

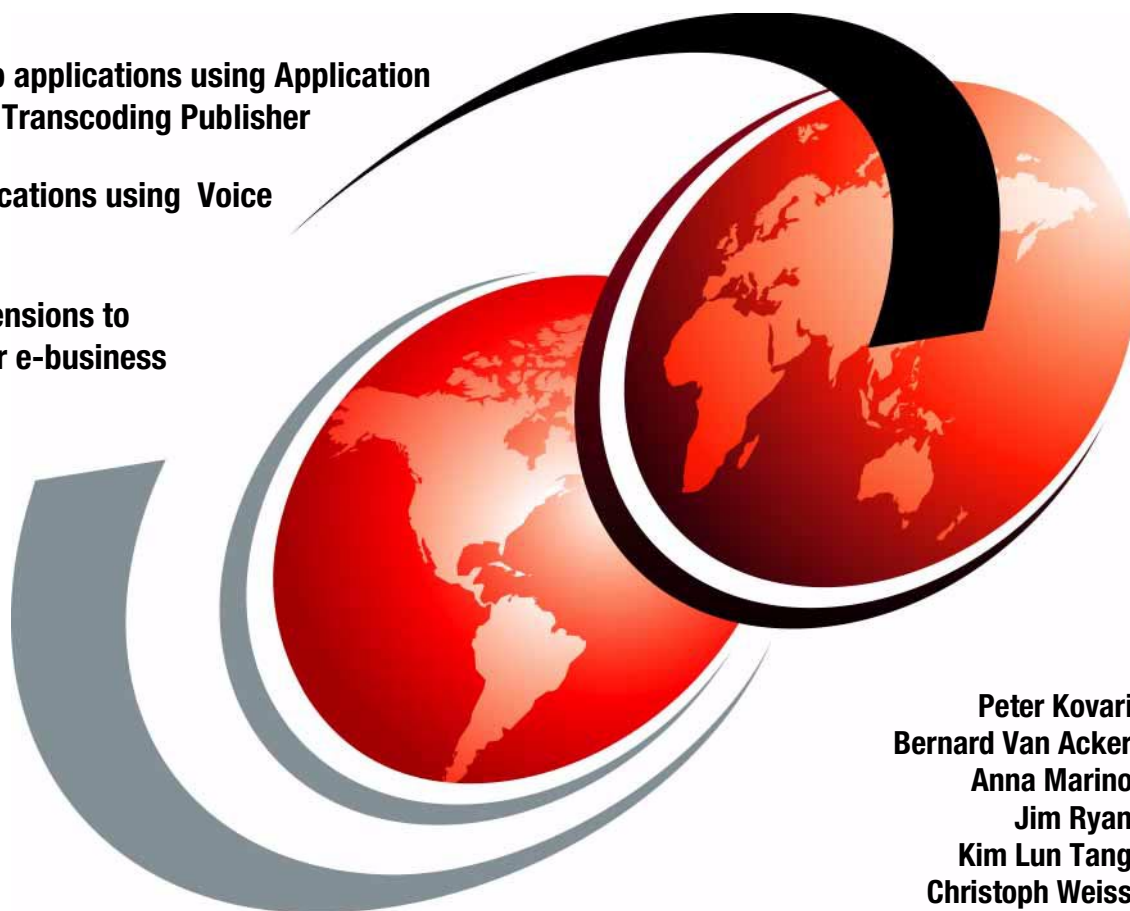


Mobile Applications with IBM WebSphere Everyplace Access Design and Development

Mobile Web applications using Application
Server and Transcoding Publisher

Voice applications using Voice
Server

Mobile extensions to
Patterns for e-business



Peter Kovari
Bernard Van Acker
Anna Marino
Jim Ryan
Kim Lun Tang
Christoph Weiss



International Technical Support Organization

**Mobile Applications with IBM WebSphere
Everyplace Access Design and Development**

October 2001

Take Note! Before using this information and the product it supports, be sure to read the general information in “Special notices” on page 421.

First Edition (October 2001)

This edition applies to WebSphere Everyplace Access Offering V1R1 on Windows NT, and AIX.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 662
P.O. Box 12195
Research Triangle Park, NC 27709-2195

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 2001. 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

Preface	xiii
The team that wrote this redbook	xiii
Special notice	xv
IBM trademarks	xvi
Comments welcome	xvi
 Part 1. Introduction	 1
 Chapter 1. Introduction to IBM WebSphere Everyplace Access V1R1	 3
1.1 Definitions	4
1.2 Business drivers	4
1.3 WebSphere Everyplace Access in this book	6
1.4 Run-time applications	7
1.4.1 WebSphere Application Server	7
1.4.2 WebSphere Transcoding Publisher	7
1.4.3 WebSphere Voice Server	7
1.5 Development applications	8
1.5.1 WebSphere Studio	8
1.5.2 VisualAge for JavaEE	8
 Chapter 2. Overview of mobile technologies	 9
2.1 Technology background	10
2.2 Mobile accessibility	10
2.3 Wireless networks	11
2.3.1 Mobile communications network history	11
2.3.2 GSM	13
2.3.3 GPRS	14
2.3.4 Mobitex	14
2.3.5 CDPD	15
2.3.6 PDC	16
2.3.7 IMT-2000	16
2.3.8 Wireless LANS	17
2.4 Wireless protocols	17
2.4.1 HTTP protocol	18
2.4.2 WAP	18
2.4.3 M-services	19
2.4.4 i-mode	19
2.4.5 Web Clipping	20
2.5 Mobile devices	22

2.5.1	Phones for voice interaction	22
2.5.2	Mobile phones	22
2.5.3	PDAs	23
2.5.4	Wireless laptops	27
2.5.5	Mobile device pros and cons	28
2.6	Emulators and mobile clients	29
2.7	Content and markup languages	30
2.8	Wireless Service Providers	30
2.9	Where to find more information	31
Chapter 3. Overview of speech technology		33
3.1	Speech-enabled versus text-based applications	34
3.1.1	Benefits of voice applications	35
3.1.2	Limitations of voice applications	35
3.2	Speech recognition	36
3.2.1	System architecture	39
3.2.2	Natural Language Understanding and Dialog Management	42
3.2.3	Application styles	43
3.2.4	Speech recognition errors	44
3.2.5	Recognition performance	46
3.3	Speech synthesis	47
3.3.1	Synthesizer architecture	49
3.3.2	Quality assessment for TTS	52
3.4	IBM ViaVoice	54
3.4.1	Multilingual support for ViaVoice	54
3.4.2	ViaVoice limitations	55
3.5	Examples of voice-enabled applications	56
3.5.1	Speech reco applications	56
3.5.2	TTS applications	57
3.6	Future development	59
3.7	Where to find more Information	62
Part 2. Patterns for e-business		63
Chapter 4. Patterns for e-business		65
4.1	Using Patterns for e-business	66
4.2	Business patterns	67
4.3	Integration patterns	69
4.4	Composite patterns	70
4.5	The patterns used in this book	71
4.6	Where to find more information	71
Chapter 5. Application patterns		73
5.1	Application patterns	74

5.2	Application patterns for Access integration	75
5.2.1	Pervasive Device Access application pattern	75
5.3	Application patterns for Self-Service	76
5.3.1	Stand-Alone Single Channel application pattern	77
Chapter 6.	Runtime patterns	79
6.1	Runtime nodes	80
6.2	Runtime pattern for the Self-Service application	82
6.2.1	Basic Runtime pattern	83
6.2.2	Runtime variation	84
6.3	Runtime pattern for the Pervasive Device Access	85
6.3.1	Base Runtime pattern	85
6.3.2	Runtime pattern variation	87
6.4	Gateway placement	88
Chapter 7.	Runtime product mappings	89
7.1	Selecting products	90
7.2	Product mappings	91
7.2.1	The WebSphere Everyplace Access V1R1 offering	94
7.2.2	Additional products for the offering	96
7.3	WebSphere Transcoding Publisher considerations	97
7.3.1	Proxy model	97
7.3.2	Filter model in WebSphere Application Server	98
7.3.3	Running JavaBean transcoders	98
7.3.4	Choosing the right model	98
7.4	Where to find more information	99
Part 3.	Wireless Internet application: guidelines	101
Chapter 8.	Solution design	103
8.1	The different modes of pervasive computing	104
8.1.1	Synchronous	104
8.1.2	Notification	105
8.1.3	Asynchronous	105
8.1.4	Voice	106
8.1.5	Multimodal	107
8.2	Decision tree	107
8.2.1	Device classes	107
8.2.2	The decision tree	108
8.2.3	When to apply tweaking	113
8.2.4	Custom solutions	115
8.3	Visual design	115
8.4	Non-visual design: voice	116
8.4.1	Challenges of voice versus visual applications	116

8.4.2 Voice applied to the decision tree	119
8.4.3 Other aspects of voice related to solution design	120
Chapter 9. Application design	121
9.1 Web application	122
9.1.1 Model-View-Controller.	126
9.1.2 Sample Web application	128
9.2 Extending a Web application to a mobile application	130
9.2.1 Extending the architecture	131
9.3 Mobile architecture	132
9.3.1 Web Intermediaries.	134
9.3.2 A mobile application	137
9.3.3 IBM WebSphere Transcoding Publisher	141
9.4 Design patterns	146
9.4.1 Object	148
9.4.2 Command pattern	149
9.4.3 Template pattern	149
9.4.4 Factory pattern	150
9.4.5 Proxy pattern.	151
9.4.6 Advantages of patterns	151
9.5 Where to find more information	152
9.5.1 IBM	152
9.5.2 Outside of IBM	152
Chapter 10. Application development	153
10.1 Web application elements	154
10.1.1 Java servlet.	154
10.1.2 JSPs	154
10.1.3 JavaBeans	156
10.1.4 Enterprise JavaBeans	157
10.1.5 XML.	159
10.1.6 XSL	160
10.1.7 XSLT	161
10.1.8 HTML, cHTML, HDML, WML	162
10.1.9 VoiceXML	164
10.2 Development tools	166
10.2.1 WebSphere Studio	166
10.2.2 VisualAge for Java	173
10.2.3 Development tools in WTP	176
10.2.4 XML tools	179
10.2.5 Voice SDK.	180
10.3 Tools for testing the application	182
10.3.1 Overview	183

10.3.2	Testing the HTML application	183
10.3.3	Testing the simplified HTML application	184
10.3.4	Testing the WML application	187
10.3.5	Testing the cHTML application	189
10.3.6	Testing the Voice application	190
10.3.7	Other emulators	190
10.4	Best practices.	192
10.4.1	Device management	192
10.4.2	Session management	193
10.4.3	Managing the content for different devices	193
10.5	Development roles and responsibilities.	194
10.6	More information	195
10.6.1	IBM related	195
10.6.2	Not IBM related	195
Chapter 11.	System management	197
11.1	General system management.	198
11.2	IBM HTTP Server	199
11.3	WebSphere Application Server.	200
11.3.1	Administration console	200
11.4	WebSphere Transcoding Publisher	202
11.4.1	Administration console	202
11.4.2	Common LDAP directory	204
11.5	WebSphere Voice Server	204
Chapter 12.	Security	205
12.1	Introduction	206
12.2	Mobile versus conventional communications	207
12.2.1	Authentication	207
12.2.2	Confidentiality and message integrity	212
12.2.3	Authorization	214
12.2.4	Non-repudiation.	215
12.2.5	Secure boundary.	217
12.3	Security technologies for wireless transactions.	217
12.3.1	Security Socket Layer (SSL).	217
12.3.2	Wireless Transport Layer Security (WTLS).	218
12.3.3	Public key cryptography	218
12.3.4	Elliptic Curve Cryptography (ECC)	219
12.3.5	Public Key Infrastructure (PKI)	219
12.4	Apparent problems in wireless security.	221
12.4.1	Broken secure connection in the WAP gateway	222
12.4.2	Other gateways.	223
12.4.3	The WebSphere Transcoding Publisher and encrypted content.	225

12.5 More Information	226
Chapter 13. Performance	227
13.1 Load balancing	228
13.1.1 Network Dispatcher approach	228
13.1.2 DNS approach	229
13.1.3 Reverse proxy approach	230
13.2 Scalability	231
13.3 Availability	232
13.4 Caching	233
13.5 Turning on transcoding	236
13.6 Where to find more information	237
Part 4. Scenarios	239
Chapter 14. Base sample overview	241
14.1 The base sample	242
14.2 The scenarios	242
14.3 The presentation logic	243
14.3.1 The three tier servlet architecture	243
14.4 The business logic	245
14.4.1 Data model	245
14.4.2 Access Beans	246
14.5 Model diagram for the application	247
14.6 Site map for the application	249
14.6.1 Publishing	250
14.7 Walkthrough	252
14.7.1 Login scenario	252
14.7.2 Portfolio	253
14.8 Summary	254
Chapter 15. Application for interactive mobile devices	255
15.1 Direct approach	256
15.1.1 Design issues	256
15.1.2 Test clients for development	263
15.1.3 New and modified code	264
15.1.4 Developing the content JSPs	264
15.2 Content transcoding (HTML source)	268
15.2.1 Preferences	269
15.2.2 Annotators	271
15.2.3 Text clipping	284
15.2.4 Guidelines for content transcoding	290
15.2.5 HTML to cHTML	290
15.3 Universal transcoding (XML source)	293

15.3.1	Converting XML to different markup languages	294
15.3.2	Trade2 example	296
15.3.3	Design decisions	296
15.3.4	Defining the class concept	297
15.3.5	Defining XML structures	298
15.3.6	Creating XML files	299
15.3.7	Developing the XSL files and the user interface	300
15.3.8	Implementing the XML solution	302
15.3.9	Registering the StyleSheets in WTP	304
15.3.10	Testing the application	307
Chapter 16.	Voice application.	309
16.1	WebSphere Voice Server	310
16.2	Direct approach	313
16.2.1	The voice site	313
16.2.2	Application design	317
16.2.3	Application development	325
16.2.4	Trade2 voice application	327
16.3	Content transcoding (HTML source)	328
16.3.1	Full HTML-to-VoiceXML transcoding	328
16.3.2	Annotators	329
16.3.3	Setting up the HTML-to-VoiceXML transcoder	330
16.3.4	Result of the HTML-to-VoiceXML transcoder	333
16.3.5	Trade2 application	334
16.4	Universal transcoding (XML source)	335
16.4.1	Design decisions	335
16.4.2	Designing the class concept	336
16.4.3	Developing XML files	337
16.4.4	Developing the XSL files	338
16.4.5	Register the StyleSheets in the WTP	339
16.4.6	Testing the application	340
16.4.7	Further directions	340
16.5	Hybrid coding	340
16.6	Where to find more information	342
Chapter 17.	Application for both interactive mobile device and voice.	343
17.1	Introduction	344
17.2	Universal transcoding	344
17.3	Content transcoding	345
17.4	Multimodal applications	346
17.4.1	Multimodal applications in WebSphere	348
17.4.2	VIWO	350
17.4.3	Future developments	352

Part 5. Working example	353
Chapter 18. Development environment for the sample application	355
18.1 The development environment	356
18.2 Application database	356
18.3 WebSphere Studio	356
18.3.1 Importing the Studio Archive File	357
18.3.2 Publishing a WebSphere Studio project	357
18.4 VisualAge for Java configuration	359
18.4.1 Adding DB2 libraries to VisualAge for Java	360
18.5 Importing the VisualAge for Java repository file	361
18.5.1 Rebuilding the EJBs	363
18.5.2 Exporting the deployed EJBs	364
18.6 WebSphere Test Environment configuration	365
18.6.1 Setting up a new Web application under WTE	365
18.6.2 Adding MIME Types to WTE	366
18.6.3 Publishing the Studio project into WTE	367
18.6.4 Starting the WebSphere Test Environment	368
18.6.5 Starting the EJB server	370
18.7 WebSphere Voice Server SDK configuration	371
18.8 WebSphere Transcoding Publisher configuration	371
18.8.1 Setting up the Voice transcoder	371
18.8.2 WTP preference profile for voice application	372
18.8.3 Registering the StyleSheets	374
18.9 TradeAppServlet configuration for voice	380
18.10 StyleSheet import	381
Chapter 19. Runtime environment for the sample application	383
19.1 Runtime enviroment for the sample application	384
19.1.1 Runtime environment	384
19.2 Installing and configuring the runtime environment	385
19.2.1 Database node	385
19.2.2 Web application node without Transcoder	386
19.2.3 Standalone Transcoder node	388
19.2.4 Web application node with Transcoder	388
19.2.5 Voice Server node	390
19.3 Deploying the sample application	391
19.3.1 Prerequisites	391
19.3.2 NT	392
19.3.3 AIX	392
19.3.4 Configuring WTP for the Trade2 application	393
19.4 Entry point for the Trade2 Web application	394
19.5 Testing the application	395

19.5.1 Test sequence	395
19.5.2 Direct	396
19.5.3 Content transcoding	396
19.5.4 Universal transcoding	396
19.6 Notes for other platforms	397
Part 6. IBM Web and wireless solutions	399
Chapter 20. Introduction to WebSphere Everyplace Suite	401
20.1 What is it for?	402
20.2 Extending capabilities	402
20.2.1 Editions	402
20.3 Connectivity services	403
20.3.1 Notification	403
20.3.2 Asynchronous communication	403
20.4 Security	403
20.5 Summary	404
Chapter 21. Other products	405
21.1 WebSphere Portal Server	406
21.2 WebSphere Personalization Server	407
21.3 SecureWay Wireless Gateway	409
21.4 WebSphere Translation Server	410
21.5 Where to find more information	411
Part 7. Appendixes	413
Appendix A. Additional material	415
Locating the Web material	415
Using the Web material	415
System requirements for downloading the Web material	416
How to use the Web material	416
Related publications	417
IBM Redbooks	417
Other resources	418
Referenced Web sites	418
How to get IBM Redbooks	419
IBM Redbooks collections	420
Special notices	421
Glossary	423
Index	429

Preface

This redbook provides application designers and developers with a broad overview of mobile e-business application design and development using WebSphere Everyplace Access V1R1.

Part 1, “Introduction” on page 1, introduces the offering itself, and some of the mobile technologies together with the voice technologies used in mobile e-business applications.

Part 2, “Patterns for e-business” on page 63, gives an overview of the Patterns for e-business, then shows the use of Patterns in the mobile e-business environment.

Part 3, “Wireless Internet application: guidelines” on page 101, provides design and development guidelines for mobile e-business applications based on the products bundled in WebSphere Everyplace Access V1R1.

Part 4, “Scenarios” on page 239, provides detailed information about the sample application by discussing different scenarios. The sample application is based on the Trade2 Web application, which has been enabled for mobile access. Scenarios are different approaches to how the application is designed and which technologies are exercised.

Part 5, “Working example” on page 353, provides detailed instructions for setting up the development and run-time environment together with the Trade2 sample application.

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, Raleigh Center.

Peter Kovari is a WebSphere Specialist at the International Technical Support Organization, Raleigh Center. He writes extensively about all areas of WebSphere. His areas of expertise include e-business, e-commerce, and mobile computing. Before joining the ITSO, he worked as an I/T Specialist for IBM in Hungary.

Bernard Van Acker is an IT Specialist candidating for IT Architect at IBM Global Services Belgium. He has over seven years of experience in developing groupware, intranet and interactive Internet applications in numerous projects. He holds a Master's degree of Sciences in Econometrics from the London School of Economics and degrees in Law and Applied Economics obtained in Belgium. He is a Certified Lotus Professional in Application Development for Notes, and a certified Solution Developer for WebSphere Application Server. His current areas of expertise include Java server programming, groupware/workflow applications and PKI. Currently, he is gaining expertise in the different aspects of pervasive computing.

Anna Marino is a Voice Systems Consultant in IBM Hursley Development Labs, UK. She holds a Master's degree in Computer Science and has recently joined IBM after having undertaken research in the field of Evolutionary Algorithms. Her interests include Artificial Intelligence, Cognitive Psychology and Security. At present, she is gaining expertise in the design and implementation of multichannel access applications.

Jim Ryan is a WebSphere Specialist with the WebSphere Innovation Centers in the United States. Jim has been with IBM for over 12 years. Prior to his work in the WebSphere Innovations Centers, he worked for five years in ibm.com. He has 14 years of experience in application design and development, specializing in networking. Currently he is working toward a Master's degree in Computer Science at Rensselaer Polytechnic Institute.

Kim Lun Tang is a Software Engineer in IBM Germany in Cologne. He holds a Bachelor's Degree in Computer Science which he obtained in the UK. After finishing his studies, he joined IBM Germany for three years. His areas of expertise include e-Commerce, payment solution, security and object-oriented software development. At present he is gaining expertise in pervasive computing.

Christoph Weiss is an IT Specialist at IBM Global Service in Heidelberg, Germany. He holds a degree in Economics and Computer Sciences from the Berufsakademie Stuttgart, Germany. He has three years of experiences in the fields of wireless computing, XML and object-oriented software development. At present he is involved in delivering multiple channel access solutions to IBM customers.

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

Kathryn Britton - Senior Technical Staff Member, WebSphere Transcoding Publisher

John Ganci - International Technical Support Organization, Raleigh Center

Jonathan Adams - IT Consultant SWG Technical Strategy, IBM UK

Leo Marland - Senior Consulting IT Architect, IBM Canada

Thanks to the following people from the International Technical Support Organization, Raleigh Center:

Gail Christensen
Rufus Credle
Linda Robinson
Juan Rodriguez
Carla Sadtler
Margaret Ticknor
Jeanne Tucker

Thanks to the following IBM employees:

Iva Anderson - Manager, WebSphere Transcoding Publisher Development
Richard K Brassel - Software Engineer, WebSphere Transcoding Publisher
Ralph Case - IBM WebSphere Connectivity and Edge Solutions
George Clelland - DirectTalk & Message Center technical support
Stan J Cox - Programmer, WebSphere Performance
Chris Cross - Engineer, Client and Server Speech
Brenda Horowitz - Software Engineer, IBM Voice Tools
Steve Ims - WebSphere Connectivity and Edge Solutions
Marshall Lamb - Chief Programmer, WebSphere Transcoding Publisher
Jennifer Lanier - WebSphere Transcoding Publisher Development
Anu Mannar - Market Manager, IBM Software Group
Nick Metianu - Manager, Voice Tools Development
Robbie J Minshall - Test / Development, WebSphere Performance
Brian Pickering - Speech Technologist, Voice System Services
Frank Seliger - Security Architect, IBM Pervasive Computing Division
Henry Welborn - Wireless Gateway Development
Anthony Wrobel - PVC Websphere Everyplace Suite Development

A special thank you to the Voice System Services Team in Hursley, UK for their invaluable help during the whole project.


Special notice

This publication is intended to help developers, I/T architects, I/T specialists, and consultants design, develop, test, and deploy mobile Web applications. The information in this publication is not intended as the specification of any programming interfaces that are provided by WebSphere Everyplace Access V1R1 offering.

IBM trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

e (logo)® 
IBM ®
AIX®
AS/400®
CICS®
DB2®
DirectTalk®
Everyplace™
IMS™
NetView®
OS/2®
PAL®

Redbooks™
Redbooks Logo ™
SecureWay®
SP™
ViaVoice®
VisualAge®
WebSphere®
WorkPad®
Lotus®
Notes®
Domino™

Comments welcome

Your comments are important to us!

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

- ▶ Use the online **Contact us** review redbook form found at:
ibm.com/redbooks
- ▶ Send your comments in an Internet note to:
redbook@us.ibm.com
- ▶ Mail your comments to the address on page ii.



Part 1

Introduction



Introduction to IBM WebSphere Everyplace Access V1R1

Today, developers and architects view mobile applications as a special case or niche. Over the next couple of years, mobile applications will be integrated into nearly all Web sites.

This book describes the WebSphere Everyplace Access offering and how it can be used to design, develop and deploy mobile Web applications.

The offering bundle includes a wide variety of software, which enables a Web application to be accessible using different types of devices, such as mobile phones with Internet capabilities, PDAs, phones using voice, and so on...

This chapter will introduce the WebSphere Everyplace Access offering.

The second chapter will give a short overview of mobile technologies.

The third chapter in this first part of the book will discuss the voice technologies.

1.1 Definitions

Pervasive, mobile, wireless and many other terms constitute a confusing terminology which makes it difficult to have a common understanding of our topic. The problem always starts with classifying sophisticated systems, when it is discovered that the terms mean different things to different people. For the sake of consistency in this book, the terms will be used as shown in Figure 1-1:

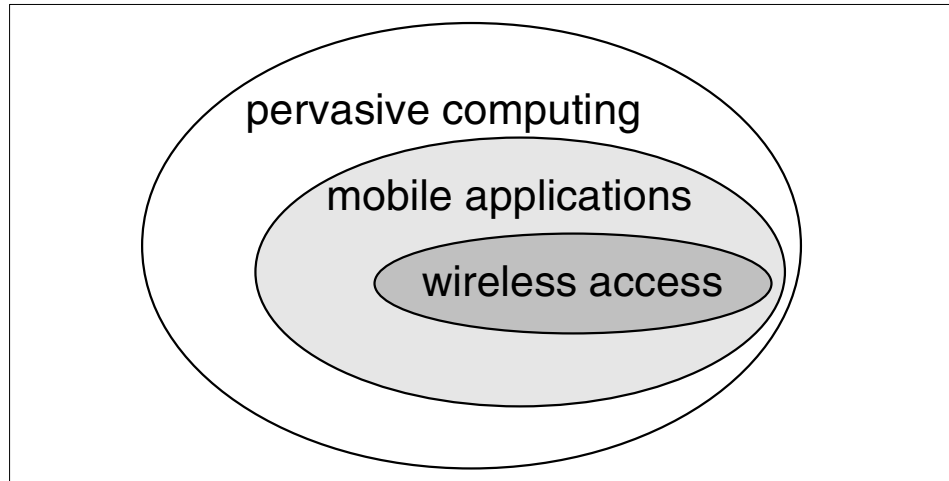


Figure 1-1 Terminology

The term *mobile applications* also needs some explanation. *Mobile* in this context means that the application is not only accessible from a desktop browser but also from different types of devices, using different connections, from different locations.

In this book, those mobile Web applications, which must be adapted to the specific requirements of mobile clients, will be introduced.

1.2 Business drivers

The Internet has built up a great network and people have learned how to take advantage of the huge amount of content available through it. Mobile applications and wireless access are the next step. Nowadays, the problem is not whether the required information is available, but how fast and easily it is accessible.

The mobile solution space is an endless landscape because there are so many possibilities and requirements. Nowadays, there are no regulations and no convergence between the different specifications; the whole field is developing fast.

To meet the maximum number of requirements, an accurate and solution-specific market study is an essential part of the project. It can help to narrow down the possibilities and determine the necessary resources and applicable technologies. The customer also gets a clear view of the possibilities, and can specify his/her requirements accurately.

“Many IT organizations today are overwhelmed by the prospect of trying to tackle wireless development with already strained resources. In addition, the demand for mobile applications is often urgent, as companies come to view wireless technologies as a strategic means of seizing competitive advantage.” - from the document Wireless Enterprise Applications for Mobile Information Management - Development Options and Business Decisions, Palm Inc.

This book discusses the WebSphere Everyplace Access V1R1 offering. We will talk about how to develop a mobile Web application to meet different business requirements using this offering.

WebSphere Everyplace Access delivers value to three types of businesses, as shown in Figure 1-2:

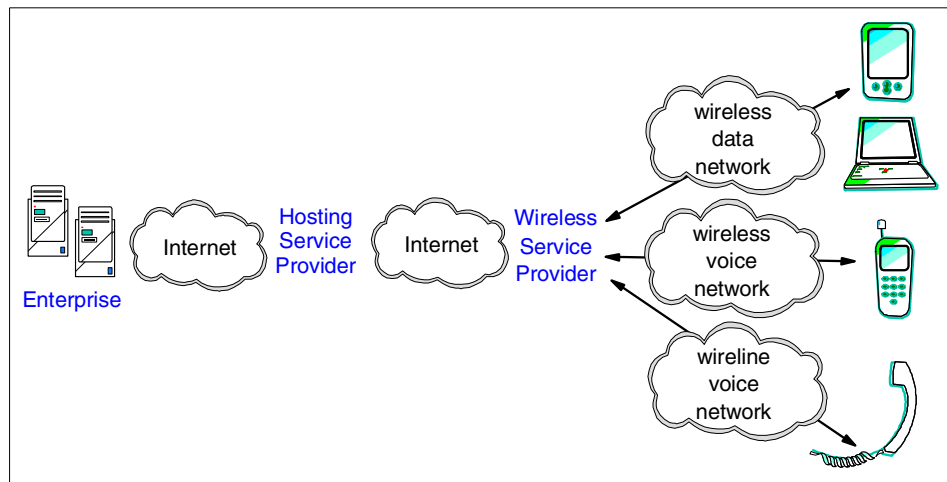


Figure 1-2 Business types using WebSphere Everyplace Access

The three types of businesses are:

1. Enterprise - The original source of information and services.
2. Wireless Service Providers - The gateway for mobile devices to the Internet. Mobile Service Providers (MSP) focus on services to the end-users.

Note: This offering is only one component of a solution for MSPs. For a complete solution, MSPs should consider IBM's WebSphere Everyplace Suite offering (which includes the principal components of the Everyplace Access offering); for more information, see Chapter 20, "Introduction to WebSphere Everyplace Suite" on page 401.

3. Hosting Service Providers - These focus on providing extended and outsourced services for enterprises.

Any of these types of businesses might deploy the WebSphere Everyplace Access offering. Furthermore, some of these deployments involve integration among the different businesses.

1.3 WebSphere Everyplace Access in this book

This book will show how to design and develop mobile e-business applications using WebSphere Everyplace Access. It is important to understand that we only include those solutions in this book which are based on this offering. There are many other mobile solutions in existence which require services that this offering cannot provide.

Note: Throughout this redbook, we will refer to WebSphere Everyplace Access as WEA.

WebSphere Everyplace Access provides great flexibility for mobile application development; it utilizes the existing infrastructure and builds from the existing solution. However, this also means that, since it relies on the underlying infrastructure, it does not provide all the required services (for example, end-to-end security).

For more information about other offerings within WebSphere Everyplace Suite, refer to Chapter 20, "Introduction to WebSphere Everyplace Suite" on page 401.

The following sections list the individual products bundled in the WebSphere Everyplace Access offering.

1.4 Run-time applications

WebSphere Everyplace Access has three different run-time server applications to run the mobile Web applications. To find out more about these run-time applications and run-time patterns, refer to Chapter 6, “Runtime patterns” on page 79.

1.4.1 WebSphere Application Server

WebSphere Application Server is the foundation for Web applications. This Java application server ensures the run-time environment for the server side application and its modules.

WebSphere Application Server Advanced Edition not only provides the run-time environment for basic Web applications, but also provides the EJB run-time for Enterprise Java Beans.

1.4.2 WebSphere Transcoding Publisher

WebSphere Transcoding Publisher is a key component in mobile solutions. It is an intermediary server between the client and the server, and handles the transformation from the source content coming from the server to the solution to the client’s specific requirements.

1.4.3 WebSphere Voice Server

WebSphere Voice Server brings voice enablement to mobile solutions. It runs VoiceXML browsers for the telephone clients, these can connect to the Web application on one end and can serve the phone clients on the other end. WebSphere Voice Server requires additional elements to switch the IP-based communication to the non-IP-based phone network.

WebSphere Voice Server comes with the WebSphere Voice SDK, which is the development toolkit for VoiceXML applications.

1.5 Development applications

WebSphere Everyplace Access also provides development applications to allow developers to implement their solution. The following are included:

1.5.1 WebSphere Studio

WebSphere Studio is an integrated development environment for WebSphere Application Server. The Studio provides a well-designed workbench for the user, and the whole project is managed with all the assets included in the project.

WebSphere Studio has recently introduced several enhancements for mobile applications, such as new markup language support, a new servlet and JavaServer Pages for mobile devices.

1.5.2 VisualAge for JavaEE

VisualAge for Java is the integrated development environment for Java. It provides a workbench to develop, debug, test and run all sorts of Java applications. The WebSphere Test Environment is built into it, which allows the developer to run and debug the Web application without a WebSphere Application Server.

WebSphere Studio and VisualAge for Java are integrated together, so they manage the Java source code and byte code between the two workbenches.



Overview of mobile technologies

This chapter provides readers not familiar with the concept with a foundation for understanding today's mobile technologies.

The chapter is organized into the following sections:

- ▶ Wireless networks
- ▶ Wireless protocols
- ▶ Mobile devices
- ▶ Content and markup languages
- ▶ Service providers

To read more about mobile technologies, refer to the IBM Redbook entitled *Mobile Commerce Solutions Guide using WebSphere Commerce Suite V5.1*, SG24-6171.

2.1 Technology background

The next generation of mobile technologies is ready, and will quickly become widespread. These technologies provide wide bandwidth, a large screen, and plenty of computing power. They have all the capabilities of the powerful, wire-connected desktops regarding network access and applications.

Handheld devices will not have larger screens in the future, but will have stronger computing power and wide bandwidth access.

Problems like session management, lack of security and standards will be solved by that time.

“The wireless world is changing quickly. Not only are new technologies and standards emerging, but the market now has such momentum that it is attracting a constant influx of new ideas, vendors and product entries. Make sure you have a high level confidence that the wireless platform you choose is open to broad vendor participation and able to incorporate advances from all directions.” - from the document *Wireless Enterprise Applications for Mobile Information Management - Development Options and Business Decisions*, Palm Inc.

2.2 Mobile accessibility

The term *mobile* covers a wide range of different devices. The mobile area is just emerging and changing very fast, and new devices are being developed every day. These devices have different types and modes of connections to the networks. The following list gives an idea of the connection types that exist today:

- ▶ Infrared (IR) or cable connection between the device and a cell phone.
- ▶ Synchronization cradle connected to a machine or directly to the network.
- ▶ Dial-up wireline connection to a network.
- ▶ Wireless connection to a network server.
- ▶ Fixed wireless LAN within office buildings, airports, hotels, convention centers, etc.
- ▶ Wireless Personal Area Networks (PAN), connecting devices together within a very short range.
- ▶ Voice - wireless, wired.

2.3 Wireless networks

Wireless networks are used to transmit data between mobile devices or personal computers using wireless adapters without the use of a physical cable or wire. In this section, we will describe these wireless networks.

2.3.1 Mobile communications network history

In cellular radio networks, the area covered by one base station is reduced and other base stations are installed with small overlapping areas. Adjacent cells need to use different frequencies to avoid interference, but the same frequency can be reused in non-adjacent cells, as seen in Figure 2-1.

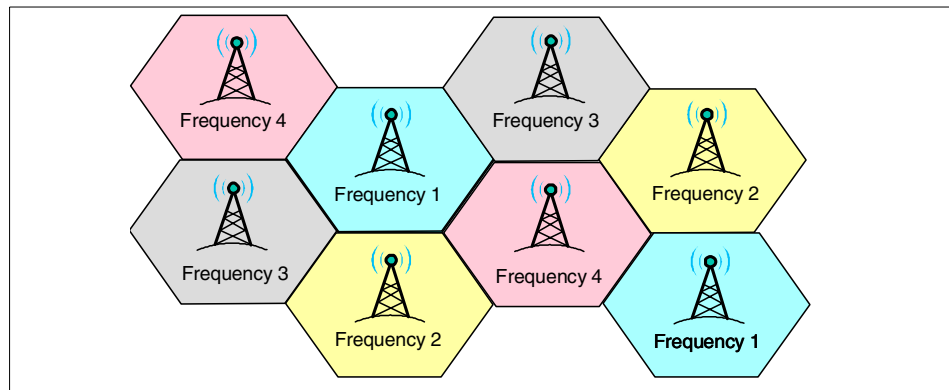


Figure 2-1 Frequency reuse

Because the coverage area is split into many cells, it may happen that a moving user passes from one cell to another. In this case, in order to guarantee the highest communication quality, the mobile device needs to switch to one of the frequencies of the new cell.

Networks are either circuit-switched or packet-switched. With circuit-switched networks, the network resources are assigned to a single connection for the whole duration of the communication. This dedicated channel remains allocated even in those time frames when no data is sent over the channel. Voice traffic is typically sent over circuit-switched connections. Data services are very different in nature from voice services. In typical voice communications, you usually have very short idle times between two consecutive transmissions, which means that the allocated channel is intensively used during the communication. The situation for data traffic is quite different. Consider the case of a user browsing the Internet. When a user sends an HTTP request from a PC browser client, they

then receive a response page for display of the content. In this phase, the user is operating offline and is not using the network. At that time, the resource could be assigned to other users waiting to transmit. This is the typical situation of data services and in such cases a packet-switched technology is highly preferable.

In packet-switched networks, no permanent channel is allocated to a single connection. Data is transmitted in the form of packets and the network resources are allocated on-demand. This also means that any single channel can be shared among different users and that the network resources are better exploited, because they are allocated only when effectively needed.

Wireless network generations

When looking at how wireless networks have evolved, the different phases and characteristics of wireless networks are often referred to as *generations*: 1G for the first generation, 2G for the second generation, and so on. Today, the latest is the 4G, which is under development.

