messages side of the box as shown in Figure 109 on page 275.

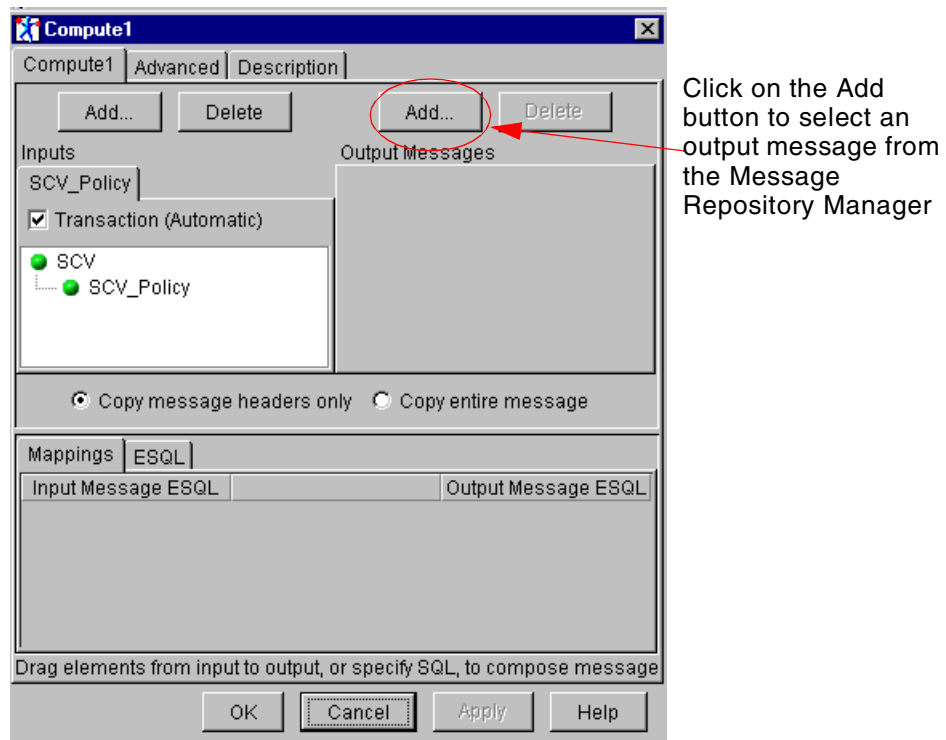


Figure 109. Selecting the MRM output message of the Compute node

Clicking the **Add** button causes the Add pop-up window, shown in Figure 110 on page 276, to appear.



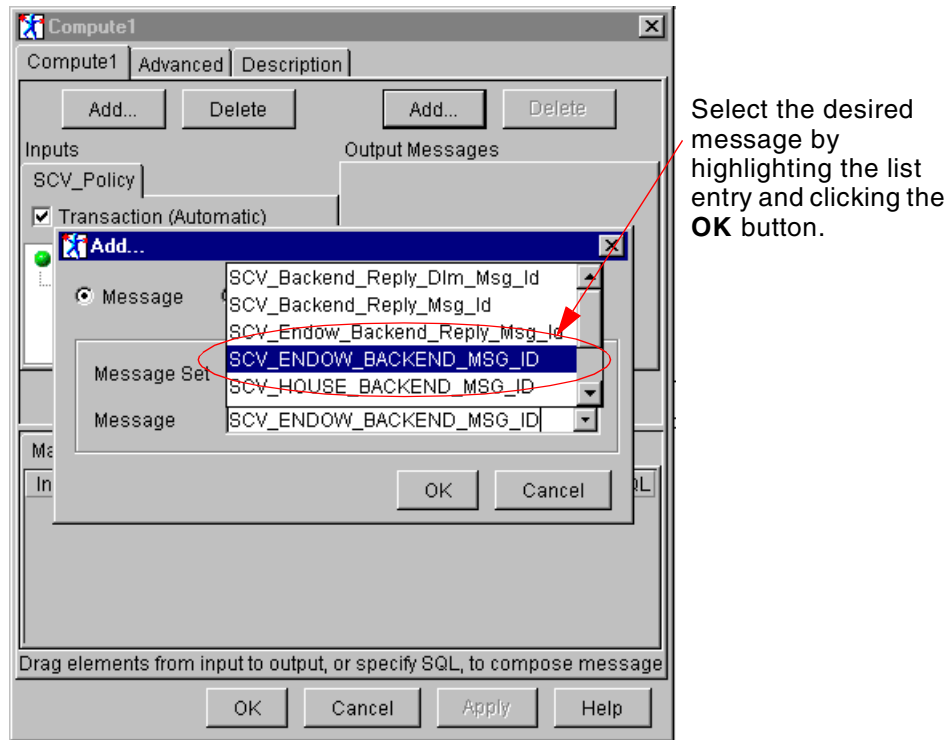


Figure 111. Selecting a Compute node output message from the MRM

Since this message flow is sending a message to the Endowment legacy back-end program, we select the SCV\_ENDOW\_BACKEND\_MSG\_ID.

Figure 112 on page 278 shows what the Compute node looks like so far.

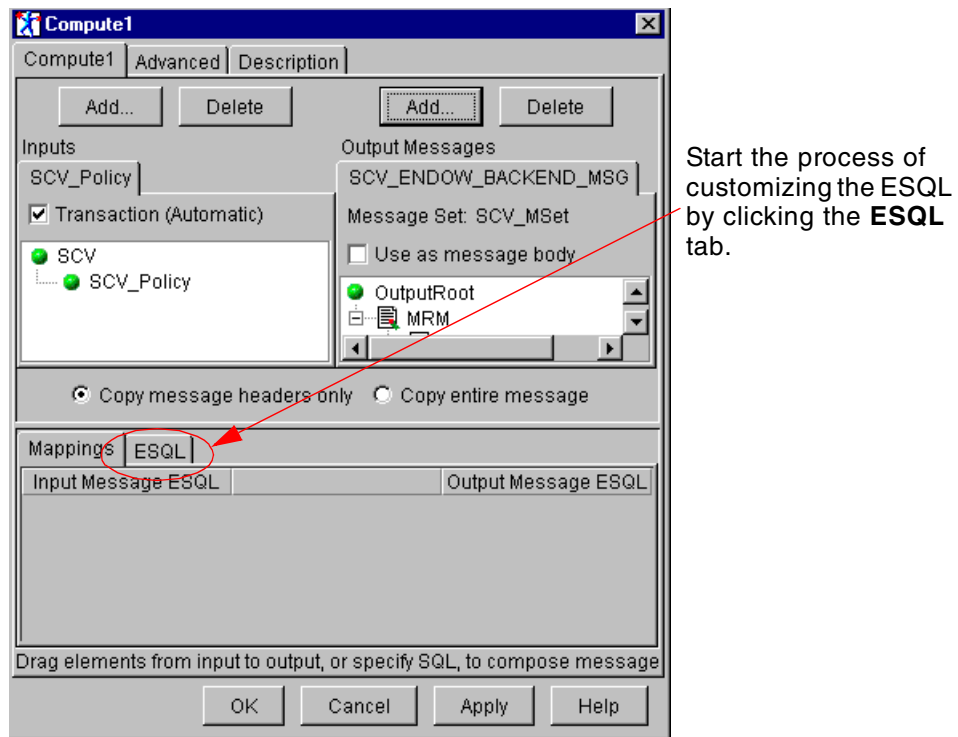
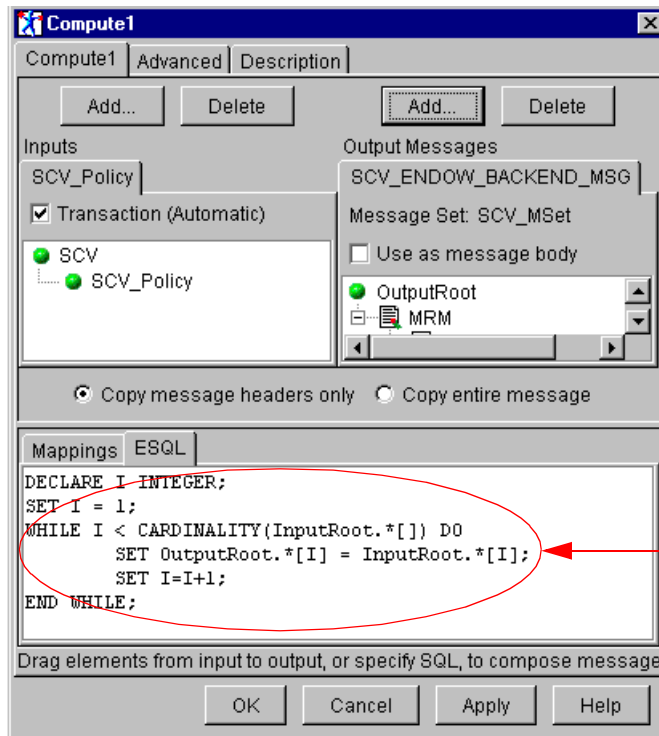


Figure 112. Compute node with inputs and outputs selected

SCV\_Policy has been selected as an input source. The output message is the SCV\_ENDOW\_BACKEND\_MESSAGE from the MRM. We have also selected to copy message headers only. The next step is to start entering the ESQL required to generate the message to be sent to the legacy back-end program. Start this process by clicking the **ESQL** tab shown in Figure 113 on page 279.



This ESQL is generated for you. It causes the message headers to be copied from the input message to the output message.

Figure 113. Compute node, generated ESQL

When you first see the ESQL palette, you'll notice that there is already some ESQL in place. This was generated by selecting **Copy message headers only** in the previous panel, shown in Figure 112 on page 278. We add the ESQL statements to populate SCV\_ENDOW\_BACKEND\_MSG following the pregenerated ESQL statements.

The three lines selected in Figure 114 on page 280 set the properties of the new message.

Mappings	ESQL
<pre> DECLARE I INTEGER; SET I = 1; WHILE I &lt; CARDINALITY(InputRoot.*[]) DO     SET OutputRoot.*[I] = InputRoot.*[I];     SET I=I+1; END WHILE; SET OutputRoot.Properties.MessageSet = 'DHQ6R5406U001'; SET OutputRoot.Properties.MessageType = 'SCV_ENDOW_BACKEND_MSG_ID'; SET OutputRoot.Properties.MessageFormat = 'CWF'; SET OutputRoot.MQMD.Format = 'MQSTR'; SET OutputRoot.MQMD.ReplyToQ = 'SCV.ENDOW.BACKEND.REPLY.QUEUE'; SET OutputRoot.MRM.SCV_Backend_Program_Id = 'ENDOWMNT'; SET OutputRoot.MRM.SCV_Customer_No_Id = InputBody.Message.CustomerId; SET OutputRoot.MRM.SCV_PolicyCountStr_ID =     CAST (THE (SELECT COUNT(*) FROM Database.SCV_POLICY AS A         WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId         AND A.POLICY_TYPE = 'ENDOW') AS CHAR); SET OutputRoot.MRM.SCV_Policy_No_Id[] =     (SELECT ITEM A.POLICY_NO FROM Database.SCV_POLICY AS A         WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId         AND A.POLICY_TYPE = 'ENDOW'); SET I = CAST (OutputRoot.MRM.SCV_PolicyCountStr_ID as INTEGER); WHILE I &lt; 3 DO     SET I=I+1;     SET OutputRoot.MRM.SCV_Policy_No_Id[I] = ' '; END WHILE; </pre>	
Drag elements from input to output, or specify SQL, to compose message	

Figure 114. ESQL to populate SCV\_ENDOW\_BACKEND\_MSG properties

This is required because the incoming message did not have an RFH header. The MessageSet, MessageType and MessageFormat properties are used by the MRM parser to format the output message.

The two lines highlighted in Figure 115 on page 281 set the MQSeries Message Descriptor (MQMD) properties.



Mappings	ESQL
<pre> DECLARE I INTEGER; SET I = 1; WHILE I &lt; CARDINALITY(InputRoot.*[]) DO     SET OutputRoot.*[I] = InputRoot.*[I];     SET I=I+1; END WHILE; SET OutputRoot.Properties.MessageSet = 'DHQ6R5406U001'; SET OutputRoot.Properties.MessageType = 'SCV_ENDOW_BACKEND_MSG_ID'; SET OutputRoot.Properties.MessageFormat = 'CWF'; SET OutputRoot.MQMD.Format = 'MQSTR'; SET OutputRoot.MQMD.ReplyToQ = 'SCV.ENDOW.BACKEND.REPLY.QUEUE'; SET OutputRoot.MRM.SCV_Backend_Program_Id = 'ENDOWMNT'; SET OutputRoot.MRM.SCV_Customer_No_Id = InputBody.Message.CustomerId; SET OutputRoot.MRM.SCV_PolicyCountStr_Id =     CAST (THE (SELECT COUNT(*) FROM Database.SCV_POLICY AS A         WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId         AND A.POLICY_TYPE = 'ENDOW') AS CHAR); SET OutputRoot.MRM.SCV_Policy_No_Id[] =     (SELECT ITEM A.POLICY_NO FROM Database.SCV_POLICY AS A         WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId         AND A.POLICY_TYPE = 'ENDOW'); SET I = CAST (OutputRoot.MRM.SCV_PolicyCountStr_Id as INTEGER); WHILE I &lt; 3 DO     SET I=I+1;     SET OutputRoot.MRM.SCV_Policy_No_Id[I] = ' '; END WHILE; </pre>	
Drag elements from input to output, or specify SQL, to compose message	

Figure 115. ESQL to populate SCV\_ENDOW\_BACKEND\_MSG MQMD

This is where we tell the legacy back-end program the name of the reply-to queue.

The next two lines, selected in Figure 116 on page 282, are simple ESQL statements to set field values in the output message.

```

Mappings ESQL
DECLARE I INTEGER;
SET I = 1;
WHILE I < CARDINALITY(InputRoot.*[]) DO
    SET OutputRoot.*[I] = InputRoot.*[I];
    SET I=I+1;
END WHILE;
SET OutputRoot.Properties.MessageSet = 'DHQ6R5406U001';
SET OutputRoot.Properties.MessageType = 'SCV_ENDOW_BACKEND_MSG_ID';
SET OutputRoot.Properties.MessageFormat = 'CWF';
SET OutputRoot.MQMD.Format = 'MQSTR';
SET OutputRoot.MQMD.ReplyToQ = 'SCV.ENDOW.BACKEND.REPLY.QUEUE';
SET OutputRoot.MRM.SCV_Backend_Program_Id = 'ENDOWMNT';
SET OutputRoot.MRM.SCV_Customer_No_Id = InputBody.Message.CustomerId;
SET OutputRoot.MRM.SCV_PolicyCountStr_ID =
    CAST (THE (SELECT COUNT(*) FROM Database.SCV_POLICY AS A
        WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId
        AND A.POLICY_TYPE = 'ENDOW') AS CHAR);
SET OutputRoot.MRM.SCV_Policy_No_Id[] =
    (SELECT ITEM A.POLICY_NO FROM Database.SCV_POLICY AS A
        WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId
        AND A.POLICY_TYPE = 'ENDOW');
SET I = CAST (OutputRoot.MRM.SCV_PolicyCountStr_ID as INTEGER);
WHILE I < 3 DO
    SET I=I+1;
    SET OutputRoot.MRM.SCV_Policy_No_Id[I] = ' ';
END WHILE;

```

Drag elements from input to output, or specify SQL, to compose message

Figure 116. Compute node simple ESQL

The first statement selected shows setting a literal value. The second statement shows setting the output message field to a value from the input message. You can see the results of these statements in the output from the Trace node shown in the screen on page 289.

The statement selected in Figure 117 on page 283 shows ESQL being used to count how many rows exist on the external database for the customer number and policy type and then assigning that calculated value to one of the fields in the output message.

```

Mappings ESQL
DECLARE I INTEGER;
SET I = 1;
WHILE I < CARDINALITY(InputRoot.*[]) DO
    SET OutputRoot.*[I] = InputRoot.*[I];
    SET I=I+1;
END WHILE;
SET OutputRoot.Properties.MessageSet = 'DHQ6R5406U001';
SET OutputRoot.Properties.MessageType = 'SCV_ENDOW_BACKEND_MSG_ID';
SET OutputRoot.Properties.MessageFormat = 'CWF';
SET OutputRoot.MQMD.Format = 'MQSTR';
SET OutputRoot.MQMD.ReplyToQ = 'SCV.ENDOW.BACKEND.REPLY.QUEUE';
SET OutputRoot.MRM.SCV_Backend_Program_Id = 'ENDOWMNT';
SET OutputRoot.MRM.SCV_Customer_No_Id = InputBody.Message.CustomerId;
SET OutputRoot.MRM.SCV_PolicyCountStr_ID =
    CAST (THE (SELECT COUNT(*) FROM Database.SCV_POLICY AS A
        WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId
        AND A.POLICY_TYPE = 'ENDOW') AS CHAR);
SET OutputRoot.MRM.SCV_Policy_No_Id[] =
    (SELECT ITEM A.POLICY_NO FROM Database.SCV_POLICY AS A
        WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId
        AND A.POLICY_TYPE = 'ENDOW');
SET I = CAST (OutputRoot.MRM.SCV_PolicyCountStr_ID as INTEGER);
WHILE I < 3 DO
    SET I=I+1;
    SET OutputRoot.MRM.SCV_Policy_No_Id[I] = ' ';
END WHILE;

```

Drag elements from input to output, or specify SQL, to compose message

Figure 117. Calculating an output message field value

The CAST function is used because the output message fields are all STRING and the COUNT function returns an integer.