The first generation (1G) cellular systems were analog and implemented with such technologies as AMPS. The network was introduced in 1983 in the United States and was used for voice capability.

Second-generation (2G) systems were introduced in the 1990s. Some of the technologies implemented include TDMA, CDMA, and GSM. 2G systems were used primarily for voice.

Some 2.5G systems have been implemented recently, such as HSCSD and GPRS. HSCSD is a circuit-switched extension of GSM that allows an increase in the bit rate by combining more than one timeslot in the GSM radio interface. GPRS can be considered a packet-based extension of GSM and provides higher data throughput.

Third-generation (3G) systems are expected to be implemented over the next few years and will be called IMT-2000. 3G networks provide higher-speed transmission to support high-quality audio and video, as well as global roaming capability.

This evolution path of wireless network generations is represented in Figure 2-2.

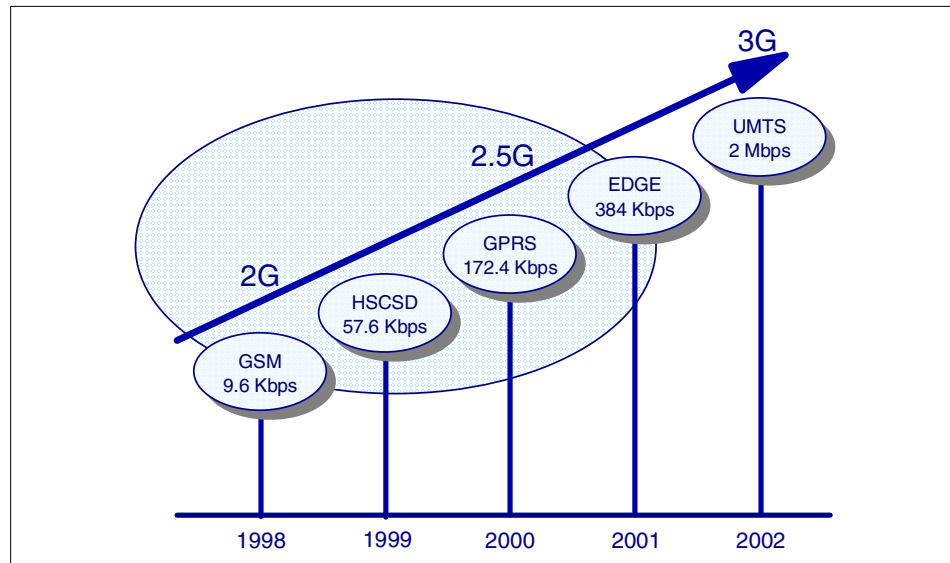


Figure 2-2 Evolution path from GSM to UMTS

2.3.2 GSM

The standard was proposed in 1982 and completed in 1990, and the first networks were deployed in 1991. The main reason behind the introduction of GSM in Europe was to provide a common standard for European cellular communications, which allowed subscribers to roam throughout Europe and access cellular networks in each country with the same equipment.

Today, GSM is the most important and widespread mobile standard worldwide. Many variants of GSM have been created for different frequency ranges. GSM technology in the form of DCS1900 is available in North America, and often referred to as Personal Communications Services (PCS) systems.

In a GSM network, the subscriber is considered an entity separate from the device. This means that the subscriber identity can be transferred from one physical phone to another, without reprogramming the device. This is accomplished by means of a Subscriber Identity Module (SIM), which is a small smartcard inserted into the mobile phone.

Each GSM subscriber is uniquely identified by the International Mobile Subscriber Identity (IMSI), which is stored on the SIM card.

The physical device is identified by the International Mobile Equipment Identifier (IMEI), which is embedded in the device.

One important service of the GSM network is the Short Message Service Point-to-Point (SMS-PP). It allows the GSM phone to send and receive short messages, using the network's control channel to transfer the information. This means that no circuit-switched connection is established and the message can be stored, then forwarded when the phone is able to accept the message. The GSM Short Message Service allows the phone to act as a two-way alphanumeric pager.

2.3.3 GPRS

General Packet Radio Service (GPRS) is the packet-based extension of the GSM network. GPRS is a fundamental step in the migration from GSM to 3G networks. GPRS can support data traffic over packet-based connections with a higher bit rate than GSM (up to 172.4 kbps). GPRS introduces three additional coding schemes (CS) for the data that is transmitted across the radio interface, with respect to the single coding scheme existing in GSM. These coding schemes provide different degrees of error correction and, consequently, different bandwidths.

One important consideration is that most GSM operators are building GPRS on top of the existing GSM infrastructure and are continuing to use GSM for voice traffic. This means that the radio resources must be shared between GSM and GPRS traffic and all timeslots cannot be allocated at one time. One important feature of GPRS and, more generally, of all packet-based networks is that the user is always connected to the network. Value-added services based on information push, such as online e-mail and remote monitoring, provide great advantages from this kind of network.

The GSM technology, even if based on a circuit-switched core network, is packet-based as far as the radio interface goes, because information is transmitted in bursts over the air interface. GPRS can be built on top of the existing GSM infrastructures, thus allowing a partial reuse of the pre-existing equipment.

2.3.4 Mobitex

The Mobitex technology was originally developed in Sweden in 1984. This network now operates in 23 countries. Mobitex is the network used in the U.S. by Palm.Net, as well as by many other wireless service providers.

The data rate of a Mobitex wireless channel is 8 kbps. The network latency is relatively high and varies significantly. Both mobile devices and fixed terminals are treated equally in terms of addressing. Any entity can communicate with every end system in the Mobitex network. It is even possible to address end systems at other network providers if they are interconnected. Mobitex takes care of all the needed routing. So, different Mobitex networks may be interconnected and packets may be transferred between them, in the same way that you would establish international phone calls or communication between two telephone carriers. Mobitex mobile and fixed terminals are both identified and addressed by a Mobitex Access Number (MAN), which has an eight-digit value. MANs are unique worldwide and are assigned by the device manufacturer. Every Mobitex network provider has its own address range to assign MANs to fixed terminals. A Mobitex mobile device even keeps its MAN when roaming between different network providers.

Mobitex has been designed to be a messaging system, where small amounts of data have to be transmitted at irregular intervals. The transmission characteristics are not well suited to routing IP traffic. One problem is the high variance in network latency, which makes it hard for the network to decide whether a packet should be regarded as lost or simply delayed.

2.3.5 CDPD

Cellular Digital Packet Data (CDPD) is a wireless, packet-switched network technology. It was built on top of the AMPS infrastructure by adding the required capabilities for packet management and routing. Roughly speaking, CDPD is the packet-based extension of AMPS, which is circuit-switched, exactly as GPRS is the packet-based extension of GSM.

CDPD inherently uses Internet Protocol (IP) as the protocol for sending and receiving data. IP includes protocols that take care of such essential functions as authentication and encryption, and provides a maximum raw data throughput of 19.2 kbps.

The wireless CDPD Network Controllers, connected to the base stations, route the packet traffic between the CDPD phones and the Internet, and also manage the handover between cells. One of the most useful features of CDPD is the ability to find open voice channels and use them for data.

Since CDPD relies on an AMPS cellular network, it is not available outside of North America and some Latin American countries.

2.3.6 PDC

Personal Digital Cellular (PDC) is a Japanese cell phone standard. The data transmission rate is 9.6 kbps. Almost all the cell phones used in Japan are based on PDC.

2.3.7 IMT-2000

International Mobile Telecommunications-2000 (IMT-2000) is a 3G wireless system. IMT-2000 offers support for a wide range of mobile devices and includes links to terrestrial and/or satellite-based networks; the terminals may be designed for mobile or fixed use.

Key features of IMT-2000 are:

- ▶ Support of different access networks, both terrestrial and satellite.
- ▶ Multi-mode and multi-band terminals.
- ▶ Virtual Home Environment (VHE) - the user should have access as much as possible to the same set of services and to the same look and feel, regardless of the access network, the terminal capabilities, and the current location.
- ▶ Very high data throughput.
- ▶ Worldwide roaming capability.
- ▶ Capability for multimedia applications.

UMTS

The Universal Mobile Telecommunications System (UMTS) is the European implementation of the 3G wireless phone system. UMTS, which is part of IMT-2000, offers global roaming and personalized features. UMTS was designed as an evolutionary system for GSM network operators, and offers impressive data rates of up to 2 Mbps. UMTS uses the W-CDMA technology. GPRS and EDGE are interim steps that will speed up wireless data for GSM.

W-CDMA (DS-CDMA)

The Direct Spread - Code Division Multiple Access (DS-CDMA) specification is supported by Ericsson (Sweden) and Nokia (Finland). This technology will be used mainly in Europe and Japan. The data translation rate is 64 kbps for upstream and 384 kbps for downstream.

cdmaOne

Code Division Multiple Access (CDMA) is a specification of wireless communication. Voices from multiple users are transformed by multiplying different codes and transferred all together as one frequency. The receiver can detect only the sender's voice and decode it. The cdmaOne is one standard of 3G cell phones that uses the CDMA protocol. It is used in North America and Asia. The data transmission rate is 14.4 kbps.

CDMA2000 (MC-CDMA)

The Multi Carrier - Code Division Multiple Access (MC-CDMA) specification is supported by Qualcomm (US) and Lucent Technologies (US) and will be the North American standard. Compared to W-CDMA, it is easier for the carriers of cdmaOne to migrate their facilities or learn management know-how. The maximum data translation rate will be 14.4 kbps while fast moving, 384 kbps while slow moving, and 2 Mbps while at a standstill.

2.3.8 Wireless LANS

A wireless LAN is a local area network that transmits over the air, typically using an unlicensed frequency such as the 2.4 GHz band. A wireless LAN does not require lining up devices for line-of-sight transmission such as IrDA. Wireless access points (base stations) are connected to an Ethernet hub or server and transmit a radio frequency over an area of several hundred to a thousand feet. This frequency can penetrate walls and other non-metal barriers. Roaming users can be handed off from one access point to another like in a cellular phone system. Laptops use wireless modems that plug into an existing Ethernet port or that are self-contained on PC cards, while stand-alone desktops and servers use plug-in cards (ISA, PCI, etc.).

There have been numerous proprietary products on the market for home and office, but now most manufacturers are adopting the standard IEEE 802.11b, which defines a maximum data rate of 11 Mbps. Bluetooth and HomeRF are other home and small-office technologies. Such systems have a more limited range and do not support roaming. Small wireless LANs are sometimes called personal area networks (PANs), since one of their primary uses is to serve an individual connecting a laptop or PDA to a desktop machine.

2.4 Wireless protocols

Wireless protocols are used to connect mobile devices to the Internet. Many of the wireless protocols have defined architectures that optimize the use of the radio resource and also minimize the capabilities required for the device.

2.4.1 HTTP protocol

The most obvious way of accessing Web-based applications from a mobile device is to use the same protocol and content type used when accessing the Internet from PC browsers over a wireline network connection.

This approach, which is very suitable for wired connections, presents many limitations when adopted over wireless networks. In fact, when using TCP/IP based protocols (such as HTTP) over wireless mobile networks, you have to deal with the fact that radio networks usually have much less bandwidth and a higher latency than local or wide area networks. Moreover, wireless connections are less stable in nature than wired connections and also unpredictable in terms of availability. TCP/IP-based protocols work well over wireless connections. However, the performance is slow.

Additionally, most mobile devices have limitations in display capabilities and may not be able to deal with all the features of full HTML. Sending full HTML to mobile devices can be useless in some cases, considering that some of this information will be ignored by the client browser. Due to the limited bandwidth and capability of mobile devices, simplified markup languages have evolved that are optimized for the limited bandwidth available and the limited capabilities of the mobile device.

As the next-generation networks (for example, IMT-2000/UMTS) become available, providing very high data throughputs, the HTTP/HTTPS protocols and HTML markup language become a much more viable solution for mobile devices.

2.4.2 WAP

The Wireless Application Protocol (WAP) is a standard for providing cellular phones, pagers, and other handheld devices with secure access to e-mail and text-based Web pages. Introduced in 1997 by Phone.com, Ericsson, Motorola and Nokia, WAP provides a complete environment for wireless applications that includes a wireless counterpart of TCP/IP and a framework for telephony integration, such as call control and phone book access.

WAP features the Wireless Markup Language (WML), which was derived from Phone.com's HDML and is a streamlined version of HTML for small-screen displays. It also uses WMLScript, a compact JavaScript-like language that runs in limited memory. WAP also supports handheld input methods such as a keypad and voice recognition. Independent of the air interface, WAP runs over all the major wireless networks in place now and in the future. It is also device-independent, requiring only minimum functionality in the unit so that it can be used with a wide variety of phones and handheld devices.

WAP was developed because of the strong limitations of both mobile devices and wireless networks. Most wireless devices are often very limited in terms of display, processing power and available memory. Moreover, wireless networks themselves are characterized by limited bandwidth and high latency. In order for wireless devices to be able to access Internet content in a way similar to wireline PC browser clients, WAP was developed.

2.4.3 M-services

Mobile Services (M-services) are a new initiative from the WAP Forum. After a couple of years developing the Wireless Application Protocol, the Forum has started to work out a new protocol to replace the WAP. WAP technology reached its new version, but the coming communication technologies require new services and support on the application level.

The new protocol is supposed to be based on WAP, building from the best technologies, and leaving the weaknesses behind.

M-services bring new functions and features to the handheld devices on the application level, for example: enhanced application support for the devices, advanced GUI and peripheral support, and modular, pluggable software.

For more information about M-services, refer to Openwave's Web site for documentation at <http://www.openwave.com>.

2.4.4 i-mode

i-mode is a wireless service developed by NTT DoCoMo in Japan. It is designed to provide mobile phone voice service, Internet and e-mail access. The i-mode protocol uses compact HTML (cHTML) as its markup language for the same reasons that WAP use WML.

In order to develop i-mode applications, you will need to enter into a confidentiality agreement with NTT DoCoMo. The agreement will allow you access to the i-mode proprietary APIs to develop secure i-mode applications. NTT DoCoMo's i-mode service is predominately used in Japan.

There are three major carriers for wireless connectivity of mobile phones in Japan:

- ▶ NTT DoCoMo
- ▶ KDDI
- ▶ J-PHONE

Each of the carriers provides Internet connectivity service for their mobile phone users. Each has a gateway center to translate the wireless transmission from the mobile phone to the TCP/IP protocol so that it can be sent over the Internet.

The carrier provides Internet e-mail service with an address such as xxxxx@docomo.ne.jp. There is a one-to-one correlation of e-mail address and phone number. The outbound e-mail is transferred through the gateway to the Internet. The inbound e-mail is routed through the gateway and notification is sent to the mobile phone that an e-mail has been received.

Each carrier in Japan has an Official Site service. The content provider (Web site or commerce site) needs to register with the carrier to become an Official Site. Once the content provider is approved as an Official Site, it gains access to many benefits from the carrier. For example, Official Site content providers are listed on the home page of mobile phones serviced by the carrier. The carrier acts as a portal site when the user of the mobile phone accesses the Internet. This feature is always turned on and is not customizable by the user.

The carrier can use the mobile phone unique identifier (in some cases, that is supplied by the carrier gateway) and the IP address of the carrier gateway to build a secure system.

Voluntary Site access is free. All that is needed to make content available for a Voluntary Site is the appropriate markup language. One disadvantage is that you cannot get any mobile device-related information from the carrier.

2.4.5 Web Clipping

Web Clipping is a proprietary technology developed by Palm. The main elements that constitute the Web Clipping architecture are the Palm device, a Web Clipping proxy server, and the content server. In order to support Web Clipping, a special piece of software called a Clipper must be present on the Palm device. This application is a built-in component of all new Palm devices since the Palm VII. However, some third-party software (for example, OmniSky) can be used to make Web Clipping applications available also on Palm V and IBM WorkPad devices. The Clipper is able to interpret and properly render a proprietary format called Web Clipping Application (WCA) format. This format makes it possible to reduce the amount of data transferred over the air and the required storage on the Palm. The content that is retrieved from the content server is in HTML format. The Web Clipping proxy is in charge of translating the HTML content into this proprietary WCA format. There are many available proxies that can be used for this, all hosted by Palm.Net. Web Clipping uses the HTTP/HTTPS protocols.

There are two main components to consider when writing Web Clipping applications:

- ▶ The Web Clipping application itself, which has to be installed on the device.
- ▶ The server side application, which returns the result pages to the device.

A Web Clipping application is like a small Web site stored locally. The starting (or index) page usually provides a form, that is, a list of links, which are the gateways to the live data provided by the server.

Result pages (clippings) are returned by the server side application as a response to the request from the Web Clipping application. The result pages are written in HTML.

Web Clipping proxy server

A key component of the Web Clipping solution is the Palm Computing Web Clipping proxy server that resides at the 3Com Corporation's data center. The Web Clipping proxy server is responsible for converting the standard Internet protocols and content from a Web page into a form that is tuned for transmission across a wireless network and for display on a small device.

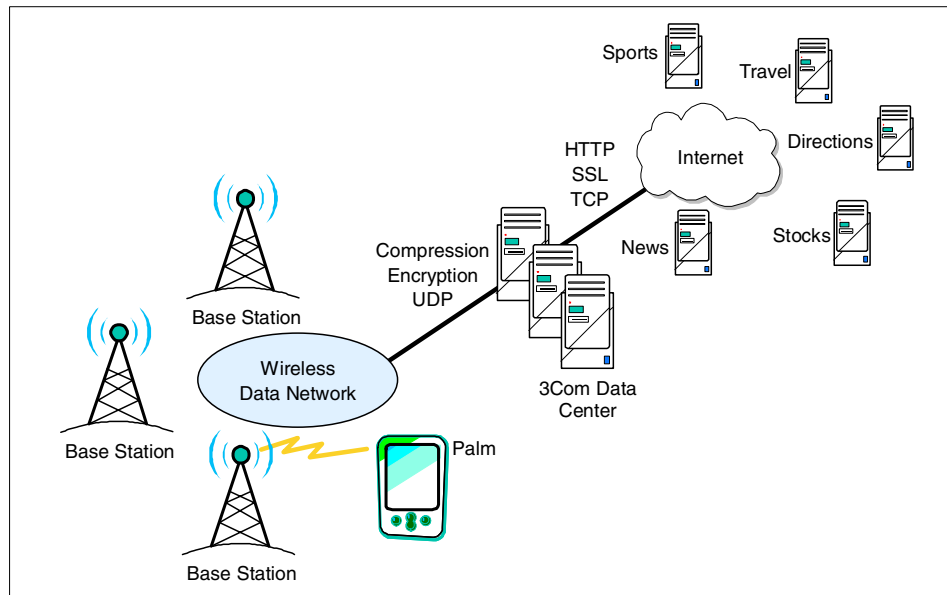


Figure 2-3 Web Clipping proxy server

As shown in Figure 2-3, the Web Clipping proxy server uses the standard Internet protocols (TCP, HTTP, SSL) to access Web servers. To ensure compatibility, the requested pages are in HTML format.

The Web Clipping proxy server implements a reliable layer over the UDP protocol to talk to the Palm device; this protocol reduces latency and conserves battery power relative to using TCP.

In Web Clipping, encryption and authentication between the handheld device and the Web Clipping proxy server is performed by Elliptic Curve Cryptography from Certicom Corporation, which offers extremely high levels of security at small key sizes. On the server side, the high-strength SSL is used for encryption and authentication between the Web Clipping proxy server and Web servers providing HTML content.

2.5 Mobile devices

We have placed the mobile devices into three categories to better describe the specific features that they offer. In practice, there is an ever-growing number of hybrid devices. For example, the Ericsson R380 WAP phone also has the capabilities of a PDA.

2.5.1 Phones for voice interaction

Everyone is familiar with phones with either analog or digital connections. In most cases, voice applications use not only the voice input and output but the numeric keypad as well. The keypad is capable of sending interactive responses from the user, which is faster, more reliable and easier than using voice for the same function.

For voice applications, it does not matter if the device is a wireless (mobile) phone or a normal desk phone. The key is that the application is mobile, not the phone itself, which means that the user can reach the same application on any phone from anywhere. If the phone is a wireless one, it means that the user has more freedom to use the same phone anywhere.

2.5.2 Mobile phones

When we refer to a mobile phone in this redbook, we are talking about mobile or wireless phones that have a microbrowser to access Internet content. Other names for a mobile phone include *cell phone* and *wireless phone*.

A standard mobile phone includes the following capabilities:

- ▶ Voice
- ▶ Messaging (SMS or WAP push)
- ▶ Data (can access Internet content through a microbrowser)

2.5.3 PDAs

A Personal Digital Assistant (PDA) is a handheld computer with a wireless interface that serves as an organizer for personal information. PDAs often have a pen-based stylus to tap selections on menus and to enter printed characters. The unit may also include a small on-screen keyboard that is tapped with the pen. Data is synchronized between the PDA and desktop computer via cable or wireless transmission.

We are interested in PDAs that have wireless transmission capability and include a Web browser. The major operating systems for PDAs are Palm OS, Epoc, and Windows CE. Table 2-1 displays the major wireless PDAs, operating systems, device manufacturers, and protocols used.

Table 2-1 PDA operating systems, manufacturers and protocols

PDA type	PDA operating system	PDA manufacturer	Protocol
Palm	PALM OS	Palm III, V, VII, m505, m510	HTTP, WAP, Web Clipping
		IBM WorkPad PC	HTTP, WAP, Web Clipping
		Handspring Visor	HTTP, WAP, Web Clipping
PocketPC	Microsoft Windows CE	Compaq iPac	HTTP, WAP
		HP Joranda	HTTP, WAP
		Casio E-125	HTTP, WAP
EPOC	Symbian Epoc32	Ericsson R380	WAP
		Nokia 9210 Communicator	WAP
		PSION Series 5mx, Series 7, REVO, REVO PLUS	WAP

Palm

The base for this solution is not the device, but the operating system (OS). The OS called PALM OS is used with a wide range of devices, including the Palm, the IBM WorkPad, and the HandSpring Visor. This device has built-in mobile capabilities, and works as a mobile phone with a data connection.

Like many other operating systems on the market, the Palm OS comes in different editions. The most widely used editions are: V3.x and the latest, V4.0. All the versions are improvements on the previous versions. It is important to note that some of the software relies on a specific version of the OS.

Palm OS has the capability of running several different browsers to access Internet content, including:

- ▶ HTTP browser
- ▶ WAP browser
- ▶ Web Clipping browser

HTTP browser

The HTTP browsers on Palm OS are like any other HTTP browser used to access Internet content, such as Netscape Navigator or Microsoft Internet Explorer.

From a mobile Web application standpoint, there are some key features required by the browser:

- ▶ Secure Socket Layer (SSL)
- ▶ Forms
- ▶ Cookies

There are several HTTP Web browsers available for Palm OS as follows:

- ▶ Intellisync Browse-it

This browser features impressive display capabilities, even for the small PDA screen. The browser supports a secure connection and cookies, and the proxy server is configurable. The browser requires a proxy server from the service provider. The software and the proxy service are available at no charge.

- ▶ Handspring Blazer

This is a great browser for a small device such as a PDA. It has all the required capabilities, such as images (black and white, gray, color), security (requires a proxy at the provider), cookies, and HTTP proxy. It supports HTML, WAP (WML/HDML), and cHTML. The browser and the service are not free anymore; however, the browser is shipped together with the Handspring PDAs.

- ▶ AvantGo AvantGo

This browser, just like the previous one, supports all the necessary functions. The presentation is average. It does not support an HTTP proxy server, but this is not required. AvantGo is a well-known and long-running product that is available at no charge. It requires a proxy server for content providing. The service is free.

- ▶ OmniSky OmniSky

The OmniSky browser is basically the same as the Palm VII built-in Web Clipping viewer.

- ▶ Qualcomm EudoraWeb

This is the only browser at this time that uses a true SSL connection. Unfortunately, the software is not free. The browser comes with all the required features, and supports a proxy. Using the software with the Palm OS Emulator, the Name Server IP address has to be defined (use the **nslookup** command at the command prompt to determine the Name Server IP).

There are many other HTML Web browsers for PALM OS; every day, a new one is coming out. The presentation is constantly improving, most of the required features (cookies, HTTP proxy, etc.) are supported, and strong encryption is used.

WAP browser

Palm uses the same WAP browsers as WAP phones. The main difference is the size of the screen.

Some WAP browsers are:

- ▶ EdgeMatrix WAPman

This is a commercial WAP browser with WTLS security features.

- ▶ 4thpass Kbrowser

This is a browser that does not require WAP gateway, and operates over HTTP.

These browsers provide a big screen with WAP wireless access. Considering the air-time and the small amount of content, it is reasonable to use WAP browsers on a Palm device.

Web Clipping browser

To achieve the goals of long battery life, low service cost and Internet-like performance with low bandwidth, Palm Computing took a different approach to accessing information on the Web. Browsing does not make sense for a handheld device with a small screen and low bandwidth. The Web Clipping solution is like clipping an article out of a newspaper to get the part that is needed, and nothing more.

Proxy servers

Some of the above-mentioned browsers require a server called a proxy, which is a server between the Web browser client and a Web server. The proxy is necessary for security purposes. These small devices are usually not capable of calculating heavy encryption keys, such as SSL. Therefore, the device uses a light (but strong enough) encryption method between the client and the proxy; then the proxy connects to the Web server using SSL.

The problem is that the client cannot do anything with server certificates, because the proxy handles the security. Usually, proxies only accept certificates predefined by the service provider (which runs the proxy). If the destination Web server certificate is not in the proxy database, the client cannot connect to the site securely. This is a common problem during the development of Palm Web applications.

PocketPC

PocketPC devices use the Microsoft Windows CE (WinCE) operating system. The latest version (3G) is called PocketPC, which is similar to the well-known Windows operating system, but optimized and developed especially for PocketPC mobile devices.

The Pocket PC uses a customized version of the Windows CE 3.0 operating system, built by Microsoft and used specifically in Personal Digital Assistants (PDAs), such as the Compaq iPaq and the Hewlett-Packard Jornada. While this customized version is only used in PDA-type devices, Windows CE 3.0 can be used in a wide variety of devices including industrial automation devices, Internet access devices, Web terminals, kiosks, consumer electronics, or retail and point-of-sale devices.

HTTP browser

WinCE/PocketPC is capable of running more advanced Web browsers, such as the WinCE/PocketPC version of Internet Explorer called PocketIE. The software is freely available for download from the Internet.

The browser supports the required features to access any Web site with support such as SSL, cookies, and proxy server configuration ability.

If the recommended design method is using simplified HTML, cut down the content; avoid using big images, nested tables, JavaScript, and Java.

WAP browser

Since WinCE/PocketPC devices are powerful enough to run sophisticated Web browsers, the availability of WAP browsers is very limited. Eventually, an application may require a WAP browser on a WinCE/PocketPC device, but there are better applications.

Some examples of WAP browsers for WinCE/PocketPC are:

- ▶ Ezos's EzWAP V1.0 for all Windows platforms
- ▶ Ezos's EzWAP V2.0 for PocketPC only

EPOC

Symbian's EPOC is an emerging operating system on the wireless PDA market. Some examples of manufacturer devices that use EPOC are:

- ▶ PSION Series 5mx, Series 7, REVO, REVO Plus
- ▶ Ericsson R380
- ▶ Nokia 9210 Communicator

You may notice that the devices are very different from each other. The operating system for these devices is EPOC, but the applications are different.

HTML browser

The most powerful browser for EPOC is the latest version of the freely available browser called OPERA. This browser was originally available for PCs.

WAP browser

Since the Ericsson R380 and the Nokia 9210 are mobile phones or communicators, they have built-in WAP browsers.

2.5.4 Wireless laptops

This category includes laptops, notebooks, or portable PC browser clients that have a wireless interface to the network for Internet access. These clients use standard TCP/IP protocols and a standard browser, such as Netscape Navigator or Microsoft Internet Explorer. The wireless connection is usually much slower than wireline-based network clients.

2.5.5 Mobile device pros and cons

The following is a list of pros and cons for mobile devices. Keep in mind that this list is not exhaustive.

Pros of mobile devices

- ▶ Portability

The user can receive information anytime and anywhere by e-mail or a direct phone call. This enables people to receive information instantly.

- ▶ Easy operation

Mobile devices can be switched on instantly. The user can operate a cell phone with a single hand, or even with just one finger or thumb. Also, a power cable is not needed.

- ▶ Low cost

Mobile devices consume very little power, and are relatively cheap compared to a PC browser client.

Cons of mobile devices

- ▶ Quality of line connection

The connection speed to the Internet is relatively low. The connection can sometimes be terminated when a user is simply going outside because of interference, being out of range, or out of frequency.

- ▶ Security

The biggest problem in this area is the lack of security standards; each manufacturer and service provider has its own proprietary solution for security.

Unfortunately, the devices cannot use strong encryption, because they have weak computing capabilities, which makes it more difficult to provide a standard for security.

Typically, the password is not visible when you enter it on a PC (* is shown). On a mobile phone, the four or six digits of numeric characters are often used and the codes are visible, resulting in lowered security.

- ▶ Poor user interface

It is difficult to enter characters from a standard keypad on a mobile device.

The key layout is different from device to device and there is no pointing device supported as in Windows.

- ▶ Small display and cache

- ▶ No cookie support (in certain devices)

It is not easy to maintain sessions between a mobile device and Web servers without a cookie. Some alternate techniques should be used.

Session management, just like security, is a problem. An alternative to cookies is using unique ID for the devices, but not all devices or gateways support retrieving a unique identifier.

- ▶ No client side scripting support

- ▶ URL length limitation

On some devices, the maximum length of a URL is 400 bytes.

2.6 Emulators and mobile clients

During development, emulators and real mobile clients (devices) are utilized. It is important to understand the difference between these terms. The following paragraphs give a high-level overview of the following:

- ▶ Real devices
- ▶ Emulators
- ▶ Simulators

Real devices are the best for development and testing. The problem with real devices is that they are expensive and usually require additional infrastructure, which makes them even more expensive. Furthermore, real devices can be slower than emulators, because of the real environment.

Emulators are the software equivalent of the original device. The wireless emulators, introduced in this book, are running on a desktop client. The emulators provide the same user experience on the screen, but they are emulating the network connection to the Internet via the operating system's network connection. Usually, emulators are running the same code as the real devices; the ROM images from the devices are interpreted by the emulator and the same applications are running real-time on the emulator machine. Emulators are easy to use and inexpensive. Unfortunately, they are not perfect copies of the original devices. However, they are perfect for development and for unit testing.

Simulators are special devices which copy the behavior of the original device. The point is to simulate the runtime environment of the device. In most cases, simulators also have monitoring, debugging and tracing facilities built in. Simulators are expensive devices, and are best for hardware development.

2.7 Content and markup languages

Markup languages are a set of labels that are embedded within text to distinguish individual elements or groups of elements for display or identification purposes. The labels are typically known as *tags*. Markup languages identify elements within a continuous stream of text.

Our intention is not to document how to use the following markup languages, but to provide insight on how they are used within the context of mobile application:

- ▶ HTML
- ▶ HDML
- ▶ WML
- ▶ cHTML
- ▶ XHTML
- ▶ VoiceXML
- ▶ XML

To find out more about the different markup languages, see Chapter 10, “Application development” on page 153.

2.8 Wireless Service Providers

A wireless service provider is an organization that provides wireless services to its customers, including cellular services, satellite services and Internet Service Providers (ISPs). In this redbook, we are interested in wireless service providers that provide Internet access. When developing mobile applications, it is very important to understand the wireless technologies supported by the wireless service provider in the targeted mobile user’s area. The wireless service provider dictates the wireless network type, wireless protocol and thus the markup language used for application development, as well as the supported mobile device type. In addition, it is important to understand the specifications of the wireless service provider protocol gateway.

The protocol gateway of the wireless service provider, such as a WAP gateway, can vary in specification and support. For example, one service provider may support session control of mobile devices by its WAP gateway, and another provider may not.

Nowadays, service providers are expanding their business and providing different services for their customers in different fields, such as e-commerce on mobile devices (m-commerce), one-way and instant messaging, entertainment, news, financial services, supplementary services for phones, and so on.

2.9 Where to find more information

To find out more about these topics, please refer to *Mobile Commerce Solutions Guide using WebSphere Commerce Suite V5.1*, SG24-6171.



Overview of speech technology

This chapter introduces the underlying mechanisms behind speech technology. Voice applications fall into two categories: those using speech recognition and those synthesizing speech from text. After a brief description of each type of voice application, we will discuss potential benefits of voice-enabled products in different business areas.

3.1 Speech-enabled versus text-based applications

As competition in the industry began to increase over a decade ago, it was no longer enough to allow users access to information via an automated system 24 hours a day and 365 days a year. Although services would continue to grow in sophistication, as shown when speech recognition was included, it was also crucial to make sure that the user's specific needs were addressed. This meant concentrating on how the user felt about the service. This move has been termed Customer Relationship Management (CRM) or, as access to call centers, automated services and information services has proliferated on the Internet, *e-CRM*.

However, businesses are exposed to customers without geographical boundaries and customers may not have constant access to devices that are physically connected to a network. Wireless devices have invaded the market and have already gained popularity among the general public. Modern customers require, then, an increased degree of freedom when contacting call centers, possibly via wireless communications. It is believed that by the end of the year at least 40% of Web access will be done from wireless devices. For this reason, companies who want to stay competitive in the market need to adapt to the new business requirements. It is not expected that traditional methods of accessing information will be replaced, but rather that new ones will be added to increase flexibility. Speech is the obvious choice for extending access modalities, since it is the most natural form of communication. Furthermore, voice frees customers from being dependent on applications where selections could only be made using Dual Tone Multi Frequency (DTMF) tones (that is, the telephone keypad) or mouse clicks. In other words, speech enables users to interact naturally with applications, freeing them from the limitations of DTMF applications. With appropriate use of technologies, it can even allow users to control the automated system rather than the system controlling the user.

We next describe the advantages and potential limitations of adopting voice-enabled solutions. At present, voice applications do not have a specific standard for their implementation. However, according to the ISO9241 (originally developed for visual media applications), interfaces should meet the following criteria:

1. Effectiveness
2. Efficiency
3. User satisfaction

This means that voice applications need to provide sufficient information to the user to allow the completion of a given task in a productive way.

3.1.1 Benefits of voice applications

The introduction of voice as an additional communication form to deploy applications undoubtedly has numerous advantages, one being the ability to access information without a computer. This feature has been a barrier for communications in the past years, since people did not always have an active network connection, for example when travelling. The use of telephony interfaces has significantly improved the availability of information and voice-enabled applications have made possible continuous access to remote data with the added benefit of anonymity (though not necessarily in all cases).

Another advantage of voice applications is the speed of information retrieval. Although voice recognition may be quite complex and not totally reliable (we will see how voice recognition depends on the language and the speaker), it facilitates the location of specific information by means of keywords or sentences.

Voice applications typically use short, meaningful sentences to present the operator with the available options. This has the advantage of focusing the attention of the caller to the important parts of the application, whereas a conventional Web site is normally filled with additional advertisements, images and other pieces of information which may confuse and distract the user, rather than guide him/her.

Additionally, voice-enabled products have introduced a new dimension to the human-computer interface, as computers are now able to emulate a distinctively human feature: speech.

3.1.2 Limitations of voice applications

Voice applications have a few significant limitations at present. Unlike their traditional text-based counterparts, voice applications are only allowed to present information sequentially. This means that the user must carefully listen to and understand each element that is spoken. For this reason, it is not trivial to decide how much information actually needs to be presented to users. Too little information can create ambiguity, misunderstandings and the feeling of a rather unnatural approach. Conversely, too many details may lead to confusion and the user may then waste time trying to correct his/her mistakes. Hence, a good voice application represents a balanced compromise that avoids too much or too little information.

Another strong limitation of voice-enabled applications is the lack of *persistence*. This means that these applications are incapable of retaining information over time. In fact, while it is possible to keep visual details on a screen indefinitely, speech is, by definition, a transient serial process. Persistence allows the user to

maintain a link with the context of the application and to review part or all the information presented by scrolling pages on a screen. The lack of persistence is usually overcome by re-prompting for information when required (or after a certain time, if no other action is requested). However, this solution is not ideal, since the user is overloaded with details he/she may not need (as opposed to using visual information, where the user can choose what he/she wishes to see again). For further details, see also “Mental load” on page 117.

Voice-enabled applications have added an extra level of communication between humans and artificial systems. However, the capacity to recognize words and to speak them out using an artificial system is not as flexible as we could imagine. Features like gender, loudness or dialects significantly affect the understanding of spoken words and sentences, while speech production lacks the capacity to associate a meaning to text, resulting in a very limited adaptability of intonation.

3.2 Speech recognition

The capability of computers to recognize human voice is called Automatic Speech Recognition (ASR). This process converts sound waveforms into text. Hence, another way of referring to it is speech-to-text. The development of speech recognition systems is not an easy task, since many problems need to be addressed:

- ▶ Human voices vary significantly from speaker to speaker because of accent, intonation and other characteristics such as the gender of the speaker.
- ▶ A voice is generally mixed with noise during a conversation.
- ▶ People usually have a multitude of ways of referring to the same concept. For example, if a user asks somebody to pick up a book from a table, any of the following sentences may be used for the request.
 - *Can you bring me that book, please?*
 - *Could you please bring me that book?*
 - *May I ask you to bring me that book, please?*
 - *Would you mind bringing me that book?*

The listener, then, needs not only to understand the words in the sentence but also to interpret them and perform the requested action. Unfortunately, being able to reproduce such a degree of flexibility into an artificial system is no easy task. Humans are trained to recognize speech with a high degree of variability both in voice and scope (that is, the topic of the conversation, like sports, travel or arts) and, although theoretically feasible, it is not possible to reproduce this skill accurately in an artificial system because of resource (like memory and storage) and performance constraints. In fact, speech recognition needs to be a real-time

process to be of practical use for automated services. To help reduce response times, speech recognition usually takes place within a well defined context, such as making travel arrangements. This restriction bounds the number of words that a speaker may use (you would not expect the speaker to describe his car or talk about his job in this context, for instance) and simplifies the recognition process.

Depending on the features associated with the identification, voice recognition systems can be classified according to their enrollment and speaking mode. Enrollment accounts for the source of the speech, that is, the same speaker or any potential one. The speaking mode refers to the temporal continuity of speech. Both categories have two options, which are briefly described here:

► **Speaker-dependent system**

This system is tuned to recognize the voice of a specific speaker. In this case, it is possible to take advantage of a detailed design, where many variables can be determined with high precision. Consequently, the identification is very accurate, but the solution is fairly rigid in the sense that another speaker would probably be less well recognized.

► **Speaker-independent system**

With this system, speech recognition occurs for any speaker of a given language. This is arguably the most flexible solution, but also the most difficult to implement. Although the underlying models are similar for speaker-dependent and the speaker-independent recognition, for the latter significantly more data has to be used for training to ensure robust and usable statistics. By comparison with speaker-dependent recognition, results may turn out to be less accurate. Furthermore, if extensive training and testing are required, this option can become a little more expensive than the speaker-dependent system.

There is also a third possibility, which is to develop a speaker-adaptive system where the recognition is automatically adapted for a new speaker. However, this approach has not been implemented as often as the previous two.

Speech applications can also be categorized according to the modality with which the recognition takes place:

► **Discrete speech recognition**

The identification is based on words, which are clearly identified by a starting and ending point. For this reason, a typical speaker is usually required to provide additional pauses to separate them. Although the identification accuracy is improved, the process is not user-friendly for the speaker and errors can be introduced because of the subjective interpretation of the pause length.

► **Continuous speech recognition**

People normally concatenate words when speaking and a recognition system should account for that. However, the continuity of speech introduces further complexity in the recognition model because it is more difficult to identify the beginning and the end of words. The sound articulation changes when two or more words are pronounced without interruption and speed tends to increase. As a result, the recognition task becomes harder.

Speech recognition is now used in a variety of applications as described in Section 3.5.1, “Speech reco applications” on page 56. However, there are only two ways of deploying solutions embedding speech recognition: *dictation* and *control and command*:

► **Dictation**

A dictation application uses recognition to interpret speech and translates it into a text format. One of the limitations typical of these applications is the number of words that the ASR engine can actually identify. Generally speaking, dictation has to tackle the problem of unseen words in such a way as to minimize errors. However, when the application is designed to respond to a specific need, for example the transcription of mathematical formulas, the vocabulary is quite restricted and the identification can be very accurate.

► **Control and command**

Applications in this category translate voice into actions. Hence, a system incorporates some logic that maps specific words to commands. For example, if such an application is used to control a computer and the operator says “*Close*”, he/she may very likely wish to “kill” the active window. Clearly, people have several ways of expressing a given concept and in the above example the operator could have also used expressions like “*Shut down the window*” or “*Exit*”. Moreover, the same control may need to be associated with different actions depending on the context in which it was issued. So, for instance, the word “*Exit*” can also refer to the ending of an application rather than the closing of the active window. For this reason, the development of the logic used to control other applications can be very complicated. One possible way to solve this problem is to set up rules to determine the receiving object of an action. For example, we could imagine a scenario where if a command is issued and there are applications running in a window, the applications are the first recipients of the controls (each with a given priority, possibly). When no applications are running, the action is received by the window itself. So, if there is a word processing session and the operator issues an “*Exit*” control, the active document is closed first.

3.2.1 System architecture

Speech is processed by a speech recognition engine. It is basically software that implements the algorithm to perform the identification. It receives input speech in waveforms and translates it into text. Figure 3-1 provides a schematic representation of speech recognition and identifies all relevant components involved in the process.

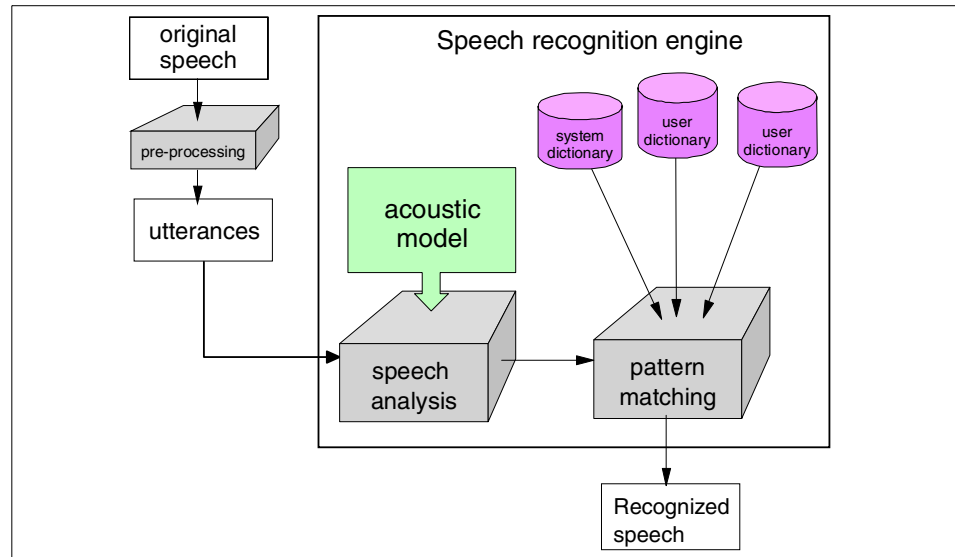


Figure 3-1 Schematic representation of the speech recognition process

When speech is input to the engine, it comes as a continuous sequence of sounds that includes meaningless information, like pauses for example. Hence, some form of preprocessing is needed before the actual recognition is performed. In technical terms, the identification takes place with utterances.

Utterances

An *utterance* is any sound occurring between two consecutive pauses (or silences). It may be a letter, a single word or a whole sentence. Both the beginning and the end of an utterance are clearly identified by pauses of a given length. This value is, in fact, a parameter that can vary depending on the application; it is tuned during training. A speech recognition engine also identifies pause time-outs, that is, lengthy pauses with no occurrence of speech. This does not necessarily mean that the operator is not speaking at all (there may still be background noise), but simply that the intensity of any incoming sound is below a certain threshold and that the recognition process will therefore fail. This feature can be quite useful when building voice-enabled applications, since it is possible to use time-outs to replay information to the user.

Dictionaries and grammars

Utterances that need to be recognized by a speech engine are stored in a dictionary. This represents the knowledge domain of the application in the sense that it specifies the only words that can be recognized. A dictionary can include multiple instances of the same word or sentence if the speaker is believed to use different ways to refer to it (for example, versions of “yes” are typically “yeah” or “yea”). However, for control and command applications, the ability to identify words is not sufficient per se. The application needs to recognize the request for an action within a given utterance and, as mentioned earlier, users can refer to the same concept in a variety of ways. When too many alternatives exist, then grammars can also be used as a mean to restrict the number of allowed word combinations. Artificial grammars represent the syntactic rules that a language must follow. It contains a description of valid concatenations of phonemes and words. Although the use of grammars and dictionaries may seem a restriction of flexibility for application design, in fact it is not. A voice-enabled application may use multiple grammars and dictionaries, each of which provides the speech engine the context in which the recognition process takes place. Speech engines do not support concurrent use of grammars, but they allow the possibility to switch between them at any time during the application. Moreover, dictionaries can be of two types: *system* and *user-defined*. The former contains general terms for the application and cannot be modified. The latter include customized expressions that need to be recognized in a given context. Applications using speech recognition can have multiple user-defined dictionaries. It is worth mentioning that in VXML applications, the concept of a dictionary is not separated from that of grammar. *Grammar* refers to both concepts.

Speech analysis

The incoming audio signal also contains background noise. This makes the identification process harder, as the actual speech needs to be separated from the background noise. For this purpose, it is necessary to provide the speech engine with a suitable acoustic model, that is, a model that describes the features of the environment where the speech occurs and accounts for the noise. Speech is normally processed before it is actually suitable for recognition. In fact, the identification process matches features of the incoming sound waves with some patterns stored during training. However, it is necessary to extract those features from the waveforms. The first step of this process is to digitalize waveforms, as shown in Figure 3-2 on page 41. The original signal is transformed into a smoother and more regular one using a filter, like FIR or IIR for example. It is then sampled to extract pulses. The number of samples taken depends on the source of the speech. A telephone conversation is normally sampled at 8 kHz (8000 times/second), but if the source is a digital audio system, the signal is sampled at 48 kHz (the average is 44 kHz).

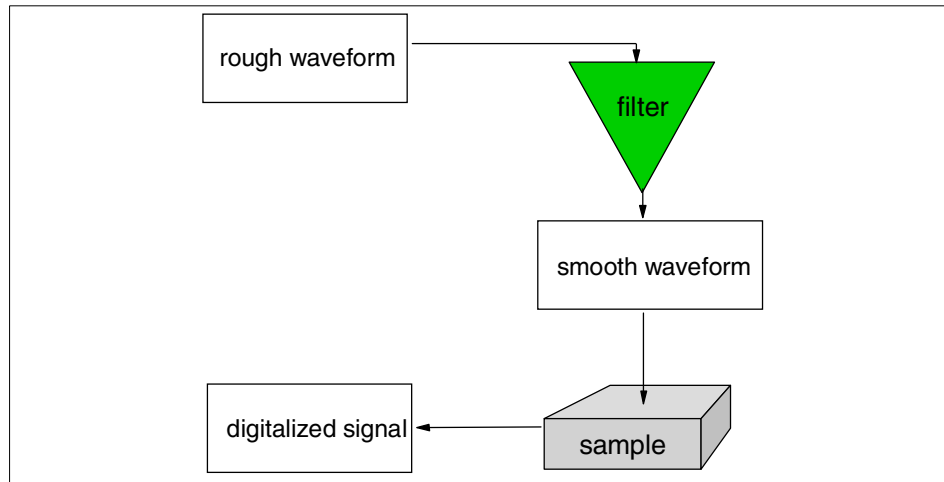


Figure 3-2 Digitalization process for a speech waveform

The digitalized waveform is subsequently processed to extract *spectral features*. These are simply fundamental characteristics of speech that are used for recognition. Although it should be possible to perform the identification with the digitalized waveforms, the use of spectral features reduces speech variability by smoothing the effects of periodicity, pitch or other source information (such as voiced or fricated segments of speech). However, it is worth mentioning that the elimination of source information affects the recognition of languages, such as Chinese for example, where tones represent a significant detail. Spectral features aim to capture the dynamics of the vocal tract movements. Typically, the information is computed over short-term windows (20-30 ms) but the details refer to shorter intervals (10-20 ms). In order to increase the robustness of the recognition against the distortion introduced by the device noise, spectral information is often processed to remove such side effects. Templates are used to estimate probabilities of the spectral features, which are needed to establish likelihood within the incoming speech.

Pattern-matching algorithm

The features extracted from the input speech are matched with an acoustic model that has been previously stored in the system, and are subsequently decoded into text using the acoustic model information. The most prevalent technique for this purpose is based on Hidden Markov Models (HMMs), a simple and fairly robust algorithm that is easy to implement and understand. The idea behind this technique is that speech is modelled by identifying regions in the space of acoustic signals. Sequences of acoustic features are treated as chains of processes, which can be described with a hidden Markov chain. When a sequence of features is presented, the model generates a match based on the maximum probability of observing a sequence of features when a given word is

pronounced. The probabilities involved in the computation are derived from the acoustic and the language models. The complexity of the search depends on the size of the vocabulary as well as the type of recognition required (that is, used for dictation or to trigger commands). Typically, for dictation and applications with a small dictionary, the search can be performed in a single step. For larger vocabularies and control applications, a multiple steps search is needed. The algorithm first computes a short list of potential candidates, then compares the probabilities with the selected hypothesis.

Other approaches have recently been considered. Some use artificial neural networks (NNs) as a pattern-matching technique. In this case, a neural network architecture is trained with examples of speech signals to tune its parameters. Later, the trained neural network is used to process an incoming speech signal and match it with the closest pattern (one of those learned during the training). Hence, in this case it is vital to provide an appropriate set of examples during training. Other approaches try to model speech using linear dynamics to account for the continuity of speech.

Once the acoustic model has been identified, the speech data can be translated into text using a decoding algorithm. Viterbi and Baum-Welch search algorithms are widely used for this purpose. Before the actual search takes place, the algorithm determines a set of valid strings of phonetic categories. This is simply a collection of information that specifies sequences of phonemes with grammar details (such as in what order words can occur). The search determines, based on probabilities, the most likely category sequence that matches the input data. The path followed by the algorithm to identify the category determines the word that was originally spoken.

3.2.2 Natural Language Understanding and Dialog Management

Along with speech recognition, two other approaches need to be considered: Natural Language Understanding and Dialog Management. Although each of them has specific features, sometimes the three terms are used interchangeably.

Natural Language Understanding (NLU) is a powerful language processing technology which significantly increases the flexibility and natural feel for speech-enabled voice applications. It takes the results of the (continuous) recognition engine and begins to interpret what the user requires from the application, extracting and interpreting the relevant pieces of information. For example, in a flight-booking application, the user may say *"I wanna fly from London to Boston next Wednesday in the afternoon."* The recognizer returns the text string; then, without NLU, the application must now try to parse the text to find out the departure airport, destination, date and time. With NLU, however, these pieces of information are automatically retrieved. London is identified as the departure airport and Boston as the destination. For the date and time, the

NLU further converts “*next Wednesday*” to an actual date (based on the platform date settings) and interprets “*in the afternoon*” to mean: anytime between 12 p.m. and 12 a.m. The application script can then continue without the need for difficult programming.

Along with NLU, Dialog Management (DM) takes care of the application flow. It is based on a description of the task to be achieved. For instance, in a flight-booking application, the caller will need to say where they are travelling to, where they are leaving from, when they intend to travel, and so forth. Each of these pieces of information is equivalent to the boxes to be filled in on an application form or indeed in an Internet-based application run through a Web browser. Some of the boxes will be optional; and some will have default information provided, which the user may wish to accept or overwrite. The DM takes the output from the NLU component and fills out as many boxes as it can. Where the caller or user has not provided the information, the DM then decides how the voice application needs to ask for the information, that is, what to do next. Coupled with continuous speech recognition, NLU and good quality TTS, the DM can produce dynamic and natural conversations each and every time the application is called; these can change automatically depending on how different users interact with the application, or how the same user might interact on different occasions.

3.2.3 Application styles

Speech recognition applications are usually in the form of interactions with the user, that is, in the form of dialogs. There are three possible types of dialogs:

- ▶ Directed
- ▶ Mixed initiative
- ▶ Natural language

In the first case, the user is not allowed to interrupt any system prompt before he/she is actually asked to provide some information. This is obviously the most rigid type of interaction, as the user has no real choice but to follow the prompts and give input when requested. We can say that the system is controlling the application. A typical directed dialog looks like this:

System: *How can I help you?*

User: *I wish to book a flight.*

System: *What is your destination?*

User: *Atlanta, Georgia.*

System: *Where do you wish to depart from?*

User: *Chicago O'Hare Airport, please.*

System: *What time do you wish to leave?*

Mixed initiative dialogs occur when the user can interrupt system prompts and supply the required information as well as take the initiative to supply extra data (useful information) that the application will request at a later stage. This way, the time the user needs to wait for an action to be performed (which could be to retrieve data, perform a transaction or transfer to an agent) is reduced. The application structure is still followed in a fairly strict manner, but the user may skip steps by providing extra data. The following example illustrates this type of dialog:

System: *How can I help you?*

User: *I wish to book a flight to Atlanta, Georgia.*

System: *Where do you wish to depart from?*

User: *Chicago O'Hare Airport, please.*

System: *What time do you wish to leave?*

The most flexible type of dialog is known as natural language because the user has the capability to interact with the application in a very natural way. He/she can provide extra information at any time during the application, and use context-dependent data that the application needs to resolve (providing the words used are included in the grammar). Of the three types of dialogs, this one offers the user the best possibility of driving the application, within certain boundaries, of course. Following is an example:

System: *How can I help you?*

User: *I wish to reserve a business class flight from Chicago to Atlanta.*

System: *What time do you wish to leave?*

3.2.4 Speech recognition errors

Speech engines typically compute confidence indexes for the utterances received in input and, based on this value, the system decides whether or not to accept the computed translation. When the output of a recognition process is not accepted, we say there is *rejection*. This does not mean that an error has occurred, but simply that the available response has a low level of confidence and might not be the most appropriate match. In fact, rejection was originally performed by human operators and was later included in the engine as part of the recognition process.

Possible types of rejections are:

- ▶ False rejection
This occurs when a correctly spoken word has a low level of confidence and is considered to be wrongly recognized. This can often happen because of bad pronunciation or other spurious errors as mentioned later in this section.
- ▶ Correct rejection
This prevents illegal or unclear words to be accepted by the engine as valid. Words can be invalid because they do not appear in any dictionary used by the speech engine
- ▶ Inter-word rejection
The matching algorithm may return several candidates for the text with the same confidence level. Unless the engine has some logic to select one of them, it is not possible to return a value and the word is rejected even though it may be valid.
- ▶ Out-of-dictionary/out-of-grammar rejection
This type of rejection occurs when a given utterance is neither identified in the dictionary nor in the grammar (if it is a sequence of words). Clearly, this error overlaps with other forms of rejection, depending on the original cause of the problem; for example, a word can be classified as out of dictionary because of added background noise.

Speech recognition is not always perfect and errors can occur at any time. Human speech perception is also affected by errors because of misunderstood words. However, from the standpoint of ASR, errors can be classified as follows:

- ▶ Insertion
When the output of a speech engine produces text that was not originally spoken, we say that an insertion has occurred. This can happen either because an illegal word is identified as correct (this condition is usually called *false acceptance*), because added noise modifies the features of the incoming speech or because the engine has performed an incorrect word segmentation and the recognition is performed on the wrong utterance.
- ▶ Deletion
This phenomenon occurs when the text resulting from a recognition process lacks part of the information that was originally spoken. Also in this case, the error can occur because of bad word segmentation or because a valid word is recognized with a low level of confidence (false rejection). In the latter case, a very likely cause of the error is pronunciation, especially when non-native speakers are involved.

- Substitution

It may also happen that the speech contains only valid words but the engine matches them with the transcription of different ones. In this case, some utterances of the original speech have been misinterpreted.

There are also some other minor causes of errors, such as for example when a speaker starts talking over a prompt and the underlying application does not allow barge-in or the prompt ends with a noticeable delay (usually more than 300 milliseconds). In this case, part of the speech is lost, resulting in an incomplete recognition; the user is very likely to repeat again what was spoken, resulting in a confusing situation for the recognition engine. This phenomenon is also called *stuttering effect*. Errors can also be triggered because of the absence of incoming speech. An operator may not be aware of the words allowed at a particular stage of an application and, by the time a decision is made, a time-out error is detected. Another typical cause of error for telephone speech recognition is the increased loudness of speech when the surrounding environment is noisy. The effect, known as *lombard speech*, causes recognition problems because of the modified feature extracted from the waveforms.

3.2.5 Recognition performance

Speech recognition can reach almost 100% accuracy on some applications. However, the performance varies greatly depending on the type of system used, such as for example speaker-dependent or speaker-independent, and the acoustic model implemented (it is obviously harder to recognize speech from a car when the system has been trained with a model suitable for telephone conversations).

Accuracy can be measured in two different ways, each of which provides information from a different perspective.

- Perfect word match

In this case, the efficiency of the identification is measured on the basis of words matching exactly some entry in the dictionary. It is not relevant if an alternative way to express the same concept exists, but it is important that the system recognizes precisely what is being spoken. Statistics like this constitute a good performance evaluation for dictation applications, where the crucial activity is the transformation of speech into its equivalent text format. The higher the percentage of exact word matches, the more reliable the system.

► Correct action

Speech recognition may not be perfect, but if the application is still able to issue the right action, then it can be considered very accurate nonetheless. In many applications it is more relevant that the command resulting from a voice instruction be correctly chosen rather than that the vocal representation of the control be perfectly matched. For example, speech engines embedded in control and command applications are better evaluated with this kind of statistics. This does not prevent us from using the previous method as well. However, data related to correct action accuracy give us an indication of the application fault tolerance, that is, the capability of the application to respond well to poorly recognized speech.

Typically, performance is analytically measured by a word error rate index, that is, the percentage of misidentified words in the vocabulary. This rate has decreased significantly over the past years because of substantial technology advances in speech recognition. For applications with limited word span, the error rate can reach values below 1% but there are domains with a 50% probability of errors. The error rate can also be estimated by the number of spoken words that are not present in the dictionary. This is, in fact, an upper limit to the former measurement. Table 3-1 reports typical figures for coverage of unseen text-based on the size of the dictionary used. This means that error rates are at most the values reported in the last column of the table.

Table 3-1 English text coverage and error rate based on dictionary size

Dictionary size (words)	Coverage	Estimated error
20000	94.1%	5.9%
64000	98.7%	1.3%
100000	99.3%	0.7%
200000	99.4%	0.6%

However, for languages with more inflections, errors are usually higher than those reported in the table. In this case, extra words are required to compensate the loss, though improvement is minimal.

3.3 Speech synthesis

The ability to generate speech from written text is called speech synthesis or Text-To-Speech (TTS). More specifically, TTS represents the ability to produce speech, using a grapheme-to-phoneme translation, with a computer-based system (also called *engine*). Voice synthesis is considerably different from the

ability to concatenate pre-recorded words or sentences using some underlying logic. The latter, known as Voice Response Unit (VRU) often uses pre-recorded human voice segments because of the limited vocabulary required for applications. This feature cannot be used in TTS, as it is practically impossible to prerecord all possible words that the system may need to read out. For this reason, a TTS engine incorporates the knowledge that is needed to perform reading.

As opposed to pre-recorded speech, automatic speech production allows larger flexibility and application domains. Similarly, its resource requirements, both in terms of hardware and software, are clearly distinct from those of pre-recorded voice segments.

Table 3-2 on page 49 shows a comparison between TTS and pre-recorded audio, based on the following criteria:

1. Hardware resources, that is, memory and storage.
2. Vocabulary, that is, the number of words that can be spoken by a given application.
3. Quality of voice (the natural feel).
4. Customizability of voice, that is, the capability to tune features like speed, gender or age of the artificial speaker, as well as language accents.
5. Intelligibility, that is, clear understanding of spoken words.
6. Flexibility, that is, the capability to adapt the synthesizer to a context different from the one normally used.
7. Cost, that is, the expenditures associated with the production of the text or pre-recorded audio.

Table 3-2 Comparison between TTS and pre-recorded audio solutions

	TTS	Pre-recorded audio
Hardware resources	low storage and memory	large storage, small memory
Vocabulary	unlimited	limited
Quality of voice	mainly mechanical	mainly natural
Customization of voice	unlimited and easy	audio needs re-recording
Intelligibility	generally quite good	very good
Flexibility	unlimited*	none
Cost	minimal	expensive**

* This include unlimited vocabulary, that is, the capacity of using TTS to generate pronunciation for ASR, the possibility of user dictionaries and phoneme mouth data (information about mouth movements associated with a given phoneme).

** This refers to the costs associated with the use of recording studios and professional speakers.

3.3.1 Synthesizer architecture

Reading a text is not one of the simplest skills to emulate using artificial resources. This is because of the different components that determine the way in which a text is pronounced. Spelling rules, intonation and context are just some of them. Each of the components needs to be modeled and implemented in a TTS engine. TTS is made up of two modules: a Natural Language Processing (NLP) module and a Digital Signal Processing (DSP) module, as shown in Figure 3-3.

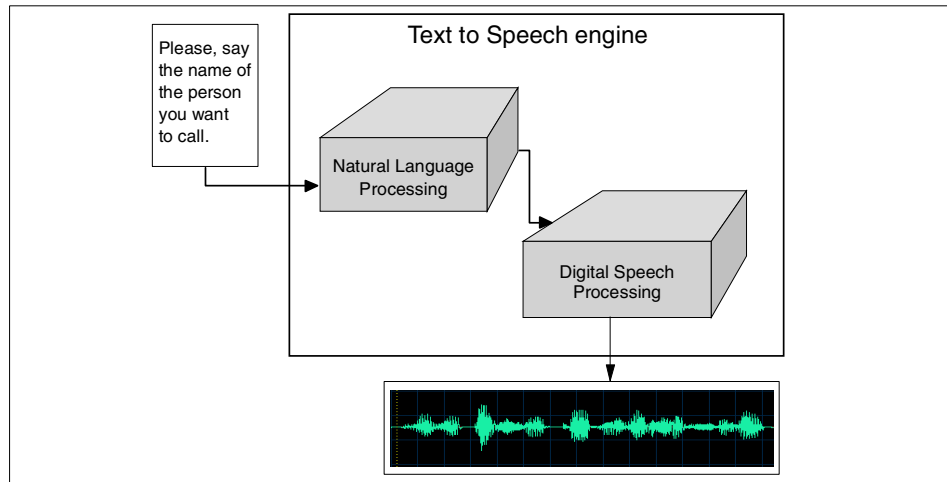


Figure 3-3 Schematic representation of a TTS process

Natural Language Processing module (NLP)

The first stage of speech synthesis is the transformation of the text to be read into its phonetic form, including intonation and rhythm. This process is carried out sequentially by different components of the NLP module, as depicted in Figure 3-4. Once the text is received by the NLP module, it undergoes two processes: a text analysis and a text-to-sound conversion. The analysis, also referred to as *text normalization*, consists of the following actions:

1. Parsing the text to identify words, numbers, abbreviations and the end of sentences. The last information can be very difficult to retrieve because of ambiguities caused by abbreviations, writing styles or misuse of punctuation. However, the problem is generally solved by using regular grammars.
2. Identifying for each word the speech categories it could represent: subject, verbs, adjectives, etc. The information is determined statically, that is, without accounting for any dependency between the words.
3. Reducing the number of possible categories by using NNs, Markov Models (MMs) or Classification and Regression Trees (CART) to identify syntactic dependencies between words.
4. Finding the text structure and organizing words in phrase-like bits to be converted into sound.

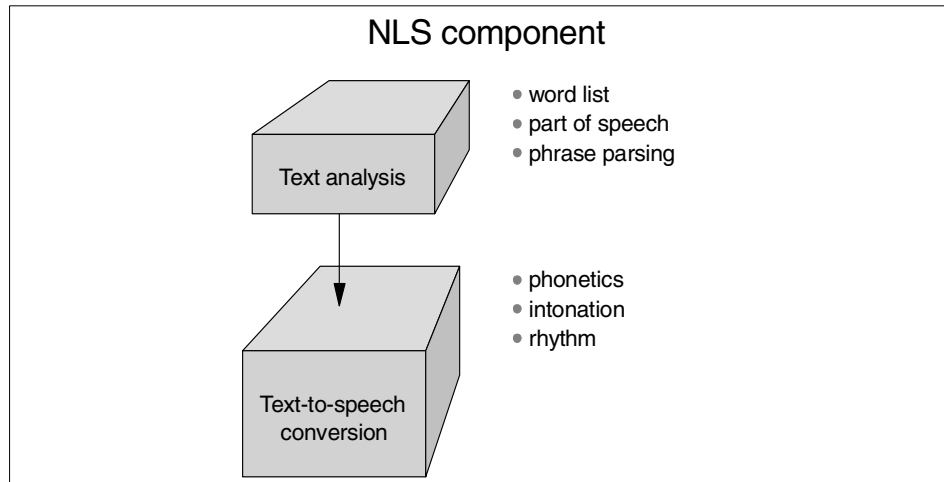


Figure 3-4 The Natural Language Processing component of the TTS engine

After analysis, the text is transformed into its phonetic equivalent using a dedicated software component. The conversion could easily be arranged using a table or a dictionary to match words with their phonetic equivalent. However, few problems need to be considered. Regardless of the size of the table, words are not usually stored with all their variants, such as masculine/feminine, singular/plural, conjugations, etc. Hence, the identification process need to be supported by a certain set of rules to help the system determine the base word to be searched. Other problems may occur because of ambiguity. There are words with the same spelling but different phonetic equivalents that depend on the context of the sentence. An example is the verb “to read”, which is spelled exactly the same way in all its tenses but it is pronounced differently. Also, there may be ambiguities because some words are used both as nouns and verbs (like “figure” for example) or because words can be pronounced differently when part of a sentence. In this case, the phonetic translation needs to be supported by rules determining exceptions to the standard conversion process. Together with the phonetic translation, the NLP module determines the intonation and rhythm indicated for the text to be read. Properties like pitch, loudness, rhythm and a few others are collectively referred to as *prosody*. These features not only make the synthesized speech sound more natural, but also provide details like word focus and relationships.

Digital Signal Processing Module (DSP)

The phonetic transcription of text is read out using the DSP module. This component of the TTS engine accounts for the production of speech from phonetics. In order to provide a realistic output, articulatory constraints need to be taken into account. This can be done either by using a set of rules describing

the influences between phonemes or by storing examples of concatenations of phonemes and using them to build up more complex acoustic units. It is at this stage that features like pitch or frequency are incorporated. The synthesis is either articulation-based, rule-based or concatenation-based.

Articulation-based synthesis

The production of speech is based on the physics of sound production and the physiology of speech. Equations of physics describe the radiation of sound waves from the mouth. Unfortunately, this approach is far too costly and complex to be used in practical applications and it leaves many problems still unsolved.

Rule-based synthesis

This approach is based on the dynamic evolution of speech. Typically, the synthesis is done using *formants*, large concentrations of energy from which it is usually possible to identify phonemes with a high degree of reliability. Formants refer to voiced phonemes only and are very difficult to estimate from speech data. For this reason, a good synthesis requires a significant trial and error process, which does not guarantee a high degree of naturalness unless the right rules are applied. A rule-based synthesizer requires many parameters (up to 60) to be tuned and analyzed and a thorough knowledge of the data to be handled. However, this method provides the possibility to study speaker-dependent voice characteristics. These can be used to build rules to switch between different synthetic voices in a relatively simple way.

Concatenation-based synthesis

This is speech synthesis based on concatenation of speech sound blocks stored in a database. The creation of the database plays a vital role in determining the quality of the synthesized speech. The major advantage of this solution is that all acoustic aspects of a real speaker are taken into account. However, because of this high degree of specialization, when a new voice or style need to be added, the database containing the speech blocks has to be re-segmented and re-analyzed. Also, a new set of recording details is required for each type of addition: whether it is a male or female voice, whether the speech is fast or slow, or whether the voice is young or age-marked.

3.3.2 Quality assessment for TTS

Synthesized speech certainly does not compare with natural human voice, but how close can we get? Nowadays, there is an increasing number of TTS products available on the shelf, each with its own features.

But what does quality mean in this context? At first glance, quality can be associated with the following characteristics:

1. Naturalness
2. Intelligibility
3. Pleasantness

A good TTS engine needs to produce speech that is as close as possible to a true human voice, while keeping speech understandable, clear and pleasant. TTS engines can usually model both female and male voices and it is often the case that female voices are clearer than their male counterpart. Other studies report male voices as being more intelligible because the fundamental frequency and formants range is more suitable for telephone conversations. Furthermore, the modeling adopted for TTS with female voice produces less robotic and more pleasant speech.

When evaluating the quality of synthetic speech, the following elements are typically taken into account:

- segmental rendering
- stress, rhythm and intonation accuracy
- variability of speaking rate
- control of intonation
- voice quality
- dialectal variation

However, on what basis is it possible to claim that a given TTS product performs better than another one? Is it because we can hear a better sound or because we there are not so many mistakes? Comparing different TTS methodologies is not easy because of all the different components that contribute to the synthesis. For example, a TTS engine may incorporate language-specific knowledge like accents or dialects, and artificial voice can be listened to in a variety of environments, such as over the telephone or through a speaker in a car. These settings affect the subjective perception of speech. Hence, if quality is evaluated by measuring the degree of human perception, the assessment may be impaired by the listener's capacity for understanding what it read out or by tiredness (when multiple products are sequentially tested). Researchers have looked into possible objective methods of assessing synthesized speech, for example the use of resynthesized speech (a method to code natural speech into parameters that can be used by a synthesizer). This way, a comparison between the natural and the artificial waveforms generated by the TTS engine gives an estimation of the degree of naturalness of the synthetic voice.

The concept of naturalness has been so far used in an intuitive manner, but it still has not been properly defined. What can be perceived as natural, especially when an objective evaluation is required? The answer is based on phonetic theories stating that a speaker usually adapts to a listener in order to minimize the cognitive load during perception. This means that the speaker is able to capture signs of fatigue, comprehension problems or other behaviors and act on the speech accordingly. Although this phenomenon can be identified in human speakers, it is not yet available for synthesized voice. Hence, naturalness is somehow compromised by the inability to adapt to the listener. Research in this area is still ongoing, but it is possible to measure the extent to which an artificial voice emulates characteristics of a natural one. These characteristics have been modeled and parametrized using human waveforms and are compared against the synthesized ones.

The quality of TTS products still lacks a solid ground for quality tests because the speaker/listener interaction does not influence the evaluation of the speech production.

3.4 IBM ViaVoice

IBM ViaVoice is a family of products providing both ASR and TTS capabilities. The speech recognition engine supports dictionaries containing approximately 200,000 words and is equipped with tools to generate customized grammars and user dictionaries.

The TTS engine is a rule-based speech synthesizer.

3.4.1 Multilingual support for ViaVoice

The IBM ViaVoice TTS engine is capable of synthesizing speech in different languages. However, an engine is only capable of producing speech for a single language. Table 3-3 shows information about the languages supported by IBM ViaVoice with respect to the platform where the engine is installed. It clearly appears that ViaVoice can be used across different platforms, although many languages are not yet available on the market.

Table 3-3 Language support for IBM ViaVoice

Language	WIN NT/95/98	AIX	Solaris	Linux
US English	Yes	Yes	Beta	Yes
UK English	Yes	Yes	Beta	Planned
French	Yes	Yes	Beta	Planned

Language	WIN NT/95/98	AIX	Solaris	Linux
German	Yes	Yes	Beta	Planned
Italian	Yes	Planned	Planned	Planned
Spanish	Yes	Planned	Planned	Planned
Portuguese	Yes	n/a	n/a	Planned
Japanese	Yes*	Planned	n/a	Planned
Chinese	Yes*	Planned	n/a	Planned
Finnish	Planned	n/a	n/a	n/a

*not all dialects of Japanese and Chinese are currently supported

3.4.2 ViaVoice limitations

Although it is possible to find products with multilingual support, a given ViaVoice engine is only capable of synthesizing speech for a single language. Multilingual support still exists but it implies the use of some form of logic to switch between several TTS engines, each of which supports a different language.

ViaVoice, like other TTS engines, still lacks a high degree of naturalness in some situations, especially because of the limited prosody associated with the text. Unfortunately, intonation and rhythm are affected by the sentence parsing process, which is itself dependent on the original text. So, for example, the list in Table 3-4 does not appear as such when spoken by a TTS engine. In fact, the parsing neither identifies logical sentences, reflecting the intention of the original writer, nor expresses correctly conventional symbols like “1/5” (one fifth), which is spelled instead.

Table 3-4 Example of parsing error in ViaVoice TTS

Original text	ViaVoice TTS parsed sentences
1. Open a new command window 2. Resize it to be 1/5 of your screen 3. cd to \temp 4. Type setup.exe	1.Open a new command window 2.Resize it to be 1 slash 5 of your screen 3. CD to slash temp 4.Type setup.exe

This does not impair the overall quality of the product, since the limitations are determined by the need for further development in speech synthesis, as mentioned in 3.6, “Future development” on page 59.

We mentioned earlier that the ViaVoice TTS engine is a formant-based synthesizer and that there are two different approaches to voice synthesis. Hence, ViaVoice does not yet offer a concatenation-based solution that allows potential customers a comparison of performance and a chance to select the most suitable approach for their application domain. The concatenation-based synthesizer has already been planned, however, and it is currently under development and testing.

3.5 Examples of voice-enabled applications

This section describes potential applications of both ASR and TTS. The list is not meant to be complete and is provided for guidance only. References to specific products are only included for information completeness.

3.5.1 Speech reco applications

This section discusses some applications built on speech reco.