The statement, selected in Figure 118 on page 284, selects all rows from the database that meet the criteria (customer number and policy type) and populate a repeating structure (SCV\_Policy\_No\_Id) with the returned row(s).

Mappings	ESQL
<pre> DECLARE I INTEGER; SET I = 1; WHILE I &lt; CARDINALITY(InputRoot.*[]) DO     SET OutputRoot.*[I] = InputRoot.*[I];     SET I=I+1; END WHILE; SET OutputRoot.Properties.MessageSet = 'DHQ6R5406U001'; SET OutputRoot.Properties.MessageType = 'SCV_ENDOW_BACKEND_MSG_ID'; SET OutputRoot.Properties.MessageFormat = 'CWF'; SET OutputRoot.MQMD.Format = 'MQSTR'; SET OutputRoot.MQMD.ReplyToQ = 'SCV.ENDOW.BACKEND.REPLY.QUEUE'; SET OutputRoot.MRM.SCV_Backend_Program_Id = 'ENDOWMNT'; SET OutputRoot.MRM.SCV_Customer_No_Id = InputBody.Message.CustomerId; SET OutputRoot.MRM.SCV_PolicyCountStr_ID =     CAST (THE (SELECT COUNT(*) FROM Database.SCV_POLICY AS A         WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId         AND A.POLICY_TYPE = 'ENDOW') AS CHAR); SET OutputRoot.MRM.SCV_Policy_No_Id[] =     (SELECT ITEM A.POLICY_NO FROM Database.SCV_POLICY AS A         WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId         AND A.POLICY_TYPE = 'ENDOW'); SET I = CAST (OutputRoot.MRM.SCV_PolicyCountStr_ID as INTEGER); WHILE I &lt; 3 DO     SET I=I+1;     SET OutputRoot.MRM.SCV_Policy_No_Id[I] = ' '; END WHILE; </pre>	
Drag elements from input to output, or specify SQL, to compose message	

Figure 118. Using the Compute node ESQL to populate a list

The repeating structure, SCV\_Policy\_No\_Id, is a fixed array of 3. The statements, selected in Figure 119 on page 285, show filling the “unused” entries in the array with spaces.

```

Mappings ESQL
DECLARE I INTEGER;
SET I = 1;
WHILE I < CARDINALITY(InputRoot.*[]) DO
    SET OutputRoot.*[I] = InputRoot.*[I];
    SET I=I+1;
END WHILE;
SET OutputRoot.Properties.MessageSet = 'DHQ6R5406U001';
SET OutputRoot.Properties.MessageType = 'SCV_ENDOW_BACKEND_MSG_ID';
SET OutputRoot.Properties.MessageFormat = 'CWF';
SET OutputRoot.MQMD.Format = 'MQSTR';
SET OutputRoot.MQMD.ReplyToQ = 'SCV.ENDOW.BACKEND.REPLY.QUEUE';
SET OutputRoot.MRM.SCV_Backend_Program_Id = 'ENDOWMNT';
SET OutputRoot.MRM.SCV_Customer_No_Id = InputBody.Message.CustomerId;
SET OutputRoot.MRM.SCV_PolicyCountStr_ID =
    CAST (THE (SELECT COUNT(*) FROM Database.SCV_POLICY AS A
        WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId
        AND A.POLICY_TYPE = 'ENDOW') AS CHAR);
SET OutputRoot.MRM.SCV_Policy_No_Id[] =
    (SELECT ITEM A.POLICY_NO FROM Database.SCV_POLICY AS A
        WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId
        AND A.POLICY_TYPE = 'ENDOW');
SET I = CAST (OutputRoot.MRM.SCV_PolicyCountStr_ID as INTEGER);
WHILE I < 3 DO
    SET I=I+1;
    SET OutputRoot.MRM.SCV_Policy_No_Id[I] = ' ';
END WHILE;

```

Drag elements from input to output, or specify SQL, to compose message

Figure 119. Initializing unused repeating structure iterations

Previously, we determined how many rows of policy information exist (SCV\_Policy\_CountStr\_Id). We use this to determine the position of the first “unused” entry in the array. We also know that the array is a fixed length of three entries; so, we can easily use a WHILE loop to initialize the remaining unused entries.

### 13.1.2.3 RequestEndowFail MQOutput node

This is an MQOutput node that receives the message if a failure occurs in the RequestEndow Compute node. The only significant properties are the queue manager name and the queue name. These are entered on the node’s Basic tab.

### 13.1.2.4 RequestEndowTrace Trace node

This node is used to show what the message being passed looks like at this point in the flow. It’s used for debugging purposes.

To configure a Trace node, drag a Trace node from the primitives list to the palette as shown in Figure 120 on page 286.

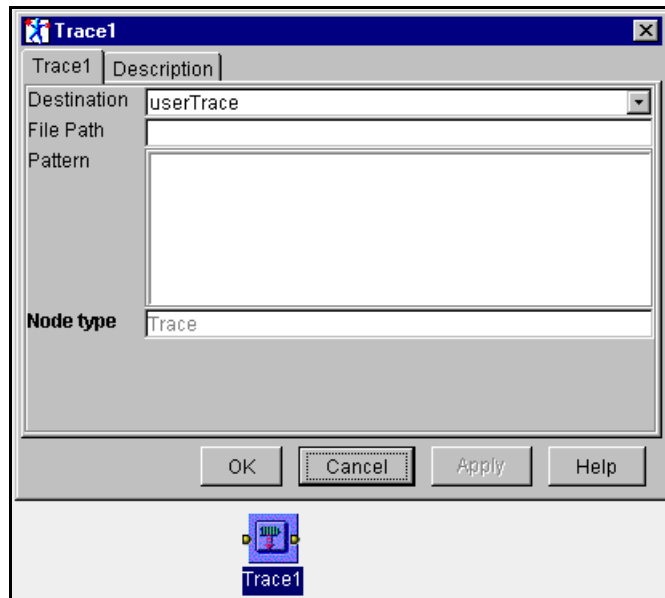
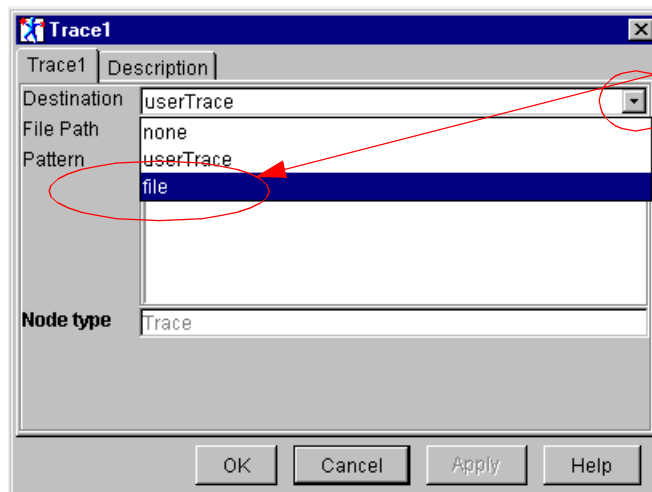


Figure 120. Configuring a Trace node

Select the **Properties** tab from the pop-up menu that is displayed when you right-click the Trace node icon you have just dragged to the palette. Select the **file** destination from the pull-down menu as shown in Figure 121.



Select the **file** destination from the pull-down menu.

Figure 121. Configuring the Trace node: Selecting the trace destination

The default destination of the trace output is the user trace log file. This is not the best place to view the Trace node output as you will need to extract the log, format it and then scan through potentially thousands of lines of output. A better choice is to output the Trace node data to a separate file. To do this you select the file option from the destination pull-down menu as shown above.

The next step is to tell the broker where you want the Trace node output to be sent. In Figure 122, we have specified that we want the Trace node data written to a file, named `scvtrace`.

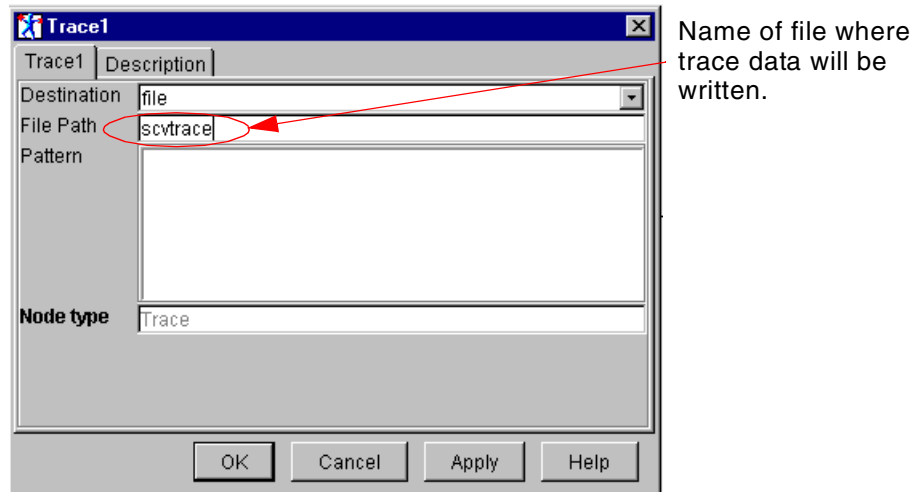


Figure 122. Configuring the Trace node: Identifying the destination file

By default, the file will be created in the `bin` subdirectory of the directory where MQSeries Integrator is installed.

Next, we need to tell the broker what to trace. The pattern property is what tells the broker what to print to the trace file. The significant statement in Figure 123 on page 288 is the third line, `${Root}`.

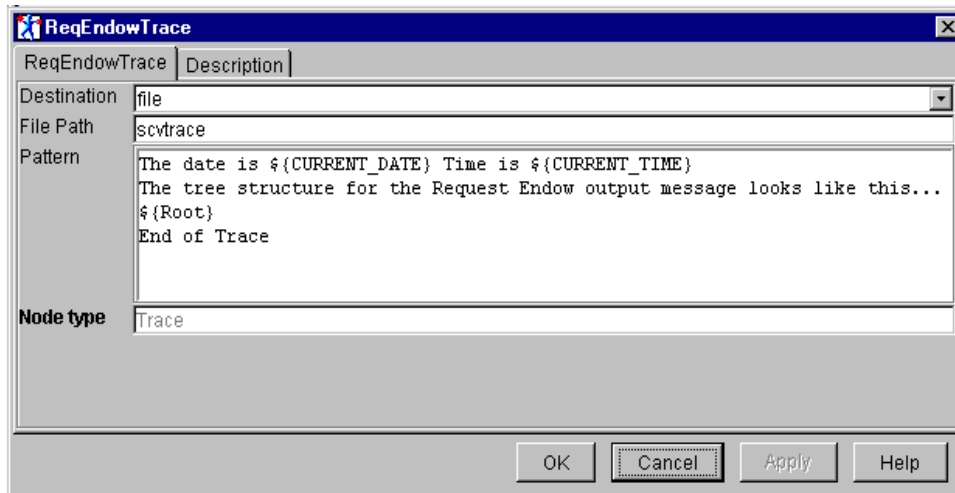


Figure 123. Configuring the Trace node: what to print in the trace

`${Root}` tells the broker that the entire message should be written to the trace file. The following figures show what the broker writes to the trace file when the ReqEndowTrace node is executed. Notice how the first two lines in the pattern are the first two lines in the trace, followed by the message and ending with last line in the pattern box. The only statement in the pattern that is necessary is the `${Root}`, but the rest of the text is descriptive information that identifies what you're looking at and distinguishes between separate trace entries written to the same trace file.





### 13.1.2.5 EndowBackend MQOutput node

This MQOutput node parses the message according to the properties set in the RequestEndow Compute node and writes the message to the queue identified by the queue manager and queue specified on the node's Basic tab.

### 13.1.3 RB\_SCV\_Request\_House message flow

The message flow, shown in Figure 124, is a sub-flow of the RB\_SCV\_1 message flow. It augments the request message passed to it with information from a DB2 database table and reformats it into the format expected by the "House" legacy back-end program.

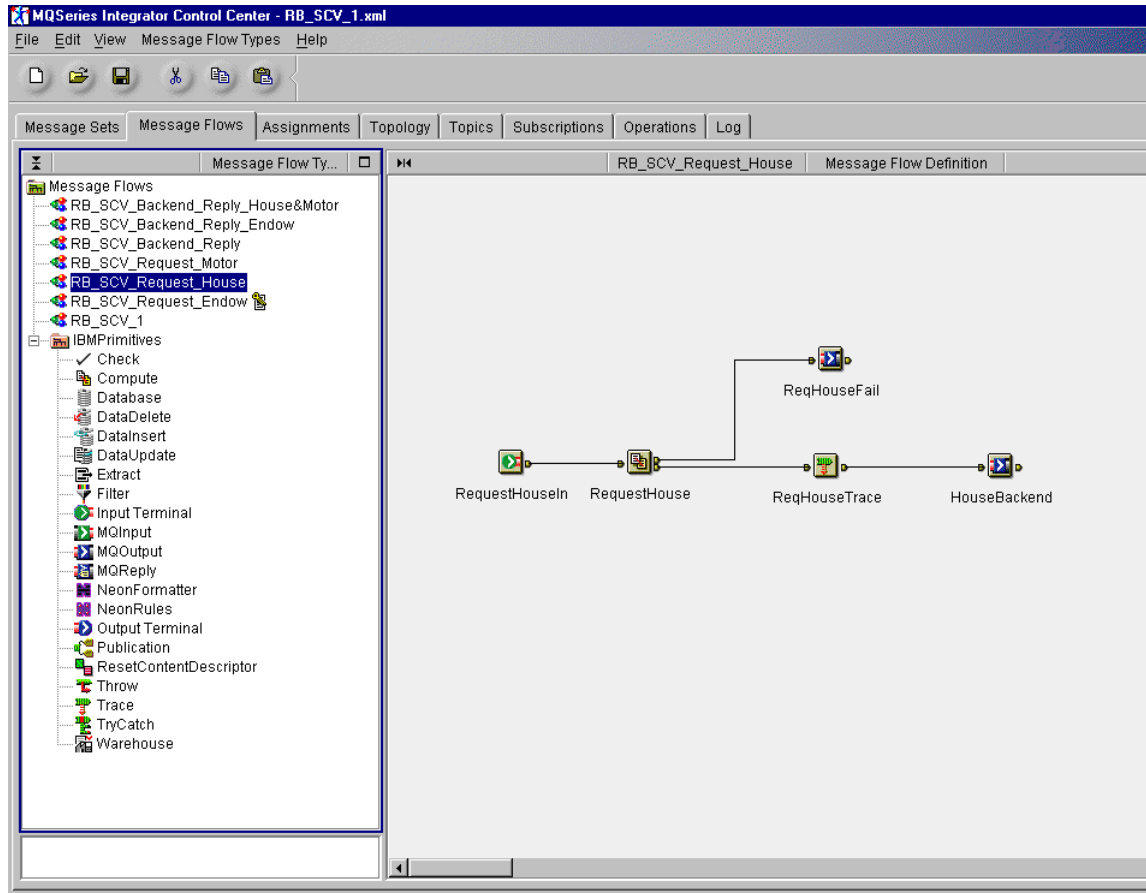


Figure 124. RB\_SCV\_Request\_House message flow

#### 13.1.3.1 RequestHouseIn Input Terminal node

This node has no properties. Its purpose is to provide an input terminal for another message flow to pass a message to this flow.

#### 13.1.3.2 RequestHouse Compute node

This node augments the request message with information from the SCV\_Policy DB2 table and reformats the message from the XML format of the request message to the Custom Wire Format expected by the “House” legacy back-end program.

For step-by-step instructions on how to add a Compute node to a message flow, see Section 13.1.2.2, “RequestEndow Compute node” on page 268.

As with the message created by the RB\_SCV\_Request\_Endow message flow, only message headers are copied from the original request message to the new message created by this node to be sent to the House legacy back-end program. This is specified by clicking **Copy message headers only** as shown in Figure 104 on page 270.

Input to this node is the SCV\_Policy DB2 database table. The same Data Source and Table Name used for the RequestEndow Compute node in the RB\_SCV\_Request\_Endow message flow is used here. For step-by-step instructions on how to specify an input to a Compute node, refer to the text and figures beginning on page 271.

Output from this node is the message, SCV\_HOUSE\_BACKEND\_MSG, in the message set, SCV\_MSet. For step-by-step instructions on how to specify an input to a Compute node, refer to the text and figures beginning on page 275.

The ESQL in this node is exactly the same as the ESQL in the RequestEndow Compute node of the RB\_SCV\_Request\_Endow message flow with a few exceptions. In fact, the ESQL was copied from the RequestEndow Compute node and pasted into this one. The ESQL is shown in Figure 125. The exceptions are indicated.