Voice dialing

A good example of a control and command application is the so-called directory dialer, a product that enables the users to place calls using the name of the person to be reached. Unlike other similar applications that need the user to type DTMF tones to make selections, the directory dialer embeds speech recognition technology to identify the name of the person to call. Then, it retrieves the correspondent telephone number from a database and automatically dials while the user is waiting. When the recognition is not sufficiently clear, the user is prompted with either a message asking him/her to say the name again or with some kind of choice. For example, when the directory contains multiple entries corresponding to the same name, the dialer prompts the user with all available entries and waits for a decision. In other cases, when the name identified by the recognition process is not a good match, the user can decide whether to keep dialing or try to query the system again.

A similar application, known as voice dialing, is available on mobile telephones. The user of the telephone can associate each entry of its personal address book with a voice segment containing the spoken version of the name. The user is required to record the segment a few times in order to allow the underlying voice reco software to train and adapt to the speaker's voice. The user can then dial simply by speaking the name of the desired person. The input sequence is matched with one of the previously stored segments and a call is placed to the associated telephone number.

Voice-activated applications

This category includes all applications where voice can be used either as a navigation tool or as a mean to trigger actions. A very good example is the use of voice to manage resources. So far, this feature has been used mainly to help people with disabilities and provide them access to computer facilities. Nowadays, voice commands can also be used, for example, for house appliances (switching TVs, microwaves or lamps on or off). In this case, an underlying computer network is needed to connect the appliances to the application that includes a speech recognition engine.

Voice can also be used in a more conventional way to browse virtual menus that are “spoken” to the user (or even traditional menu-based information). In this case, the speaker follows virtual links by pronouncing a specified word or sentence that is recognized by the engine.

3.5.2 TTS applications

TTS applications are countless and the last years have seen an increasing use of TTS technology in a variety of areas, especially because a growing number of languages is supported. Synthesized speech has become particularly helpful to people with disabilities, as it has enabled them to access facilities that could have not been used otherwise. However, this does not represent an exhaustive domain for TTS, which can provide significant improvements in everyday activities. Following is a list of some application areas.

Augmentative communication

This category refers in a general way to applications designed to enhance information accessibility. People with speech disabilities can use a device to compose text and convert it into voice. Similarly, blind people can be read out information that would normally be available only in a visual format.

Computer access

Applications in this area usually include a speech recognition engine. The idea is to interact with other computer applications using voice-activated commands (where applicable) and to receive information in a spoken format. Clearly, the level of detail of the information returned does not equal that which can be provided using other devices (like video or paper for example).

Talking pages

Many services today are available in vocal format. Some companies have redesigned their applications to be suitable for a vocal media. The user dials a dedicated number and selects the required service using DTMF tones or speech. The application reads back the selected information. In some cases, applications like this can instead be implemented using VRUs if the content to be spoken is static or follows a predetermined format.

In-car navigators

These applications are designed to provide drivers with directions to reach their selected destination. The driver simply provides details about the desired location (optionally, the starting point if no GPS information is available) and the system gives back driving directions via speech, such as *“Turn right on highway 54.”*

Games

An increased number of game companies try to make their characters as close as possible to reality. This involves the use of voice technology, both for ASR and TTS (depending on the game). More sophisticated characters also simulate facial movements when speaking.

IBM MessageCenter for DirectTalk

This application uses both speech recognition and speech synthesis; it is part of the IBM Universal Messaging solution.

Communication nowadays plays a vital role in business and the possibility to retrieve and send information anytime and anywhere is definitely an advantage. The application is designed to provide the user with the capability to access messages using a variety of devices. This means that information from different sources (such as e-mail, fax, voicemail) can be accessed using the same device. The base product on which this application runs, named DirectTalk, is a VRU application with integrated Computer Telephony Interface (CTI) capabilities.

How can a user access messages using a single device? Currently supported devices are:

► Telephone

Voicemail messages can be accessed with no problem using a telephone, as this is the typical device used to create them. However, a remote connection to an e-mail server allows the user to handle e-mail messages, including faxes. The connection can be listened to using a TTS engine and a reply can be sent by recording a voice message over the telephone (this message will be translated into text using speech recognition). Fax handling is a bit more restricted, as the fax can only be redirected to a different user or to a local fax machine for printing.

- ▶ Web

A GUI shows the user the login session needed to identify the settings and access permission for that user. E-mail and voicemail messages are shown in a list. Voicemail messages are presented using audio files (.wav, .au) that the user can play and listen to. Faxes are converted to a graphical format (.tiff) and shown on the screen as images. The user can decide to redirect the image to a printer for a hard copy.

- ▶ WAP

The wireless component of this product has limited functionality at present but can already support simple message handling (without fax) and basic change of settings via the WAP interface.

- ▶ Fax machine

Currently, this is only used to receive faxes as a result of a telephone redirection.

3.6 Future development

Despite the fact the speech recognition research has made much progress in the last years, there is still a lot to be done. The algorithms currently used fail in a variety of situations, from a simple voice alteration to a change of environment. Therefore, most of the efforts will be directed towards an enhancement of performance for the recognition.

Portability

Current speech recognition works very well in a given context, but performance gets significantly worse when the situation changes. Normally, one should collect new data and retrain the system. However, this is a rather expensive and time-consuming procedure and it is not always worth the effort when the context might be changing again. One possible way to improve portability is to increase the size of the vocabulary and the number of rules in the grammar. However, regardless of the size of these databases, there is always a chance for unpredicted context and the problem then resurfaces. The introduction of some form of dynamic update for both the grammar and the vocabulary may be the answer, but other questions need to be addressed, such as deciding when the upgrade should be done and how.

Adaptation

This refers to the ability of the recognition process to adapt to new conditions, such as for instance a change in the hardware used to input speech. The more constraints one can put on the incoming speech, the better performance can be achieved. However, this approach leads to a lack of flexibility in the system and

may turn out to be a negative point. It is not unusual to upgrade pieces of technology like microphones or telephones (they may also simply need to be replaced due to breakage), but such a simple action can have a serious impact on recognition. Some of the characteristics of the device used to input speech are included in the acoustic model to improve performance and these features are often device-dependent. Hence, when the device changes, the model is no longer a valid description of the underlying environment, and performance is negatively affected.

Out-of-vocabulary words

As described in Section 3.2.1, “System architecture” on page 39, a vocabulary contains all the words that a voice recognition engine should identify. However, it is not possible to guarantee that a speaker will never use a word not included in the vocabulary, since he/she may not be aware of what the vocabulary contains. At present, the recognition process tries to identify the closest match for the input received, because it is not able to distinguish whether a given word belongs to the vocabulary or not. This behavior is not desirable for control and command applications since a wrong action will be invoked. A good way to solve this problem might be to use minimum threshold levels for the word match. In this case, when a match is very poor, an out-of-vocabulary exception can be fired and a message can be sent to the speaker. The negative side of that is that, depending on the threshold, words that *do* belong to the vocabulary might be misclassified as not belonging to it.

Robustness

Robustness and its various aspects will possibly be one of the most challenging areas in speech recognition in the coming years. Communications are changing very rapidly and speech engines need to quickly adapt to the new media and to the needs of the potential users of the system. For example, though speech recognition can be very accurate for certain applications, it is also true that accuracy can drop down sensibly when a non-native speaker is involved. The same is also true for native speakers with a strong regional accent. Although our ear can easily be trained to identify words regardless of accent and intonation, a speech recognition system cannot do so yet. In fact, this aspect of robustness is strictly linked with problems of adaptability and out-of-dictionary words, as regional accents influence the perception of speech. Robustness is also affected by transient interferences on telephone conversations and recognition of speech coming from devices with low signal-to-noise ratio. Hence, this represents another potential area for improvements.

Spontaneous speech

During a normal conversation, it may well happen that people sneeze, cough or hesitate before takings there may also be a conversation between people in the background. In all these circumstances, speech recognition performs quite poorly because the added noise affects the quality of the actual speech. It is certainly desirable to have a system where all these conditions could easily be dealt with.

Language modeling

Currently, speech technology is far from being human-like because of the various limitations imposed to perform both recognition and synthesis. However, the constant rise in mobile device use is causing a push for a wider use of voice technologies as well. ASR and TTS can offer functionalities that are not available in natural language, such a random access of data, remote data access and sorting. A breakthrough in the use of these resources will only be possible when improved language modeling is available. New models should be able to remove much speech variability caused by accents and external noise (for ASR) as well as synthesize voice using appropriate intonation and pauses. Statistical models are not able to capture all speech features and the use of other techniques (especially for the prosody) could be a successful approach to improving the accuracy of both ASR and TTS.

Dynamics modeling

ASR and TTS normally treat the information contained in window frames as is, without dynamics. However, speech is very dynamic and this form of variability need to be taken into account.

Prosody

As mentioned earlier in this chapter, prosody provides information about intonation, rhythm, etc. Prosody is significantly affected by the use of punctuation in TTS. In fact, when no punctuation is found in a long sentence, a human speaker naturally breaks the words into phrases of smaller length. However, an artificial system is not able to perform this task because of the inability to associate a meaning to the words being spoken. Consequently, sentences are identified by proper punctuation marks (like full stops, question marks and exclamation marks), as a TTS engine is unable to place pauses autonomously.

Prosodic information is neither available for synthesized speech nor captured during recognition, although it notably improves naturalness (for speech synthesis) and understanding (for speech recognition). Currently, the major problem to be solved is the integration of such information in the overall engine architecture.

3.7 Where to find more Information

- ▶ MFCC: see <http://ccrma-www.stanford.edu/~unjung/mylec/mfcc.html>.
- ▶ Text coverage: see <http://cslu.cse.ogi.edu/HLTSurvey/ch1node8.html>.
- ▶ Speech rejection: see B. Balentine, D. P. Morgan and W. Meisel, *How to Build a Speech Recognition Application*.
- ▶ Direct Talk: see http://www-4.ibm.com/software/speech/enterprise/ep_2.html for further details.
- ▶ Speech recognition: M. Schroeder (ed.), *Speech and Speaker Recognition*.
- ▶ A. Waibel and K.-F. Lee (ed.), *Readings in Speech Recognition*.



Part 2

Patterns for e-business



Patterns for e-business

Patterns for e-business are a group of proven, reusable assets that can help speed the process of developing applications. Some of these assets are:

- ▶ Architectural design patterns
- ▶ Runtime patterns and matching product mappings
- ▶ Design and implementation guidelines
- ▶ Code

Patterns for e-business provide development architects with a finite number of well-defined requirements which can be used for optimization of their solutions.

Application developers will find similarities between the Patterns for e-business and object-oriented application design patterns. Both of these are based on proven experience and provide systematic solutions for well-defined situations. While one deals with objects within an application, the other deals with elements within a solution architecture.

4.1 Using Patterns for e-business

The Patterns for e-business approach enables architects to implement successful e-business solutions through the reuse of components and solution elements from proven successful experiences. It is based on an analytical and systematic approach to solution design. This approach uses a sequence of steps that cover much of the implementation process, from the business definition to the architectural design, the application development and the system management.

The patterns approach is based on a set of layered assets which can be exploited by any existing development methodology. These assets include:

- ▶ Business patterns that identify the interactions between users, businesses and data.
- ▶ Integration patterns that tie multiple Business patterns together when a solution cannot be provided based on a single Business pattern.
- ▶ Composite patterns that represent commonly occurring combinations of Business patterns and Integration patterns.
- ▶ Application patterns that provide a conceptual layout describing how the application components and data within a Business pattern or Integration pattern interact.
- ▶ Runtime patterns that define the logical middleware structure supporting an Application pattern. Runtime patterns depict the major middleware nodes, their roles, and the interfaces between these nodes.
- ▶ Runtime product mappings that identify tested, optimal software implementations for each Runtime pattern.
- ▶ Best practice guidelines for design, development, deployment and management of e-business applications.

You can see these assets and their relation to each other in Figure 4-1.

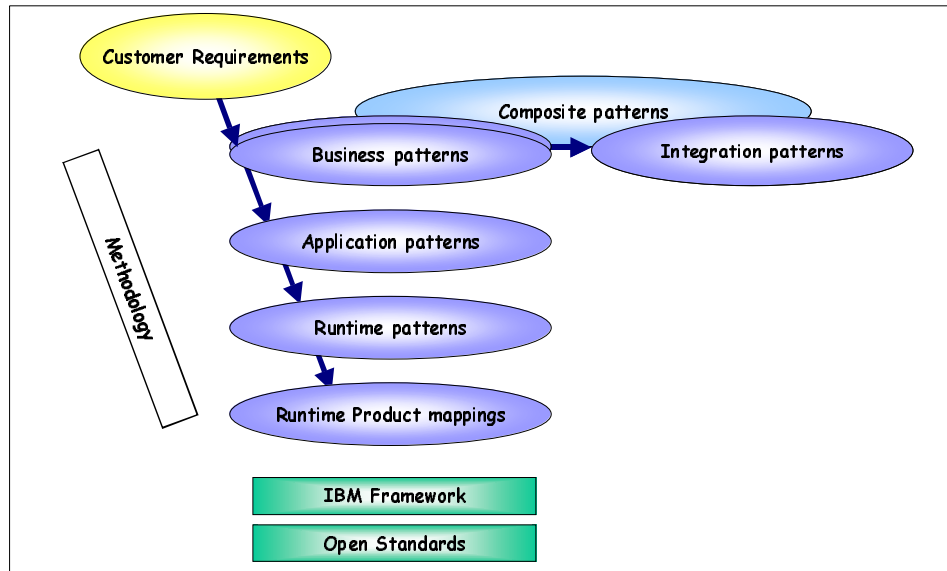


Figure 4-1 The patterns approach

Customer requirements are typically based on business needs or problems. Let us assume that these requirements are clear and that the objectives are well defined. The design starts from an abstract description based on the customer's requirements and reaches its final stage, based on a set of tools and components, in our case the IBM framework and Open Standards.

Finding the right solution is only possible by going through many architectural steps, where decision after decision leads you to the optimal solution. A set of well-defined options and proven experiences can help the architect make the necessary decisions. The Patterns for e-business include both these options and proven experiences.

4.2 Business patterns

Business patterns, as the name implies, represent common business problems. To understand a business requirement from an IT perspective, it is often easier to find an appropriate Business pattern for that requirement, then start the discussion from that point.

The four primary Business patterns are defined in Figure 4-2.

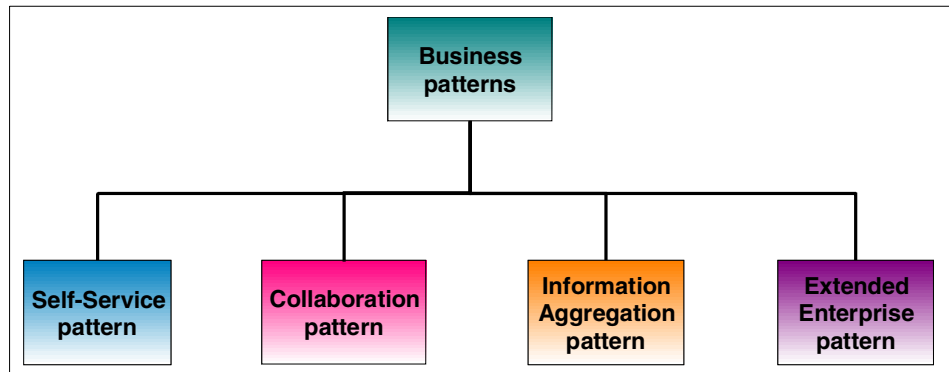


Figure 4-2 Business patterns

Self-Service business pattern

The Self-Service business pattern, also known as the User-to-Business pattern, represents the general case of internal and external users interacting with enterprise transactions and data.

Collaboration business pattern

The Collaboration business pattern, also known as the User-to-User pattern, applies to the use of e-mail and shared documents. The most popular implementation of this pattern is synchronous peer-to-peer applications, where users are connected to each other so that they can exchange messages and data online. An example is instant messaging.

Information Aggregation business pattern

The Information Aggregation business pattern, also known as the User-to-Data pattern, represents the use of tools to aggregate and distill useful information from large volumes of data, text, images, video, and so on. Some examples of this are business intelligence, knowledge management, and Web crawlers.

Extended Enterprise Business pattern

The Extended Enterprise business pattern, also known as the Business-to-Business pattern, represents programmatic communications between parties who do not belong to the same company. An example is supply chain management.

4.3 Integration patterns

Integration patterns provide the “glue” to combine Business patterns to form solutions (see Figure 4-3).

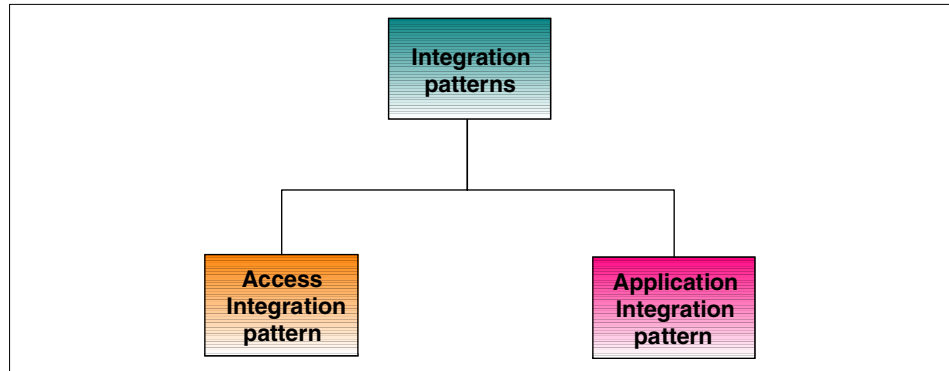


Figure 4-3 Integration patterns

There are two patterns in this category, the Access Integration pattern and the Application Integration pattern.

Access Integration pattern

The Access Integration pattern provides users with consistent and seamless access to various applications using different access mechanisms. It becomes relevant when:

- ▶ These applications need to be accessed using multiple devices, such as PC browsers, mobile devices, voice response units, or PDAs.
- ▶ Users need access to multiple applications and information sources without every application requiring its own sign-on in order to establish a separate security context.
- ▶ It is necessary to provide a common look and feel to these applications.
- ▶ The user wishes to customize the choice of applications and how they are presented.
- ▶ The customer does not want to rewrite the applications and does not want to make significant modifications to the code.

Application Integration pattern

Application patterns that apply to Application Integration patterns are different from those that apply to Business patterns in that they do not, by themselves, solve a business problem. They are used in combination with Business patterns to provide for the seamless integration of back-end applications and data.

4.4 Composite patterns

Often, the solutions to e-business problems will be more complex than any which can be defined by any individual Business or Integration patterns. To solve these problems, we need a custom design.

Composite patterns provide the ability to combine multiple Business and Integration patterns into one solution. For example, Figure 4-4 shows an example of a custom design. In this example, a composite pattern for a trading exchange solution is built using the two Integration patterns, Self-Service business pattern and Information Aggregation pattern. The Business patterns in italics are optional.

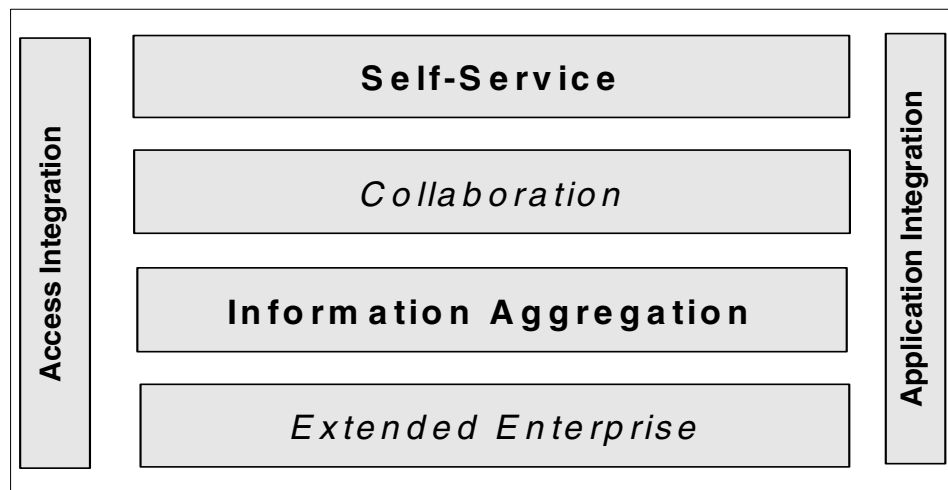


Figure 4-4 A Composite pattern for a trading exchange

Some Composite patterns that have been identified are:

- ▶ Account Access
- ▶ Electronic Commerce
- ▶ Portal
- ▶ Buy-side Hub
- ▶ Sell-side Hub
- ▶ Trading Exchange

4.5 The patterns used in this book

This book will focus primarily on the Access Integration pattern. It will illustrate it using the WebSphere Everyplace Access offering to provide a seamless common access to back-end applications. The application in this instance will be an example of the Self-Service business pattern.

4.6 Where to find more information

- ▶ Patterns for e-business Web site available at <http://www.ibm.com/developerworks/patterns>
- ▶ Patterns for e-business resources available at <http://www.ibm.com/developerworks/patterns/library/index.html>



Application patterns

This chapter introduces Application patterns, and discusses the following questions:

- ▶ What is the coarse-grained application structure?
- ▶ How are the tiers linked?
- ▶ Where is the data?

In particular, we will take a look at the Application patterns used in this book.

As mentioned previously, the discussion in this book centers around a sample solution based on the Access Integration pattern and the Self-Service business pattern. The Application pattern used for the Access Integration pattern provides pervasive device access to the application and we will examine this. The application itself is built based on the Self-Service business pattern and we will take a look at the Application pattern it implements also.

5.1 Application patterns

Application patterns use logical tiers to illustrate ways of configuring the interaction between users, applications, and data. The chosen Application pattern is later associated with a Runtime pattern by mapping the logical tiers to runtime nodes. An Application pattern 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 can be stored.

In the previous chapter, we discussed the Business and Integration patterns, including how to select the appropriate pattern for your solution. The Application patterns represent the next step necessary in designing the solution. This level is still abstract, but we are getting closer to the realization of the solution.

Each Application pattern describes the following:

- ▶ Structure
 - Coarse-grained components (tiers) of the application
 - Interactions between the coarse-grained application components
- ▶ Placement
 - How do we split up processing?
 - Where do we place data?
- ▶ Integration
 - Loosely coupled versus tightly integrated
 - Impact on back-end systems

These elements represent an abstract layer describing the application components. They represent application and data placement, derived from the business requirements, but also taking real IT requirements into consideration.

Figure 5-1 shows a legend for all the Application pattern's diagrams.

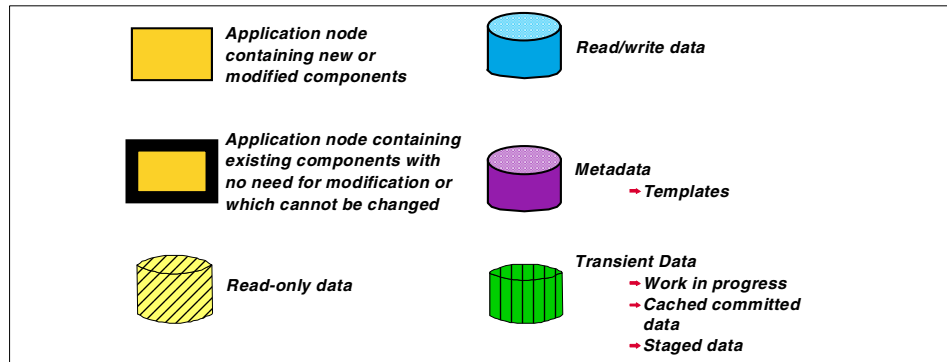


Figure 5-1 Legend for Application pattern diagrams

5.2 Application patterns for Access integration

Three Application patterns have currently been identified for the Access integration pattern:

- ▶ Pervasive Device Access
- ▶ Single Sign-On
- ▶ Personalized Delivery

Each of these Application patterns is designed to provide a single, consistent, and seamless access mechanism to applications that would otherwise require the use of several different access mechanisms. These patterns are not mutually exclusive and you could very well find more than one implemented in a solution.

This book focuses on the Pervasive Device Access pattern, which supports mobile e-business solutions.

5.2.1 Pervasive Device Access application pattern

The Access Integration pattern is used to provide consistent access to various applications using multiple device types. In order to provide pervasive device access to an existing Business pattern, we therefore need to use an Access Integration application pattern. This Application pattern brings a new tier into the architecture. This tier is responsible for the pervasive extensions to the original application. The function of this tier is to convert the HTML issued by the application presentation logic into a format appropriate for the pervasive device.

Figure 5-2 on page 76 shows the Application pattern for pervasive device access.

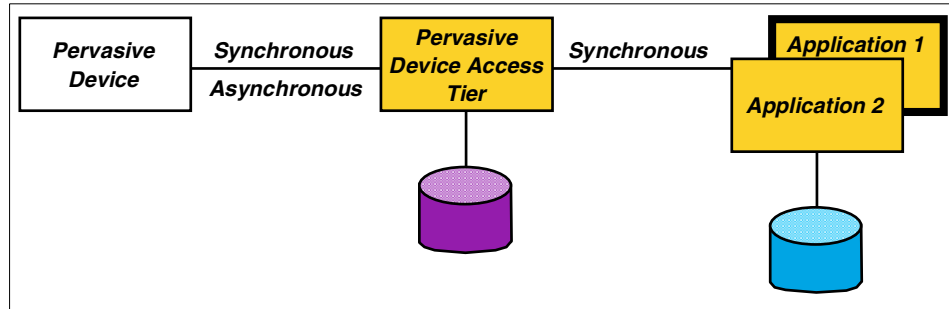


Figure 5-2 Pervasive Device Access application pattern

The connections between the elements used in this book are all synchronous. Although pervasive devices can maintain asynchronous connections like notification synchronization, these services are beyond the scope of this book.

5.3 Application patterns for Self-Service

The Self-Service business pattern is important from our perspective, because in many cases the mobile application is based on an existing Web application. Since the most popular Web applications for mobile e-business are the Self-Service Web applications, we have to pay particular attention to this Business pattern.

Figure 5-3 shows the seven currently identified Application patterns for the Self-Service business pattern.

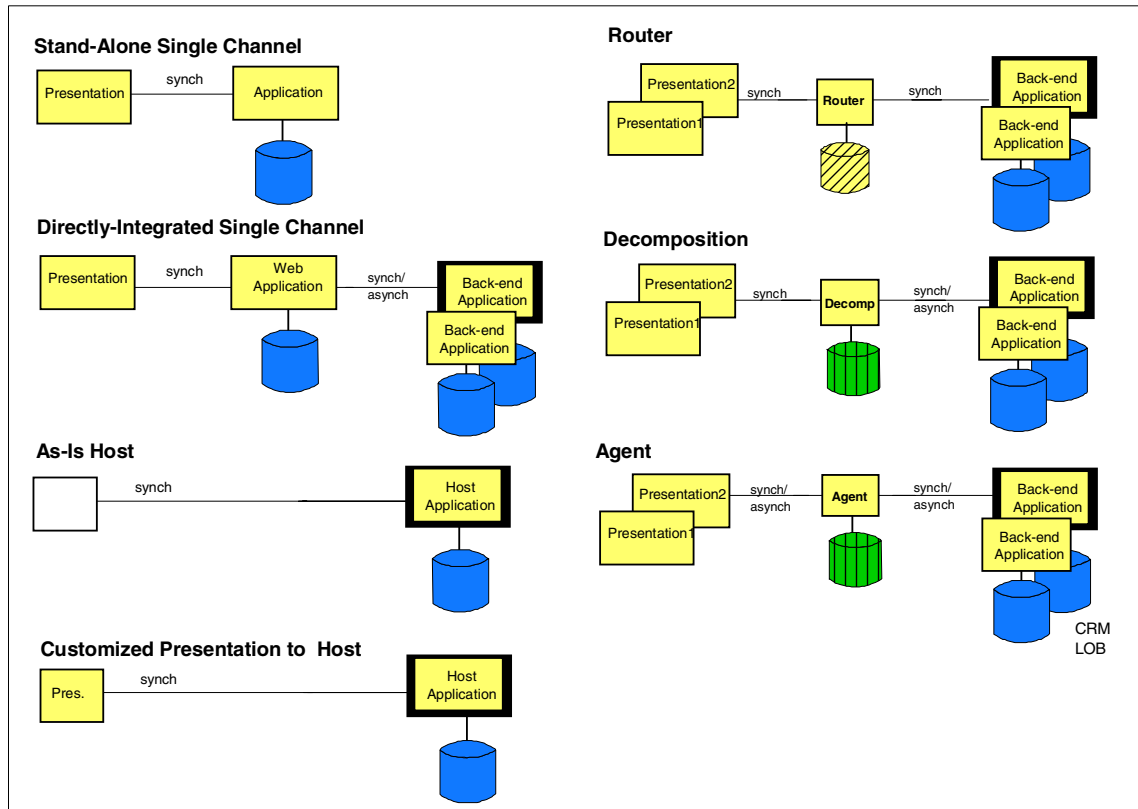


Figure 5-3 Self-Service business patterns

5.3.1 Stand-Alone Single Channel application pattern

The application in this book is based on the simplest pattern, the Stand-Alone Single Channel application pattern. This pattern provides for stand-alone applications that have no need for integration with existing applications or data. It assumes one delivery channel, most likely a Web client, although it could be something else. It consists of a presentation tier that handles all aspects of the user interface, and an application tier that contains the business logic to access data from a local database. The communication between the two tiers is synchronous. The presentation tier passes a request from the user to the business logic in the Web application tier. The request is handled and a response sent back to the presentation tier for delivery to the user.



Runtime patterns

The Application pattern can be further defined into more explicit functions to be performed. Each function is associated with a runtime node. In reality, these functions, or nodes, can exist on separate physical machines or may co-exist on the same machine. In the Runtime pattern, this is not relevant. The focus is on the logical nodes required and their placement in the overall network structure.

This chapter will give a general overview of Runtime patterns and how they are used, but will focus on the Runtime patterns used in this book.

6.1 Runtime nodes

A Runtime pattern is represented by a logical node diagram, where each node has a specific role in the architecture. It defines the topology of the architecture and node placement. Each Runtime pattern will have nodes that demonstrate certain service level characteristics, including:

- ▶ Availability
- ▶ Performance
- ▶ Scalability
- ▶ Security

Most patterns will consist of a core set of common nodes, with the addition of one or more nodes unique to that pattern. To understand the Runtime patterns presented in this book, you will need to review the following node definitions.

User node

The user node is most frequently a personal computing device (PC, etc.) supporting a commercial browser, for example, Netscape Navigator or Internet Explorer. The level of the browser is expected to support SSL and some level of DHTML.

Client node

The client node represents the user interface client such as a browser, mobile phone, or PDA. This is a more specific instance of the user node.

Gateway node

Gateway nodes switch between the different networks to establish communication between pervasive devices and the Web applications. This only means that the two parties can communicate with each other. It does not mean that they will understand each other. Communicating and passing data between the two parties is one thing, but adapting the content and translating between different protocols is another.

Voice Server node

The voice server node has the responsibility of transforming the special voice application content to voice. Basically it is running numerous VoiceXML browsers, which are browsing the content from the server side, and handling the client interaction via the phone (voice synthesization, voice recognition). The voice server node connects to the servers in the DMZ using HTTP. On the client side, it has to be switched (using a gateway) to the phone network.

Load balancer node

The load balancer node provides horizontal scalability by dispatching HTTP requests among several, identically configured Web server nodes. Using a load balancer node introduces the need to ensure session affinity.

Transcoding proxy node

The transcoding proxy is an intermediary server that transforms the content going through it. Different client types require different representations of the original content. The transcoding proxy transforms the content to a suitable format for the client. The transcoding proxy can be either a forward-proxy or a reverse-proxy.

Web application server node

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 node. 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.

Web server redirector node

In order to separate the Web server node from the application server node, a so-called *Web server redirector node* (or just *redirector* for short) is introduced. The Web server redirector is used in conjunction with a Web server. The Web server redirector serves HTTP pages and 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.

Application server node

The application server node provides the infrastructure for application logic and may be part of a Web application server node. 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.

Protocol firewall and domain firewall nodes

Firewalls provide services that can be used to control access from a less trusted network to a more trusted network. Traditional implementations of firewall services include:

- ▶ Screening routers (the protocol firewall in this design)
- ▶ Application gateways (the domain firewall)

The two firewall nodes provide increasing levels of protection at the expense of increasing computing resource requirements. The protocol firewall is typically implemented as an IP router, while the domain firewall is a dedicated server node.

Database server node

The database server node provides a persistent data storage and retrieval service in support of transactional interactions. The data stored is relevant to the specific business interaction, for example, bank balance, insurance information, current purchases by the user, etc.

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.

Directory and security services node

The directory and security services 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, to verify certificates. The authentication in most current designs validates the access to the Web application server node part of the Web server, but it can also authenticate for access to the database server node.

6.2 Runtime pattern for the Self-Service application

In order to understand the complete concept, it is best to start with the basic elements. In this case, the Access integration pattern provides a service to enhance the Self-Service application. If we were running our application without the mobile access integration, we would choose a Runtime pattern for the Self-Service application pattern. There are multiple Runtime patterns defined for this business pattern and you can see each of them in detail by referring to *Self-Service Patterns using WebSphere Application Server V4.0*, SG24-6175.

Of the available Runtime patterns, we chose two basic patterns as a basis for our Self-Service application implementation.

6.2.1 Basic Runtime pattern

This Runtime pattern provides an initial implementation with an entry-level footprint. It is a simple yet effective way to make the solution available. The basic pattern uses a minimum of runtime nodes, yet provides a measure of security by putting all sensitive persistent data behind a firewall.

While it does not provide scalability or failover capabilities, it is a good starting point from which you can easily progress to Runtime patterns that do provide these functions.

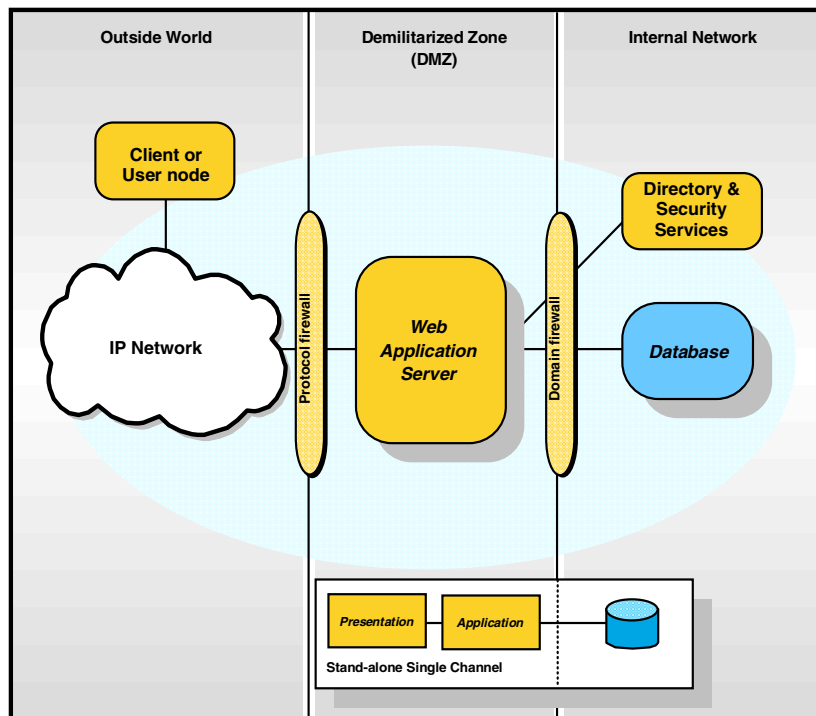


Figure 6-1 Basic Runtime pattern: Stand-alone Single Channel application pattern

The presentation logic and business logic of the application are provided by a single Web application server node in a Demilitarized Zone (DMZ). The data to be accessed from the business logic is behind the domain firewall in the internal network.

In addition to the network security provided by the firewalls, application-level security is provided by the Web application server node. The user information required for authentication and authorization is stored in the directory and security services node behind the domain firewall in the internal network.

6.2.2 Runtime variation

This Runtime pattern focuses on the network security aspects. A typical requirement is often the need to separate the Web server node from the Web application server node, most likely in order to place the application server node behind a DMZ into a secure environment. The corresponding Web server node can be placed inside the DMZ behind a firewall.

This separation of the Web server node from the application server node is done by means of the Web server redirector. The Web server node, acting as a redirector, serves static HTML pages but forwards servlet and JSP requests to a dedicated server.

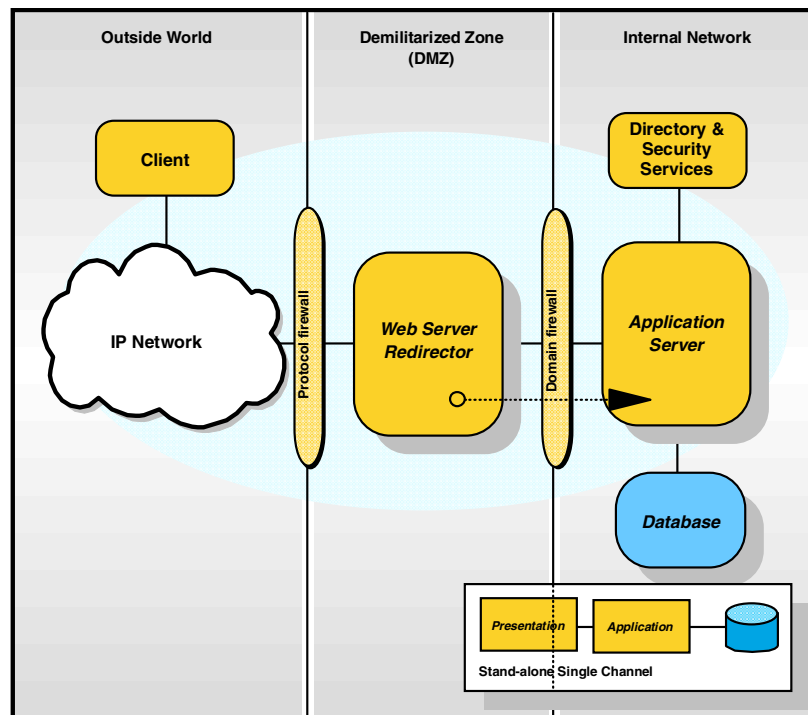


Figure 6-2 Basic Runtime pattern variation: Stand-alone Single Channel

This variation uses one Web server node and one application server node, effectively splitting the function of a Web application server node across two machines. In this case, the application server resides in the internal network to provide it with more security. The application server node will run both presentation and business logic.

The Web server node remains in the DMZ and serves static pages. A Web server redirector is used to forward the requests from the Web server node to the application server. Using a redirector could affect performance, so the options for implementation should be compared.

This pattern is especially useful when you do not expect many clients to access the server simultaneously and when you want to have additional security by separating the Web server node from the application server node.

6.3 Runtime pattern for the Pervasive Device Access

Now that we have the basic patterns identified for the application, we will go a step further and combine these with the nodes needed to incorporate pervasive access functions. The various combinations will become the basis to form Runtime patterns for the Pervasive Device Access application pattern.

6.3.1 Base Runtime pattern

The base Runtime pattern shown in Figure 6-3 on page 86 extends the basic Self-Service runtime pattern to allow a wide variety of pervasive devices to participate in the e-business solution. This gives a much richer user experience on the one hand, and extended functionality on the other.

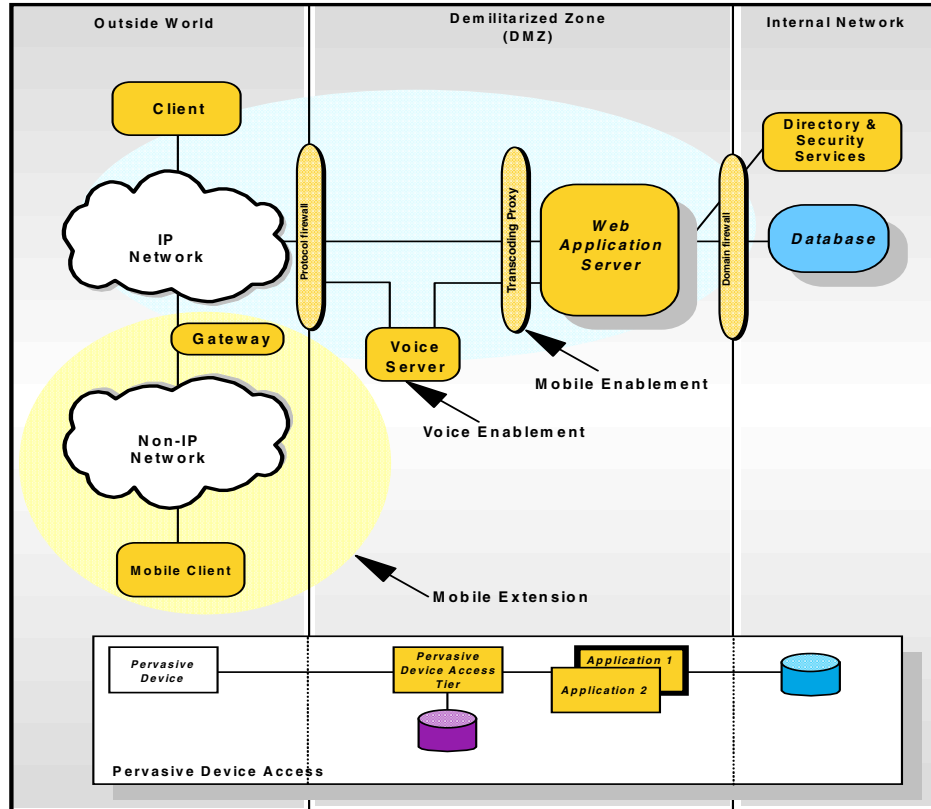


Figure 6-3 Runtime pattern for mobile Web applications

This pattern assumes that there will be incoming requests for the application from both IP and non-IP networks. The non-IP network consists of wireless networks (GSM, CDPD, etc.), and phone networks (PTSN). In order to access the IP-based networks (Internet or intranet), a special gateway is required. Phone networks require Voice over IP gateways to connect the analogue or digital lines connection to the IP-based, packet-switched networks. Wireless networks also have to use gateways to handle the connections, translate the protocols, and connect to IP networks (for example WAP or i-mode).

The voice server node provides voice enablement, while the transcoding proxy (Web intermediary) provides mobile enablement. The term *mobile enablement* may sound strange, since this pattern by definition implies enablement for mobile devices. However, in this instance *mobile enablement* refers to enabling the Web application, not the architecture, for mobile access. Although this mobile enablement does not include all the pervasive devices, remember the terminology we follow in this book (refer to 1.1, “Definitions” on page 4).

6.3.2 Runtime pattern variation

This variation changes the basic runtime pattern slightly by moving the business logic into the internal network to provide enhanced security.

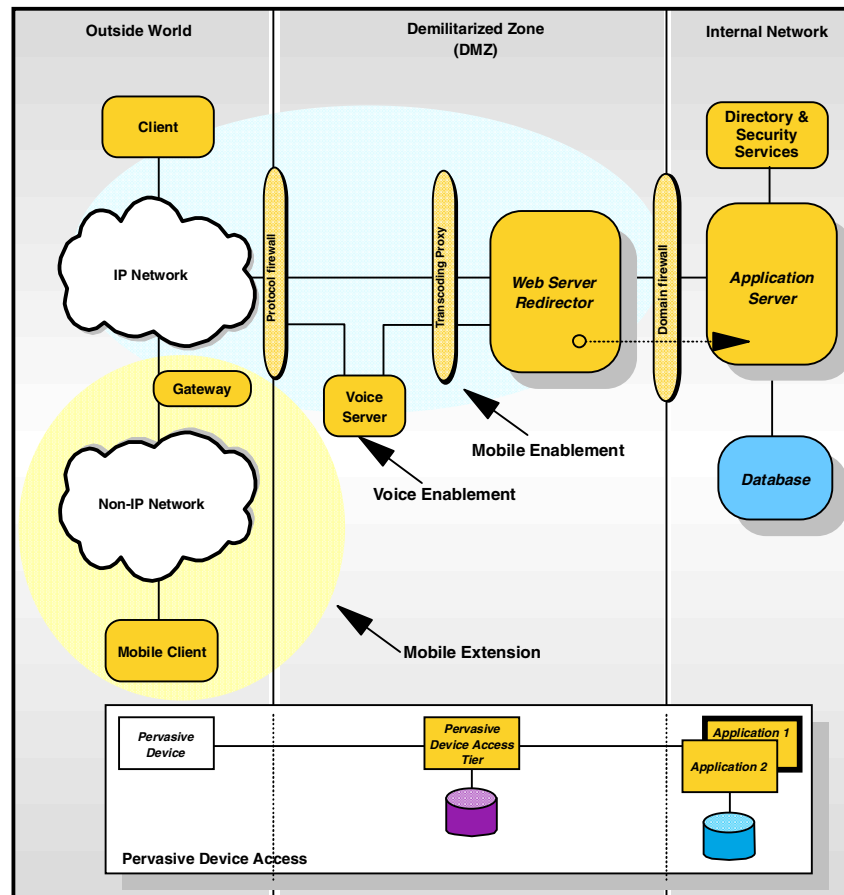


Figure 6-4 Mobile variation extended

The Web application server is split into two nodes, the Web server redirector and the application server. The Web server redirector performs the functions of an HTTP server, serving static pages. Other requests are forwarded to the application server, where the bulk of the application logic resides.

The transcoding proxy node is connected to the Web server redirector.

6.4 Gateway placement

In every mobile solution, there must be a gateway somewhere in the architecture. It is fairly obvious in the logical architecture where to put the gateway. The question is, where should the gateway be physically? There are three choices:

- ▶ In the enterprise
- ▶ At the ISP
- ▶ At the carrier

Placing the gateway in the enterprise is the most expensive choice, not only because of the cost of a gateway, but because of having to achieve the availability required.

The ISP, on the other hand, can absorb the expenses of a gateway. In addition, high availability can be achieved. However, security could be weak in this case.

The most obvious choice is to put the gateway at the carrier's site. The costs are absorbed by the carrier, they guarantee availability, and you have the assurance that the gateway will work with that carrier. However, keep in mind that this last point could also turn out to be a disadvantage because the gateway will only serve the connections coming through that provider.



Runtime product mappings

Runtime product mappings are the key needed to complete the solution. In product mappings, the nodes in the Runtime patterns are populated with products. Refining the solution down to the product level allows you to take into consideration configuration and deployment options of the products that can impact the architecture or application design. The best example of this can be seen in the WebSphere Transcoding Publisher deployment modes, which can change the architecture as well as the application itself.

7.1 Selecting products

Selecting products means selecting hardware and software elements as well. Choosing the right elements is always difficult because of various considerations which must be taken into account. In some cases, decisions are made considering not only the technical and logical points but other factors such as expenses, existing knowledge, or time.

The following considerations should be taken into account during the design:

- ▶ Platform

Choosing the right platform depends on several other points. The platform in this context means the operating system and the underlying hardware.

- ▶ Homogenous/heterogeneous solution

This is closely related to the platform, whether the solution deals with different platforms (heterogeneous), or just one (homogenous).

- ▶ Availability

Availability is very complex from the architecture standpoint. It is related to the number of systems. The higher the redundancy, the higher the availability; this also means higher costs. It is very important to ensure just the right level of availability; less than required can cause loss of business, more than required can raise the costs.

- ▶ Performance

Performance has influence on the end user experience and has a significant impact on the business. Users want fast responses, and the system has to perform well, no matter what kind of application is involved.

- ▶ Security

Security has always been an issue. It depends on the business, and becomes very important for mission-critical applications. Just as with availability, the right level of security has to be found. Security can be very strict or stay at a basic level; this depends on the requirements.

- ▶ System management

Solutions cannot stand alone without any system management. E-business solutions are moving towards centralized system management, but users still have to deal with multiple tools to manage the system within their solution.

► Time

Building a solution takes some time. There is always the question: how long is it going to take?

Choosing the right technologies and finding the fastest way to apply them are two different things. Developing the solution is a matter of time; it depends on the chosen technology, the existing knowledge and the available resources.

► Money

Money has the biggest influence on the solution. Since this is a technical book, it is beyond its scope to analyze the various aspects of the influence of money on the implementation.

7.2 Product mappings

Up to now, the designs have not specified concrete products or even concrete technology for the solution. These patterns are open for any solution; they show a path for any implementation.

The following product mappings will show that everything required to implement the Runtime patterns is included in the WebSphere Everyplace Access offering. The offering provides all the components required for a mobile e-business solution. Not only is the runtime environment provided, but products for the development environment are provided as well. The development environment and the products will be discussed later in this book.

Figure 7-3 shows products that can be used to implement the basic Runtime pattern for the Pervasive Device Access application pattern. This Runtime pattern is discussed in 6.3.1, “Base Runtime pattern” on page 85.

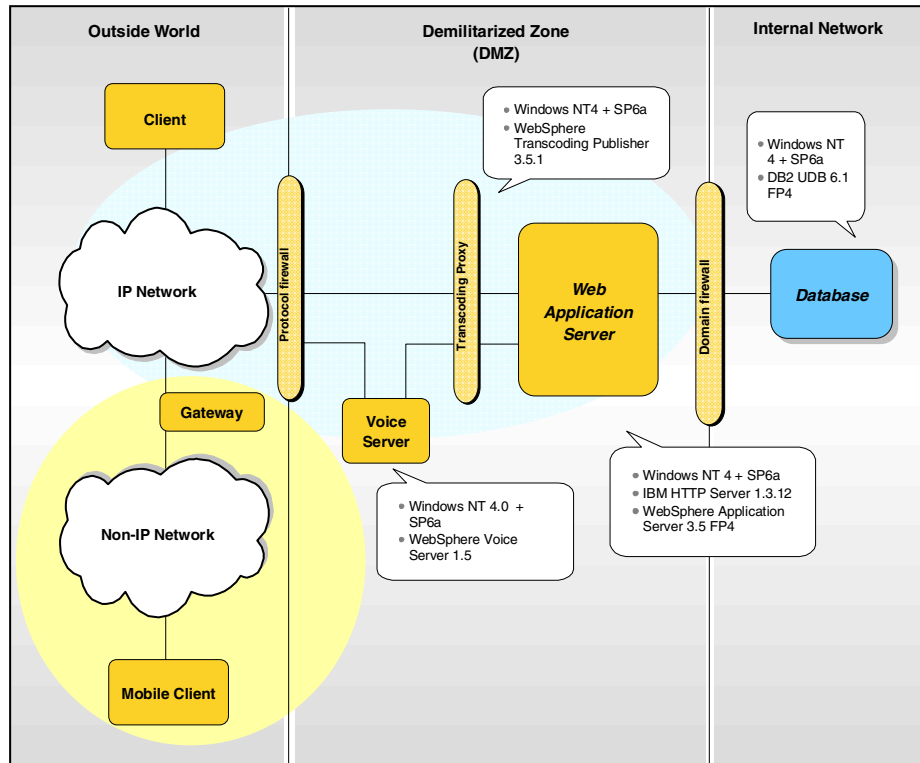


Figure 7-1 Product mapping for basic Runtime pattern

Figure 7-2 shows products that can be used to implement the basic Runtime pattern variation discussed in 6.3.2, “Runtime pattern variation” on page 87.

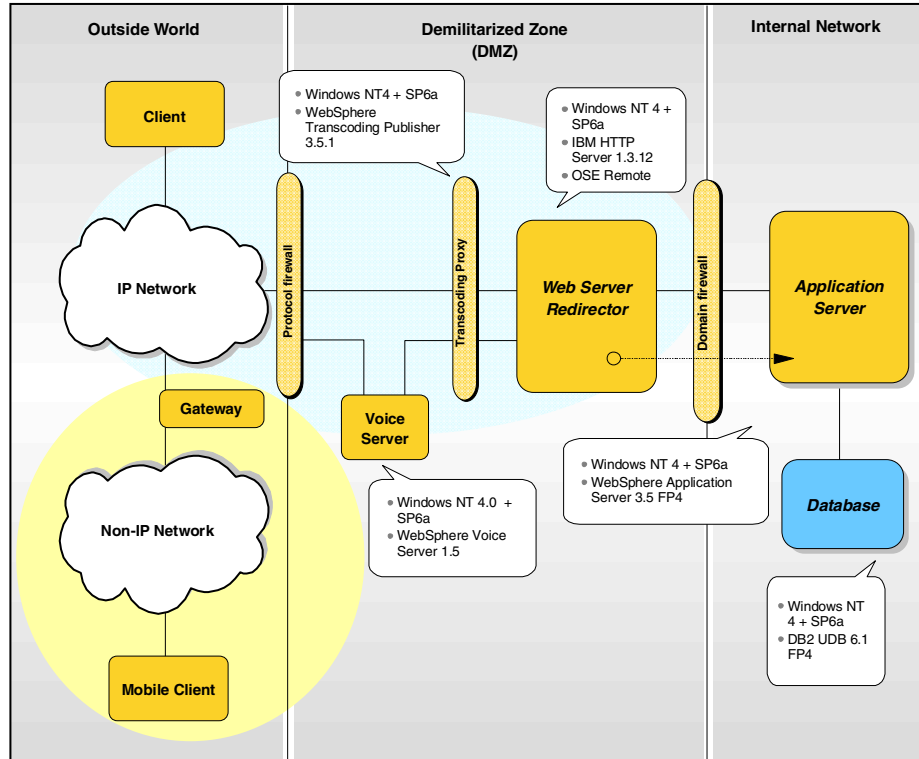


Figure 7-2 Product mapping for Runtime pattern variation

The last product mapping, shown in Figure 7-3, does not have a matching Runtime pattern. This is because it is an implementation specific to a particular product, WebSphere Transcoding Publisher. Runtime patterns show common architectural patterns that could be implemented by multiple product sets. This implementation shows a specific feature available in a particular product. More information on the placement of the WebSphere Transcoding Publisher can be found in 7.3, “WebSphere Transcoding Publisher considerations” on page 97.

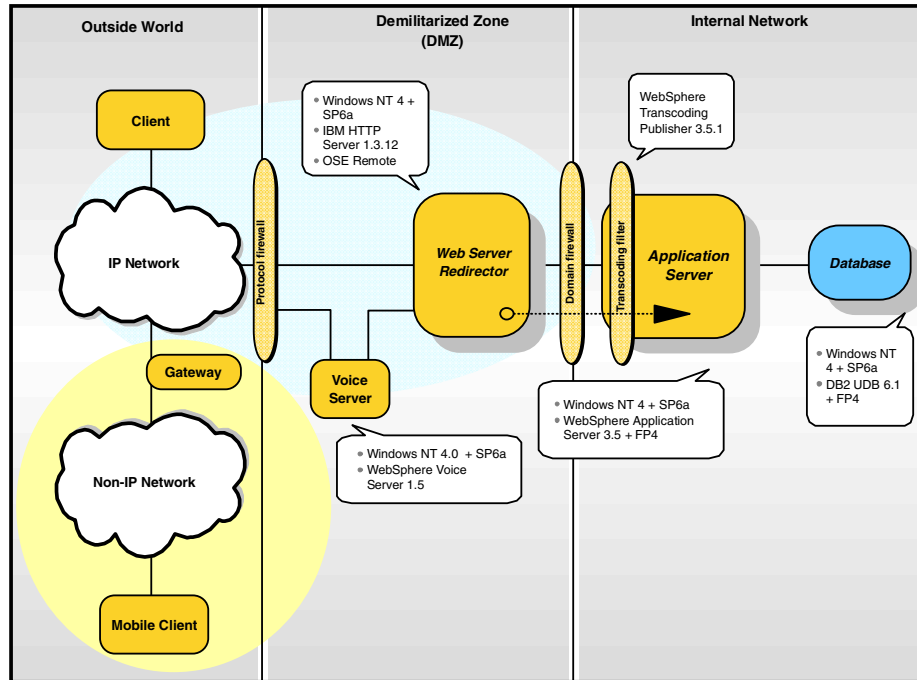


Figure 7-3 Product mapping specific to the product set

In all of these product mappings, the products shown were implemented in Windows NT. It is important to note, however, that they are available for other platforms as well. This gives you a wide range of choice for hardware and operating systems. In Chapter 19, “Runtime environment for the sample application” on page 383, the runtime environments for two different platforms and two different scenarios are discussed.

7.2.1 The WebSphere Everyplace Access V1R1 offering

The following tables show the products that were used in the product mappings and the platforms on which they are available. These products are available with the WebSphere Everyplace Access offering. To find out more about these products, see the list of references in 7.4, “Where to find more information” on page 99

The WebSphere Application Server serves as the Web application server node in the solution. It has a tight integration with the Web server (IBM HTTP server).

Table 7-1 WebSphere Application Server

WebSphere Application Server	
Version	Advanced Edition 3.5
Update	Fixpack 4
Available platforms	<ul style="list-style-type: none"> ▶ AIX ▶ Windows NT 4 / Windows 2000 ▶ Linux ▶ Solaris ▶ HP-UX
Additional	<ul style="list-style-type: none"> ▶ IBM JDK 1.2 ▶ DB2 Runtime Client v6.2 (fixpack 4)

The WebSphere Transcoding Publisher is the intermediary application which transcodes the content from one transcoder to another. The transcoders are pluggable items under the server, and new ones are being introduced on a regular basis.

Table 7-2 WebSphere Transcoding Publisher

WebSphere Transcoding Publisher	
Version	3.5.1
Update	<ul style="list-style-type: none"> ▶ HTML-to-VoiceXML transcoder ▶ Additional device profiles
Available platforms	<ul style="list-style-type: none"> ▶ AIX ▶ Windows NT 4 / Windows 2000 ▶ Linux ▶ Solaris

WebSphere Voice Server is a standalone server application which runs the VoiceXML browser for each user. It also talks to the VoIP gateway on one side and to the Web application on the other. The server has different editions for the different platforms.

Table 7-3 WebSphere Voice Server

WebSphere Voice Server	
Version	1.5
Available platforms	<ul style="list-style-type: none"> ▶ AIX (DirectTalk platform) ▶ Windows NT 4

The database server that comes with the WebSphere Everyplace Access offering is intended for use by WebSphere Application Server as a administrative repository.

Table 7-4 DB2 UDB

DB2 UDB	
Version	6.1
Update	Fixpack 4
Available platforms	<ul style="list-style-type: none"> ▶ AIX ▶ Windows NT 4 / Windows 2000 ▶ Linux

The IBM HTTP Server comes with WebSphere Application Server. This server is based on the Apache OpenSource HTTP Server. It provides the Web server services for the Web application.

Table 7-5 IBM HTTP Server

IBM HTTP Server	
Version	1.3.12
Available platforms	<ul style="list-style-type: none"> ▶ AIX ▶ Windows NT 4 / Windows 2000 ▶ Linux

7.2.2 Additional products for the offering

In addition to the products illustrated by the product mappings, the WebSphere Everyplace Access offering includes several other products.

Secureway Directory Server

The Secureway Directory Server provides an LDAP directory server and services for the solution. LDAP is utilized in several situations, for example centralized security and common directory services for distributed components.

WebSphere Edge Server

WebSphere Edge Server is a multipurpose server. It provides load balancing, a caching proxy, and content distribution.

Secureway Firewall

In our architecture, the firewalls are running the Secureway Firewall product. Since it is more like a network component, it is not closely related to the other products. It is only mentioned in this list because it is an essential part of the architecture and security.

7.3 WebSphere Transcoding Publisher considerations

WebSphere Transcoding Publisher provides three different deployment modes for running a transcoder. These options are all available for building a mobile e-business application, though not every one is optimal.

The different WTP deployment modes are:

1. As a proxy (network intermediary)
2. As a filter running in IBM WebSphere Application Server
3. As JavaBeans

7.3.1 Proxy model

In this configuration, WebSphere Transcoding Publisher is a single service that tailors content coming from many different Web servers. The proxy intercepts HTTP requests and responses as they flow between the user and the Web server. However, this configuration does not tailor content that is encrypted between the user and the Web server.

If you run Transcoding Publisher as a network proxy, you can use it with or without a cache server. If you use a cache server in your network, WebSphere Transcoding Publisher can use it to store and retrieve transcoded Web pages and intermediate results. This may enable WebSphere Transcoding Publisher to avoid repeating the transcoding of frequently accessed pages.

If you run WebSphere Transcoding Publisher as a network proxy, and your network uses a cache server or a firewall, you will need to supply the addresses and port numbers used by these servers when you configure WebSphere Transcoding Publisher.

7.3.2 Filter model in WebSphere Application Server

In this configuration, WebSphere Transcoding Publisher tailors the content generated by a single Web server. When running as a filter, Transcoding Publisher can tailor content before it is encrypted and sent to a user. However, when you run WebSphere Transcoding Publisher as a filter, you cannot use network preference profiles or the Request Viewer from the WTP Toolkit.

If you use WebSphere Application Server in your network, running Transcoding Publisher as a WebSphere Application Server filter enables you to tailor the output of other filters and servlets for your pervasive devices. If you use encryption within the WebSphere Application Server, then you must use WebSphere Transcoding Publisher as a filter so that it can modify content before it is encrypted. If you do not use encryption, then you could run WebSphere Transcoding Publisher either as a filter or as a proxy. The filter deployment model changes the runtime environment a bit. Figure 7-3 on page 94 reflects the new Runtime pattern.

7.3.3 Running JavaBean transcoders

The modules inside the Transcoding Publisher transcoding server that modify content are called *transcoders*. The transcoders are also provided as Java beans, which can be run independently of WebSphere Transcoding Publisher (WTP). This provides a means for other server programs, such as servlets, independent content-providing programs, or JavaServer Pages (JSPs) to invoke single transcoders directly. The JavaBean wrapper provides the transcoder with the same information about the system and request that it receives inside the WebSphere Transcoding Publisher transcoding server, so that it operates the same way in both contexts.

7.3.4 Choosing the right model

In the runtime environment, you can set up WTP either as a proxy or as a servlet filter. The primary question is whether to apply security to the front-end (in this case, how strict must the security be?) or to run the site without security.

If secure connections are not required for the front-end, then WTP can run either as a forward or a reverse proxy. If security is an essential component, and in most cases it is, then you should consider using the servlet filter option. There are trade-offs to using this model, but if security is an issue, it is always worth it.

Although security is a requirement, you can always put the transcoder proxy in-house together with the gateway. The main problem is that the Web intermediaries are wedged between the client and the content server. In this case, the content transmitted between the two parties cannot be encrypted

because the Web intermediary has to work on the content. If you put the gateway in-house together with the Web intermediary, then the client can connect to the gateway securely, and you have a better chance of securing the connection between the gateway and the content server, even if the transmitted content is not encrypted between the two ends.

Choosing between forward-proxy and reverse-proxy

- ▶ **Forward-proxy**

This is the easiest configuration; during development, this is also the most efficient. WTP can be used for accessing any HTTP server, which gives freedom to the client, but also puts a load on WTP.

The problem with this setup is that not all of the devices support configuring a proxy for accessing the Web. They cannot use WTP to pull up the correct information.

- ▶ **Reverse-proxy**

The advantage of this setup is that the client does not have to specify the proxy for accessing the Web, so every client can reach WTP. The clients are only able to access the preconfigured HTTP server, which, in terms of performance and loading, is much more efficient.

The problem with the reverse-proxy is that the URLs in the content have to be rewritten; so the hard-coded links (for example JavaScript) will not work within this setup.

In a runtime environment where security is not an issue, in terms of front-end content provisioning, the reverse-proxy provides better loading on WTP and gives more flexibility to the clients.

7.4 Where to find more information

- ▶ For more information about WebSphere Application Server, go to:
<http://www.ibm.com/software/webservers/appserv/>
- ▶ WebSphere Transcoding Publisher:
<http://www.ibm.com/software/webservers/transcoding/>
- ▶ WebSphere Voice Server:
http://www.ibm.com/software/speech/enterprise/ep_1.html
- ▶ IBM HTTP Server: <http://www.ibm.com/software/webservers/httpservers/>
- ▶ WebSphere Edge Server:
<http://www.ibm.com/software/webservers/edgeserver/>

- ▶ DB2 UDB: <http://www.ibm.com/software/data/db2/udb/>
- ▶ Secureway Firewall:
http://www.tivoli.com/products/index/secureway_firewall/



Part 3

Wireless Internet application: guidelines



Solution design

This chapter introduces the possible solutions for implementing mobile Web applications. The discussion will focus on design details, in preparation for the next two chapters, where application design and application development will take place.

First, we introduce the different pervasive computing modes to provide a general overview on communication; then the discussion will focus on synchronous mode.

The most essential part of the chapter describes the decision tree, which depicts the most important decisions and options for the appropriate solution during design time.

Finally, we briefly mention the distinctions between the design for a visual and a non-visual (voice) application.

8.1 The different modes of pervasive computing

In a pervasive world, mobile devices are a subset of all those devices in our daily life which have something to do with computing, from the intelligent Internet-enabled refrigerator to a PDA organizer or the massive “intelligent” buildings (where computers are controlling the whole building).

What is a computing mode?

Computing modes are a kind of representation at the highest level, where different devices are communicating through various networks using specific protocols.

These modes are not only possible options; they can be mixed in one solution, even in one application on one device.

Pervasive computing is beyond the scope of this book. We will only address the mobile computing scenarios.

Figure 8-1 and the following sections describe in some detail the possible modes for solution design.

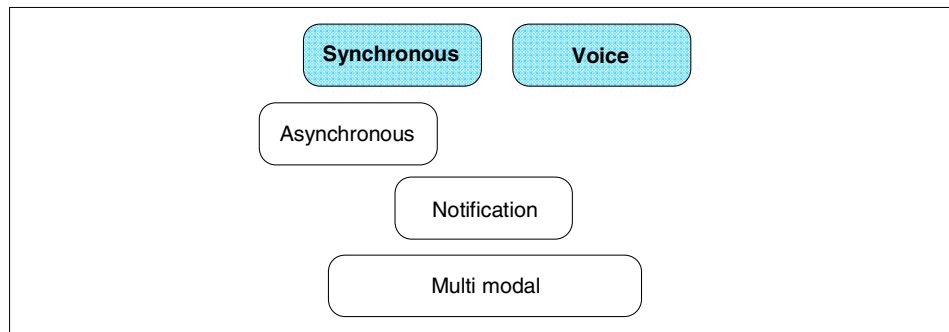


Figure 8-1 Computing modes

WebSphere Everyplace Access only deals with the communication modes shown within the shaded areas.

8.1.1 Synchronous

The synchronous mode is the best known mode, since it has received great attention from the public. In fact, one of the goals of the “wireless Web” is to extend e-business applications to mobile devices.

In our case, we made the following assumptions:

1. The computing mode is synchronous.
2. The end user is a thin client and the application runs on the server side.

Definition: *Synchronous communication* requires that each end of a communication exchange respond in turn without initiating a new communication.

For example, a WAP phone connects to a Web application. The communication between the WAP phone and the Web server is synchronous, as are the requests and the responses during the session.

8.1.2 Notification

There is a notification application at work when the client receives information without having to take the initiative; this is also called *push*. This communication mode requires the network to be always on, which enables applications to send messages to the device over the network.

We can make a distinction between:

- ▶ The PC-based notification applications, for example, in the context of a chat application, the notification received when a new person comes in.
- ▶ The notification of pagers and PDAs via the paging network.
- ▶ The Short Message Services (SMS) notification:
 - A widely used message service over GSM networks.
 - The WAP push is a more general architecture (also defined by the WAP forum) that allows to push information to WAP devices. This can be provided by SMS but also via other protocols such as SMTP (Simple Mail Transfer Protocol).
- ▶ Voice notification, an application that provides acknowledgement via a telephone using pre-recorded messages.

Note: In this book, we will not cover the notification communication mode.

8.1.3 Asynchronous

The best known asynchronous applications are mail and messaging applications. A common characteristic of these applications is that synchronization has to be done explicitly in order to get the data through.

Definition: *Asynchronous* mode pertains to communications that proceed independently of each other until one connection needs to interrupt another connection with a request.

Another form of asynchronous communication is one where synchronization, possibly wirelessly, occurs between devices and user workstations. In this case, a set of data or content is synchronized between the two parties. After the synchronization, both devices can work with the same set of data; which could be different until the next synchronization. For example, the address book in the PDA is synchronized with the address book in a desktop PC with a mailing application.

An emerging data format is the SyncML, a synchronization Markup Language providing a unified way of synchronizing data between the mobile device and the networked device (for example PDA and a networked PC). A concrete example of this is to have mail messages marked “read” on a PDA when the corresponding message has actually been viewed in the original application. More information about SyncML can be found at <http://www.syncml.org>.

Note: In this book we will not cover the asynchronous communication mode.

8.1.4 Voice

Voice applications access back-end applications using different methods. In our case, speech recognition or DTMF tones are used to process user inputs. On the other end, synthesized speech or pre-recorded audio snippets provide output facilities for information.

Voice applications require additional elements to be implemented into the architecture, since the end-user device is a phone. While the previous communication modes already had some kind of network connection to transfer data, human voice, as a form of interaction, requires a special gateway, which enables the transmission of voice over a data network (for example the Voice over IP gateway).

An emerging standard is VoiceXML, a markup language specifically designed for voice applications. Further details about the design of voice applications will be provided later in the book in section 8.4, “Non-visual design: voice” on page 116.

Note: VoiceXML is governed by the VoiceXML forum, which was founded in 1999 by IBM, Lucent, Motorola and AT&T. More information can be found at <http://www.voiceXML.org/>

8.1.5 Multimodal

The ability of a device or application to support several modes of interaction simultaneously is called “Multimodal interaction”.

In 17.4, “Multimodal applications” on page 346 we will introduce a multimodal application known as VIWO (Voice in WAP out), where the input is human voice and the output is a set of WML pages.

8.2 Decision tree

The decision tree in Figure 8-2 seeks to provide guidance in the design of an effective front-end solution for your mobile Web applications. Starting from a high-level description, the discussion will move down to the final solution by following a branch at each node on the tree. When multiple options are available at a node (decision point), questions and a set of answers regarding the solution will help to find the right path.

8.2.1 Device classes

The expression *device class* refers to an identifiable class of devices (or set of devices) that has a different realization of content, that is, a particular view of the document. These classes are easily separated based on the adopted markup language or a particular network or device profile, as shown in Table 8-1; the list is not exhaustive and it is provided for clarity of exposition.

Table 8-1 Device classes

Device	Markup language	Bandwidth
Desktop PC/Workstation	HTML	wide
Wireless PC	HTML	low
WAP devices	HDML, WML	low
i-mode devices	cHTML	low
PDA's	*ML	low
Phone using voice	VoiceXML	not applicable

8.2.2 The decision tree

We start from the top node, which has the label System. This also assumes that the solution designer has a clear understanding of the customer requirements, and has a plan about the application itself and about the functionalities.

The white label boxes are options; the designer can choose between the options on a level.

The shaded boxes are also options, but they represent technology options, which the solution will be based on.

Decision points raise questions and also provide a set of answers. They help the designer choose between options.

We use the tables (following Figure 8-2) together with the decision tree. Whenever we get to a decision point, we look up the table related to that decision point and go through the questions. Depending on our answers, and the answers we find below the questions, we make a decision to get to the next option. We do this until we get to a final solution.

The objective is not only to find the right solution for the applications, but also to learn about the possibilities and the applicable technologies while making the decisions.

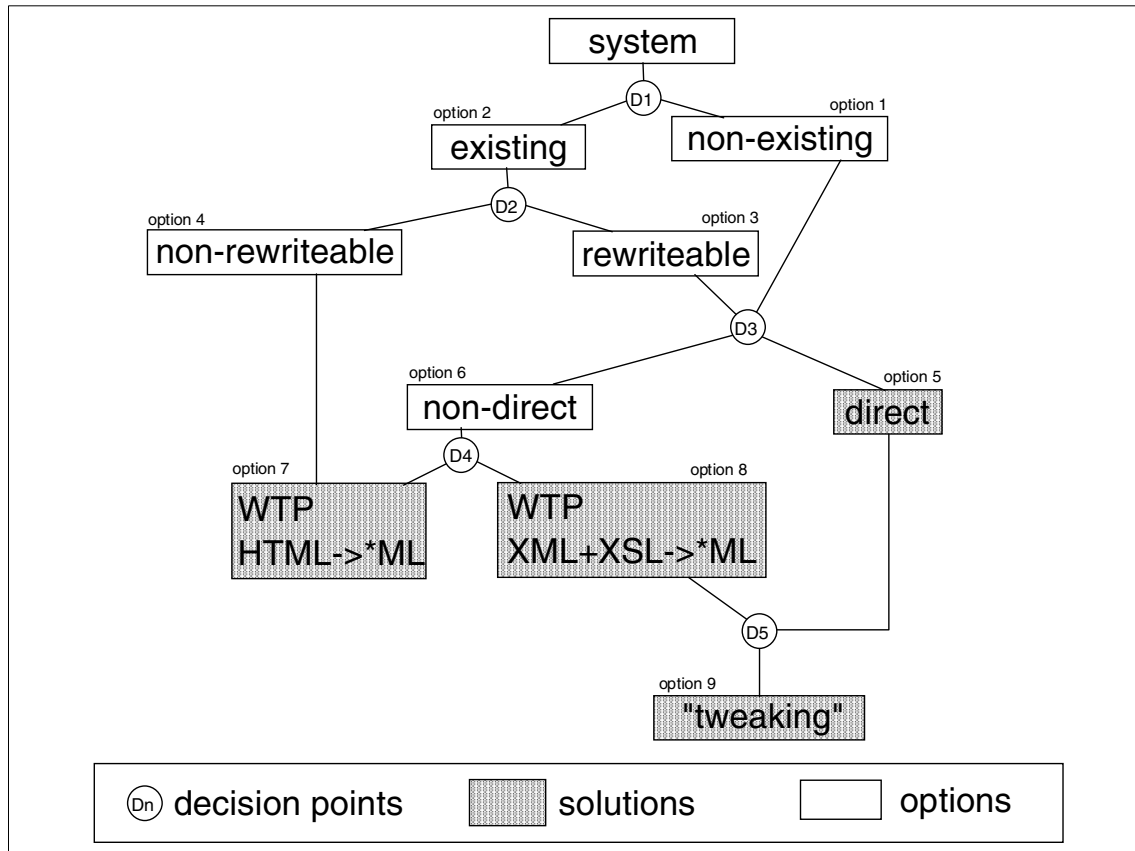


Figure 8-2 Decision tree

The following tables represent the decision points (nodes); each table includes the questions and answers for a decision point.