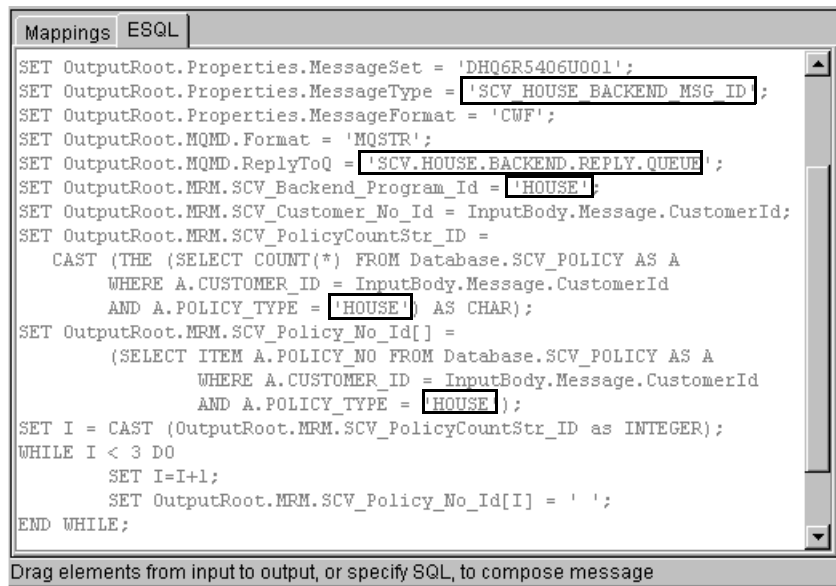


Figure 125. RequestEndow Compute node ESQL

### 13.1.3.3 ReqHouseFail MQOutput node

This node puts messages that fail during processing of the RequestHouse Compute node on the queue identified by the queue manager and queue name values specified on the node's Basic tab as shown in Figure 126.

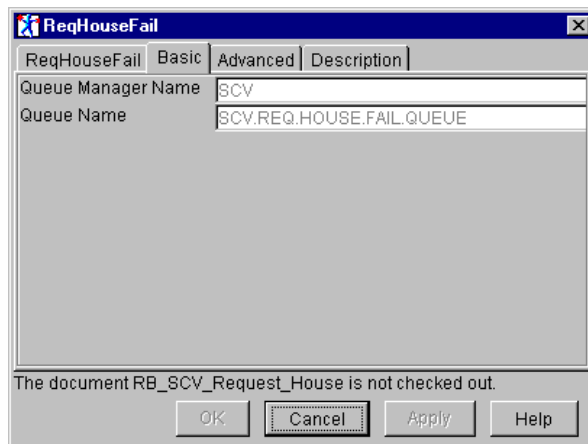


Figure 126. ReqHouseFail MQOutput node

#### 13.1.3.4 ReqHouseTrace trace node

The ReqHouseTrace trace node, shown in Figure 127, prints the message to the destination specified in the Trace node properties.

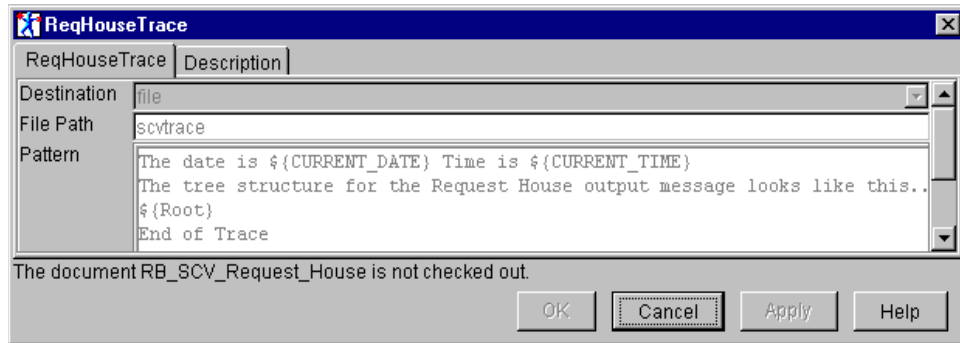


Figure 127. ReqHouseTrace Trace node

#### 13.1.3.5 HouseBackend MQOutput node

The HouseBackend MQOutput node, shown in Figure 128, parses the message according to the message's properties and writes it to the queue identified on the node's Basic tab.

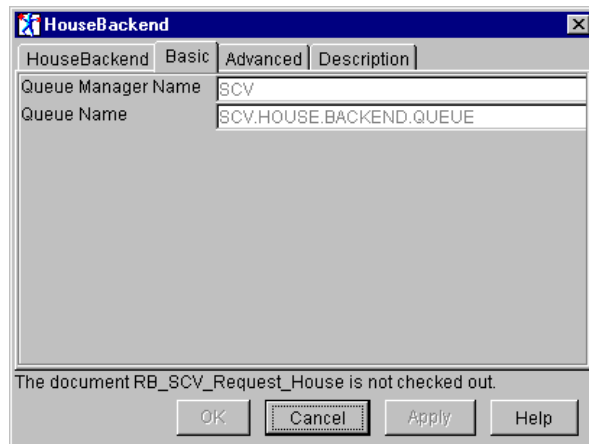


Figure 128. HouseBackend MQOutput node

#### 13.1.4 RB\_SCV\_Request\_Motor message flow

This message flow, shown in Figure 129 on page 294, is a sub-flow of the RB\_SCV\_1 message flow. It augments the request message passed to it with policy number information from a DB2 database table, reformats the

message, and writes it to the MQSeries queue monitored by the Motor legacy back-end program.

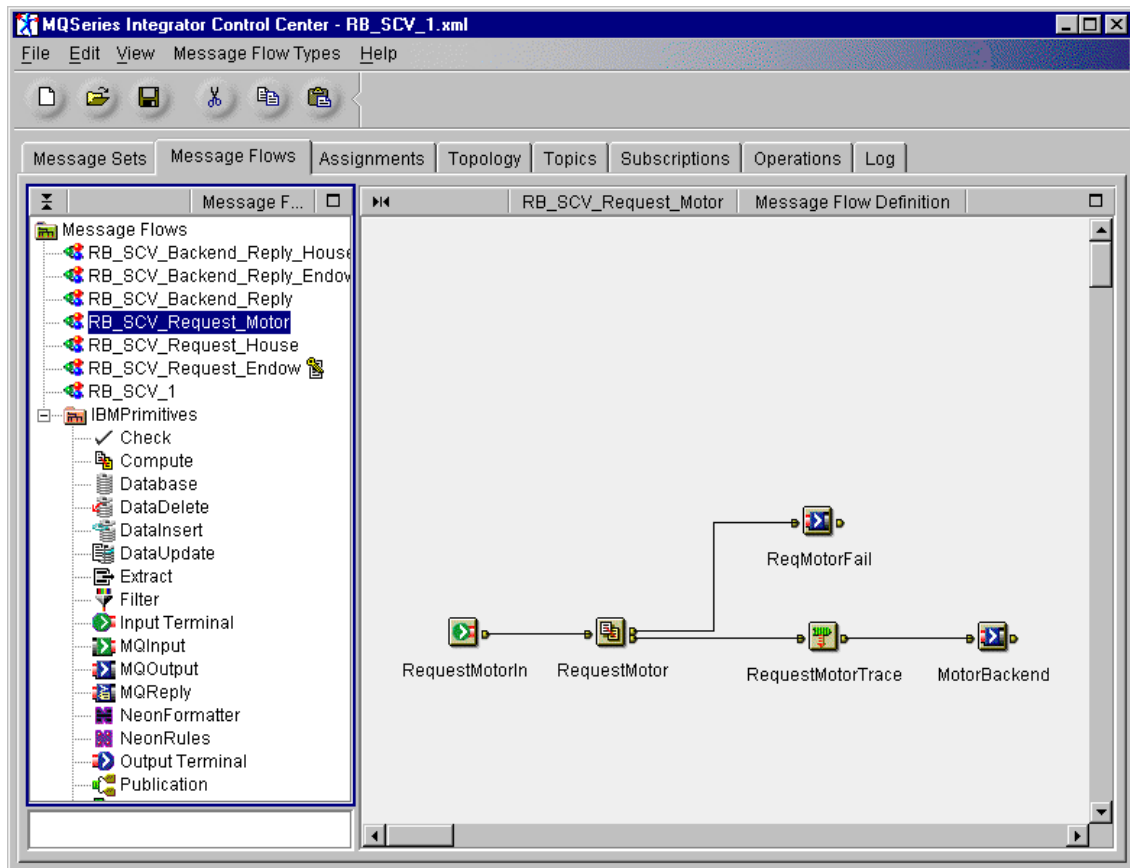


Figure 129. RB\_SCV\_Request\_Motor message flow

#### 13.1.4.1 RequestMotorIn Input Terminal node

This node has no properties. Its purpose is to provide an input terminal to the flow so that a message can be passed in from another flow.

#### 13.1.4.2 RequestMotor Compute node

This node uses information extracted from the SCV DB2 database SCV\_Policy table to augment the request message with a list of policy numbers to be retrieved. It also reformats the message from XML to a customer wire format expected by the legacy back-end program and writes the message to the queue monitored by the Motor legacy back-end program.

For step-by-step instructions on how to add a Compute node to a message flow, see Section 13.1.2.2, “RequestEndow Compute node” on page 268. Figure 130 shows the RequestMotor Compute node properties.

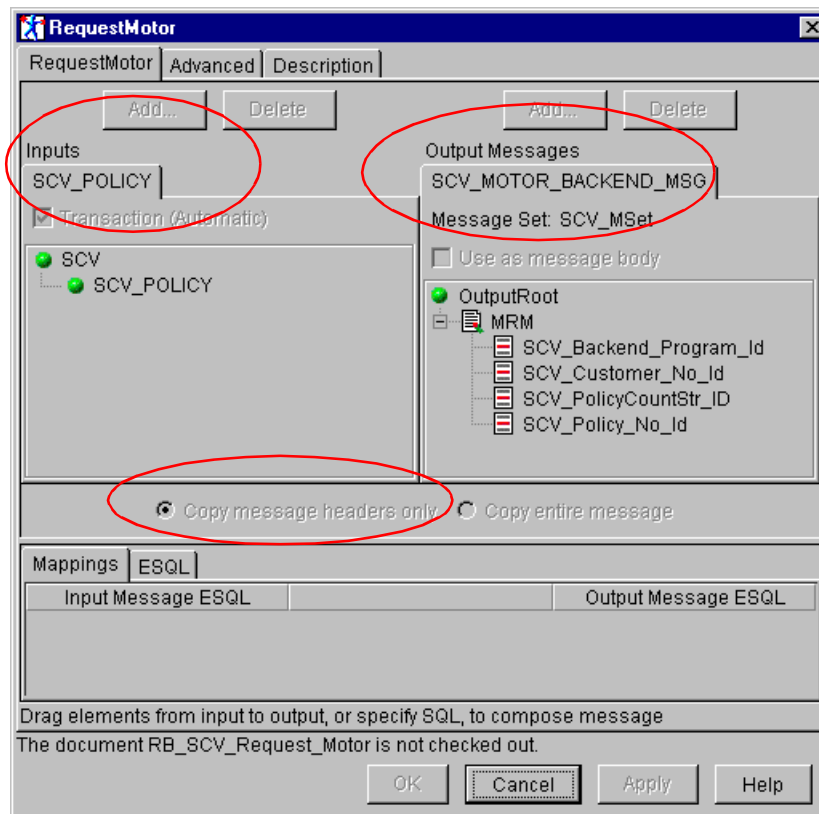


Figure 130. RequestMotor Compute node properties

As can be seen in the fields marked in Figure 130, only message headers are copied from the input message to the output message. A DB2 database table, SCV\_Policy is input to the Compute node, and the output message is defined in the Message Repository Manager (MRM).

The ESQL in this Compute node was copied from the RequestEndow Compute node, and then, the necessary changes were made to tailor the references for a Motor request. Figure 131 on page 296 shows the values that were changed when tailoring the ESQL.

```

Mappings ESQ
DECLARE I INTEGER;
SET I = 1;
WHILE I < CARDINALITY(InputRoot.*[]) DO
    SET OutputRoot.*[I] = InputRoot.*[I];
    SET I=I+1;
END WHILE;
SET OutputRoot.Properties.MessageSet = 'DHQ6R5406U001';
SET OutputRoot.Properties.MessageType = 'SCV_MOTOR_BACKEND_MSG_ID';
SET OutputRoot.Properties.MessageFormat = 'CMF';
SET OutputRoot.MQMD.Format = 'MQSTR';
SET OutputRoot.MQMD.ReplyToQ = 'SCV_MOTOR_BACKEND_REPLY_QUEUE';
SET OutputRoot.MRM.SCV_Backend_Program_Id = 'MOTOR';
SET OutputRoot.MRM.SCV_Customer_No_Id = InputBody.Message.CustomerId;
SET OutputRoot.MRM.SCV_PolicyCountStr_ID =
    CAST (THE (SELECT COUNT(*) FROM Database.SCV_POLICY AS A
        WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId
        AND A.POLICY_TYPE = 'MOTOR') AS CHAR);
SET OutputRoot.MRM.SCV_Policy_No_Id[] =
    (SELECT ITEM A.POLICY_NO FROM Database.SCV_POLICY AS A
        WHERE A.CUSTOMER_ID = InputBody.Message.CustomerId
        AND A.POLICY_TYPE = 'MOTOR');
SET I = CAST (OutputRoot.MRM.SCV_PolicyCountStr_ID as INTEGER);
WHILE I < 3 DO
    SET I=I+1;
    SET OutputRoot.MRM.SCV_Policy_No_Id[I] = ' ';
END WHILE;

```

Figure 131. RequestMotor Compute node ESQ

### 13.1.4.3 ReqMotorFail MQOutput node

This node, shown in Figure 132 on page 297, puts messages that fail during processing of the RequestMotor Compute node on a queue identified by the queue manager and queue name values specified on the node's Basic tab.



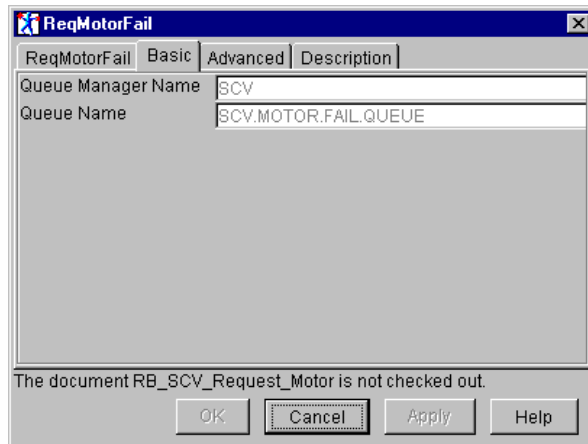


Figure 132. ReqMotorFail MQOutput node

#### 13.1.4.4 RequestMotorTrace Trace node

This Trace node prints the message to the destination specified in the Trace node properties. The `${Root}` statement specifies that the entire message is to be printed.

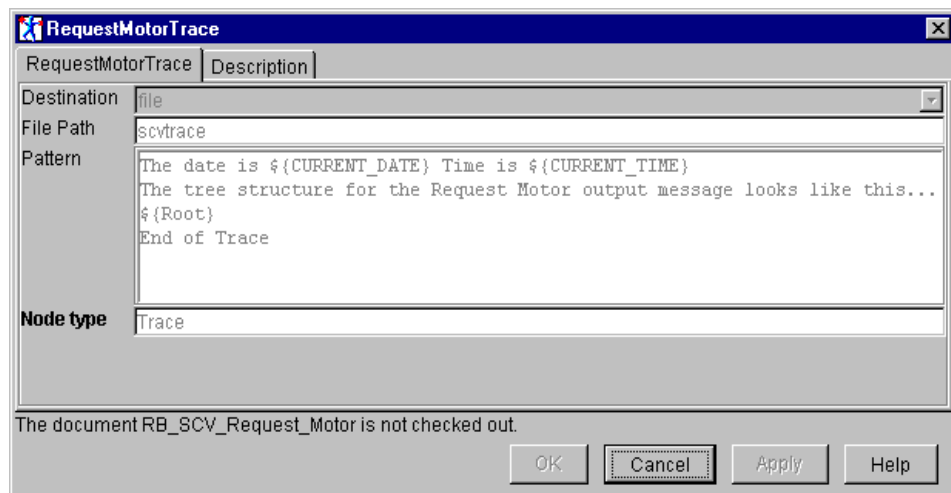


Figure 133. RequestMotorTrace Trace node

#### 13.1.4.5 MotorBackend MQOutput node

This MQOutput node parses the message according to the properties set in the RequestMotor Compute node and writes the message to the queue

identified by the queue manager and queue specified on the node's Basic tab as shown in Figure 134.