Table 8-2 Decision #1

Is the starting point an existing system or new development?	
Option #1: Non-existing	Option #2: Existing
The whole solution has to be developed.	The mobile solution has to suit to the existing system. The key is having all the documentation of the application to start with.

A multitier Web application separates the business and the presentation logic within the application logic.

Presentation logic handles the user interaction, controlling the flow and generating the content. The presentation logic maps and interprets the user interface to the business logic, which is the digital realization of all the business related data and functions.

It is important to understand that the presentation logic is not only generating the content, but also controls the user interaction flow. This makes the presentation logic capable of handling not only different kinds of content types but different kinds of interaction flows as well.

Table 8-3 Decision # 2

O #3: Rewriteable	O #4: Non-rewriteable
Is the presentation rewriteable?	
The presentation layer components (see definition above) are accessible in “write mode”, or adding new components. Source code required in most cases.	There is no way to redesign or rewrite the presentation. In this case a Web Intermediary (transcoder) is required to do the job.
An existing site can already support mobile devices. In this perspective, is there a case for modifying or adding presentation layer objects?	
The current site, equipped with new technologies, is not yet considered to be fully ready. In other words, there are benefits to modifying or adding presentation layer components. The presentation layer components are accessible at a reasonable cost.	The site is running fine, the content is presented in a well-formatted fashion, and there is no need to rewrite the logic. It is rather difficult to access the presentation layer components. The cost of modifying the existing presentation layer components is too high compared with the benefits.

The term *rewriteable* deserves some additional explanation. It means that the environment in which the presentation layer (typically servlets and JSPs) is developed is accessible for new development; this also means that one should at least be able to develop new servlets and new pages (either dynamic or static) in the existing application environment. We call these components *presentation layer components* (seeFigure 8-3).

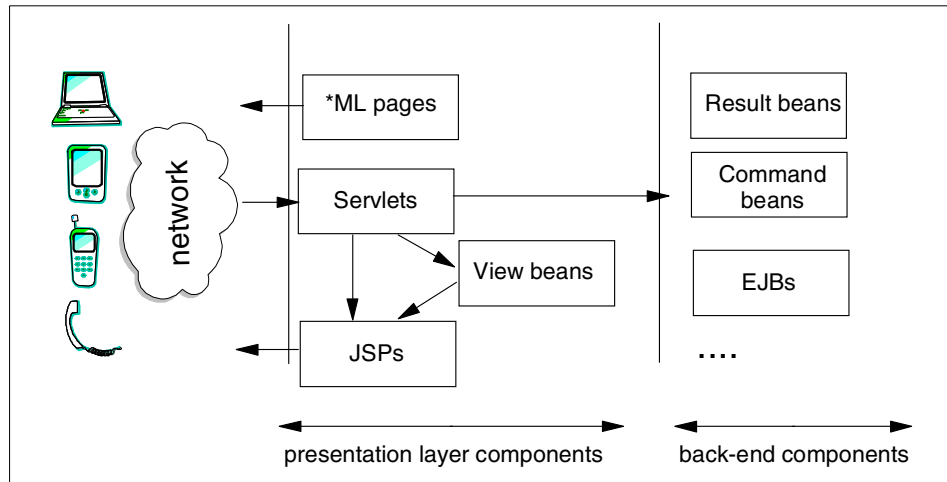


Figure 8-3 Presentation layer components

The presentation layer components are responsible for representing the content to the users. The presentation logic itself depends on the business logic, shown implemented, as in Figure 8-3, by the back-end components. The Web applications are responsible for producing the content (using the back-end components) and provide interaction with the user (via the presentation components).

The term *non-rewriteable*, as opposed to *rewriteable*, means that there is no write access at all to the presentation layer components (not even to add new ones) or that this access is too costly compared to the benefits. The only thing on which we can base the new content, if applicable, is the output content, for example HTML output. A typical example for this scenario is the situation of a service provider, which is usually not allowed to access the application of its customers.

Please note that *rewriteable* does not mean that there is access to the other components. We assume that the business logic behind the application is well designed so that there is no need to modify it; we only have to work on the presentation layer.

As shown in the decision tree, a non-existing application is also rewriteable, which is logical; application development starts from scratch.

This does not mean that we can restrict the changes to any of the pages and the JavaServer Pages only. There are many cases in which the devices are so different that access to other components, such as servlets and view beans, is affected. The capabilities of the different devices should be taken into account; in some cases, the existing presentation logic behind those pages could not match (or had difficulty matching) the other device classes.

For example:

- ▶ A sophisticated query with a desktop PC's browser can be very easy: it only requires filling out a couple of fields on a form. But the same query on a mobile phone using a WAP browser cause difficulties because of the limitations of screen, memory, and so on.
- ▶ Using image maps on a Web page to determine the position on a map can cause problems. Some of the small devices do not have pointing devices, not to mention voice access.

Table 8-4 Decision #3

O #5: Direct	O #6: Non-direct
Existing system or not?	
In the case of a non-existing system or if adding a new device class, the whole UI has to be written for each device class.	In an existing system, there are methods to leverage the existing content, and take advantage of it, to make the development easier.
Number of different devices accessing the application?	
The application is designed to be accessible by a maximum of two different types of devices.	The site should support a wide variety of device types (more than two types).
Number of pages that have to be written?	
The wireless site consists of only a couple of pages. For example, these may be only some short database queries that have to be written in JavaServer Pages.	The wireless site has many pages. For example, the whole site content should be accessible using wireless devices.
Is voice application also required?	
Implementing a VoiceXML application directly is possibly the easiest way.	There are solutions to re-use the existing content to produce VoiceXML, but they all need some workaround. Probably the easiest solution is to use XML and XSL to generate VoiceXML.

In the case of a non-direct solution, the following table helps to make the decision between two different approaches, where both use WebSphere Transcoding Publisher.

Table 8-5 Decision #4

O #7: WTP using transcoding	O #8: WTP using XML and XSL
Quality of the expected result?	
Using simple, easily transcodable pages can result in good quality, but usually a draft result is achieved on most devices.	This is used to reach the best presentation on the devices.
How much effort is invested in development?	
This solution requires the least amount of work.	Much more work is required to achieve the result. Different XSLs have to be written for each device.
Is voice implemented?	
Voice using transcoding is the easiest solution, and requires the least effort, compared with the XML solution.	VoiceXML pages are structured differently from other content, and applying Stylesheet could be difficult. The voice application has a different flow, and the pages are more fragmented.

XML and XSL

In the previous section, we made a distinction between using a transcoding node and using XML together with XSL. The two technologies are completely different, and require different application designs as well.

Transcoding works on the existing content, and only addresses the content. XML and XSL address the content from two perspectives, one being the data (XML) and the other the visualization (XSL).

Find more information about XML and XSL in 10.1.7, “XSLT” on page 161.

8.2.3 When to apply tweaking

Tweaking means that we want to adapt a given content to a device (to get the most out of it) using the same class of devices.

For example, we created content for WAP-enabled devices in WML. WAP devices have different characteristics. There are differences in screen size, and some might accept images, while others might not. These devices use the same content, but each of them might have a different profile.

In order to adapt the content to those devices, we could categorize the devices according to different preferences. For each category, we would then use different transcoders to adapt the specific features.

Table 8-6 Decision #5

O #7: Not worth using tweaking	O #8: Worth using tweaking
Complexity of the site? (frames, tables, content...)	
There are no frames; we have very simple tables (queries), straight, linear content, few images, no client-side scripts (for example JavaScript)	The site uses many frames, sophisticated, nested tables, many images, different topics on one page, client-side scripts.
How do we reach the maximum capabilities on most devices?	
A good, but not perfect result can be achieved with WTP solutions. The result is close to the expected content.	We get the most out of the device capabilities. A sharper, more accurate result is achieved on different devices.

Technically, *tweaking* refers to the application transcoding to the final (device-specific) content. Transcoding is done by using different device profiles or deploying customized transcoders, that is, developed MEGs or MEGlets.

Tweaking can work on directly coded content, or on content generated using XML data and XSL.

Using device profiles

Defining device profiles is the easiest way to implement some form of tweaking. WTP has the following classes of features for a device, which are applicable to a profile as a parameter:

- ▶ Output type preferences
- ▶ Image preferences
- ▶ HTML browser preferences
- ▶ Java/XML preferences
- ▶ Fragmentation preferences
- ▶ Device-specific preferences

The developer can set up the profiles using the Profile Builder; to create a new profile, refer to “Creating a new device profile” on page 331, or modify an existing one.

A good example of tweaking is when the user accesses a site with a wireless PC (laptop with wireless adapter) and the low bandwidth requires cutting down the content. The site already has simplified HTML content for wireless PCs, but the user only needs text based documents without pictures, so the client is the command window browser: Lynx. With tweaking, the transcoder can easily remove the images from the document and the JavaScripts.

8.2.4 Custom solutions

WTP supports transcoders written either using Web Intermediaries (WBI) v4.5 API or Java Servlet v2.1 API. They are referred to as MEGs or MEGlets, depending on the applied API.

Custom solutions give the highest flexibility to the developer with WebSphere Transcoding Publisher. Developers are encouraged to develop their own MEGs or MEGlets. Of course, more work is required, but this will give the best results using transcoding.

It is also possible to develop a totally new transcoder for WebSphere Transcoding Publisher, which can transcode the content more precisely to the required device or into the required format. For example:

- ▶ Document transcoding into PDF (Portable Document Format)
- ▶ Video or audio stream transcoding

The possibilities with custom solutions are endless; it is up to the solution designer to find out if the solution requires a new MEG, MEGlet or transcoder, or if it can be built upon the existing ones.

Refer to *Adding Transcoders to Transcoding Publisher* in the online development guide, and to Section 15.2.3, “Text clipping” on page 284 for more information.

For more information about WTP custom solutions, refer to the IBM Redbook *New Capabilities in IBM WebSphere Transcoding Publisher Version 3.5: Extending Web Applications to the Pervasive World*, SG24-6233.

8.3 Visual design

The visual design of a mobile application is well documented; you can find documents, papers, and books about designing visual user interfaces for Web applications.

Refer to the IBM Redbook *Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*, SG24-5755 for more information.

The IBM Redbook *The Front of IBM WebSphere Building e-business User Interfaces*, SG24-5488, is also a relevant source of information.

In this book, we will also cover the aspects of visual design in Chapter 9, “Application design” on page 121.

We will see that designing a mobile Web application poses more challenges when one wants to implement mobile services as well, and the level of difficulty increases if voice is included.

Our visual mobile applications follow the well-known Model-View-Controller pattern for front-end application design. Our applications also leverage from the Web Intermediary technologies to make the transition from content less difficult. To find out more about the visual application design, please refer to Chapter 9, “Application design” on page 121.

8.4 Non-visual design: voice

As we mentioned before, a voice user interface provides input and output to the application via speech recognition and speech synthesis; it is optionally completed with other means of input (for example DTMF) and output (for example pieces of prerecorded audio).

VoiceXML is an XML-based markup language for creating distributed voice applications in which the input and/or output are through a spoken (rather than graphical) user interface. Users can access deployed VoiceXML applications anytime, anywhere, from any telephony-capable device.

8.4.1 Challenges of voice versus visual applications

Most dynamic content that is being considered for a voice application has either already been rendered visually, such as through HTML pages, or has no interface representation at all, such as a collection of XML documents or data from a database. The first thing to understand is that a voice interface implies a completely different set of rules and expectations for the end user than do traditional visual interfaces. For instance, visual elements which may make the document easy to use (navigational bars, “twisties” or expand/collapse nodes) or visually appealing (embedded graphics, animation or color scheme) may not work at all in a voice application. Also, visual elements which are not relevant to the subject of the document, such as advertisements or company banners, interfere with the end user’s ability to get to the core content quickly and easily.

The elements of a good interface are the same, regardless of whether it is viewed or listened to: it must be clean, intuitive, efficient, and productive.

Note: *“Still, high-speed networks will not improve the experience all that much. If users are wasting time scrolling through Web pages laid out for desktop viewing and trying to locate the “needle” of information they are seeking in an enormous “haystack” of content, they are not working productively. The whole point of providing wireless mobile access to mission-critical applications is lost in the execution” - from the document Wireless Enterprise Applications for Mobile Information Management - Development Options and Business Decisions, Palm Inc.*

Dynamic data which is in some other, non-visual representation, such as XML, saves you from these visual elements. Even so, an interface must be built for this data. Understanding the capabilities of the voice browsing system and the user's expectations for the interface is equivalent to building a usable voice interface, especially if your background is in developing visual Web applications.

The following paragraphs explore in more detail some of the most important differences between a visual application and a voice application.

Mental load

One of the most important differences between a visual and audio interface is the amount of data which can be represented and processed. Web (HTML) pages often offer a vast amount of information, but users have been trained to use common navigational elements to retrieve the desired data. The page sits in front of the user for as long as it is needed by the user to analyze the information. It is also possible to move back quickly through previous pages to review the content.

Contrast this with a voice application. Users typically access such an application through a phone, meaning that once they have heard it, it is gone, and is usually difficult or time consuming to get back. Information in a voice application is transient just like any human-to-human spoken conversation.

Voice application designers need to limit the amount of information presented to the user, so as to avoid overpowering short term memory. Remember, there is no document to view and no back button. For example, where a large list of options may be presented in a visual form, common usability guidelines for voice dictates that approximately 7 (plus or minus two) options in a list is best.

Navigation and error recovery

We are all familiar with navigating HTML pages. The common use of navigation bars, self-describing hyperlinks, and the Forward and Back buttons make it easy to get where you need to go and understand where you are. If there is an error, the Back button gets you back to a safe location where you can try again.

In voice applications, it is not obvious to the user where he/she is in the application. For navigation, you must provide a consistent approach to traversing menus and pages of data, such as always offering a link back to the main menu first, followed by links to content, then by a link to exit the application. The link wording needs to be concise (one to three words) yet descriptive and representative of the current area of the application. In visual Web pages, you can intersperse links to distantly related material and not cause the user to lose focus of the subject matter. In a voice application, the user can quickly lose track of what he/she has heard and forget where he/she is, forcing him/her to go through the rest of the links to find the one that repeats the options or returns to the previous menu. Exercise restraint in how much data is presented to the user and ensure that all of the data is closely related in subject.

Error recovery is often simpler in a visual application. You recognize that you are typing something incorrectly and fix it before it is submitted, or resubmit input by going back and trying again. You click the wrong link, so you use the Back button to try another one. With voice applications, you not only have to contend with the possibility of providing the wrong input, but with the system's ability to understand what you say.

Voice applications need to be designed in such a way as to make it easy for the user to retry an input, or limit the user's choices of input to minimize the risk of error. In a visual representation, data is randomly accessed, and input can be given at almost any time. In voice applications, data is read sequentially and input (voice recognition) can only be processed at certain times. *Barging-in*, which refers to a voice application's allowing the user to interrupt the data being read with a request, should be considered. Controlling when voice input is allowed in general is a function of the voice browser, not VoiceXML.

User characteristics

More than in visual applications, the user has to be familiar with the tasks proposed, since there are no elements illustrating them. In that respect also, one has to recognize the typical vocabulary to describe tasks, which can vary from region to region, even for the same language.

Wording and input processing

Modern speech recognition processing is built around recognizing known words and phrases. It is easier for software to distinguish between spoken phrases made up of several phonemes and dynamic phonetic characteristics between spoken letters, such as when spelling a word. It is very difficult for voice recognition software to tell the difference between P and B, especially if the user is on a cell phone in a public place or in a moving vehicle, but it is easier to recognize a specific phrase.

When a voice application requires text input, that input must be limited to a well-understood and bounded grammar, or specification of valid input. When the grammar can be limited to a (short) list of words from which to choose, such as through a navigation menu or selection list, then the voice browser needs only to concentrate on understanding those set inputs. The grammar must either be generated as part of the page, or dynamically generated as part of the application logic, such as from a database of grammars.

Requiring the user to spell an entry, such as a name, account number, or restaurant name, can cause unrecognized input or erroneous results. The possible exception is numeric input, where the grammar is limited to the numbers 0 through 9. These ten characters are more or less phonetically unique and most voice recognition systems have less difficulty differentiating between them. For this very reason, many voice applications owners will issue numeric IDs or personal identification numbers (PINs) to their users to authenticate themselves to the voice application system. Since most billing and banking account numbers are numeric, voice-based financial applications have become widespread.

8.4.2 Voice applied to the decision tree

As mentioned in the previous section, the differences between visual and non-visual applications are such that in most situations, voice applications can be directly coded in VoiceXML using the IBM WebSphere Voice Server SDK or an equivalent product. However, WTP may be a more productive solution, particularly when it is used in combination with a native VoiceXML application, if any of the following situations occur:

- ▶ The user already has an existing visual Web site (in HTML) or data in XML.
- ▶ The user wants to present the same basic data on multiple output devices, one of which is a telephone.

In this case, which corresponds to the non-direct approach (see option 6 in Figure 8-2 on page 109), you should refer to the sections below, which discuss in more detail the following usage scenarios:

1. Converting XML data directly to VoiceXML through the application of XSL stylesheets (see option 8 in Figure 8-2),
2. Transcoding (see option 7 in Figure 8-2) by either:
 - a. converting a Web site into a VoiceXML application, or
 - b. mining Web content for reuse in a VoiceXML application.

8.4.3 Other aspects of voice related to solution design

Currently, applications with embedded voice synthesis, which may be used to read transcoded HTML content, sound very artificial. Until voice synthesis becomes more natural (hopefully in the near future), most companies will still prefer voice applications written totally in VoiceXML and incorporating prerecorded human voice cues. Only dynamically generated content needs to be read synthetically, but, as previously described, it can still make use of prerecorded voice snippets.

Deciding whether to use voice snippets instead of synthesized voice is important. If voice snippets are to be used, where should this happen? The following considerations should be taken into account:

- ▶ Voice snippets sound much better, and provide a better user experience.
- ▶ Maintaining the pre-recorded voice parts requires some organization. The same voice should be used throughout. However, maintaining a voice site is probably the cheapest solution of all.
- ▶ Do not forget to put the identical text for the voice snippets into the VoiceXML, so the text browser can also handle that part.
- ▶ Prerecorded audio takes up much more disk space, depending on its quality, than the synthesized one. On the other hand, prerecorded audio requires less processing from the server.



Application design

This chapter addresses application design, which is a very important issue in software development. The main focus point will be an architecture that can deal with the new challenges in mobile application. Time is changing as well as the demands on applications. New technologies are emerging every day and good Web applications must be able to adapt to these new technologies very quickly. Speed is a key to the success of an application. Who is the first on the market with that new technology? Also, the flexibility to adapt to these technologies will be a crucial asset.

There are Web applications implemented for offering services in the intranet of a company and there are Web applications implemented for offering services for all users on the Internet. Obviously, these two types of applications are of a different nature and also intended for different types of users, but they have gone through the same consistent application design. The functions and features that each application should support are implemented, and the flexibility for easy maintenance and enhancement is present.

The market is irrevocably changing and mobile applications regularly show as many new features as the Web itself now. The technology of mobile applications is emerging steadily and many already established applications on the Web will need to be enhanced to adapt to this new technology. There are many issues to

consider in order to design an application with a high degree of flexibility and enhance an application having mobile features. Some examples are object technology, design patterns, object reuse, and so on. All these issues will be discussed in the forthcoming sections.

9.1 Web application

Normal Web applications include some interaction with the user. An initial request is triggered by the user when the URL is typed into the browser. Most applications will reply with a simple welcome page to the user. Then the user has to input some information to the page; for example, if it is a login page, the user will input a user name and password to gain access to that site. The request is passed back to the server, which checks and validates the inputs made by the user, then responds with a dynamically generated result page.

This scenario is a very simple one, but there is actually a well-thought design behind it. Commonly, the data flow of an application is concentrated on the server side and little or none of the business logic is revealed on the client side. The reason for that is to avoid distributing the logic. The server should be in control of all the logic and every change can then be easily realized on the server side. Web application architecture often implements a design pattern called *Model-View-Controller* (MVC). Section 9.1.1, “Model-View-Controller” on page 126 will discuss the MVC pattern in more detail.

Figure 9-1 depicts a sample Web application that provides a high degree of flexibility at a low cost.

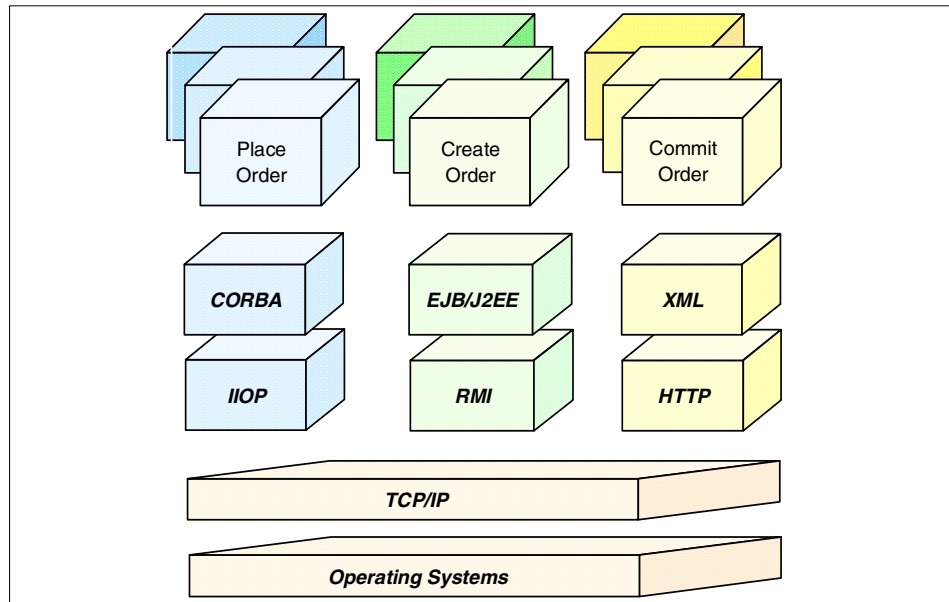


Figure 9-1 A sample Web application (1)

An implemented Web application based on this pattern can have many additional features.

- ▶ All components are defined logically.
- ▶ Each one is assigned one function.
- ▶ A defined interface controls the data flow between components. Hence, only the interface has to be changed in case of an extension.
- ▶ Each component can be implemented as another pattern as well. This leads to a higher degree of flexibility and reusability of each component.
- ▶ Cost is reduced.
- ▶ Quality is higher.
- ▶ Flexibility is high.
- ▶ Reusability is high.

Not only are the functions reusable, but the design, patterns and framework are as well. The application is divided into an application service layer and a server-side business layer.

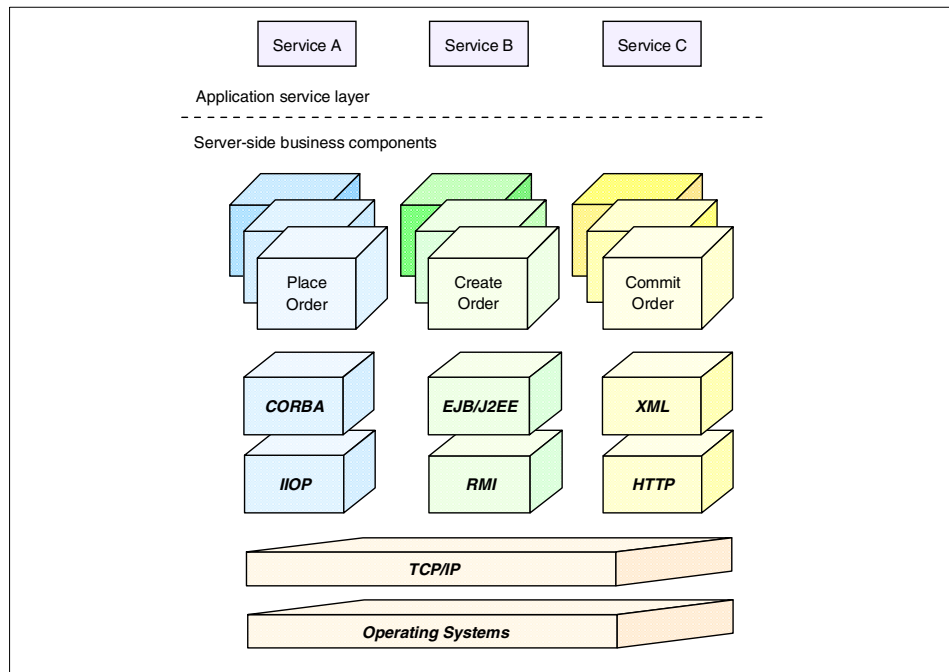


Figure 9-2 A sample Web application (2)

► Application layer

The application service layer will take care of all components supported by the application. There can be as many components as needed. Each function is encapsulated in its own component. Any changes required to the service result in the modification of this function without affecting others.

► Server-side business layer

The Server-side business layer handles all functionalities and business logic required to process a service. An advantage of this is that every process required by a specific logic is hidden from the application service layer. Only a known interface is used to communicate between these two layers. Therefore, the interface is the main connection for the application service layer to pass any information to the server-side business layer.

Of course, the same interface is the only connection for the server-side business layer to send information back to the application. Hence, the interface must also be carefully maintained. Through the separation of the application layer and the business layer, services can be created rapidly and customer feedback is received more quickly. Also, it is easy to adapt to any future market changes. Although the application is so well-defined, the server-side business layer and its components are only one part of a complete application.

A server-side component can be seen as a component which is small enough to reuse and maintain a function. On the other hand, the whole server-side business layer should be large enough to deliver, deploy and support the whole business logic. Through this separation, the whole server-side business layer can be delivered as a self-contained package. To do this, a standardized interface is the only means that should be used. This makes this layer easier to build for customization, composition and collaboration with other components. For example, if an already established system has to be extended with new functionalities, but the whole system cannot be modified, then the customization, composition and collaboration feature comes into effect.

This key issue will remain important through the whole chapter. In the upcoming chapters, this issue will be further described, but first a pattern will be discussed that is widely used to support such an implementation.

9.1.1 Model-View-Controller

The Model-View-Controller paradigm is a way of breaking up an application into three parts: the Model, the View, and the Controller. Each component handles a specific type of logic. The primary objective of MVC when it was first developed was to map input, processing and output into the GUI realm. For example, the user clicks a button, an event is triggered and processed, and the output is shown in the window.

Further details about this model can be found in E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*.

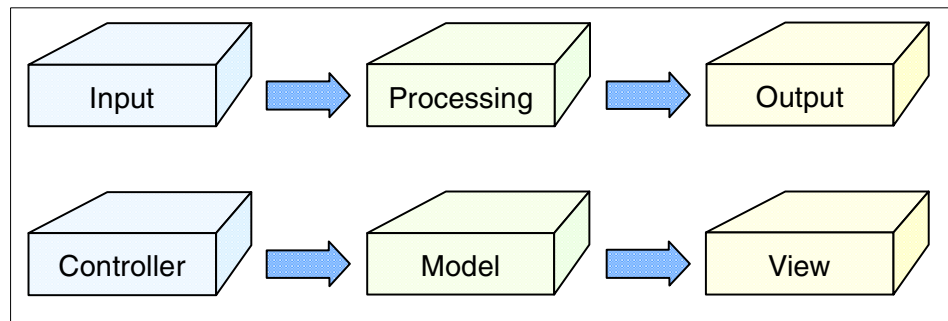


Figure 9-3 Input/output processing

Actually, this is a simple design, but a powerful one. The application development very much depends on input and output. How to process a specific event depends on the underlying business logic. In the GUI realm, the output of an event depends on the operating system and on the Windows system, but if we apply this consideration to a general application, then the output depends very much on the input.

In a general application, the input can be a specific hardware device or some specific data that identifies a client. For example, in the mobile realm, a piece of hardware can be a wireless phone or a PDA. These are different hardware devices that need differently supported presentations. The same output has a different presentation on a wireless phone and on a PDA. The important fact to remember is that an application should be client-independent.

An application that is only designed to support a specific type of client can be considered a “bad” application. For example, consider information that is requested by a mobile phone and by a PDA. The same information is being requested by the two different hardware devices. Why should applications be differently designed and implemented if they use the same business logic? Having two applications serving the same information results in duplicate components, duplicate maintenance and duplicate costs.

How to avoid this problem will be explained later. First, the features and responsibilities of this methodology will be discussed.

► **Controller**

The Controller is the means by which the user interacts with the application. The user makes a request to the server. The Controller must first validate the request and the current session of the user. The user's input data is then made available by the Controller to the Model. For example, parameters are mapped to an independent data format of the Model. The business logic module is then invoked with the parameters as input arguments. This way of invoking the business logic might also be implemented as a pattern.

► **Model**

The Model defines a means to represent the business logic of the application. Model objects represent the data objects of the application. An MVC application should have all of its data encapsulated in Model objects. Hence, data that is part of the persistent state of the application can be stored on a local file system or database server. This is very helpful in initializing the application when the data is loaded. A Model object contains only information presented by the View and methods that allow the Controller to change information in response to user interaction. A Model object should not deal with how data is presented; this also means that a Model object has no connection with the user interface.

For example, if a Model object represents a person and his/her date of birth, then there should be no information about the birth date formatting in the Model object, because birth date formatting is actually the presentation of a piece of information, which has nothing to do with the business logic. In general, a Model object should have nothing to do with interface and presentation issues.

► View

The View provides the interface communication with the client. All data which is a response to a request from the user will be passed to a View object. It is then the responsibility of a View object to decide which part of the Model object should be displayed. A View object can display all of the responding data or part of it. The data can also be displayed in a different language according to the user's language settings. The main purpose of a View object is only to present the data correctly to the user.

Developers should design the classes according to these three components. The MVC makes it easier to maintain an application, if the look and feel of this application is going to be changed frequently. In that case, only the View component needs to be changed without interfering with the business logic. Also, if a Web site needs to maintain different languages, then different interfaces can be used without interfering with the business logic. For more information, see A. Leff and J. Rayfried, *IBM Research Report Web-Application Development Using Model/View/Controller Design Pattern*.

An example of a Web application built according to this model will be introduced next.

9.1.2 Sample Web application

A simple Web application architected with the MVC design pattern (see Section 9.1.1, "Model-View-Controller" on page 126) can be, for example, a Java-based approach running on an application server.

- The servlet is implemented as the Controller.
- Enterprise JavaBeans is implemented as the Model.
- The JavaServer Pages are implemented as the View.

This approach uses servlets in front as the Controller to receive all input from the user. Some kind of validation of the input is done inside this module and passed to the Model. In this case, Enterprise JavaBeans functions as the Model to implement the business logic behind it. Important information is made persistent and stored in a database. A response is created to the user's initial request and the response data is passed to the View, which is implemented using JavaServer Pages. In this module, all business independent data is formatted and prepared for presentation. Depending on the input, some data, or all of it, is presented to the user. Here, the View can decide which language to use to suit the user's preferences.

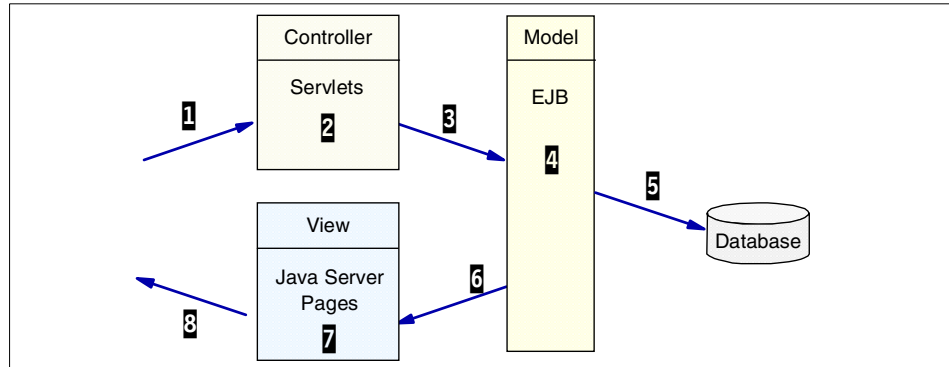


Figure 9-4 Data flow of a Web application

Here is a description of the main steps implemented to support this data flow (see Figure 9-4):

1. The servlets receive the input data.
2. The servlets apply validation to the data and format them to application-independent format for Enterprise JavaBeans.
3. The data is passed to the Enterprise JavaBeans.
4. The business logic is applied to the data by Enterprise JavaBeans.
5. Important data is serialized and stored in the database.
6. Response data is prepared for the View.
7. The View takes the data and apply the presentation of the data through JavaServer Pages.
8. The formatted data is sent back to the client.

This is a very simple approach, but the necessary functions and components are divided logically according to the MVC design. All features and advantages of an MVC design pattern are therefore adapted to this approach.

A Web application with this type of architecture may have the flexibility to be enabled for mobile access. However, when this kind of traditional Web application was built, no one had ever thought about a mobile environment. What does a Web application need to support a mobile client? What does the mobile application look like? All components used in this Web application are limited to clients using desktop HTML browsers. To enable the application for other types of clients, additional components are needed to the ones introduced in Figure 9-4 on page 129.

Consider that the mobile world includes many different mobile devices. Each of them has different underlying standards, interfaces and presentations. To accommodate all of these features, some new components must be introduced. For example, mobile devices use other markup languages than a PC's desktop browser, such as VoiceXML for Voice Server, WML for WAP, XML or cHTML for i-mode. Please refer to Figure 10 on page 153 for more details on these markup languages. How can an application be extended in order to fulfil these requirements?

9.2 Extending a Web application to a mobile application

Recently, the usage of mobile devices has increased. Whether it is a PDA, a mobile phone or a wireless laptop, more and more people are using mobile devices to receive information or trigger an action. Nowadays, people can also use their mobile phones to buy goods. The technology of mobile devices has come to a point where it offers the capabilities to make business attractive and encourages consumers to invest more in mobile services. As mobile devices become more and more powerful, their functionality is improving as well. The challenge is to develop new mobile applications or set up new mobile infrastructures for new businesses or services.

As the Internet grows, new technologies have emerged to help build Web applications. Early on, there were several CGI modules, then, as the Java programming language emerged, servlets became the counterpart to CGI modules in this approach. Later, to support the presentation layer, the JavaServer Pages technology was introduced. At the time, these technologies were designed for use in a Web application and not in a mobile application. Therefore, easy adaptation of these technologies to a mobile application cannot automatically be taken for granted. Mobile applications must handle several different markup languages to support different type of devices. Recall that different mobile devices communicate differently.

Mobile devices speak a different markup language than normal Web application do. Mobile applications must not only understand these markup languages, but also respond in the same markup language as the client. Also, the output of a mobile device is limited to its size and dimensions. Nowadays, widely used markup languages are XML, HTML, WML, cHTML, VoiceXML, and so on.

9.2.1 Extending the architecture

When talking about extending a Web application, two scenarios must be considered:

1. A Web application is already established and needs some kind of migration.
2. A mobile application is being built from scratch.

The first scenario is actually the more complicated one, because the Web application might be in production already and delivering services to thousands of users. All the data presented to users exists already either in static or dynamic HTML pages. The customer does not want a significant overhaul of the design or structure of the Web application. The main concern of the customer will be to keep costs to a minimum and time spent to a minimum. Therefore, a temporarily shutdown of the Web application is not affordable. Because of this, many developers will work to make their application highly flexible and reusable.

The second scenario is one in which the developer can build a whole new system from scratch. The developer can, and should, take all the mobile application's additional features into consideration. If that is to be feasible, the developer needs a consistent process to create a mobile application which is highly flexible, inexpensive to implement and easy to extend.

One way to overcome the challenges faced within the two scenarios is to put some new components in front of the Controller module and View module. These new components must have the following abilities:

1. The ability to receive inputs from clients of any type, such as a normal Controller in an MVC design pattern. Input devices must be examined and its type determined by the application, input data need to be translated to a format that a Web application can understand. Specific information pertaining to any hardware devices should be stored temporarily in order to have it available for the presentation.
2. The ability to present the data to the user, independently of the client's nature. Device-specific values should be used to format the response data to suit the client's device.
3. The ability to modify the response data that is provided by the Web application and to insert it back into the stream. For example, if the client is a WAP client and cannot display large pictures, then it should be removed.
4. The ability to format to the device-specific language. For example, if the client is a WAP client, then the response data has to be formatted as needed.

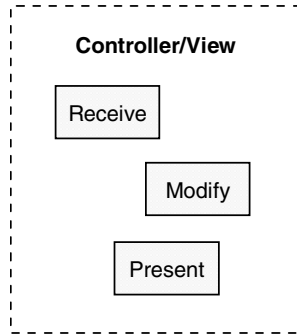


Figure 9-5 A new component

Each new component functions as a *Web Intermediary*. This is because it sits between the client and the Web server (see Figure 9-5) and has the capabilities to alter the data, as described above.