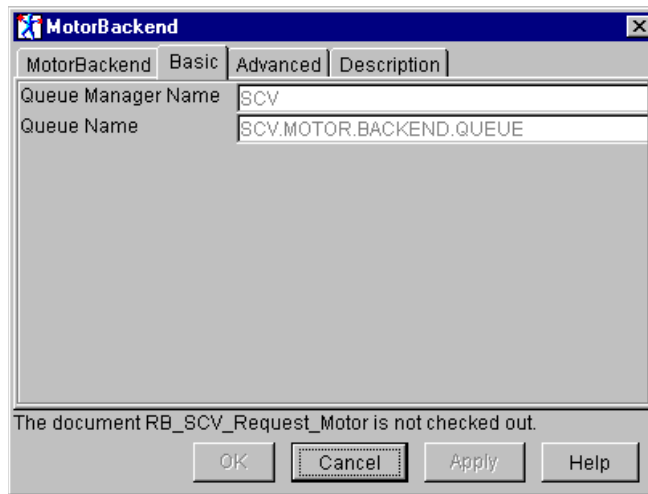


Figure 134. MQOutput node

---

## 13.2 Inbound flow

This section describes the inbound message flow.

1. The legacy back-end application writes its response message to the queue name specified as the reply-to queue name in the request message.
2. The response message is picked up by the RB\_SCV\_Backend\_Reply message flow or the RB\_SCV\_Backend\_Reply\_Endow message flow, depending on which legacy back-end application generated the reply.
  - The RB\_SCV\_Backend\_Reply message flow receives and processes messages from the House and Motor legacy back-end applications.
  - The RB\_SCV\_Backend\_Reply\_Endow message flow receives and processes messages from the Endow legacy back-end application.
3. The message flow parses the reply message and generates an output message that consists of a subset of the fields on the input message with semi-colon delimiters inserted between the fields in the output message.
4. The Message Merger Java application retrieves messages and "merges" messages having matching correlation IDs into a single message and writes the merged message to a queue monitored by the ClientServlet Java application.

5. The ClientServlet application retrieves the reply messages by correlation ID and updates the Web page with the response data.

### 13.2.1 RB\_SCV\_Backend\_Reply message flow

The RB\_SCV\_Backend\_Reply message flow processes reply messages from the House and Motor legacy back-end applications as shown in Figure 135.

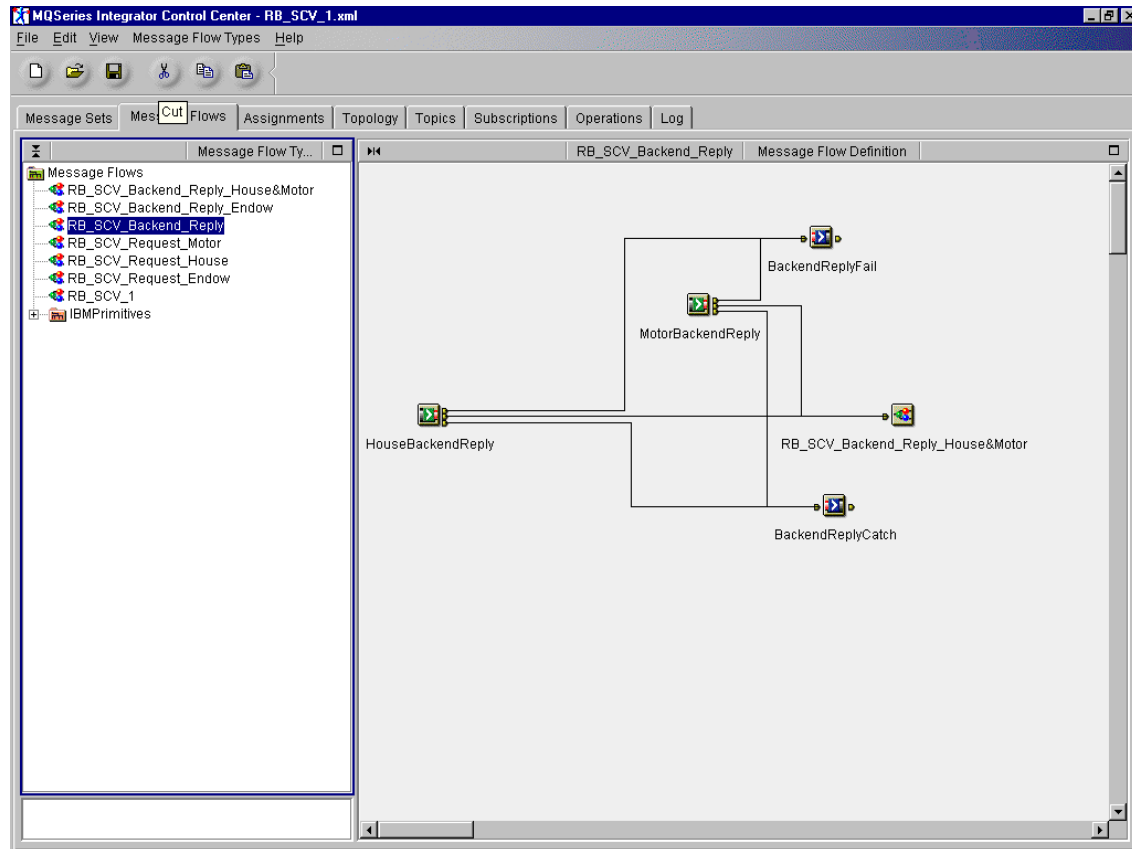


Figure 135. RB\_SCV\_Backend\_Reply message flow

This flow demonstrates that two separate queues are being monitored for reply messages from the two back-end applications and that the reply message, regardless of which back-end application it came from (House or Motor), is being processed by a single sub-flow, `RB_SCV_Backend_Reply_House&Motor`.

#### 13.2.1.1 HouseBackendReply MQInput node

The reply message sent by the House legacy back-end application does not carry an RFH; so, the message properties must be set on the Default tab of the MQInput node. Figure 136 shows that the message is defined in the MRM, where the message set, message (type), and message format are identified.

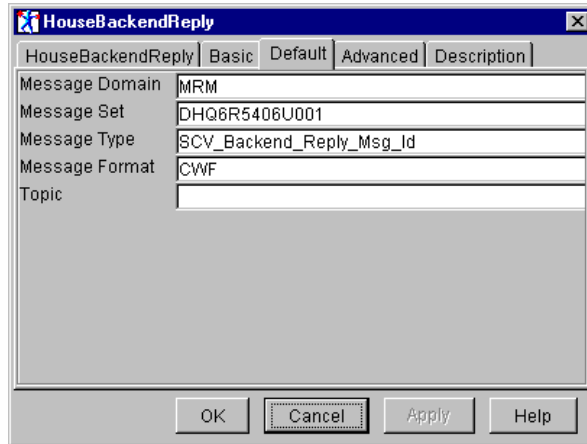


Figure 136. HouseBackendReply MQInput node default properties

The Default tab of the MotorBackendReply is exactly the same.

#### 13.2.1.2 RB\_SCV\_Backend\_Reply\_House&Motor node

This is a message processing node whose function is to pass the message to another message flow. It has no properties.

#### 13.2.1.3 BackendReplyFail MQOutput node

This node receives message propagated to the Failure terminal of the BackendReply MQInput node. It writes the message to the queue identified on the node's Basic tab.

#### 13.2.1.4 BackendReplyCatch node

This node receives message propagated to the Catch terminal of the BackendReply MQInput node. It writes the message to the queue identified on the node's Basic tab.

### 13.2.2 RB\_SCV\_Backend\_Reply\_House&Motor message flow

This flow processes messages passed to it from the RB\_SCV\_Backend\_Reply message flow as shown in Figure 137.

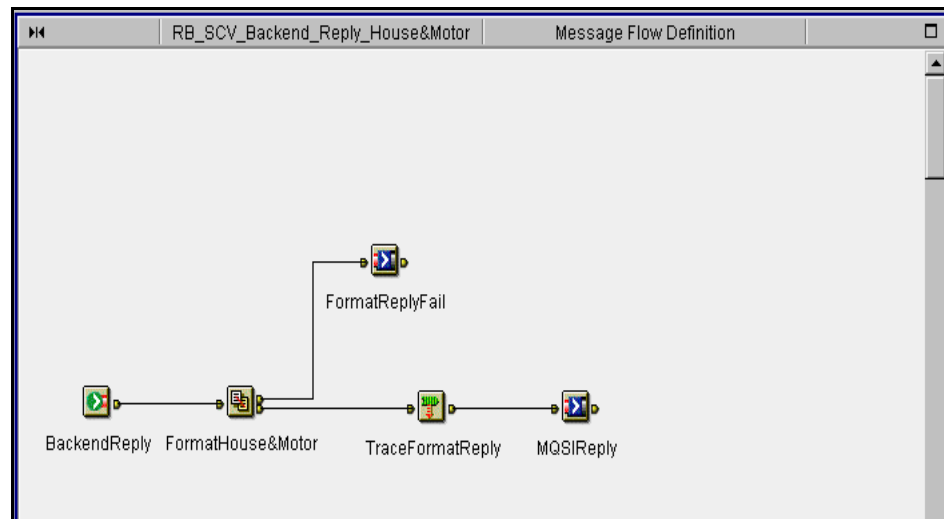


Figure 137. RB\_SCV\_Backend\_Reply\_House&Motor message flow

#### 13.2.2.1 BackendReply Input Terminal node

This node provides an input terminal into this flow. It has no properties.

#### 13.2.2.2 FormatHouse&Motor Compute node

This node takes the reply message from the House and Motor legacy back-end applications, extracts the fields required for the output message, and inserts a semicolon delimiter between each of the fields in the output message as shown in Figure 138 on page 302.

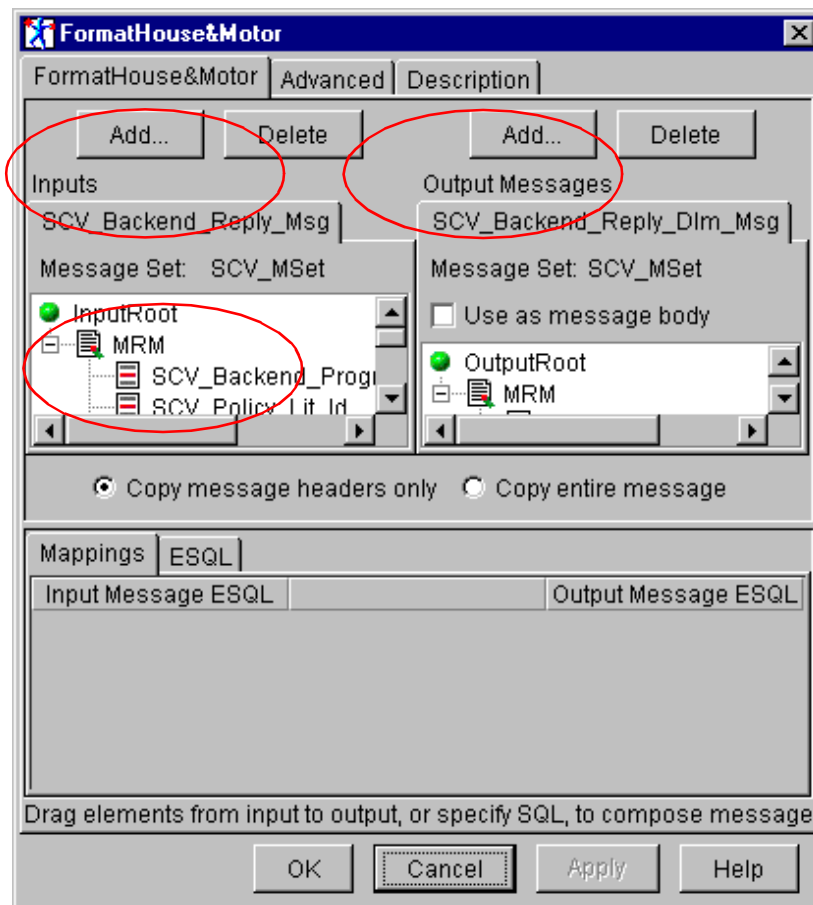


Figure 138. FormatHouse&Motor Compute node

Input to this node is the reply message sent by either the House or the Motor legacy back-end application. The input message is a legacy format predefined in the MRM. It is added to this node as input by clicking the **Add** button and selecting it from the drop-down list in the pop-up menu.

Output from this node is a predefined format expected by the Web front-end application. It too has been defined in the MRM and is added to the Compute node by clicking the **Add** button on the Output Messages side of the window and selecting the appropriate message set and message from the drop-down list in the pop-up menu.

Since the input and output messages are of different formats, we select **Copy message headers only**.

```

DECLARE I INTEGER;
SET I = 1;
WHILE I < CARDINALITY(InputRoot.*[]) DO
    SET OutputRoot.*[I] = InputRoot.*[I];
    SET I=I+1;
END WHILE;

```

Figure 139. FormatHouse&Motor Compute node ESQL

The first five lines of ESQL, shown in Figure 140, are automatically generated because the Copy message headers only option was selected.

```

SET OutputRoot.Properties.MessageSet = 'DHQ6R5406U001';
SET OutputRoot.Properties.MessageType = 'SCV_Backend_Reply_Dlm_Msg_Id';
SET OutputRoot.Properties.MessageFormat = 'CWF';
SET OutputRoot.MRM.SCV_Reply_Field_Count_Id = '06';
SET OutputRoot.MRM.FldDelim_Id = ',';

```

Figure 140. FormatHouse&Motor Compute node ESQL

The next five lines set the message properties, that is, the identification for the message set and the name of the message type.

```

SET OutputRoot.MRM.SCV_PolicyInfo_Repeat_Dlm_Id[] =
    (SELECT T.SCV_Policy_No_Id,
           T.SCV_Policy_Desc_Id,
           T.SCV_Policy_StartDate_Id,
           T.SCV_Policy_EndDate_Id,
           T.SCV_Policy_Status_Id
    FROM InputBody.SCV_PolicyInfo_Repeat_Id[] AS T);

```

Figure 141. FormatHouse&Motor Compute node ESQL

This SET statement selects the policy information from the input message and copies the values in the corresponding fields of the output message.

Figure 142 on page 304 shows the FormatHouse&Motor Compute node ESQL.

```

DECLARE POLICYLIT CHAR;
SET POLICYLIT = SUBSTRING(InputBody.SCV_Backend_Program_Id FROM 1 FOR 5);
SET I = 1;
WHILE I < 4 DO
    IF OutputRoot.MRM.SCV_PolicyInfo_Repeat_Dlm_Id[I].SCV_Policy_No_Id = ' ' THEN
        SET OutputRoot.MRM.SCV_PolicyInfo_Repeat_Dlm_Id[I].SCV_Policy_Lit_Id = ' ';
        SET OutputRoot.MRM.SCV_PolicyInfo_Repeat_Dlm_Id[I].Fld2Delim_Id = ' ';
        SET OutputRoot.MRM.SCV_PolicyInfo_Repeat_Dlm_Id[I].Fld3Delim_Id = ' ';
        SET OutputRoot.MRM.SCV_PolicyInfo_Repeat_Dlm_Id[I].Fld4Delim_Id = ' ';
        SET OutputRoot.MRM.SCV_PolicyInfo_Repeat_Dlm_Id[I].Fld5Delim_Id = ' ';
        SET OutputRoot.MRM.SCV_PolicyInfo_Repeat_Dlm_Id[I].Fld6Delim_Id = ' ';
        SET OutputRoot.MRM.SCV_PolicyInfo_Repeat_Dlm_Id[I].Fld7Delim_Id = ' ';
    ELSE
        SET OutputRoot.MRM.SCV_PolicyInfo_Repeat_Dlm_Id[I].SCV_Policy_Lit_Id = POLICYLIT;
        SET OutputRoot.MRM.SCV_PolicyInfo_Repeat_Dlm_Id[I].Fld2Delim_Id = ';';
        SET OutputRoot.MRM.SCV_PolicyInfo_Repeat_Dlm_Id[I].Fld3Delim_Id = ';';
        SET OutputRoot.MRM.SCV_PolicyInfo_Repeat_Dlm_Id[I].Fld4Delim_Id = ';';
        SET OutputRoot.MRM.SCV_PolicyInfo_Repeat_Dlm_Id[I].Fld5Delim_Id = ';';
        SET OutputRoot.MRM.SCV_PolicyInfo_Repeat_Dlm_Id[I].Fld6Delim_Id = ';';
        SET OutputRoot.MRM.SCV_PolicyInfo_Repeat_Dlm_Id[I].Fld7Delim_Id = ';';
    END IF;
    SET I=I+1;
END WHILE;

```

Figure 142. FormatHouse&Motor Compute node ESQl

This ESQl is filling in the delimiter between fields. For policy entries that contain real data, the delimiter is set to a semicolon. For empty entries it is set to blanks.



---

## Appendix A. Hardware and software specifications

This appendix lists the hardware and software used for the application solution described in Chapter 13, “Getting a single customer view with MQSeries” on page 245.

The solution used two 350 MHz Pentium II IBM Netfinity 3000 servers. One server hosted an IBM WebSphere Application Server and an IBM HTTP Server. The other server hosted MQSeries for Windows NT V5.1 and MQSeries Integrator V2.0 in addition to the legacy programs and application data. IBM DB2 UDB V6 was the database software used in the solution. TCP/IP was the communications protocol used. Details of the software installed on each server follow:

### Server #1 - WebSphere

- IBM Netfinity 3000 350 MHz Pentium II
- 260 MB memory
- Windows NT Workstation V4.0 with ServicePac 5
- IBM HTTP Server V1.3.6
- IBM WebSphere Application Server V3.02
- IBM Java Development Kit V1.1.7
- IBM DB2 Universal Database (UDB) V6.1
- MQSeries Java Client V5.1

### Server #2 - MQSeries

- IBM Netfinity 3000 350 MHz Pentium II
- 320 MB memory
- Windows NT Workstation V4.0 with ServicePac 5
- IBM MQSeries for Windows NT V5.1 with CSD04
- IBM MQSeries Integrator for Windows NT V2.0
- IBM DB2 Universal Database (UDB) V6.1
- IBM Java Development Kit V1.1.7



---

## Appendix B. MQSeries Internet pass-thru

IBM MQSeries Internet pass-thru:

- Is an MQSeries base product extension that can be used to implement messaging solutions between remote sites across the Internet
- Makes the passage of MQSeries channel protocols into and out of a firewall simpler and more manageable by tunneling the protocols inside HTTP or acting as a proxy
- Operates as a stand-alone service that can receive and forward MQSeries message flows. The system on which it runs does not have to host an MQSeries Queue Manager
- Helps you to provide business-to-business transactions using MQSeries
- Enables existing, unchanged MQSeries applications to be used through a firewall
- Provides a single point of control over access to multiple queue managers

This appendix describes the latter product. For detailed information, including installation and configuration details, refer to the Internet related SupportPacs at <http://www-4.ibm.com/software/ts/mqseries/txppacs/mqguides.html>

This appendix is an abstract of the MQSeries SupportPac MS81 available at the aforementioned URL.

In this appendix, MQSeries Internet pass-thru is often termed “MQIPT” for convenience.

---

### B.1 Introduction

MQSeries Internet pass-thru is an extension to the base MQSeries product. MQIPT runs as a stand-alone service that can receive and forward MQSeries message flows, either between two MQSeries queue managers or between an MQSeries client and an MQSeries queue manager. MQIPT enables this connection when the client and server are not on the same physical network. One or more MQIPTs can be placed in the communication path between two MQSeries queue managers or between an MQSeries client and an MQSeries queue manager. The MQIPTs allow the two MQSeries systems to exchange messages without needing a direct TCP/IP connection between the two systems. This is useful if the firewall configuration prohibits a direct TCP/IP connection between the two systems.

MQIPT listens on one or more TCP/IP ports for incoming connections, which can carry either the normal MQSeries messages or an MQSeries protocol

tunneled inside HTTP. It can handle multiple concurrent connections. The MQSeries channel that makes the initial TCP/IP connection request is referred to as the “caller”; the channel to which it is attempting to connect is the “responder”, and the queue manager it is ultimately trying to contact is the “destination queue manager”.

The anticipated uses of MQIPT are:

- MQIPT can be used as a channel concentrator so that channels from or to multiple separate hosts can appear to a firewall as if they are all from or to the MQIPT host. This makes it easier to define and manage firewall filtering rules.

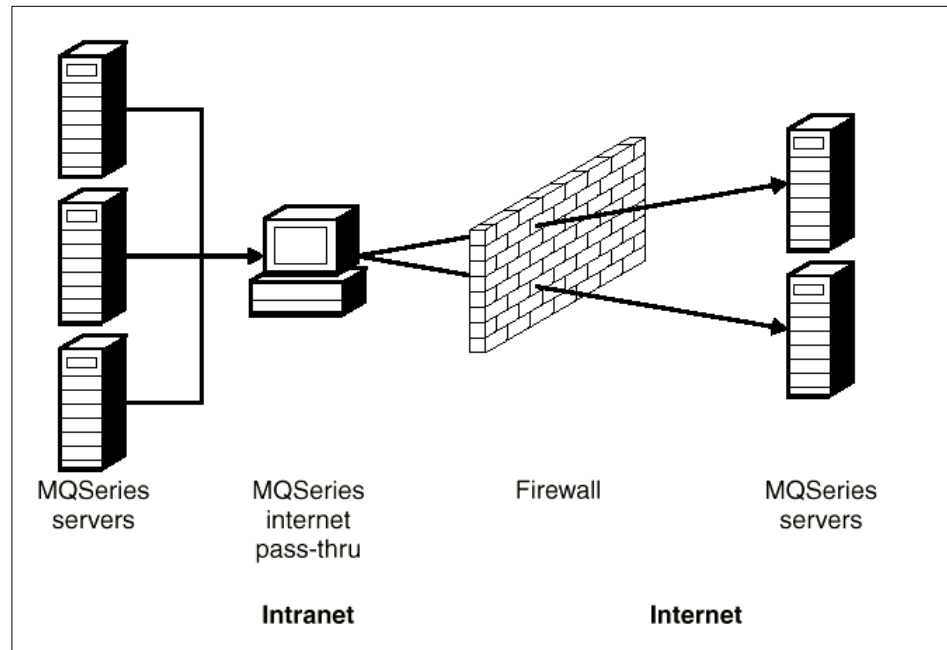


Figure 143. Example of MQIPT as a channel concentrator

- If it is placed in the firewall’s “demilitarized zone” (DMZ), on a machine with a known and trusted Internet protocol (IP) address, MQIPT can be used to listen for incoming MQSeries channel connections that it can then forward to the trusted intranet; the inner firewall must allow this trusted machine to make inbound connections. In this configuration, MQIPT prevents external requests for access from seeing the true IP addresses of the machines in the trusted intranet. Thus, MQIPT provides a single point of access. Figure 144 on page 309 shows an example of MQIPT with a “demilitarized zone”.

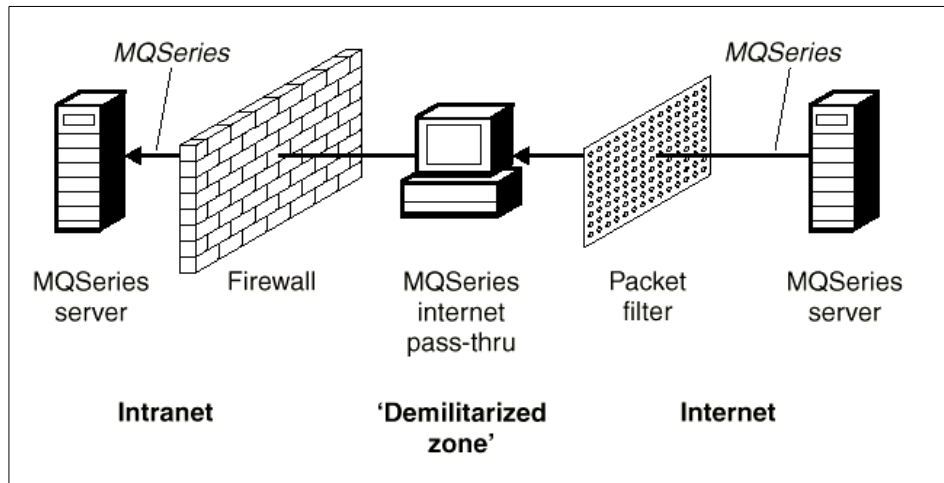


Figure 144. Example of MQIPT with a “demilitarized zone”

- If two MQIPTs are deployed inline, they can communicate using HTTP. Provision of this HTTP tunneling feature enables requests to be transmitted through corporate firewalls by the use of existing HTTP proxies. The first MQIPT inserts the MQSeries protocol into HTTP and the second extracts the MQSeries protocol from its HTTP wrapper and forwards it to the destination queue manager as shown in Figure 145.

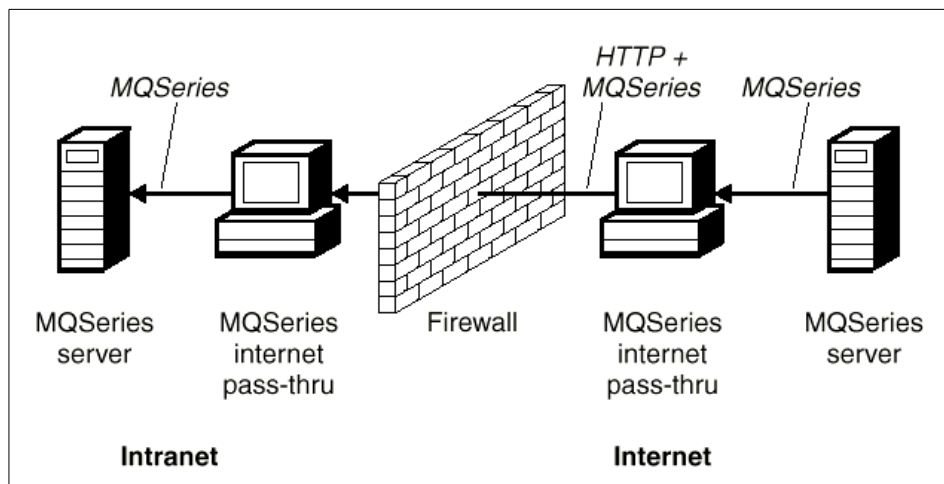


Figure 145. Example of MQIPT and HTTP tunneling

MQIPT provides a number of options for inbound connections through firewalls; it does not provide for outbound connections through firewalls. For outbound connections, it is assumed that the MQSeries client or queue manager is running on a system that is SOCKS-enabled and is configured to communicate with a SOCKS gateway.

MQIPT holds data in memory as it forwards it from its source to its destination. No data is saved on disk (except for memory paged to disk by the operating system). The only time MQIPT accesses the disk explicitly is to read its configuration file and to write log and trace records.

The full range of MQSeries channel types can be made through one or more MQIPTs. The presence of MQIPTs in a communication path has no effect on the functional characteristics of the connected MQSeries components, but there will be some impact on the performance of message transfer.

MQIPT can be used in conjunction with MQSeries publish/subscribe or the MQSeries Integrator message broker.

---

## **B.2 Overview of how Internet pass-thru works**

In its simplest configuration, MQIPT acts as an MQSeries protocol forwarder. It listens on a TCP/IP port (1414 by default) and accepts connection requests from MQSeries channels. If a well-formed request is received, MQIPT establishes a further TCP/IP connection between itself and the destination MQSeries queue manager. It then passes all protocol packets it receives from its incoming connection to the destination queue manager and returns protocol packets from the destination queue manager back on the original incoming connection.

No change to the MQSeries protocol (client/server or queue manager to queue manager) is involved because neither end is directly aware of the presence of the intermediary; so, new versions of the MQSeries client or server code are not required.

To use MQIPT, the caller channel must be configured to use MQIPT's hostname and port, not the hostname and port of the destination queue manager. MQIPT does not examine the channel name; this is simply passed through to the receiving queue manager. Other configuration fields, such as the user ID and password in a client/server channel, are similarly passed to the receiving queue manager.

MQIPT can be used to allow access to one or more destination queue managers. For this to work, there must be a mechanism to tell MQIPT which

queue manager to connect to; so, MQIPT uses the incoming TCP/IP port number to determine which queue manager to connect to as described in the next paragraph. To allow access to more than one destination queue manager, MQIPT can be configured to listen on multiple TCP/IP ports. Each incoming port is mapped to a destination queue manager through an MQIPT “route”. The MQIPT administrator may define up to 100 such routes, which associate an incoming TCP/IP port with the hostname and port of the destination queue manager. This means that the hostname (IP address) of the destination queue manager is never visible to the originating channel. Each route can handle multiple connections between its listening port and destination and acts as a different MQIPT “instance”.

---

### **B.3 HTTP support**

As an option, MQIPT can be configured so that the data packets it forwards are encoded as HTTP requests. MQIPT supports HTTP tunneling with or without chunking.

Because today’s MQSeries channels do not accept HTTP requests, a second MQIPT is required to receive the HTTP requests and convert them back into normal MQSeries protocol packets. The second MQIPT strips off the HTTP header to convert the incoming packet back into a standard MQSeries protocol packet before passing it on to the destination queue manager.

When using HTTP tunneling without chunking, an HTTP reply is sent back to the first MQIPT. This reply can be the response from the destination queue manager or a dummy acknowledgement.

If either MQSeries system has to send a chain of successive MQSeries protocol packets (as happens when transferring a large message), several HTTP request/reply pairs are used to transfer the data. To achieve this, MQIPs insert additional request or reply flows.

When using HTTP tunneling with chunking, only the first packet is wrapped in an HTTP header. Middle and last packets have chunking headers. This arrangement removes the wait for a dummy acknowledgement from the second MQIPT and, thus, offers slightly better performance than that provided by HTTP tunneling without chunking.

When HTTP is being used between two MQIPs, the TCP/IP connection on which the HTTP requests and replies are flowed is kept open for the lifetime of the message channel. The MQIPs do not close the TCP/IP connection between request/reply pairs.

If two MQIPTs are communicating through HTTP, it is possible that an HTTP request might stay outstanding for an extended period. An example is in a requester/server channel when the server side is waiting for new messages to arrive on its transmission queue. The MQSeries channel protocol provides a “heartbeat” mechanism, which requires the waiting end periodically to send heartbeat messages to its partner (the default channel heartbeat period is five minutes) and MQIPT uses this heartbeat as the HTTP reply. To avoid causing problems with timeouts in some firewalls, do not disable this channel heartbeat or set it to an excessively high value.

---

## **B.4 Supported channel configurations**

All MQSeries channel types are supported, but configuration is restricted to TCP/IP connections. To an MQSeries client or queue manager, MQIPT appears as if it is the destination queue manager. Where channel configuration requires a destination host and port number, the MQIPT host name and listener port number are specified.

### ***Client/server channels***

MQIPT listens for incoming client connection requests, and then forwards them (either using HTTP tunneling or as standard MQSeries protocol packets). If MQIPT is using HTTP tunneling, it forwards them on a connection to a second MQIPT. If it is not using HTTP tunneling, it forwards them on a connection to what it sees as the destination queue manager (although this could in turn be a further MQIPT). Once the queue manager has accepted the client connection, packets are relayed between client and server.

### ***Sender/receiver***

If MQIPT receives an incoming request from a sender channel, it forwards it to the next MQIPT or destination queue manager in exactly the same way as for client connection channels. The destination queue manager validates the incoming request and starts the receiver channel if appropriate. All communications between sender and receiver channel (including security flows) are relayed.

### ***Requester/server***

This combination is handled in the same manner as the types above. Validation of the connection request is performed by the server channel at the destination queue manager.



***Requester/sender***

The “callback” configuration could be of use if the two queue managers are not allowed to establish direct connections to each other, but are both allowed to connect to MQIPT and accept connections from it.

***Server/requester and server/receiver***

These are handled by MQIPT just like the Sender/Receiver configuration.

---

**13.3 Normal termination and failure conditions**

When MQIPT detects closure (either normal or abnormal) of an MQSeries channel, it propagates the channel closure. If the administrator closes down a route through the MQIPT, all channels going through that route are closed.

MQIPT provides an optional idle time-out facility. If MQIPT detects that a channel has been idle for a period of time exceeding the timeout, it performs an immediate shutdown on the two connections in question.

The two MQSeries systems at either end of the channel observe these abnormal termination conditions either as network failures or as termination of the channel by their partner. The channels in question are then able to restart and recover (if the failure happens during a protocol in-doubt period) just as they would do if there were no MQIPTs.

---

**B.5 Security considerations**

MQIPT does not contain any mechanism to authenticate the originating channel or provide user-based access control to destination queue managers. MQIPTs allow channel security flows, so that MQSeries channel exits can be used to provide security over the entire channel from end to end.

MQIPT has several additional functions that help a designer build a secure solution:

- If there are many clients in an internal network all trying to make outgoing connections, they can all go through an MQIPT located inside the firewall. The firewall administrator then has to grant external access only to the MQIPT's machine.
- MQIPT can connect only to queue managers for which it has been explicitly configured in its configuration file.
- It provides a connection log. When enabled, this facility logs all connections, successful or otherwise, detailing the host from which the connection was made and the responding hostname. It also logs connection timeouts and disconnects.

- MQIPT verifies that the messages it receives and transmits are valid and conform to the MQSeries protocol. This helps prevent MQIPTs being used for security attacks outside of the MQSeries protocol.
- It allows channel exits to run their own end-to-end security protocols.
- MQIPT allows you to restrict the total number of incoming connections. This helps protect a vulnerable internal queue manager from denial-of-service attacks.

You must protect the MQIPT's configuration file, `mqipt.conf`, because this file controls access to the internal hosts, and you must prevent unauthorized access to the command port (if it is enabled) because such access allows an external person to shut down MQIPT.

---

## Appendix C. Using the additional material

This redbook also contains additional material in the form of Web material. See the sections below for instructions on using or downloading this material.

---

### C.1 Locating the additional material on the Internet

The Web material associated with this redbook is also available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG246010>

Alternatively, you can go to the IBM Redbooks Web site at:

[ibm.com/redbooks](http://ibm.com/redbooks)

Select **Additional materials** and open the directory that corresponds to the redbook form number.