This component must also have the flexibility to run as a stand-alone server or a servlet in a servlet engine. This gives more flexibility to an already established application in the way it will integrate this component.

The next section will present a whole new architecture, including the new Web Intermediaries. We will also address how this component can be integrated into the application.

Note: The technical term *Web Intermediary* is not an industry standard term, but it is widely used. In this book, we will use this term, because it is very descriptive and more general than *transcoder* or even the product name IBM WebSphere Transcoding Publisher.

9.3 Mobile architecture

Section 9.1, “Web application” on page 122 has already revealed the problems and concerns with the process of a service, which could lead to high costs and a high degree of inflexibility for a Web and mobile application. Section 9.1.1, “Model-View-Controller” on page 126 and Section 9.1.2, “Sample Web application” on page 128 have described a Web application architected with the MVC pattern. This section will discuss the integration of a new component into these systems without the need for significant modification to a current system, all the while keeping the advantages of MVC patterns.

This means that the new architecture will still have the logical divisions of Model, View and Controller components, but will have the advantage of being client-independent.

The term *independent*, in this situation, means that:

1. Requests are accepted regardless of their markup languages: this is a basic requirement for supporting multiple clients.
2. The content of the data is presented in a uniform way: no matter the form of the incoming request, the content must be representable first.
3. Data can be modified, but only if the content of the data is understood.
4. The client's device type can be determined: with this ability, the response can be created to suit client's device.
5. The content can be translated to another format: this is the main requirement to support multiple clients.
6. Some parts of the content can be cut out: different client devices have different resources to handle a response; a mobile phone, for example, cannot handle a huge load of data as a normal desktop browser can.

If the new mobile architecture is extended with these features, suddenly the architecture becomes more powerful. How can this be implemented?

Delivering all these features without changing the current system means that a new component must be introduced and placed between the request and the Controller components. This component must be able to accept the request and forward it to the server. A proxy will fit very well into this situation. Additionally, this component can also monitor and edit the incoming data, and even generate new data and replace it with the incoming data. With these capabilities, this new component has the power to achieve the above listed goals. The problem of how to customize existing content to a new application and deliver it is not new. The introduction of Extensible Markup Language (XML) has created a new opportunity to address this problem. However, with the growth of mobile devices, new constraints on this solution have also emerged. Somehow, a better solution must be thought out and defined.

These features are already defined as a concept called Web Intermediaries (WBI, pronounced “webby”), which will be introduced in detail in the following chapter. First, the concept of Web Intermediaries and its underlying techniques will be discussed, then the whole new architecture will be presented, and lastly, an example of an implementation will be provided.

9.3.1 Web Intermediaries

Web Intermediaries are a kind of entity that lies along an information stream. Once positioned on a stream, data can be modified, monitored or even generated and inserted back into the stream. This concept allows incoming or outgoing data of a system to be altered. Intermediary-based components are extremely useful if the data producer or server and the data consumer (a browser client or a wireless device) cannot be altered. The original data flow is not disturbed by these Web Intermediaries. On the contrary, all data is passed back to the flow.

The fundamental communication mechanism of the Web is the Hypertext Transfer Protocol, or HTTP. HTTP is a stateless request-response protocol. A client connects to the server, a request is submitted to the server, which responds with the requested data and closes the connection. This is a direct connection of a client to a server. Most Web sites are secured by a firewall and a proxy. In the case of a proxy, the request is sent to the proxy, which retrieves the requested data from the server and returns it back to the client. Web Intermediaries can function as a proxy to observe and alter data on the flow.

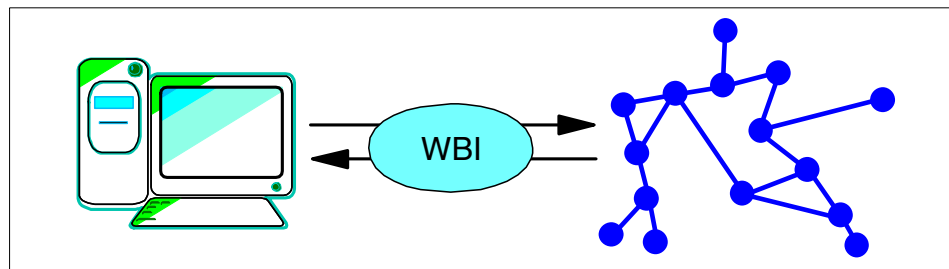


Figure 9-6 WBI is between the client and the application

Web Intermediaries architecture

Web Intermediaries are composed of three kind of agents:

► **Monitor agents**

The monitor agent receives a copy of the request-response stream, but is not able to alter any data in that stream. Although a monitor agent cannot participate in the processing of the data, it can be used to perform related and supporting tasks. For example, a monitor can keep track of which pages a user has visited.

► Editor agents

The editor agent intercepts the communication stream, receiving the information of that stream and delivering a modified version of it. Edited data can be created from the incoming stream or any other available information source. Editors can be placed either in the request part or the response part of a stream. One fundamental role of the request editor is to transform incoming data from one format to another. For example, if the body of incoming data is in WML, then the request editor can transform it into HTML and place it back into the stream.

► Generator agents

The generator agent accepts the request either directly from the stream or as output from a request editor. It has the ability to convert a request into a response. It passes a request to the Web server, retrieves the response and returns it to the client. In general, it performs the job of an HTTP proxy.

These three agents are also referred to as MEG (Monitor, Editor, Generator). MEG represents the building block of Web Intermediary plug-ins. Each agent has a defined priority and circumstance when an agent is triggered. The whole flow of data is controlled by a set of rules and, under a certain condition, an agent is triggered (see Figure 9-7).

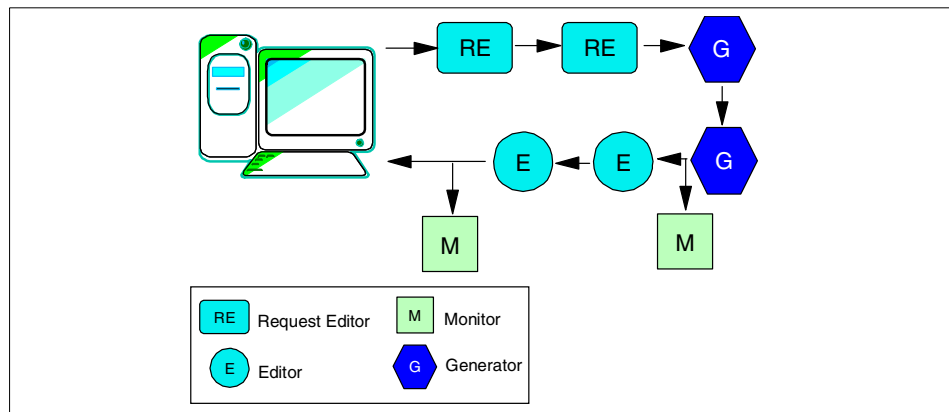


Figure 9-7 A transaction flows through a series of WBI MEGs

Depending on the function of an agent, fields such as request header, response header, content-type, user-agent or even the body of the data can be altered accordingly. The content of the body can also be transformed from one markup language to another. This is what makes Web Intermediaries so powerful and useful for integration into a Web application, so that the features may be enhanced to the level of a wireless application.

Furthermore, in the Web Intermediaries environment, it is also possible to register as many MEGs as needed. Depending on the number of devices an application supports, there can be a MEG registered for each device. This allows an application to become multiple device independent. Such a scenario is shown in Figure 9-8.

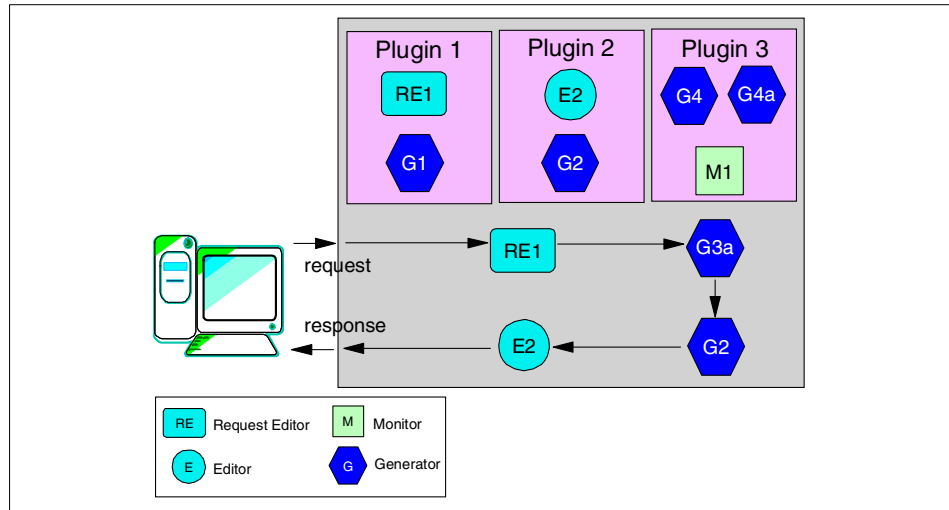


Figure 9-8 Multiple MEG plug-ins in WBI

With the integration of Web Intermediaries, an application does not need to contend with a specific type of client. Furthermore, the other components of the application, such as the Web server, application server or database server, do not need to be modified or adapted to this new component. Web Intermediaries will be simply put in front of the Web server running as a proxy or as a servlet inside an application server. This way of adding a new feature to an already existing or new application will not harm the system, but provides a highly customizable environment.

We will now look at what the architecture of an application looks like after the integration of Web Intermediaries.

9.3.2 A mobile application

We will now discuss the new mobile architecture integrated with the Web Intermediaries. As previously discussed, the integration of Web Intermediaries will enhance the Web application to the level of a mobile application. Recall that a recommended design pattern for an application is the MVC design pattern (see Section 9.1.1, “Model-View-Controller” on page 126 for more information). How does the integration of WBI affect the architecture? Some modifications will be expected, but to what extent?

The next scenario concerning the architecture will use Web Intermediaries as a proxy, because this is the best way to extend an application and does not affect the other components in the system. Later on, the proxy can be easily extended to load-balance the traffic. A proxy provides a single access point to all clients, and if the traffic reaches a peak that cannot be handled by a single proxy, then an extension through a second proxy will relieve the load. It is also easier to implement the high availability of a proxy.

Data flow with Web Intermediaries

In the MVC design pattern, the Controller module has the responsibility of receiving input from the clients. Presentation of the response data to the client is provided by the View module. Now, with the integration of the Web Intermediaries, the responsibility and functionalities of the Controller and View modules have shifted. Web Intermediaries have the power to monitor, edit and generate the data stream as it arrives. As stated above, the easiest way to run a Web Intermediary is to implement it as a proxy. Keeping this in mind, the whole mobile application has a new data flow (see Figure 9-9).

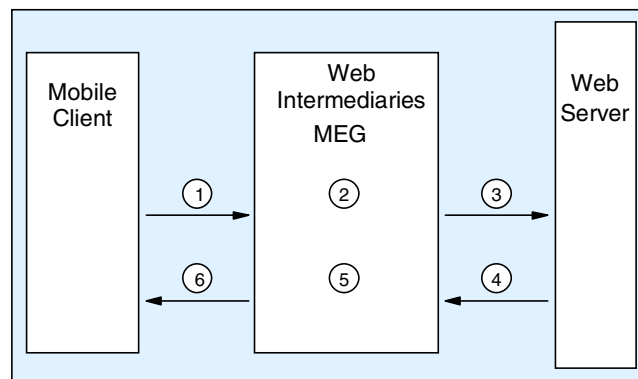


Figure 9-9 A mobile data flow

1. A client makes a request to the Web site.
2. The proxy (Web Intermediary) receives the request. Depending on the request header and the data stream, the body of the data may be transcoded.
3. Modified request headers and transcoded data are forwarded to the Web server.
4. The Web server responds to the proxy with the requested data.
5. The proxy modifies the response header as appropriate and transcodes the data back to the client's markup language for viewing.
6. The transcoded data is sent back to the client.

Web Intermediaries also have the ability to modify the incoming and outgoing data stream. The incoming stream needs some modification because of the request header fields and perhaps also to transcode the body of the data so as to allow the Web server to understand it. The outgoing stream needs some modification as well, perhaps to transcode the body of the data back to the client, to adapt the response header values to the client and/or to format the presentation of the data stream.

For example, let us imagine that a mobile phone is accessing a Web site through Web Intermediaries set up as a proxy, and that the request is in WML. The request cannot be understood by the Web server. Having a MEG registered in WBI, acting as a transcoder that is able to understand WML and also able to transform the content of the data into, for example, HTML, allows this Web application to serve WAP clients. The MEG transcodes WML into HTML, modifies the request header values as needed and forwards the transcoded data to the Web server. The Web server responds with some data, the MEG transcodes the data back to WML, modifies the response header, the presentation and the language to suit the client's device and passes the data back to the WML client.

This leads to a new separation of the logical modules inside the MVC design pattern (see Section , "Mobile architecture" on page 138).

Mobile architecture

In this scenario, some of the Controller module's responsibility has shifted to the Web Intermediaries. This is because Web Intermediaries can alter and transcode the data submitted by the client, but only so as to allow the Web server to understand it. Some of the responsibility of the View module has also shifted to the Web Intermediaries, because Web Intermediaries can format the presentation of the response data for a specific client device. For example, images can be replaced with links to accommodate the client's limitations. Figure 9-10 on page 139 shows a sample architecture.

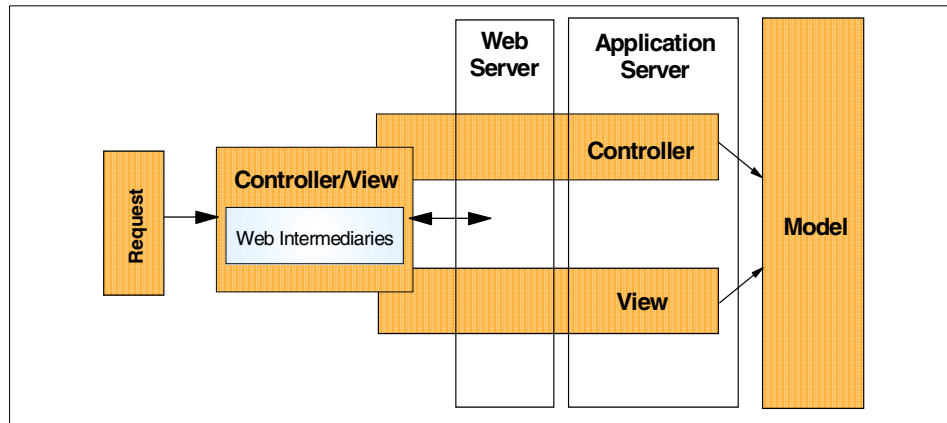


Figure 9-10 Sample mobile architecture

The tasks and responsibilities of each logical module have shifted between Web Intermediaries, Controller and View, except for those of the Model. There is no longer a clean separation between them. To what extent have the responsibilities changed for each component?

► Controller

This component should still have the function and responsibility to prepare and map incoming data to device- and business-independent data, because it remains the entry point of the business logic. If the incoming data is written in another markup language, then the Controller should simply reject the request. The interface to invoke the business logic is only known to this component and makes the whole system easier to control.

The Controller has to handle the application flow as well, which could be different for each kind of device. For example, filling out a long, sophisticated form on a PC browser is different from using a WAP phone, and very different from using a voice application. The PC browser can pull up the whole form, then the server can validate after submission; if anything was wrong, or missing, the server can send back the page pointing out the errors. The WAP phone does not have that much space at its disposal; the application has to collect the information field by field, performing the validation each time. Using voice is the most difficult; the user does not want to read a whole page to fill out a form. Design must also be impeccable because of the grammar involved.

- View

This component still has the function and responsibility of deciding which parts of the data are going to be presented to the client. The response should only contain the overall presentation layout. Issues like language selection and tweaking should not be handled here. The decision as to what data is going to be presented to the client is still the responsibility of this module.

- Web Intermediaries

The final presentation has to be created here. Web Intermediaries must look up the request and response header values to determine the correct final presentation to the client. If transcoding is needed, it should be performed here appropriately. Language selection is performed here as well, because it is very easy to apply some StyleSheets in Web Intermediaries instead of changing the logic of the View module. Tweaking should be performed here as well, for example by inserting an image with a lower rate or by removing an image, to accommodate the client's device.

Though the View and the Web Intermediaries are symbionts, the View part has to stand without the Web Intermediaries. There are different levels of integration and dependencies between them:

- The View component can represent the whole content without any Web Intermediaries; we can see this in any traditional Web application, or in our case in the direct approach for mobile devices.
- The View component is independent of the Web Intermediaries, but not for all the clients. For example, the application may work well producing HTML content, but the devices are not able to access the content. The Web Intermediary runs independently and transcodes the original content to the required format. The same thing happens with tweaking.
- The View component and Web Intermediaries are tightly related; when the View component generates only one part of the content, the final presentation is produced by the Web Intermediaries. For example, a View element, a JavaServer Page, creates XML, content then the transcoder applies the StyleSheet to that XML.

It seems that this approach is difficult to maintain and implement, but with well-defined tasks assigned to each component it is much easier to achieve than to modify the existing Controller and View components. The approach now shows a clean separation of all components. This makes this design easier to maintain and enhance in the future. Each component can be easily load-balanced to gain a higher performance. It is also easier to turn this system into a highly available system. A very high degree of flexibility has also been achieved .

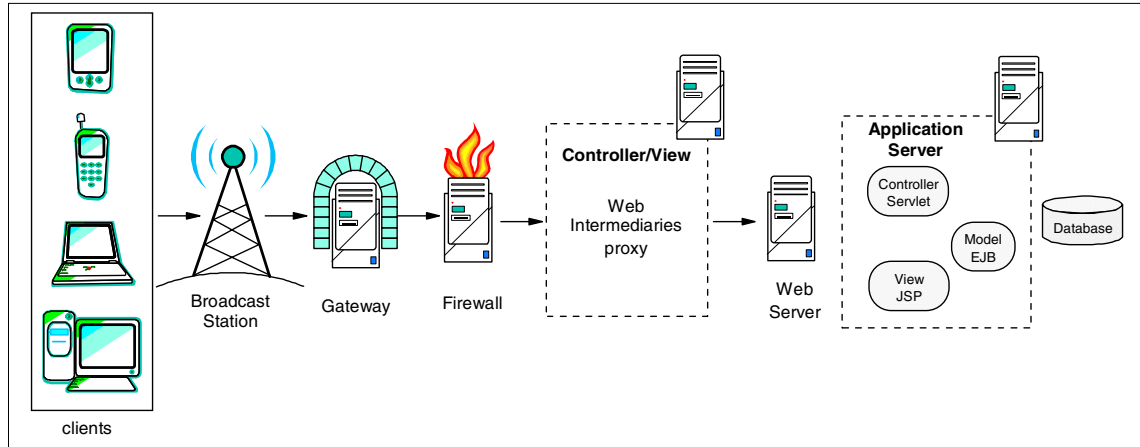


Figure 9-11 Mobile application with Web Intermediaries

This approach is the final recommended design pattern for a mobile application (see Figure 9-11). Certain issues such as devices, performance, high availability, reusability, client independence, multiple languages, and patterns are addressed and can be handled appropriately. Following is an example of the mobile approach using this architecture.

9.3.3 IBM WebSphere Transcoding Publisher

IBM is leading the way in the mobile computing field to bring the technologies of the mobile world together with the strength and integration of the IBM server and software family of products in order to create mobile solutions. Mobile computing and pervasive computing provide a series of technologies that enable people to accomplish personal and business tasks. Pervasive computing allows users to access enterprise applications, synchronize their data and receive updates from the network. Mobile devices give user access to information regardless of time and location.

IBM has defined a set of pervasive computing products to satisfy user requirements. One product which allows the two described scenarios to be extended to a mobile application is IBM WebSphere Transcoding Publisher. This application acts as a transcoding service that can filter, enhance, convert or reformat content to enable access to data by different devices. Transcoding is a general concept, not bound to any markup languages, formats, or products. In the pervasive realm, transcoding has been found useful in adapting markup languages, in exchanging data between two systems, and in preserving bandwidth where limited. Transcoding can be done on binary data, such as images, as well as on text data, but most transcoding will be applied to text

streams. Transcoding can be a very powerful service that relieves the workload and difficulty of having a page designed in several versions, each specific for one client type. Also, it eliminates the necessity of writing interface code for heterogeneous systems.

For example, a user is using a browser to access the welcome page of a specific Web site; the Web site answers with a respond page, the transcoder formats this respond page to suit the output to the browser. Another user is accessing the same Web site with a mobile phone; the Web site sends the same respond page to the transcoder and the transcoder formats the respond page to suit the mobile phone. The Web site does not need to have two versions of the welcome page to maintain, one version for a browser and the other version for a PDA. One page, for example in HTML or XML, is all the Web site needs to maintain. The rest of the formatting to a specific device or markup language is the responsibility of the transcoder. The Web site can also maintain different types of markup languages.

Editors

IBM WebSphere Transcoding Publisher contains two types of editors, the text editor and the image editor. Beyond that, IBM WebSphere Transcoding Publisher has a standard set of transcoders available to convert HTML documents and reformat images. The transcoder also has the function of caching the respond page. If the same page is requested again by another client, the Transcoding Publisher can retrieve that page from the cache and return it to the client without forwarding the request to the Web server again.

This powerful product brings about the following result for the two scenarios mentioned above: existing Web applications do not need to modify their current resources, because the transcoding part will be carried out in the transcoder; this is the only application that needs to be integrated into the existing system. New systems can still build the application as a usual Web application and integrate the transcoders into the system.

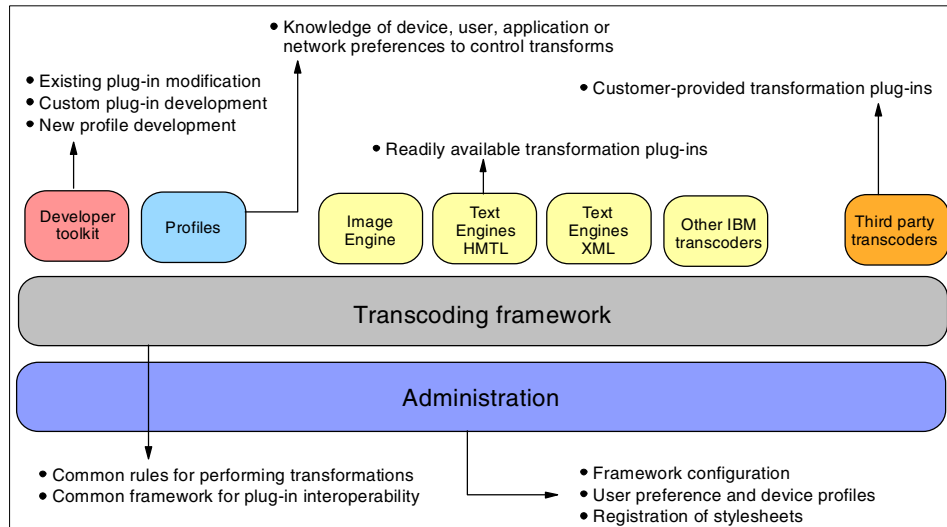


Figure 9-12 WebSphere Transcoding Publisher components

The concept of transcoders and plug-ins inside IBM WebSphere Transcoding Publisher is based on the concept of Web Intermediaries which was discussed in 9.3.1, “Web Intermediaries” on page 134. It is built upon a transcoding framework which provides common services to the registered transcoders and plug-ins. The framework also supports transcoders written against the Web Intermediaries API or servlets written against the Java servlet API.

Architecture

IBM WebSphere Transcoding Publisher can run in three different modes. It can be deployed as a proxy, as a servlet filter, or as a set of JavaBeans. For more information about the runtime modes of WebSphere Transcoding Publisher, refer to Section 7.3, “WebSphere Transcoding Publisher considerations” on page 97. Delivering a well-defined solution requires careful considerations in application design. There are many important key issues to be considered, such as the devices to be supported, the format of the data source, the developer’s skills, as well as flexibility, maintenance and performance considerations.

The mobile architecture that we will introduced will use IBM WebSphere Transcoding Publisher running as a proxy on a separate machine, because this provides the most flexible and high performance solution. The transcoder can also be easily clustered into a high availability environment. The performance will take an extra boost as well.

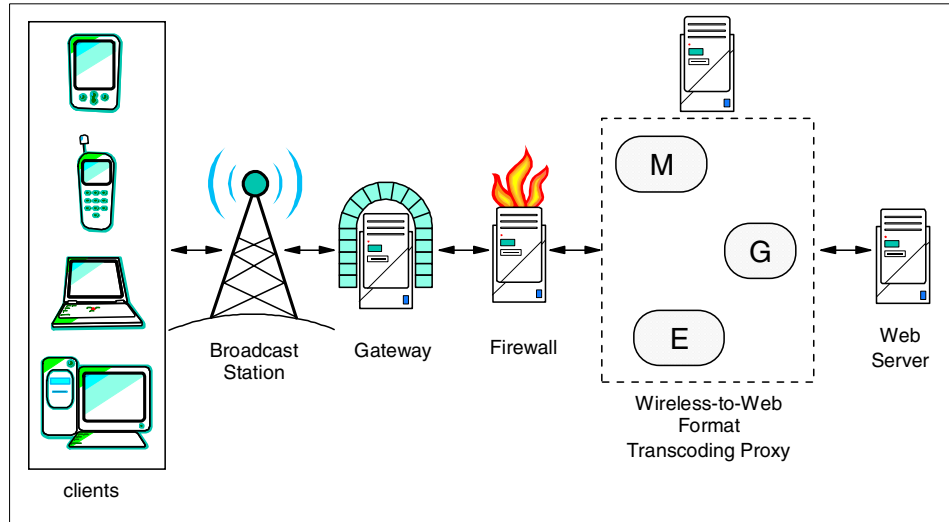


Figure 9-13 A mobile architecture

In this architecture, IBM WebSphere Transcoding Publisher is set up in front of the network path as a proxy. The function of a proxy inside the Internet or intranet is to cache responses and therefore reduce a repoll of each request to the data source. In the lab environment, there is no cache server set up for this purpose. If the user so wishes, he/she can choose to use a cache server. Another function of a proxy is to provide a single point of entry/exit to/from an internal network. A firewall can be placed between the client and the transcoder, and between the transcoder and the application server to ensure security. The application server and its servlet engine remain in the same chain of the network. What does the data flow look like if a request is received?

Data flow

A sample description of the data-flow for an HTTP client and WAP is as follows:

HTTP

1. The client makes a request to the transcoding proxy for a resource, which can be an image, a text/HTML page, or a text/xml page.
2. The transcoding proxy is able to edit the URL and other header fields and header values. Information about the device given in the request is also saved. The transcoding proxy then forwards the request to the Web server to acquire the requested object.
3. The Web server returns the object that the transcoding proxy asked for.

4. The transcoding proxy uses the initially stored information about the client to apply the converting and formatting on the response page.
5. The converted or formatted page is sent to the client.

WAP

1. The wireless client makes a request that is converted to HTTP by the WAP gateway.
2. The WAP gateway forwards the request to the transcoding proxy.
3. The transcoding proxy edits the URL and other header fields and header values, if necessary. Information about the device given in the request is also saved. It then forwards the request to the Web server to acquire the requested object.
4. The Web server returns the object that the transcoding proxy asked for.
5. The transcoding proxy uses the initially stored information about the client to apply the converting and formatting to the response page. In this case, the requested page is formatted into WML.
6. The WML page is sent to the WAP gateway.
7. The WAP gateway converts the text WML page into wireless-ready form for the wireless client.

According to the description of the data flow, the logical separation of components is, in this environment, no longer transparent, as in the presented MVC pattern. Recall that each component was assigned a task. In this architecture, the transcoding proxy mixes up two logics together. These are the Controller and the View components. The transcoding proxy must validate and modify the information of each request made by a client initially before forwarding it to the Web server, also storing information if necessary. This is to some extent controlling the request as the Controller should do, but the transcoding proxy is not validating and modifying the content of the request. The main data in the content is disregarded by the transcoding proxy.

The transcoding proxy does not need to know anything about the interface. This is done by the Controller, in our example the servlets. The servlets map the data supplied by the client to independent data that is understood by the Model. Also, the transcoding proxy applies its presentation to the result of the View component, in our example the JavaServer Pages. Here, the transcoding proxy is converting and formatting the resulting data to a new presentation, but it does not have the ability to decide which part of the Model object is going to be presented to the client.

Section 9.1.1, “Model-View-Controller” on page 126 stated that the View model takes control of which data to present and this ability does not apply to the transcoding proxy. The transcoding proxy disregards the content of the response object, simply applies converting and formatting to the object and returns it to the client. Of course, converting and formatting changes the presentation, but only to suit the client’s device. In this case, the client has limitations in displaying the information. No loss of important information due to converting or formatting should occur. The transcoding proxy has a hybrid status inside the architecture. Although there is now an overlapping in the architecture related to the purpose of each logical component, this will not harm the application. The logic of the application was not changed for the transcoding proxy. There was no need to change the data source or any objects in the application because of the integration of the transcoding proxy. An important design consideration here is *state management*. As of today, WAP phones do not support cookies, which means that state management and session identity are maintained by the WAP gateway.

The architecture presented in Figure 9-13 on page 144 provides a new way to design a mobile application. Many types of devices are supported through this architecture, but there are also some limitations because of some types of devices. Still, the underlying MVC pattern is not altered. Instead, a new logical component is introduced to handle device-specific data. This is important to expand the usability of the system. A higher flexibility is achieved and the system is easily maintained. If the performance and throughput of the system need to be improved, this scenario is easy achievable.

More details about IBM WebSphere Transcoding Publisher can be found on the following Web site: <http://www-4.ibm.com/software/webserver/transcoding/>.

9.4 Design patterns

Design patterns are common strategies for developing reusable object-oriented components. More precisely, they are a special way to solve a specific group of problems. Contrary to design concepts or application frameworks, patterns are not out-of-the-box solutions that can be taken off the shelf and deployed in the application. It is an approach that helps design by analyzing the code from other projects or applications.

You might find that you have used a special way to solve a class of problems in your application. This is what we call a *pattern*. The process of discovering a pattern is also widely called *pattern mining*.

For the application design, we want to take a better look at the design patterns. Beyond those, there are also architectural and analytical patterns.

The design patterns can be classified into three groups: creational patterns structural patterns and behavioral patterns.

The **Creational patterns** are used generate new objects instead of using the common “new” operator. This allows the system to be more flexible. One example is the Prototype pattern, which can be used for complex objects. Instead of creating a new instance of the class, a copy of the object is used. This can save time and make the instantiation easy for the new object.

The **Structural patterns** are used to combine and compose groups of objects into larger structures. The main advantage of Structural patterns is that they does not modify the connection between the objects when there are changes in the system. One example is the Proxy pattern.

The third group includes the **Behavioral patterns**. They are used for communication between the objects as well as for the flow control in a program. One example is the Chain of Responsibility pattern. This pattern provides the ability for different classes to handle one request without knowing the other requests. The request is passed in a specific order along the classes until one class can handle it.

Each group contains a set of different patterns. You can find the definitions for them in discussion or e-mail groups or in other publications. As new patterns appear, others disappear. The most important design patterns are defined in Gramm, Helm, Johnson and Vlisside, *Design Patterns: Elements of Reusable Software*. It includes descriptions and examples of twenty three of the most commonly used design patterns.

In the next sections, we will focus on the three-tier class model of our Trade2 example. We will outline the different patterns that we have either used or recognized in the application. We will discuss and outline the impact of the patterns on the Trade2 application. Figure 9-14 on page 148 gives an overview of the different Trade2 example patterns.

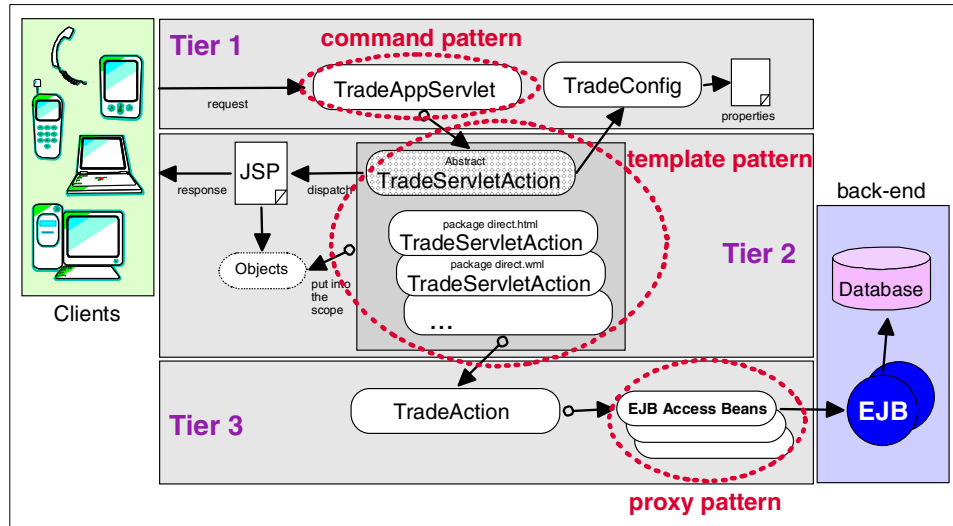


Figure 9-14 Used patterns in the Trade2 example

9.4.1 Object

This section explains the term *object* in a few sentences to get a common understanding before discussing the patterns.

Objects are abstract entities of the real world or of a system. An object consists of the following three parts:

- ▶ **Identity:** Every object is unique and has its own identity. The identity can be used to distinguish between objects.
- ▶ **State:** The state of an object includes its structure as well as the values for the property. Other objects can be referenced by using them as a value.
- ▶ **Behavior:** The behavior covers the functionality of an object. The functionality is given in the defined fields and methods of the object. In every object-oriented language, methods consist of a signature (including the name, parameter and type of the method) and the implementation itself.

Objects are defined by object type. Class concepts can be used to group single or similar object types. Every object has to include at least one constructor method for creating the object. Additionally, it can contain a destructor for removing the object and several methods for implementing functionality.

For more information about object-oriented technology, please refer to <http://java.sun.com/docs/books/tutorial/java/concepts/object.html>

9.4.2 Command pattern

Although the Command pattern is not used in the Trade2 example, it has some similar concepts. It is important for us to introduce this pattern because it can be widely found in other projects. Most of the well-designed, structured Web applications use the Command pattern as the Controller for the application.

First, the Command pattern belongs to the Behavioral pattern class. As the Chain of Responsibility pattern, it deals with requests from the user interface. More precisely, the Command pattern has the responsibility of forwarding the request to only one module, which was specified beforehand.

The functionality is normally provided by using a command interface. In this way, the user interface is separated from the program modules. The client does not need to know anything about the action itself and other objects. Changes in the program modules do not affect the client-side programs. Another advantage of the Command pattern is that every action is stored in a separate object. Therefore, it is possible to write a simple undo algorithm for each action.

9.4.3 Template pattern

The Template pattern belongs to the Behavioral patterns class. The idea of the Template pattern is to provide a predefined parent schema; you only have to add additional implementations to the derived classes and objects of the schema. The best example is an abstract class, with predefined fields and methods. The derived classes have to define and implement additional logic to this.

The design and implementation of the abstract `TradeServletAction` of the Trade2 example follows the Template pattern. Subclasses like the `direct.html.TradeServletAction` or `direct.wml.TradeServletAction` are derived from this class and implement additional logic.