---

### C.2 Using the Web material

The additional Web material that accompanies this redbook includes the following:

<i>File name</i>	<i>Description</i>
<b>SwitchNode.zip</b>	Sources for the SwitchNode
<b>SCV.zip</b>	Sources for the single customer view application

#### C.2.1 System requirements for downloading the Web material

The following system configuration is recommended for using the additional Web material.

<b>Operating System:</b>	Windows NT Version 4
<b>Processor:</b>	Pentium II
<b>Memory:</b>	256 MB

#### C.2.2 How to use the Web material

Create a subdirectory (folder) on your workstation and copy the contents of the Web material into this folder.



---

## Appendix D. Special notices

This publication is intended to help IT architects and IT specialists design and deploy B2B Integration solutions. The information in this publication is not intended as the specification of any programming interfaces that are provided by any of the products mentioned. See Appendix E, "Related publications" on page 321, for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.


Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.


The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers

attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

e (logo)   
Redbooks  
System/36  
System/390  
VisualAge  
VTAM

IBM  
Redbooks Logo   
System/360  
ViaVoice  
VSE/ESA  
WebSphere

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere., The Power To Manage., Anything. Anywhere., TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Lotus Notes is a registered trademark of Lotus Development Corporation

Other company, product, and service names may be trademarks or service marks of others.





---

## Appendix E. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

---

### E.1 IBM Redbooks

For information on ordering these publications see “How to get IBM Redbooks” on page 325.

- *Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*, SG24-5755
- *Developing an e-business Application for the IBM WebSphere Application Server*, SG24-5423
- *VisualAge for Java Enterprise Version 2: Data Access Beans - Servlets - CICS Connector*, SG24-5265
- *Connecting e-business to the Enterprise by Example*, SG24-5514
- *Business Integration Solutions with MQSeries Integrator*, SG24-6154
- *The XML Files: Using XML and XSL with IBM WebSphere V3.0*, SG24-5479
- *Patterns for e-business: User-to-Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*, SG24-5864
- *The XML Files: Using XML for Business-to-Business and Business-to-Consumer Applications*, SG24-6104
- *An Early Look at Application Considerations Involved with MQSeries*, GG24-4469
- *Connecting the Enterprise to the Internet with MQSeries and VisualAge for Java*, SG24-2144

---

### E.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at [ibm.com/redbooks](http://ibm.com/redbooks) for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
IBM System/390 Redbooks Collection	SK2T-2177
IBM Networking Redbooks Collection	SK2T-6022
IBM Transaction Processing and Data Management Redbooks Collection	SK2T-8038

CD-ROM Title	Collection Kit Number
IBM Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
IBM AS/400 Redbooks Collection	SK2T-2849
IBM Netfinity Hardware and Software Redbooks Collection	SK2T-8046
IBM RS/6000 Redbooks Collection	SK2T-8043
IBM Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

---

### E.3 Other resources

- *Design Patterns - Elements of Reusable Object-Oriented Software*, ISBN 0-2016-3361-2, by E. Gamma, R. Helm, R. Johnson, J. Vlissides
- *A Pattern Language*, ISBN 0-1950-1919-9, by C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, S. Angel.
- *Pattern-Oriented Software Architecture - A System of Patterns*, ISBN 0-4719-5869-7, by Buschmann, et al.
- *Pattern Hatching - Design Patterns Applied*, ISBN 0-2014-3293-5, by J. Vlissides
- *Object-Oriented Analysis and Design with Applications*, ISBN 0-8053-5340-2, by Booch and Grady
- *Object-Oriented Software Engineering; A Use Case Driven Approach*, ISBN 0-2015-4435-0, by Jacobson and Ivar
- *Object-Oriented Modeling and Design*, ISBN 0-1362-9841-9, by Rumbaugh, James et al.
- *UML Distilled: Applying the Standard Object Modeling Language*, ISBN 0-2013-2563-2, by Fowler, Martin, Scott, Kendall, and Jacobson
- *Developing Object-oriented Software - An Experienced-Based Approach*, ISBN 0-1373-7248-5, by John Barry, Tom Bridge, Paul Fertig, Tom Guinane, Geoff Hambrick, Daniel Hu, Tom Kristek, Dave Livesey, Guillermo Lois, Mike Page, Branko Petch, Frank Seliger, Thomas Wappler, Brian Watt, Martin West, George Yuan

The following publications are product documentation:

- *MQSeries Integrator Introduction and Planning*
- *MQSeries Integrator Using the Control Center*
- *MQSeries Clients*
- *MQSeries Application Programming Guide*
- *MQSeries Intercommunication Manual*

- *MQSeries Using Java*
- *MQSeries Messaging Interface*
- *MQSeries Planning Guide*
- *MQSeries Integrator Introduction and Planning Guide*
- *MQSeries for AS/400 V5.1 System Administration Manual*
- *MQSeries for Digital Open VMS System Management Guide*
- *MQSeries for Digital UNIX System Management Guide*
- *MQSeries for Tandem NonStop Kernel System Management Guide*
- *MQSeries for VSE/ESA System Management Guide*
- *MQSeries System Administration Manual*

---

## E.4 Referenced Web sites

These Web sites are also relevant as further information sources:

- Web magazine focused on Application Integration issues:  
<http://www.eaijournal.com/>
- Articles from IBM Research on a number of software disciplines:  
<http://www.research.ibm.com/journal>
- Industry leading Web site for middleware and systems management:  
<http://www.messageq.com>
- e-commerce solution provider:  
<http://www.fastwater.com/>
- <http://www.eaijournal.com>
- <http://eai.ebizQ.net/>
- <http://www.ibm.com/software/developer/web/patterns>
- <http://www.ibm.com/iac/papers/icdcsws99/index.html>
- [www.ansi.org](http://www.ansi.org)
- [www.edifact.org](http://www.edifact.org)
- <http://www.dtmf.org>
- <http://www.oasis-open.org>
- <http://www.xml.org>
- <http://www.ebxml.org>
- <http://www.cxml.org>

- <http://www.biztalk.org>
- <http://www.dmtf.org>
- [eco.commerce.net](http://eco.commerce.net)
- <http://www.openapplications.org>
- <http://www.biztalk.org>
- <http://www.rosettanet.org>
- [http://www.xml.org/xmlorg\\_registry/index.shtml](http://www.xml.org/xmlorg_registry/index.shtml)
- <http://msdn.microsoft.com/xml/general/soapspec.asp>
- <http://www.commerceone.com/xml/cbl/>
- <http://www.xmledi.com>
- <http://www-4.ibm.com/software/ts/mqseries/library/manualsa/>
- <ftp://ftp.software.ibm.com/software/ts/mqseries/library/books/csqzae03.pdf>
- <http://www.ibm.com/software/developer/web/patterns/>
- <http://www.omg.org>
- <http://www.omg.org/uml>
- <http://www.rational.com/products/rose/>
- <http://www-4.ibm.com/software/ts/mqseries/library/manualsa/index.htm>
- <http://java.sun.com>
- <http://www-4.ibm.com/software/ts/mqseries/library/manualsa/index.htm>
- <http://www-4.ibm.com/software/ts/mqseries/txppacs/ma0f.html>
- <http://www.ibm.com/software/ts/mqseries/txppacs/txpml.html>
- <http://www-4.ibm.com/software/ts/mqseries/txppacs/mp16.html>
- <http://www-4.ibm.com/software/ts/mqseries/txppacs/mp19.html>
- <http://www-4.ibm.com/software/ts/mqseries/txppacs/mqguides.html>
- <http://www-4.ibm.com/software/ts/mqseries/txppacs/mp43.html>
- <http://www-4.ibm.com/software/ts/mqseries/txppacs/mp67.html>
- <http://www-4.ibm.com/software/ts/mqseries/txppacs/mp74.html>
- <http://www-4.ibm.com/software/ts/mqseries/txppacs/mp11.html>
- <http://www-4.ibm.com/software/ts/mqseries/txppacs/mp12.html>
- <http://www-4.ibm.com/software/ts/mqseries/txppacs/mp14.html>

---

## How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** [ibm.com/redbooks](http://ibm.com/redbooks)

Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

Send orders by e-mail including information from the IBM Redbooks fax order form to:

	<b>e-mail address</b>
In United States or Canada	<a href="mailto:pubscan@us.ibm.com">pubscan@us.ibm.com</a>
Outside North America	Contact information is in the "How to Order" section at this site: <a href="http://www.elink.ibm.link.ibm.com/pbl/pbl">http://www.elink.ibm.link.ibm.com/pbl/pbl</a>

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the "How to Order" section at this site: <a href="http://www.elink.ibm.link.ibm.com/pbl/pbl">http://www.elink.ibm.link.ibm.com/pbl/pbl</a>

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the "How to Order" section at this site: <a href="http://www.elink.ibm.link.ibm.com/pbl/pbl">http://www.elink.ibm.link.ibm.com/pbl/pbl</a>

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

---

### IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at <http://w3.itso.ibm.com/> and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at <http://w3.ibm.com/> for redbook, residency, and workshop announcements.

---

## IBM Redbooks fax order form

Please send me the following:

Title	Order Number	Quantity

---

First name	Last name
------------	-----------

---

Company
---------

---

Address
---------

---

City	Postal code	Country
------	-------------	---------

---

Telephone number	Telefax number	VAT number
------------------	----------------	------------

---

<input type="checkbox"/> Invoice to customer number	
---	--

---

<input type="checkbox"/> Credit card number	
---	--

---

Credit card expiration date	Card issued to	Signature
-----------------------------	----------------	-----------

**We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.**

---

## Glossary

**ACL.** Access Control List.

**AD/SI.** IBM Application Development/Systems Integration Method Group within WSDDM.

**API.** Application Programming Interface.

**APPC.** Advanced Program-to-Program Communication.

**applet.** A Java program designed to run within a Web browser. Contrast with application.

**application.** In Java programming, a self-contained, stand-alone Java program that includes main() method. Contrast with applet.

**ASCII.** American Standard Code for Information Interchange. The 8-bit character encoding scheme used by most PCs and UNIX systems. It supersedes an earlier 7-bit ASCII standard.

**ASP.** Microsoft's Active Server Pages

**AVM.** Accelerated Value Method. This is a rapid deployment method based on iterative prototyping, used by Lotus Consulting.

**Bamba.** Bamba is a brandname for IBM technology used to develop network-enabled multimedia applications. One function of the technology is streaming audio and video (both live and stored) across the Internet and over intranets via modem or LAN connections.

**BB.** Building Block. Many IBM design and development methods, including the one described in this redbook, construct a design from building blocks that are generally defined in more detail as the design evolves.

**Bean.** This is a small component that can be used to build applications. See JavaBean.

**Browser.** An Internet-based tool that lets users browse Web sites.

**Call Level Interface (CLI).** This is a callable application program interface (API) for database access that is an alternative to an embedded SQL application program interface. In contrast to embedded SQL, CLI does not require precompiling or binding by the user, but instead

provides a standard set of functions to process SQL statements and related services at run time.

**Customer Information Control System (CICS).** This is a distributed online transaction processing system designed to support a network of many terminals. The CICS family of products is available for a variety of platforms ranging from a single workstation to the largest mainframe.

**CICS Access Build.** This is a VisualAge for Java Enterprise tool that generates beans to access CICS transactions through the CICS Gateway for Java and CICS Client.

**CICS Client.** This is a server program that processes CICS ECI calls, forwarding transaction requests to a CICS program running on a host.

**CICS Gateway for Java.** This is a server program that processes Java ECI calls and forwards CICS ECI calls to the CICS Client.

**Class.** This is an aggregate that defines properties, operations, and behavior for all instances of that aggregate.

**Client.** As in client/server computing, this is the application that makes requests to the server and, often, handles the necessary interaction with the user.

**Client/server.** This is a form of distributed processing in which the task required to be processed is accomplished by a client portion that requests services and a server portion that fulfills those requests. The client and server remain transparent to each other in terms of location and platform.

**COBOL.** Common Business Oriented Language.

**Commit.** The operation that ends a unit of work to make permanent the changes it has made to resources (transaction or data).

**Common Gateway Interface (CGI).** A standard protocol through which a Web server can

execute programs running on the server machine. CGI programs are executed in response to requests from Web client browsers.

**CGI.** Common Gateway Interface.

**CICS.** Customer Information Control System.

**COM.** Microsoft's Component Object Model.

**Common Object Request Broker Architecture (CORBA).** A middleware specification that defines a software bus – the Object Request Broker (ORB) – that provides the infrastructure.

**Communications area (COMMAREA).** In a CICS transaction program, this is a group of records that describes both the format and volume of data used.

**Conversational.** A communication model where two distributed applications exchange information by way of a conversation; typically one application starts (or allocates) the conversation, sends some data, and allows the other application to send some data. Both applications continue in turn until one decides to finish (or deallocate). The conversational model is a synchronous form of communication.

**CORBA.** Common Object Request Broker Architecture.

**Data Access Builder.** A VisualAge for Java Enterprise tool that generates beans to access and manipulate the content of JDBC/ODBC-compliant relational databases.

**Database.** (1) A collection of related data stored together with controlled redundancy according to a scheme to serve one or more applications. (2) All data files stored in the system. (3) A set of data stored together and managed by a database management system.

**Database Management System (DBMS).** A computer program that manages data by providing the services of centralized control, data independence, and complex physical structures for efficient access, integrity, recovery, concurrency control, privacy, and security.

**DB2 Call Level Interface (CLI).** The DB2 call level interface is an alternative SQL interface for the DB2 family of products and takes full

advantage of DB2 capability. This implementation closely follows industry standards, such as X/OPEN, to enhance application portability. Currently, the DB2 Call Level Interface functions are compatible with ODBC 2.0, and contain DB2-specific APIs to help exploit DB2 capability.

**DB2 for MVS/ESA.** An IBM relational database management system for the MVS operating system.

**DCE.** Distributed Computing Environment. Adopted by the computer industry as a de facto standard for distributed computing. DCE allows computers from a variety of vendors to communicate transparently and share resources, such as computing power, files, printers, and other objects in the network.

**DB2.** IBM Relational Database Family.

**DCOM.** Microsoft's Distributed Object Model.

**DECS.** Domino Enterprise Connection Services. A standard component of Domino 4.6.3 and Domino 5 that provides connectivity from Domino to back end relational data base servers. This connectivity is table-driven and requires no programming.

**DMZ.** DeMilitarized Zone. This term is now commonly used in the industry to describe a subnetwork, typically used for Web servers, that is protected by firewalls from both the external Internet and a company's internal network.

**DNS.** Domain Name Services.

**Distributed Processing.** Distributed processing is an application or systems model in which function and data can be distributed across multiple computing resources connected on a LAN or WAN. See client/server computing.

**Distributed Program Link (DPL).** This enables an application program executing in one CICS system to link (pass control) to a program in a different CICS system. The linked-to program executes and returns a result to the linking program. This process is equivalent to remote procedure calls (RPCs). You can write applications that issue RPCs that can be received by members of the CICS family.



**Dynamic Link Library (DLL).** This is a file containing executable code and data bound to a program at run time rather than at link time. The C++ Access Builder generates beans and C++ wrappers that let your Java programs access C++ DLLs.

**E2E.** IBM End-to-End System Design Method. See ISD.

**EBCDIC.** Extended Binary Coded Data Interchange Code. EBCDIC is the 8-bit character encoding scheme used by IBM and compatible mainframes since the introduction of System/360 in the mid 1960s.

**ECI.** External Call Interface is a way of interfacing with CICS at a program-to-program level.

**EJB.** Enterprise JavaBean. An EJB is a non-visual, remote object designed to run on a server and be invoked by clients. An EJB can be built out of multiple non-visual JavaBeans. EJBs are intended to live on one machine and be invoked remotely from another machine. They are platform-independent. Once a bean is written, it can be used on any client or server platform that supports Java.

**e-business.** This is either (a) the transaction of business over an electronic medium, such as the Internet, or (b) a business that uses Internet technologies and network computing in their internal business processes (via intranets), their business relationships (via extranets), and the buying and selling of goods, services, and information (via electronic commerce).

**External Call Interface (ECI).** This is an API that enables a non-CICS client application to call a CICS program as a subroutine. The client application communicates with the server CICS program using a data area called a *COMMAREA*.

**External Presentation Interface (EPI).** An API that allows a non-CICS application program to appear to the CICS system as one or more standard 3270 terminals. The non-CICS application can start CICS transactions and send and receive standard 3270 data streams to those transactions.

**Enterprise Access Builders (EAB).** In VisualAge for Java Enterprise, this is a set of code-generation tools. See also CICS Access Builder and Data Access Builder.

**EPI.** External Presentation Interface is a way of interfacing with CICS at a program to 3270 level.

**ERP.** Enterprise Resource Planning.

**EXCI.** External CICS Interface allows programs within MVS (including UNIX Services) to communicate with CICS programs.

**Firewall.** A computer (or programmable device) with associated software that can be used to restrict traffic passing through it according to defined rules. Controls would typically be applied based on the origin or destination address and the TCP/IP port number.

**FTP.** File Transfer Protocol

**File Transfer Protocol (FTP).** This is the basic Internet function that enables files to be transferred between computers. You can use it to download files from a remote host computer as well as to upload files from your computer to a remote host computer. See Anonymous FTP.

**Gateway.** This is a host computer that connects networks that communicate in different languages. For example, a gateway connects a company's LAN to the Internet.

**Graphical User Interface (GUI).** This is a type of interface that enables users to communicate with a program by manipulating graphical features rather than entering commands. Typically, a graphical user interface includes a combination of graphics, pointing devices, menu bars and other menus, overlapping windows, and icons.

**HotJava.** This is a Java-enabled Web and intranet browser developed by Sun Microsystems, Inc. HotJava is written in Java.

**Hypertext Markup Language (HTML).** This is the basic language that is used to build hypertext documents on the World Wide Web. It is used in basic plain ASCII-text documents, but, when those documents are interpreted (called *rendering*) by a Web browser, such as Netscape, the document can display formatted text, color, a

variety of fonts, graphic images, special effects, hypertext jumps to other Internet locations, and information forms.

**HTTP.** HyperText Transport Protocol

**Hypertext Transfer Protocol (HTTP).** The protocol for moving hypertext files across the Internet. Requires an HTTP client program on one end, and an HTTP server program on the other end. HTTP is the most important protocol used in the World Wide Web (WWW). See also Client, Server, WWW.

**HTTP request.** A transaction initiated by a Web browser and adhering to HTTP. The server usually responds with HTML data, but can send other kinds of objects as well.

**Hypertext.** Text in a document that contains a hidden link to other text. You can click a mouse on a hypertext word and it will take you to the text designated in the link. Hypertext is used in Windows help programs and CD encyclopedias to jump to related references elsewhere within the same document. The wonderful thing about hypertext, however, is its ability to link—using HTTP over the Web—to any Web document in the world, yet still require only a single mouse click to jump clear around the world.

**IBM.** International Business Machines

**IEC.** The IBM International Education Centre in La Hulpe, Belgium.

**IE.** Microsoft's Internet Explorer.

**IGS.** IBM Global Services.

**IIOP.** Internet Inter-ORB Protocol

**Internet Inter-ORB Protocol (IIOP).** An industry standard protocol that defines how General Inter-ORB Protocol (GIOP) messages are exchanged over a TCP/IP network. The IIOP makes it possible to use the Internet itself as a backbone ORB through which other ORBs can bridge.

**Integrated Development Environment (IDE).** A software program comprising an editor, a compiler, and a debugger. IBM's VisualAge for Java is an example of an IDE.

**Interface.** A set of methods that can be accessed by any class in the class hierarchy. The Interface page in the Workbench lists all interfaces in the workspace.

**Internet.** The vast collection of interconnected networks that all use the TCP/IP protocols and that evolved from the ARPANET of the late 1960's and early 1970's.

**Intranet.** A private network inside a company or organization that uses the same kinds of software that you would find on the public Internet, but that is only for internal use. As the Internet has become more popular, many of the tools used on the Internet are being used in private networks. For example, many companies have Web servers that are available only to employees.

**IMAP4.** Internet Message Access Protocol - Version 4.

**IMS.** Information Management System.

**IP.** Internet Protocol.

**IPSec.** IP Security Protocol. Provides cryptographic security services at the network layer.

**ISAPI.** Internet Server API.

**ISC.** The Installation Support Centre. Based in Hursley in the UK the EMEA ISC provides early education and support for new technologies.

**ISD.** InfraStructure Design. IBM ISD is a path of the AD/SI method component of WSDDM. ISD is used to design the delivery platform supporting an application. ISD phases are Requirements, Architecture, Infrastructure Specification, and Component Specification and Selection. ISD is based on the original E2E method that provided a formal way of designing complex systems.

**ISD.** The Integrated Service Offering Development process of IBM Global Services.

**ISP.** Internet Server Provider.

**I/T.** Information Technology.

**ITSO.** International Technical Support Organization

**Java.** Java is a new programming language invented by Sun Microsystems that is specifically designed for writing programs that can be safely downloaded to your computer through the Internet and immediately run without fear of viruses or other harm to your computer or files. Using small Java programs called *applets*, Web pages can include functions, such as animations, calculators, and other fancy tricks. We can expect to see a huge variety of features added to the Web using Java, since you can write a Java program to do almost anything a regular computer program can do, and then include that Java program in a Web page.

**Java archive (JAR).** A platform-independent file format that groups many files into one. JAR files are used for compression, reduced download time, and security. Because the JAR format is written in Java, JAR files are fully extensible.

**JavaBeans.** In JDK 1.1, the specification that defines the platform-neutral component model used to represent parts. Instances of JavaBeans (often called *beans*) may have methods, properties, and events.

**Java Database Connectivity (JDBC).** In JDK 1.1, the specification that defines an API that enables programs to access databases that comply with this standard.

**Java Development Kit (JDK).** The Java Development Kit 1.1 is the latest set of Java technologies made available to licensed developers by Sun Microsystems. Each release of the JDK contains the following: the Java Compiler, Java Virtual Machine, Java Class Libraries, Java Applet Viewer, Java Debugger, and other tools.

**Java Foundation Classes (JFC).** Developed by Netscape, Sun, and IBM, JFCs are building blocks that are helpful in developing interfaces to Java applications. They allow Java applications to interact more completely with the existing operating systems.

**JavaBean.** A JavaBean is a component that can be integrated into an application with other beans that were developed separately. This single application can be used stand-alone, within a

browser, and as an ActiveX component. JavaBeans are intended to be local to a single process, and they are often visible at runtime. This visual component may be, for example, a button, list box, graphic, or chart.

**JDBC.** This is a trademark, often referred to as *Java DataBase Connectivity*.

**JDK.** Java Developer's Kit.

**JFC.** Java Foundation Class.

**JIT.** Just In Time.

**JVM.** Java Virtual Machine.

**Local Area Network (LAN).** This is a computer network located at a user's establishment within a limited geographical area. A LAN typically consists of one or more server machines providing services to a number of client workstations.

**LDAP.** Lightweight Directory Access Protocol.

**LEI.** Lotus Enterprise Integration. This is the product formerly known as NotesPump.

**LSX.** Lotus Script Extensions.

**LU type 6.2 (LU 6.2).** A type of logical unit used for CICS intersystem communication (ISC). LU 6.2 architecture supports CICS host-to-system-level products and CICS host-to-device-level products. APPC is the protocol boundary of the LU 6.2 architecture.

**Logical unit of work (LUW).** An update that durably transforms a resource from one consistent state to another consistent state. A sequence of processing actions (for example, database changes) that must be completed before any of the individual actions can be regarded as committed. When changes are committed (by successful completion of the LUW and recording of the synch point on the system log), they do not need to be backed out after a subsequent error within the task or region. The end of an LUW is marked in a transaction by a synch point that is issued by either the user program or the CICS server, at the end of task. If there are no user synch points, the entire task is an LUW.

**Messaging.** A communication model whereby the distributed applications communicate by sending messages to each other. A message is typically a short packet of information that does not necessarily require a reply. Messaging implements asynchronous communications.

**Method.** A fragment of Java code within a class that can be invoked and passed a set of parameters to perform a specific task.

**Multipurpose Internet Mail Extension (MIME).** The Internet standard for mail that supports text, images, audio, and video.

**MIB.** Management Information Base.

**MIME.** Multipurpose Internet Mail Extensions.

**MOM.** Message-Oriented Middleware.

**MQ.** Message Queue.

**MVS.** Multiple Virtual Storage. For many years the flagship operating system for IBM large Enterprise Servers. In its latest releases, it is formally known as OS/390, although the term MVS is still used unofficially.

**NC.** Network Computer or Network Computing.

**NCF.** Network Computing Framework.

**NNTP.** Network News Transfer Protocol.

**NSAPI.** Netscape Server API.

**NSF.** Notes database file extension.

**NT.** Windows NT (New Technology).

**Online Transaction Processing (OLTP).** A style of computing that supports interactive applications in which requests submitted by terminal users are processed as soon as they are received. Results are returned to the requester in a relatively short period of time. An online transaction-processing system supervises the sharing of resources to allow efficient processing of multiple transactions at the same time.

**Object.** (1) A computer representation of something that a user can work with to perform a task. An object can appear as text or an icon. (2) A collection of data and methods that operate on that data, which together represent a logical entity in the system. In object-oriented

programming, objects are grouped into classes that share common data definitions and methods. Each object in the class is said to be an instance of the class. (3) An instance of an object class consisting of attributes, a data structure, and operational methods. It can represent a person, place, thing, event, or concept. Each instance has the same properties, attributes, and methods as other instances of the object class, although it has unique values assigned to its attributes.

**ODBC Driver.** An ODBC driver is a dynamically linked library (DLL) that implements ODBC function calls and interacts with a data source.

**ODBC Driver Manager.** The ODBC driver manager, provided by Microsoft, is a DLL with an import library. The primary purpose of the Driver Manager is to load ODBC drivers. The Driver Manager also provides entry points to ODBC functions for each driver and parameter validation and sequence validation for ODBC calls.

**Open Database Connectivity (ODBC).** A Microsoft-developed C database application programming interface (API) that allows access to database management systems calling callable SQL, which does not require the use of a SQL preprocessor. In addition, ODBC provides an architecture that allows users to add modules called *database drivers* that link the application to their choice of database management systems at run time. This means applications no longer need to be directly linked to the modules of all the database management systems that are supported.

**Object Request Broker (ORB).** A CORBA term designating the means by which objects transparently make requests and receive responses from other objects, whether they are local or remote.

**OS/390.** The latest version of MVS. OS/390's standard OpenEdition function provides certified POSIX compliant interfaces in addition to enhanced support for its traditional programming interfaces.

**ODBC.** Open DataBase Connectivity

**OMG.** Object Management Group.

**PERL.** Practical Extraction and Reporting Language.

**PGP.** Pretty Good Privacy

**PKI.** Public Key Infrastructure.

**POP3.** Post Office Protocol 3

**Port.** A TCP/IP terminology; a port is a separately-addressable point to which an application can connect. For example, by default, HTTP uses port 80 and Secure HTTP (HTTPS) uses port 443.

**Protocol.** (1) The set of all messages to which an object will respond. (2) Specification of the structure and meaning (the semantics) of messages that are exchanged between a client and a server. (3) Computer rules that provide uniform specifications so that computer hardware and operating systems can communicate. It's similar to the way that mail in countries around the world is addressed in the same basic format so that postal workers know where to find the recipient's address, the sender's return address, and the postage stamp. Regardless of the underlying language, the basic protocols remain the same.

**Proxy.** This is an application gateway from one network to another for a specific network application, such as Telnet or FTP, for example, where a firewall's proxy Telnet server performs authentication of the user and then allows the traffic to flow through the proxy as if it were not there. Function is performed in the firewall and not in the client workstation causing more load in the firewall. Compare with socks.

**RACF.** Resource Access Control Facility.

**RDMS.** Relational Database Management System.

**RFC.** Request For Comment. Internet Standards are defined in documents known as RFCs.

**Remote Method Invocation (RMI).** In JDK 1.1, the API that allows you to write distributed Java programs allowing methods of remote Java objects to be accessed from other Java virtual machines.

**Remote Procedure Call (RPC).** A communication model where requests are made by function calls to distributed procedure elsewhere. The location of the procedures is transparent to the calling application.

**RSA.** Rivest-Shamir-Adleman algorithm.

**Sandbox.** A restricted environment provided by the Web browser in which Java applets run. The sandbox offers them services and prevents them from doing anything naughty, such as doing file I/O or talking to strangers (servers other than the one from which the applet was loaded). The analogy of applets to children led to calling the environment in which they run the *sandbox*.

**SAP.** Originally "Systemanalyse und Programmentwicklung" and now named Systems, Applications, and Products in Data Processing, SAP supplies widely-used software for integrated business solutions.

**Schema.** In the Data Access Builder, the representation of the database that will be mapped. In the Data Access Builder, the mapping contains a set of definitions for all attributes matching all the columns for your database table, view, or SQL statement, as well as information required to generate Java classes.

**Server.** A computer that provides services to multiple users or workstations in a network; for example, a file server, a print server, or a mail server.

**SES/workbench.** A simulation product for behavioral and performance modeling of complex client/server, network, software, and hardware systems.

**SET.** Secure Electronic Transaction.

**SHTTP.** Secure Hypertext Transfer Protocol.

**SI.** Systems Integration. One of the six competency areas of IBM Global Services.

**SI/AD.** Systems Integration/Application Development. One of the six competency areas announced by IBM Global Services in May 1998, later renamed simply SI.

**SMTP.** Simple Mail Transport Protocol

**SNMP.** Simple Network Management Protocol

**Socket Secure (SOCKS).** The gateway that allows compliant client code (client code made socket secure) to establish a session with a remote host.

**SQL.** Structured Query Language.

**SSL.** Secure Sockets Layer.

**S/MIME.** Secure MIME.

**Telnet.** U.S. Dept. of Defense virtual terminal protocol.

**TME.** Tivoli Management Environment.

**Transmission Control Protocol/Internet Protocol (TCP/IP).** This is the basic programming foundation that carries computer messages around the globe via the Internet. It is the suite of protocols that defines the Internet. Originally designed for the UNIX operating system, TCP/IP software is now available for every major kind of computer operating system. To be truly on the Internet, your computer must have TCP/IP software.

**Thin client.** Thin client usually refers to a system that runs on a resource-constrained machine or that runs a small operating system. Thin clients do not require local system administration, and they execute Java applications delivered over the network.

**Transaction.** This is a unit of processing consisting of one or more application programs initiated by a single request. A transaction can require the initiation of one or more tasks for its execution.

**Transaction Processing.** This is a style of computing that supports interactive applications in which requests submitted by users are processed as soon as they are received. Results are returned to the requester in a relatively short period of time. A transaction processing system supervises the sharing of resources for processing multiple transactions at the same time.

**UDB.** Universal Data Base. The latest release of IBM DB2.

**Uniform Resource Locator (URL).** This is the standard to identify resources on the World Wide Web.

**VA.** Visual Age.

**VAJ.** Visual Age for Java.

**VB.** Visual Basic.

**Virtual Machine (VM).** A software program that executes other computer programs. It allows a physical machine, a computer, to behave as if it were another physical machine.

**VM.** Virtual Machine.

**VPN.** Virtual Private network.

**VTAM.** Virtual Telecommunications Access Method.

**Web server.** This is the server component of the World Wide Web. It is responsible for servicing requests for information from Web browsers. The information can be a file retrieved from the server's local disk or generated by a program called by the server to perform a specific application function.

**Widget.** In this context, it is a generic term for something that can be put on a window, such as a button, scrollbar, label, listbox, menu, or checkbox.

**Workstation.** This is a configuration of input/output equipment at which an operator works. A terminal or microcomputer, usually one that is connected to a mainframe or a network, at which a user can perform applications.

**World Wide Web (WWW or Web).** A graphic hypertextual multimedia Internet service.

**WSDDM.** IBM WorldWide Solution Design and Delivery Methods. WSDDM is a collection of generic plans and methods representing IBM best practices for planning, managing, and delivering projects.

**WYSIWYG.** What You See is What You Get.

**XML.** Extensible Markup Language.

---

## Abbreviations and acronyms

<b>IBM</b>	International Business Machines Corporation	<b>DPL</b>	Distributed Program Link (CICS)
<b>ITSO</b>	International Technical Support Organization	<b>DRDA</b>	Distributed Relational Database Architecture
<b>API</b>	Application Program Interface	<b>EBCDIC</b>	Extended Binary Coded Decimal Interchange Code
<b>ARM</b>	Application Request Manager	<b>ECI</b>	External Call Interface (CICS)
<b>ASCII</b>	American National Standard Code for Information Interchange	<b>EPI</b>	External Presentation Interface (CICS)
<b>AWT</b>	Abstract Windowing Toolkit	<b>ESA</b>	Enterprise Systems Architecture
<b>CA</b>	Certification Authority	<b>EXCI</b>	External CICS Interface
<b>CAB</b>	Cabinet (Microsoft)	<b>FTP</b>	File Transfer Protocol
<b>CAE</b>	Client Application Enabler	<b>GUI</b>	Graphical User Interface
<b>CGI</b>	Common Gateway Interface	<b>HTML</b>	Hypertext Markup Language
<b>CICS</b>	Customer Information Control System	<b>HTTP</b>	Hypertext Transfer Protocol
<b>CICS TS</b>	CICS Transaction Server for OS/390	<b>IBM</b>	International Business Machines Corporation
<b>CLI</b>	Call Level Interface	<b>IDE</b>	Integrated Development Environment
<b>COMMAREA</b>	Communication Area (CICS)	<b>IIOP</b>	Internet Inter-ORB Protocol
<b>CORBA</b>	Common Object Request Broker Architecture	<b>IMS</b>	Information Management System
<b>DBMS</b>	Database Management System	<b>IS</b>	Information System
<b>DB2</b>	Database 2	<b>ITSO</b>	International Technical Support Organization
<b>DCE</b>	Distributed Computing Environment	<b>JAR</b>	Java Archive
<b>DDCS/2</b>	Distributed Database Connection Services/2	<b>JDBC</b>	Java Database Connectivity
<b>DLL</b>	Dynamic Link Library	<b>JDK</b>	Java Development Kit
		<b>JFC</b>	Java Foundation Classes

<b>JVM</b>	Java Virtual Machine	<b>SIT</b>	System Initialization Table
<b>LAN</b>	Local Area Network		
<b>LDAP</b>	Lightweight Directory Access Protocol	<b>SMTP</b>	Simple Mail Transfer Protocol
<b>LUW</b>	Logical Unit of Work	<b>SNA</b>	Systems Network Architecture
<b>MIME</b>	Multipurpose Internet Mail Extensions	<b>SNMP</b>	Simple Network Management Protocol
<b>MVS</b>	Multiple Virtual Storage	<b>SQL</b>	Structured Query Language
<b>NC</b>	Network Computer	<b>SSL</b>	Secure Sockets Layer
<b>NCF</b>	Network Computing Framework	<b>TCP/IP</b>	Transmission Control Protocol/Internet Protocol
<b>NetBIOS</b>	Network Basic Input/Output System	<b>TSO</b>	Time Sharing Option
<b>NNTP</b>	NetNews Transfer Protocol	<b>URL</b>	Uniform Resource Locator
<b>NT</b>	Microsoft Windows NT (New Technology)	<b>XA</b>	Extended Architecture
<b>ODBC</b>	Open Database Connectivity	<b>XML</b>	Extensible Markup Language
<b>OLTP</b>	Online Transaction Processing		
<b>ORB</b>	Object Request Broker		
<b>OS/2</b>	Operating System/2		
<b>OSF</b>	Open Software Foundation		
<b>PC</b>	Personal Computer		
<b>POP</b>	Post Office Protocol		
<b>RACF</b>	Resource Access Control Facility		
<b>RAD</b>	Rapid Application Development		
<b>RMI</b>	Remote Method Invocation		
<b>SDK</b>	Software Developer's Kit		
<b>SET</b>	Secure Electronic Transaction		
<b>SHTTP</b>	Secure Hypertext Transport Protocol		



---

## Index

### A

- abstract windowing toolkit, See AWT
- Access protocol 77
- ActiveX 205
- adapter
  - placement 182
  - processing flow 182
  - transformation 188
  - types 191
- adapter placement 181
- adapters
  - Bridges 191
  - Configurable 191
  - Custom 191
- administrative agent 160
- AIX 241
- Apache 72
- API
  - implementation functions 225
  - utility functions 225
- APPC 241
- application
  - elements 68, 171
  - factors 27
  - General Principles 179
  - influencing factors 27
  - node types 53
  - request manager, See ARM
  - runtime topology 41
- Application Framework for e-business 25, 67
- Application Integration 20
- application server 63
- Application services 73
- application topology 27
- applications 7
  - adapter placement 181
  - classification schema 7
  - distribution 179
  - existing 60
  - Inter-business 7
  - Intra-business 7
  - legacy 60
  - partner interface 200
  - performance 179
  - Role of the Adapter 182
  - synchronization 179

- transform 188
- Application-to-Application 13
- Ariba 90
- Asynchronous Communication 174
- authentication 238
- Authorization 238

### B

- B2B
  - application elements 68
  - Business Context 28
  - challenges 13
  - frameworks 91
  - Functional components 28
  - node types 53
  - runtime topology 41
  - Solution Components 14
- B2BI
  - General Principles 179
  - partner interface 200
- B2Bi
  - Role of the Adapter 182
- B2M2B 11
- BEA 75
- BizTalk 93
- BLOB
  - message domain 165
- business context 28
- Business patterns 19
- Business-to-Business 20

### C

- cache 73
- Capacity 209
- CBL 97
  - description 97
- CICS 79, 103
- client 70
- Client Connection 142
- Client Reply
  - MQSeries
    - Client Reply 146
- Client Request 144
- clustering 73
- COM 70, 73
- COM/DCOM 89

- Commerce Business Library 97
- Commerce One 90
- Commerce XML 96
- Commerce XML (cXML) 92
- common gateway interface, *See* CGI
- common message interface 151
- communication area, *See* COMMAREA
- CompCode 149
- Component model 73
- Compute node 152
- Configuration Management 237
- Configuration Manager 153
  - dependencies 163
- configuration repository 153
- connection complexity 187
- connection pooling 73
- connectors 77
- content management 73
- Control Center 154
- CORBA 13, 70, 73, 89
- Custom Wire Format
  - See* CWF
- CWF 166
- cXML 90, 92, 96

## D

- Data confidentiality 238
- Data integrity 238
- Database node 152
- Database Resilience 209
- Database server node 62
- Databases
  - dependencies 164
- DB2 103
- DCE 183
- demilitarized zone 54, 308
- Demilitarized Zone (DMZ) 62
- description 96
- Design patterns 18
- Development
  - Build cycle 214
  - Macro design 214
  - Micro design 214
  - Deployment 214
  - Solution outline 214
- DHCP 76
- Directory and Security Services 56

- Distributed Management Task Force 92
- DMTF 92
- DMZ 54, 63, 308
- Document Type Definitions 92
- domain firewall 59
- Domain Name Service (DNS) 60
- DTD 91, 92

## E

- e-business
  - about 3
  - adoption process 5
  - applications 7
  - competitiveness 7
  - transformation 4
  - value 6
- ebXML 91
- ebXML project 92
- eCo 92
- eCo Specification 92
- EDI 12
  - description 99
- EDI translation package 54
- EDI/MIME Translator 55
- EDIFACT 54
- eMarketPlace 11
- encryption 238
- enterprise 28
- Enterprise Java Beans 13
- Enterprise JavaBeans 73
- Enterprise Solution Structure (ESS) 17
- ERP 28
- ESQL 152
- ESS 22
- external call interface, *See* ECI
- external CICS interface, *See* EXCI
- external presentation interface, *See* EPI
- Extranet 53

## F

- fail-over 73
- Filter node 152
- filtering 70
- filtering technologies 70
- framework 67, 69
- frameworks 91
- full-duplex 174

## G

generic XML messages 164  
graphical user interface, See GUI

## H

half-duplex 174  
HP-UX 241  
HTML 62  
HTTP 62, 72, 90  
Hub 184  
Hub and spoke 184

## I

IBM Application Framework for e-business 67, 69  
IBM Global Services methodology 17, 213  
IBM Software  
    Application Accelerators 106  
    Customer and Partner Applications 108  
    Foundation 102  
    Foundation Extensions 102  
IBMPrimitives 152  
ICE 96, 98  
    description 98  
IDE 103  
Identification 238  
IMS 103  
influences 70  
Influencing factors 27  
Information & Content Exchange 96, 98  
Innovation 4  
Inprise 75  
Integration services 78  
Inter-business 7  
Internet Gateway 56  
Internet Information Server 74  
Internet Open Trading Protocol 98  
Internet pass-thru 307  
Internet services 72  
Intra-business 7  
Intranet 54  
Iona 77  
IOTP 98  
    description 98  
iPlanet 74, 75  
IPSEC 62  
ISAPI 73  
ISV 108

## J

Java archive, See JAR  
Java Database Connectivity, See JDBC  
Java development kit, See JDK  
Java IDE 103  
Java Server Pages 62, 103  
Java virtual machine, See JVM  
JD Edwards 79  
JDBC 79  
JSP 103

## L

LDAP 57, 76, 77, 183  
lightweight directory access protocol 57  
lil 160  
load balancer 62  
load balancing 73  
Lotus Domino 107

## M

message broker 156  
    dependencies 163  
    instance name 158  
message channel agent 240  
message domain 165  
message flow 151  
message flow execution engine 160  
message format 165  
message repository 153  
message set 165  
message type 165  
messaging  
    Message Broker 60  
Microsoft 70, 72  
    COM 70  
    Internet Information Server 74  
    ISAPI 73  
MIME 55  
MIME-based Secure EDI 56  
MIMS 74  
Mobile 77  
MQ Adapter 61  
MQ Server 61  
MQBACK 148  
MQBEGIN 148  
MQCMIT 148  
MQInput node 152  
MQINQ 148

- MQOutput node 152
- MQPUT1 148
- MQRFH 208
- MQRFH2 208
- MQSeries 102
  - Adapter 192
  - Adapter Builder 192
  - Adapter Kernel 193
  - Adapter Sets 194
  - AMI 196
  - Automation Classes for ActiveX 205
  - C/S Connection details 143
  - Client Connection 142
  - Client Server Connection 142
  - Code Fragment 148
  - description 102
  - guidelines 194
  - Integrator Libraries 194
  - JMS 196
  - LotusScript Extension 205
  - management 236
  - MQI 196
  - Partner Interface 200
  - Security 238
  - Server Connection 142
  - Server Reply 146
  - Server Request 145
  - Systems management products 237
  - Topology 2 171
  - Topology 3 171
  - XML 166
- MQSeries
  - Client Request 144
- MQSeries Integrator 105
  - brokers 151
  - Components 151
  - Configuration Manager 151
  - Control Center 151
  - Databases 162
  - Dependencies 162
  - Description 151
  - description 105
  - Message Repository Manager 151
  - MQSeries Queue Manager 151
  - Security subsystem 162
  - User Name Server 151
  - XML 167
- MQSeries WorkFlow 107
- MQSET 148

- MRM
  - message domain 165
- Multi-Purpose Internet Mail Extensions 55
- multi-threaded 73

## N

- namespaces 91
- NEON
  - message domain 165
- Netscape 72
  - iPlanet 74
  - NSAPI 73
- node types 53
- Non-repudiation 238
- NSAPI 73

## O

- OAG 92
- OASIS 91, 92
- OBI 94
  - definition 94
  - description 94
- object request broker, See ORB
- ODBC 79
- OFX 98
  - description 98
- OMG 70
- ompetitiveness 7
- online transaction processing , See OLTP
- Open Applications Group 92
- Open Financial Exchange 98
- Open Trading Protocol 98
- Operations Management 237
- Orbix 77
- OTP 98
  - description 98

## P

- P3P 98
  - description 98
- parser 225
- Partner Interface Processes 90
- Pattern Development Kit (PDK) 23
- PeopleSoft 79
- Performance 209
- performance 73
- Performance Management 237

- persistence 71, 77
- PIPS 90
- PIX 98
  - description 98
- PKI 77
- Platform for Privacy Preferences Project 98
- plug-in 224
- point to point integration 183
- Portal 104
- predefined messages 164
- Problem Management 237
- Product Information Exchange 98
- project management 88
- protocol firewall 59
- Protocols 89
  - CBL 97
  - cXML 90
  - DMTF 92
  - ebXML 91
  - ICE 96, 98
  - IOTP 98
  - OFX 98
  - OTP 98
  - P3P 98
  - PIX 98
  - RosettaNet 90
  - SET 97
  - SOAP 96
  - UDDI 107
  - VoiceXML 104
  - WBEM 92
  - xCBL 90
  - XML 91
  - XML/EDI 99
- Public Key Infrastructure 77
- Public Key Infrastructure (PKI) 59

## Q

- Queue Manager 58

## R

- rapid application development, See RAD
- Rational Rose 219
- Reason Field 149
- reliability 73
- Resilience 209
- reverse proxy 63
- RFC1767 56

- RFC2026 56
- RMI 89
- robust communication 174
- robustness 73
- RosettaNet 90, 93, 95
  - description 95
  - PIP 95
- RPC 97
- runtime topology 41
  - definition 41
  - Direct with Adapter/Bridge 46
  - Document Exchange 42
  - Message Broker 51

## S

- SAP 79
- scalability 73
- schemas 91
- Secure Electronic Transaction 97
- secure electronic transaction, See SET
- SecureWay Software 77
- Security 57, 97
- security 77
- self-defined messages 164
- Server Connection 142
- Server Reply 146
- Server Request 145
- Services
  - Application 72
  - Application Development 72
  - Integration 72
  - Internet 72
  - Management 72
- servlet redirector 63
- SET 97
- simple mail transport protocol 56
- Simple Object Access Protocol 96
- SMTP 56
  - Server 56
- SNA 71
- SOAP 96
  - contents 96
  - description 96
- Spoke 184
- SSL 62
- Standards 89
- state management 73
- Store and Forward 174

- connecting to 180
- invasive insertion 180
- passive adaptation 180
- straight-through processing 28
- Sun 70
- Sun EJB 70, 75
- Synchronous Communication 174
- System integrators 108
- Systems Management 235
- systems management 188

## T

- technology classification 67
- technology influences 70
- threads 73
- Tivoli Policy Director 106
- topology
  - application 27
  - runtime 41
- TPA 91
- Trading Partner Agreement 91
- Transformation 4
- transformation 188
- tunneling protocols 307
- Tuxedo 79
- Two System Update 175

## U

- UDDI 107
- UN/CEFACT 91, 92
- Unified Model Language (UML) 219
- uniform resource locator , See URL
- Universal Description, Discovery and Integration (UDDI) services 107
- user
  - interface, See GUI
- User Name Server 161
  - dependencies 164
- User-to-Business 20
- User-to-Data 20
- User-to-Online Buying 20
- User-to-User 20
- user-written node 224

## V

- value added network 55
- VAN 55

- access point 55
- mailbox 55
- Virtual Private Network 59, 77
- Visigenic 77
- VisualAge Application Rules 103
- VisualAge for Java 103
- VisualAge Generator 103
- Voice 104
- VoiceXML 104
- VPN 59, 77

## W

- Warehouse node 152
- WBEM 92
- Web application server 61, 63
- Web application server node 61
- Web Based Enterprise Management 92
- Web integrators 108
- Web server redirector 62
- WebSphere Application Server 102
- WebSphere Business Components 104
- WebSphere Business-to-Business Integrator 107
- WebSphere Commerce Suite 106
- WebSphere Edge Server 105
- WebSphere Everyplace Suite 105
- WebSphere Homepage Builder 104
- WebSphere Host Integration Solution 106
- WebSphere Personalization 105
- WebSphere Portal Server 104
- WebSphere Site Analyzer 105
- WebSphere Studio 103
- WebSphere Transcoding Publisher 104
- WebSphere Voice Server 104
- wire format
  - See CWF

## X

- X509 94
- xCBL 90
- XML 91, 99, 164
  - message domain 165
  - MQSeries 168
  - MQSeries Integrator 168
- XML.org 91
- XML/EDI 99
- XML/EDI Group 93

## IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, **please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.**

- Use the online **Contact us** review redbook form found at [ibm.com/redbooks](http://ibm.com/redbooks)
- Fax this form to: USA International Access Code + 1 845 432 8264
- Send your comments in an Internet note to [redbook@us.ibm.com](mailto:redbook@us.ibm.com)

<b>Document Number</b>	SG24-6010-00
<b>Redbook Title</b>	Business-to-Business Integration Using MQSeries and MQSI Patterns for e-business Series
<b>Review</b>	<div></div> <div></div> <div></div> <div></div> <div></div> <div></div>
<b>What other subjects would you like to see IBM Redbooks address?</b>	<div></div> <div></div> <div></div>
<b>Please rate your overall satisfaction:</b>	<input type="radio"/> Very Good <input type="radio"/> Good <input type="radio"/> Average <input type="radio"/> Poor
<b>Please identify yourself as belonging to one of the following groups:</b>	<input type="radio"/> Customer <input type="radio"/> Business Partner <input type="radio"/> Solution Developer <input type="radio"/> IBM, Lotus or Tivoli Employee <input type="radio"/> None of the above
<b>Your email address:</b> The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.	<input type="radio"/> Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.
<b>Questions about IBM's privacy policy?</b>	The following link explains how we protect your personal information. <a href="http://ibm.com/privacy/yourprivacy/">ibm.com/privacy/yourprivacy/</a>







## Business-to-Business Integration Using MQSeries and MQSI







# Business-to-Business Integration Using MQSeries and MQSI Patterns for e-business Series



**Redbooks**

**Select topologies  
and mappings to  
build B2B Integration  
e-business solutions**

**Gain insight into  
available products  
and design  
guidelines**

**Learn from an  
implementation  
example**

Patterns for e-business are a group of proven, reusable assets that can help speed the process of developing applications. The patterns discussed in this book, Business-to-Business Integration patterns two and three, form the basis for many of the more complex and functional B2B patterns. It is relevant to all enterprises dealing with partner integration issues over the Internet.

Application topology 2 describes a scenario in which messages are being passed between two enterprise applications and no routing is performed. Topology 3 extends topology 2 to describe the scenario where routing is required for multiple cross enterprise applications to communicate.

Part 1 of the redbook takes you through the process of choosing an application topology and a runtime topology. It gives you possible product mappings for implementation of the chosen runtime topology and introduces all the topologies, even though only two of the patterns are covered in this book. Part 2 is a set of guidelines, based on topologies 2 and 3, for building your e-business application. It includes information on application design, technology options, application development, performance, and security.

Part 3 takes you through a working example showing the simple implementation of an integration pattern using application topology 3.

## **INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION**

### **BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
[ibm.com/redbooks](http://ibm.com/redbooks)

SG24-6010-00

ISBN 0738418579