Figure 9-15 gives an overview of the inheritance of the classes for the Trade2 example:

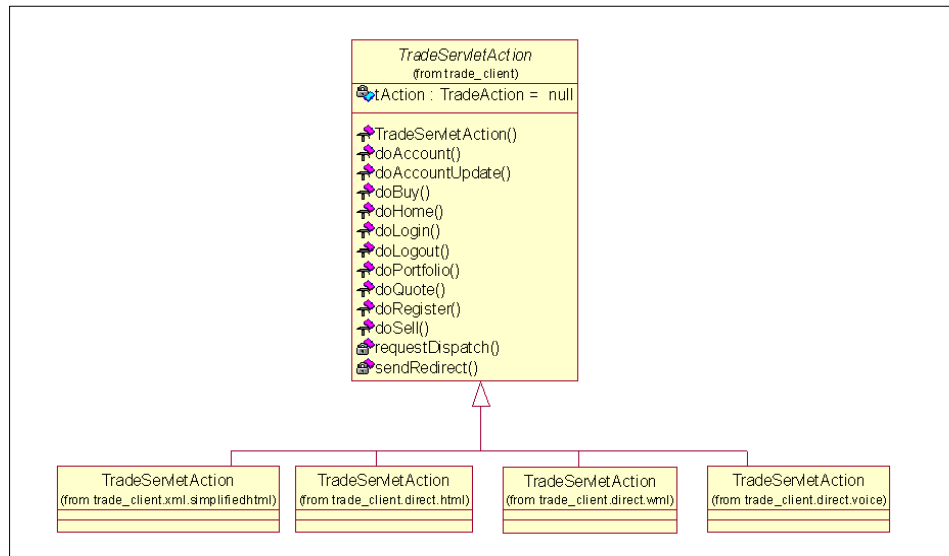


Figure 9-15 Class diagram of the Trade2 example

9.4.4 Factory pattern

The Factory pattern belongs to the class of Creational patterns. It helps you to create new objects of a specific group with a given set of attributes. Therefore, it has to handle several subclasses of the group. Depending on the attributes, one instance of the subclasses is returned. Figure 9-16 gives you an overview how Factory patterns work:

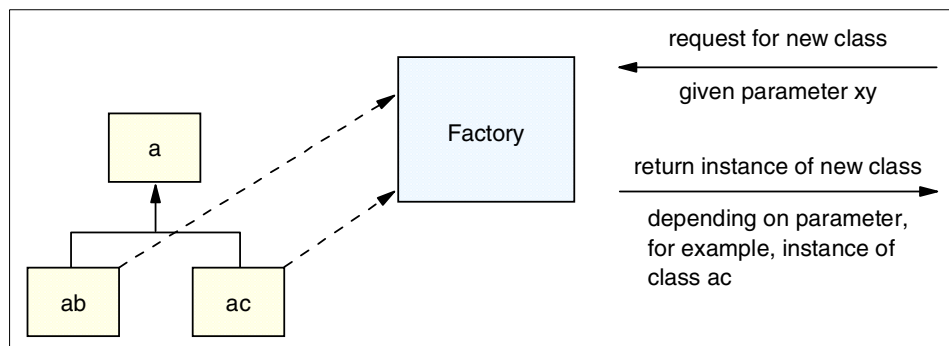


Figure 9-16 Overview of the Factory pattern

On the left-hand side of the figure, you can see the parent class **a** and the corresponding two subclasses **ab** and **ac**. The Factorypattern manages the instantiation of the classes and returns the appropriate object. In the figure's example, it returns the class **ac** for the given parameter **xy**.

We have talked about the abstract TradeServletAction in the previous section. We have outlined that the abstract class identifies the use of the Template pattern. However, the creation of the derived TradeServletAction classes can also be categorized as a Factory pattern. Depending on the attribute User-Agent and the chosen method (xml or direct, for example), the appropriate class and package are chosen. The only exception is that the Trade application is not using a separate class for this function. However, we can define the process of the creation of the object as a kind of Factory pattern. This is just an example of how easily patterns can overlap in an application.

9.4.5 Proxy pattern

The Proxy pattern belongs to the Structural patterns class. It helps you to hide the complexity of an object by providing a simple object interface. The simple object provides the same methods. It passes the request to the complex model and takes care of returning the results. This way, all the methods and the entire functionality of the complex model can be accessed. In addition, it can provide access control to the original model or it can create the model before the object has been accessed. This means that the object is not immediately created with the instance call. This can save additional computing resources.

In the Trade2 application, you can find the Proxy pattern within the EJB Access Bean. The EJB Access Bean helps you to hide the complexity of the EJB interface. You do not have to write code to access the Remote and Home interface of the EJB object.

For more information about EJB, refer to <http://java.sun.com/products/ejb/>

For more information about Access Beans, refer to the Visual Age for Java product and programming guides.

9.4.6 Advantages of patterns

As mentioned in the introduction, the main focus of patterns is reusability. The Solutions patterns can be easily adapted to new projects. Reusable components can be, for example, object models, class designs or even code. Therefore, patterns help to improve and to speed up the design and the development phases.

Another advantage of using patterns is the possibility to change the code without having to change the whole application (see for example the Bridge or the Adapter patterns). In addition, less effort is required to maintain the application. New functionality can be added more easily in the application and the system is substantially more flexible.

9.5 Where to find more information

9.5.1 IBM

- ▶ Avrahm Leff and James T. Rayfried, *IBM Research Report Web-Application Development Using Model/View/Controller Design Pattern*.
- ▶ Information about WBI:
<http://www.cssrv.almaden.ibm.com/wbi/intermediaries.html>

9.5.2 Outside of IBM

- ▶ Gramm, Helm, Johnson and Vlisside, *Design Patterns - Elements of Reusable Software*.



Application development

This chapter discusses application development and explores with a greater level of detail the topics presented in the previous two chapters (see Chapter 9., “Application design” on page 121 and Chapter 8, “Solution design” on page 103). The chapter is divided into four parts:

1. The Web application elements section gives a short overview of the components used by Web applications. The common Java elements, as well as the different markup languages, are discussed.
2. The second section describes in detail the different development tools proposed for the offering, and how the tools can be used effectively. This includes a developer’s guideline as well as the interlink between the products and the supported team development.
3. The third section introduces the various tools for testing the applications. It presents mainly the desktop-based simulating tools to access and test a mobile Internet application.
4. In the last section, we talk about the best practice pattern. We provide answers to some common questions as well as tips and hints for a fast start to setting up a mobile Internet solution.

10.1 Web application elements

A Web application environment consists of different components. In this section, we will give an overview of the most common technologies. The focus is set on the Java elements and the different markup languages for the mobile devices.

10.1.1 Java servlet

Java servlets are used to provide dynamic Web content based on an HTTP request. In general, they are server-side Java programs, so they offer all the advantages of the Java language. Servlets are object-oriented and can utilize the full Java API. This allows the developer to use predefined and implemented code and methods easily. The Java servlet can take advantage of the built-in memory control of the Java Virtual Machine (JVM). For example, unused objects are automatically removed by the garbage collector.

Sun's Java Servlet API also defines a session object. The Java servlets are able to store information about the session, such as user data input or user preferences, in the context memory.

Unlike common server-side programs using CGI, Java servlets can be deployed on each Web server (with servlet support) without advance compilation. Furthermore, the servlets are only compiled and initialized once in the runtime environment. For the incoming request, an instance of the appropriate servlet is used. This strategy ensures a higher performance compared to scripting languages.

There are many tools on the market for developing Java servlets. The offering also provides the development tools: IBM VisualAge for Java 3.5 and WebSphere Studio 3.5. Both tools can be combined for quick application development. We describe these components in detail in 10.2.1, "WebSphere Studio" on page 166 and 10.2.2, "VisualAge for Java" on page 173.

For more information, please refer to the redbook *Programming with VisualAge for Java*, SG24-5264.

10.1.2 JSPs

JavaServer Pages (JSPs) are used to generate the formatted output for a Web page response. JSPs do not include any business logic. They are useful in separating the development of the Web site from the Web page design. It is possible to define clear responsibilities for the Web application development. On

one hand, the Web page designer does not have to know how to obtain dynamic data, but only where to place it, and can therefore concentrate on the look and feel of the Web page. On the other hand, Java programmers do not have to handle layout issues. Their focus is to provide the logic for the application.

The dynamic content of the Web pages is typically provided by a JavaBean (see 10.1.3, “JavaBeans” on page 156). JavaBeans are reusable software components that can be created by other Java objects, such as for example Enterprise Java Beans (EJBs) or Java servlets. Through the use of a special scripting language, the data in the JavaBean can be accessed from the JSP.

The output of a JSP can be any type of document; in our case the following ones were used:

- ▶ HTML
- ▶ WML
- ▶ XML
- ▶ Related HTML markup languages, like cHTML
- ▶ Other XML-based languages, like VoiceXML

JSPs are used in the common MVC model in the View component. The servlet dispatches the request to the JSP. Figure 10-1 gives an overview of how JSPs are used in this context.

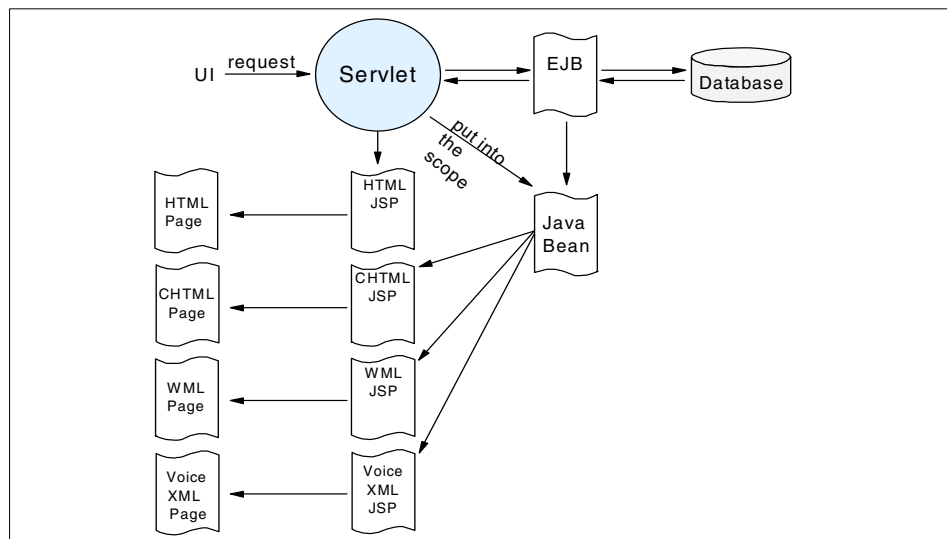


Figure 10-1 Interaction between Java components and Web application objects

The servlet receives an request from the client and invokes an EJB to fulfill this request. The EJB collects the data from a database, then instantiates and sets a JavaBean.

The JavaBean then has to be put into the scope, so the JSP can reach it as an object through the application server.

Depending on the client, the servlet now dispatches the request to the appropriate JSP file. For example, WML JSP files are used for WAP devices, while for a normal desktop browser, HTML JSP files will be invoked from the servlet.

A simple way to develop JSP files is to use WebSphere Studio (see 10.2.1, “WebSphere Studio” on page 166). It provides wizards and additional functionalities for creating and editing JSPs. The WebSphere Test Environment in VisualAge for Java allows you to test the JSP files in conjunction with your Web application (see 10.2.2, “VisualAge for Java” on page 173).

10.1.3 JavaBeans

JavaBeans are simple component classes in the Java language and are mainly utilized as reusable objects. The following rules must be applied to a JavaBean:

- ▶ Public class
- ▶ Public constructor without arguments
- ▶ Setter and getter methods for accessing the properties

The setter methods are used to set values for certain fields of an object. It is not possible to access and set them directly, so with getter methods, values can be retrieved from the object. Furthermore, it is possible to define events for the bean, in case the JavaBean is able to interact with other beans.

Most development environments support the introspector feature for JavaBean development. This feature allows users to modify the bean using a visual interface. Note that you can normally use this feature only when the syntax rules are implemented correctly. The developer of a JavaBean can also provide the structure and further information about the bean via the JavaBean Information class.

In the Java servlet and JSP Web application context, JavaBeans are mainly used as a kind of a container for data exchange between different components. For example, an EJB object providing the business logic can instantiate the bean and set the properties values. The JSP is then able to retrieve the data by using the getter methods of the bean.

JSPs can reach the JavaBeans as objects via the application server. In order to be accessible, the bean has to be put into the scope of the Web application. The following three scopes are available during runtime:

1. Application: the object is accessible while the application server is running.
2. Session: the object is only available for the session it has been created in. The object disappears together with the session.
3. Request: the object is available during the request; after the response has been sent out, the object disappears.

JavaBeans are typically distributed in .jar files.

For JavaBeans development, we recommend the two IBM tools IBM WebSphere Studio 3.5 and IBM VisualAge for Java 3.5 (see 10.2.1, “WebSphere Studio” on page 166 and 10.2.2, “VisualAge for Java” on page 173 for more details).

For more informations on JavaBeans, please refer to
<http://java.sun.com/products/javabeans/>

10.1.4 Enterprise JavaBeans

The EJB 1.0 specification was announced by Sun Microsystem in March of 1998. The main focus and advantages of Enterprise JavaBeans (EJBs) are to separate the business logic of an application from the middleware supporting it.

The EJBs are deployed and managed in the application server in a container framework. One container can include one or more beans. To split up the development, management and handling of EJBs, the following roles are defined:

- ▶ Provider: develops the business logic
- ▶ Deployer: deploys the EJBs
- ▶ Application Assembler
- ▶ Container Provider
- ▶ Server Provider
- ▶ System Management

Every EJB consists of a Remote and a Home Interface. The name of the EJB Home Interface is stored in the JNDI directory. One way for clients to access the EJB is to use this Home Interface. It allows clients to create, find and remove EJB instances. Another approach is to use AccessBeans. They hide the complexity of the EJB interface, but offer the client the same functionality.

An Enterprise JavaBean can be either a session bean or an entity bean.

Session beans

There are two different kinds of session beans:

- ▶ stateful
- ▶ stateless

A stateful session bean typically lives as long as the client session does. One important feature is that it can be made passive or active by the container. That means that the EJB, including all information about the session, can be stored and accessed again by the application.

Stateless session beans are instantiated for each client request. They cannot be made active or passive by the container. Therefore, they are not able to store any information about the session and the request flow. The instances of a stateless session bean are equal to one another. It is common for the container to have a small pool of instantiated stateless session beans of the same type, in case an instance of the same stateless session bean can be used for different requests.

Entity beans

Entity beans are used to represent persistent data. One common practice is to obtain the data from a relational database. Another is invoking another application or, for example, a CICS transaction.

The entity beans can be categorized into two groups, namely BMP and CMP.

BMP - Bean Management Persistence

Bean management persistence means that the EJB provider has to ensure the persistence of the EJB. It requires additional logic and code; because of this, BMP is not recommended for large projects with a focus on portability and scalability. There is no guarantee that an BMP EJB can be easily deployed to another Web application server.

CMP - Container Management Persistence

Here, the persistence of the EJB is ensured by the container. No additional code has to be added to the business logic of the EJB. The business logic remains untouched and can be deployed easily on different Web application servers; for example, the IBM WebSphere Application Server provides CMP for EJBs.

An entity bean uses a unique key to identify its home. This unique key can be used to create or find the instance of an EJB. Note that the unique key has to be serializable to match the specifications.

More information on EJBs can be found on the following Sun Web page:

<http://java.sun.com/products/ejb/>

10.1.5 XML

XML stands for eXtensible Markup Language. It is a subset of the SGML standard and is also used to define markup languages like WML or SyncML.

XML is an easy to use language for describing any content. Developers can define their own markup language tags or elements. They can ensure that their XML dialect best fits their application needs. In general, XML documents include only data. They do not include any formatting information used to present the content; this information is stored in a separate Extensible StyleSheet Language (XSL) file. Only this XSL file is used to display the data in an appropriate way on the client browser. An XSL file and the XML data document are merged by using the Extensible StyleSheet Language Transformation (XSLT). We will discuss XSL and XSLT later (see 10.1.6, “XSL” on page 160 and 10.1.7, “XSLT” on page 161). Below is a very basic XML example of the enhanced Trade2 application:

Example 10-1 Enhanced Trade2 application

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE account SYSTEM "/trade/xml/simphtml/res/Trade.dtd">
<account>
  <userid>James</userid>
  <fullname>James Howart</fullname>
  <address>Bakerman street downtown Raleigh</address>
  <email>jhowart@wherever.com</email>
  <creditcard>56789-99-1234</creditcard>
</account>
```

The example shows that an XML document typically consists of three parts. The first part is the XML declaration. It specifies the used XML version as well as the character encoding. Furthermore, the XML declaration may indicate if the document is logically complete; otherwise, a reference to an external entity is included.

The second part is the Document Type Definition (DTD). This element is optional. Normally, it is used to define the grammar for the XML file and to validate the XML data against this grammar. If there is a DTD, the user can reference it internally or externally. In the first case, one DTD is included in the XML file. In the second case, the DTD is referenced by a pointer and stored in the server file directory.

Furthermore, the DTD can be defined by a public or non-public entity. For a public entity, the keyword PUBLIC is used. Non-public entities are referenced by the keyword SYSTEM. For the shown XML example, we built our own DTD file and stored it in the directory of the server. You can find the reference as well as the keyword SYSTEM in the XML data files. For more information about the XML Trade2 application, please refer to 15.3, “Universal transcoding (XML source)” on page 293.

The third part is the body of the document, which contains the data described in a structured way. Each piece of information is represented by a tag that can contain elements and attributes. As discussed before, the XML document does not include any formatting information to present the data.

An XML document must obey specific syntax rules. For example, one rule states that every element must consist of a start and end tag. An example for this rule is the tag `userid` (`<userid>James</userid>`). If an element is empty, no explicit end tag is needed. Instead, the end sign is included in the first tag (`<userid/>`). Nested elements are allowed if the proper order is matched.

The XML documents can be classified into two categories:

- ▶ **Well-formed XML documents:**
A well-formed XML document satisfies the strong syntax requirements. No special additional DTD is needed. Therefore, a non validating XML parser is able to read and parse the document
- ▶ **Valid XML documents:**
A valid XML documents applies the strong syntax rules and matches the legal structure defined in the DTD. A validating XML parser is able to read the document.

In practice, XML is used widely to share, store and exchange information between applications or computer systems in business settings. An example of an application where XML is becoming more and more popular is EDI (Electronic Data Interchange). Here, XML is used to exchange and share data and information between different companies.

More information about XML can be found at the following Web address:
<http://www.w3.org/XML>

10.1.6 XSL

The Extensible Stylesheet Language (XSL) is used for the presentation of an XML document. XSL can produce various output documents, for example HTML, WML, cHTML and VoiceXML.

XSL is fundamentally based on two existing standards: Cascading StyleSheets (CSS) and Document Style Semantics and Specification Language (DSSL).

By using XML and XSL, data description and data presentation are separated. In this case, it is possible to define clear roles and responsibilities within the Web application development. As a consequence, the XSL developer does not have to deal with the dynamic data and can change or amplify the user interface without touching the business logic. Note that XSL does not require a DTD for either the source or the result document.

Examples of XSL files can be found in the XML Trade2 applications (see 15.3, “Universal transcoding (XML source)” on page 293).

For more information on XSL, please refer to <http://www.w3.org/Style/XSL>

10.1.7 XSLT

Extensible Stylesheet Language Transformation (XSLT) is the common language used to translate an XML document into another output format by applying the appropriate StyleSheet for it.

Figure 10-2 gives an overview of the translation process using a XML input source and a XSL file.

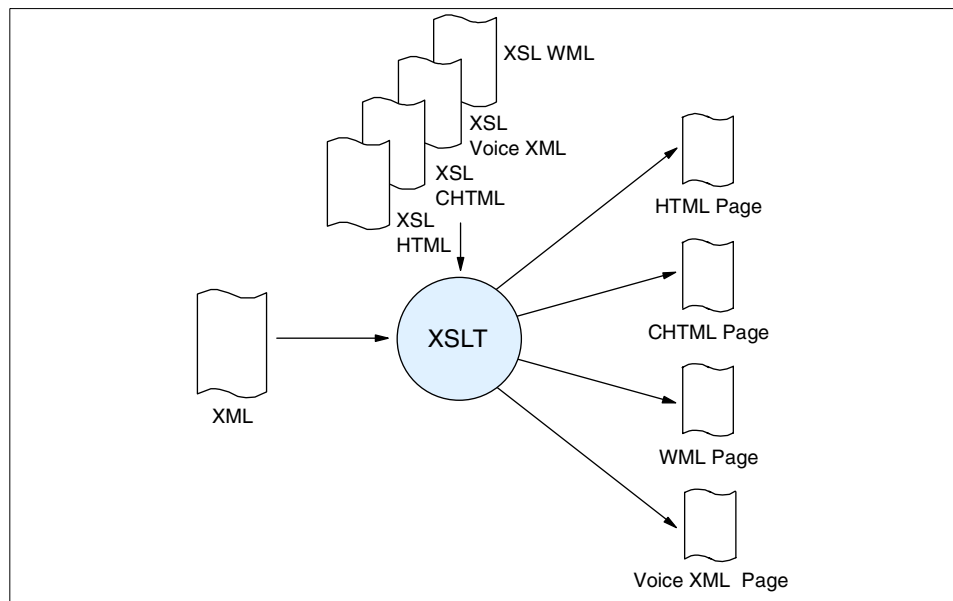


Figure 10-2 Interaction between XML, XSL and XSLT

In translating an XML document, the following steps are processed by the XSLT:

- ▶ Parsing and validating the XML document
- ▶ Parsing and validating the XSL document
- ▶ Applying appropriate patterns to parts of the source tree
- ▶ Calling and applying the templates within the matching patterns

As mentioned before, the output format of the XSLT process can be another XML document or, for example, an HTML or WML file. Typically, a control element in the application has to choose the right XSL file. In our Trade2 example, we used the WTP to do the matching (see 15.3.9, “Registering the StyleSheets in WTP” on page 304).

To develop XML applications, one can use predefined classes from Apache. These classes cover the whole XML, XSL and XSLT development, giving the developers the whole API. For details, please refer to <http://xml.apache.org>.

The specification for XSLT can be found at the following Internet address:
<http://www.w3.org/TR/xslt>

10.1.8 HTML, cHTML, HDML, WML

This section provides an overview of the most common and important visual markup languages for mobile Web applications.

HTML

Hypertext Markup Language (HTML) is the common language for creating and publishing documents for the World Wide Web. It is based on SGML and used to display hypertext information on traditional Web browsers such as Netscape Navigator or Internet Explorer.

The simplest method to create and process an HTML document is to use a text editor. But there is also a wide range of visual programming tools for WYSIWYG development, one of which is included in the offering the IBM WebSphere Studio 3.5. It offers several wizards and the Page Designer as a visual editor to create the HTML files (see 10.2.1, “WebSphere Studio” on page 166).

In addition to the defined tag set, you can use Cascading StyleSheets (CCS), JavaScript and Java Applets to improve the functionality, look and feel of an HTML page.

For more information about HTML, please refer to <http://www.w3.org/Markup/>

Compact HTML

Compact HTML (cHTML) was developed and announced in 1999 by the Japanese telecommunications provider NTT DoCoMo. It was specified to fit the new requirements of small wireless devices such as mobile phones. cHTML is actually the used markup language for the i-mode standard.

Basically, cHTML is a well-defined subset of HTML versions 2.0, 3.2 and 4.0, so it is based on SGML. It has no support for Java or any scripting language. Only GIF images can be displayed on an i-mode browser.

More information on cHTML can be found at
<http://www.w3.org/TR/1998/NOTE-compactHTML-19980209/>

For more information on i-mode, please refer to <http://www.nttdocomo.com/i/>

HDML

Handheld Device Markup Language (HDML) is a specialized version of HTML. It is designed to enable wireless pagers, cell phones, mobile phones and other handheld devices to obtain information from Web pages. HDML was developed by Phone.com (formerly Unwired Planet) before the WAP specification was standardized. It is a subset of WAP with some features that were not included in WAP.

HDML is still widely used in the US and has several million users in Japan, although the dominant markup language and service are cHTML used with i-mode. The UP.Browser developed by OpenWave Phone.com provides support for HDML markup. While HDML is widely used, this standard is rather dated, and in some areas is disappearing.

WML

The Wireless Markup Language (WML) is defined by the WAP Forum. Like cHTML, it was developed to meet the requirements of small appliances such as mobile phones or PDAs; WML is based on XML.

The content of an WML file can be viewed with any WAP browser. Unfortunately, as in HTML, the visual presentation of a WML document depends on which browser is used for display. For example, the user can notice differences in the look and feel when testing a WML application with a Nokia or Phone.com browser.

The defined tag set for WML is similar to those of HTML and cHTML. However, only so-called “wbmp images” can be displayed on the device. A WML document is organized into cards and decks. a deck is the unit sent to a wap Browser to answer a request. It can contain one or more cards.

A WML card is the unit that is shown on the screen of a WML browser. Cards are typically interlinked with each other using the common link technology. Note that the possible deck size permitted for a specific device can vary because of different browsers.

For details about WML and WAP, please refer to <http://www.wapforum.org>

XHTML

Extensible HTML (XHTML) is a combination of HTML 4.0 and XML 1.0 streamlined into a single format for the Web. XHTML is expected to become the standard format for Web pages. XHTML also makes it possible for Web pages to be developed with different sets of data, for different types of browsers used to access the Web. Increasingly, handheld devices are used for the Web that must download abbreviated pages because they do not have screen displays large enough to handle the graphics.

10.1.9 VoiceXML

The Voice Extensible Markup Language (VoiceXML) is based on XML. It is used to create distributed voice applications based on existing or new Web applications. These applications can be accessed from the user by telephone. It is also possible to refer to VoiceXML as a *non visual* markup language, because the output is not presented visually to the user.

VoiceXML includes the following features:

- ▶ recognition of spoken input
- ▶ synthesized speech output
- ▶ digitalized audio
- ▶ DTMF (telephone key input)
- ▶ recording of spoken words
- ▶ dialog flow control
- ▶ scoping of input

The following example gives you an overview of what a VoiceXML file looks like:

Example 10-2 A VoiceXML file

```
<vxml version="1.0">
  <form id="intro">
    <property name="bargein" value="false"/>
    <block>
      <prompt>
        <audio src="/trade/direct/vxml/intro.au">
Welcome to the IBM WebSphere Everyplace Access Redbook Trading Demo. Please
speak your userid and password when prompted. If you do not have a userid,
please go to the web site to register.
        </audio>
      </prompt>
      <goto next="/trade/direct/vxml/login.vxml"/>
    </block>
  </form>
</vxml>
```

The main difficulty VoiceXML applications have is recognizing the spoken input. Therefore, a *grammar* is used. All possible and expected input from the user has to be defined and stored in the grammar.

As mentioned above, it is also possible to include pre-recorded audio files in a VoiceXML document. This offers the user real human interface instead of the all too common robotic touch of synthesized speech. However, a greater effort is needed to maintain pre-recorded audio files, and the application loses some of its flexibility.

Other important features of VoiceXML are *say what you hear*, *barge-in* and built-in commands such as *quit* and *help* that are automatically handled by the voice browser for each VoiceXML document.

To develop VoiceXML applications, IBM WebSphere Studio 3.5 can be used (see 10.2.1, “WebSphere Studio” on page 166); this tool offers different wizards and an editor with code assistance to create the documents. To test the VoiceXML files within your application, you can use the IBM WebSphere Voice Server SDK (see 10.2.5, “Voice SDK” on page 180).

More information on VoiceXML can be found at the following Web address:
<http://voicexml.org>

A very useful resource for VoiceXML is the Programming Guide, which comes with WebSphere Voice SDK.

10.2 Development tools

The next section gives an overview of the development tools recommended for building a multiple channel Web application. The focus is on the main features which support a quick and effective development.

10.2.1 WebSphere Studio

WebSphere Studio is an integrated development environment which can be used to create, manage and process Web application resources such as HTML documents, images, Java objects, and so on.

The new 3.5 version of WebSphere Studio includes an enhancement for developing applications for mobile devices. It supports the widespread markup languages VoiceXML, cHTML and WML.

In the next sections, we will discuss and outline the most important features of WebSphere Studio. We will introduce the different wizards, the team development and the publishing facility in detail.

For the Trade2 example application, the Advanced Edition of WebSphere Studio 3.5 was used. Note that not all features described in the next paragraphs are available in the Entry Edition or the Professional Edition.

More information about the WebSphere Studio can be found in the following Redbooks:

- ▶ *How about Version 3.5? VisualAge for Java and WebSphere Studio Provide Great New Function*, SG24-6131.
- ▶ *WebSphere Personalization Solution Guide*, SG24-6214
- ▶ *Version 3.5 Self Study Guide: VisualAge for Java and WebSphere Studio*, SG24-6136

Wizards of WebSphere Studio

For straightforward application development, WebSphere Studio contains the following wizards:

- ▶ SQL Wizard
- ▶ Database Wizard
- ▶ JavaBean Wizard
- ▶ Content Wizard
- ▶ User Wizard

Please note that while wizards are used to help in the development process, they will not provide final solutions. Most of the results from the wizards are only useful for creating prototype applications, or generate a skeleton for application development.

The wizards can be found and invoked from the Tools menu in the Studio Workbench (see Figure 10-3).

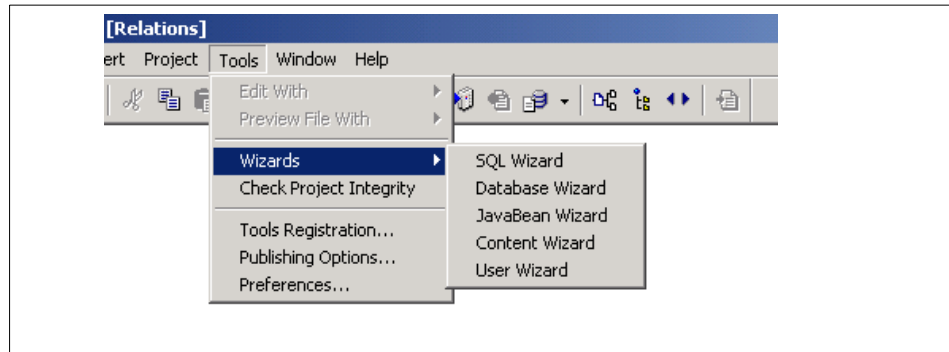


Figure 10-3 Wizards in the Studio Workbench

The SQL Wizard allows the developer to create SQL statements. A visual menu guides the developer through the steps. In the first step, the database connection is chosen. Then the developer can choose all the necessary tables and rows from a list. Later on, the tables may be joined. In the last steps, the guide allows the developer to add conditions to the query and enhance the statement with other SQL commands.

The Database Wizard uses the created SQL statement to generate documents and classes for the Web query. In the first step of this wizard, the developer has to decide between creating a Servlet Model or a JSP Model.

- First, we want to take a better look at the Servlet Model wizard. In the first step of the Servlet Model, the developer can select the markup languages for the application. At the moment HTML, cHTML, WML and VoiceXML are supported; multiple selections are allowed. The developer must define which of the parameters of the SQL statement should be provided by the user. For the chosen parameters, the wizard creates an input page. This input page will be presented to the user before the SQL statement at the beginning of the application. For processing the request, the wizard creates a Java servlet and a servlet configuration file. Database access is handled by a JavaBean using a JDBC connection. To print out the results to the user interface, the wizard generates a result page JSP. Two additional error-handling JSPs for database and general errors are provided as an option.

- In the JSP Model, only JSP files are used. This means that the request processing, database access and generating of output is handled by a JSP file. All tasks are centralized in the JSP file.

To match the common MVC model and gain the use of a code that is easy to maintain, the use of the Servlet Model is recommended.

Example 10-3 shows the hierarchy of the generated servlet:

Example 10-3 Hierarchy of the servlet

```
Object
  javax.servlet.GenericServlet
    javax.servlet.http.HttpServlet
      com.ibm.servlet.PageListServlet
        com.ibm.webtools.runtime.AbstractStudioServlet
          com.ibm.webtools.runtime.StudioPervasiveServlet
```

The JavaBean Wizard helps you to generate code for input HTML files, as well as a servlet and a result JSP page for a given JavaBean. As in the Database Wizard, a JSP file for general and database errors can be produced.

The User Wizard and the Content Wizard are used to define personalization rules for the Web application. With the User Wizard, the developer can define attributes and roles for the users. The Content Wizard is used to define attributes of the content. By defining these parameters settings, the application gains the possibility of dynamically generating and providing content depending on the user settings and preferences.

Personalization is not used in the Trade2 example. For more details about this topic, please refer to the *WebSphere Personalization Solution Guide*, SG24-6214.

Team development support

The most common way of enabling WebSphere Studio for team development is to place the project file on a shared network drive. In this case, WebSphere Studio automatically manages the resources using *check out* and *check in* on assets.

Check out is used to modify and edit a specific document from the project. For every checked out document WebSphere Studio creates a local copy in the following directory of the developers machine:

<WebSphere Studio Install Folder>\check_out\<project name>\<folder>

All changes are stored on the local copy first, until the document is checked in again. Checked out files are marked with a red check mark placed in front of the file name. Other developers can only obtain non writable copies of checked out files. In order to avoid conflicts, it is recommend that the publishing of the resources only be done when all resources are checked in. It is possible to get a list of all the checked out documents and users by choosing **Project -> Check Out Info** from the workbench menu.

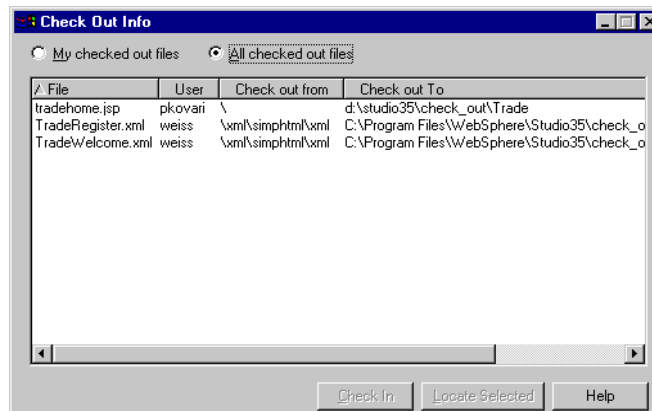


Figure 10-4 Example for the overview of the checked out files

To extend the team development support, additional Version Control software can be added to the WebSphere Studio environment. If you are using WebSphere Studio together with VisualAge for Java, it is recommend that you use the same repository.

Publishing

Publishing is used to deploy the project resources. WebSphere Studio has multiple publishing stages; by default, every project provides two: the *Test stage* and the *Production stage*. These can be customized by defining the publishable resources and setting up the publishing targets (Web application directories). In general, documents can be published to a file system or to an FTP host. An example for a Test stage is the VisualAge for Java WebSphere Test Environment (WTE). In WTE, all resources can be tested together with the other Web components before they are deployed to a productive Web application server. It is also possible to define new Publishing Stages in WebSphere Studio to meet specific needs. The developer is able to deploy the resources to any Web application environment (for example, a multiple productive Web applications server).

For information about setting up a Publishing Stage and for more details, please refer to Section 18.3.2, “Publishing a WebSphere Studio project” on page 357.

Site tools

To develop and enhance the user interface, WebSphere Studio provides the following additional tools:

- ▶ **Page Designer:** This tool is used to create and edit documents such as JSPs and HTML files. It offers three different views of the source: a WYSIWYG (What You See Is What You Get) editor referred to as the *normal view*, the HTML *source view* and the *preview* in a browser.

The Page Designer cannot be used for mobile markup languages such as VoiceXML and WML. For VoiceXML, WebSphere Studio provides a special VoiceXML editor with code assistance.

The WebSphere Studio default application for WML editing is Notepad. As long as Notepad is only a simple text editor, we recommend that you use the standard WML SDKs, such as the Nokia Toolkit or others. The best WML developer environment provides visual code assistance, debugging facilities and a graphical user interface for testing (see 10.3.4, “Testing the WML application” on page 187).

WML, cHTML, and VoiceXML support in WebSphere Studio includes document parsing and validating.

- ▶ **Animated GIF Designer and WebArtDesigner:** Both tools can be used to create and process images and clip art.
- ▶ **Page Detailer:** This tool is an additional program for WebSphere Studio. It is used to measure the performance of a Web application, on a single page level.

Using WebSphere Studio with VisualAge for Java

WebSphere Studio 3.5 has the capability of interlinking itself with VisualAge for Java. The developer has the possibility of making Java classes and source code exchanges between both applications.

To set up the link, make sure that VisualAge for Java and WebSphere Studio are correctly installed on the same machine. The other necessary tools are installed by selecting **Project -> VisualAge for Java - Install Studio Tools in VisualAge** from the WebSphere Studio menu bar. In addition the **Remote Access to Tool API** in the Option panel in VisualAge for Java has to be started. We recommend that you enable the automatic start option for the tool.

The established interlink can be used as follows:

- ▶ From WebSphere Studio:
 - Sending Java source files to the VisualAge for Java Workbench:
Select the Java file; by right-clicking it, you obtain a pull-down menu with the option to send the file to the VisualAge for Java Workbench.
 - Importing Java source files and classes from VisualAge for Java.
Choose the **Import** option from the menu bar.
 - Updating Java source files and class files from VisualAge for Java:
Select the files again; the pull-down menu invoked by right-clicking them allows you to update the files.
- ▶ From VisualAge for Java:
 - Importing the Java source file from WebSphere Studio project.
 - Sending Java class files to a WebSphere Studio project.
 - Checking the status of existing VisualAge for Java Java files in a WebSphere Studio project.

Note: WebSphere Studio and VisualAge for Java have to run on the same machine in order to exchange files.

Please make sure that you have selected the right WebSphere Studio project for use with VisualAge for Java (to do this, select **Tools -> WebSphere Studio Tools -> Set Studio Project**).

If you are only using VisualAge for Java to develop and deploy (compile) the code for your Web application, it is recommend that you use the same CVS facility for WebSphere Studio (see “Team development support” on page 168).

Other features of WebSphere Studio

The following sections will introduce some other features of WebSphere Studio related to our mobile Web application development.

Annotators

Annotators (see Figure 10-5) are useful in customizing a Web page using clippers added to the document as XML tags. This code defines which elements should be kept, removed or replaced. The interpretation of a file including annotators is done by the annotator engine. The engine is provided by the WebSphere Transcoder Publisher (see 15.2.2, “Annotators” on page 271).

To add and edit annotators in WebSphere Studio, the developer can use the Page Designer. Modifications can be easily made in the normal view using the context-sensitive options of an element, or coded directly using the HTML editor.

Please note that only internal annotators are supported with WebSphere Studio at the moment.

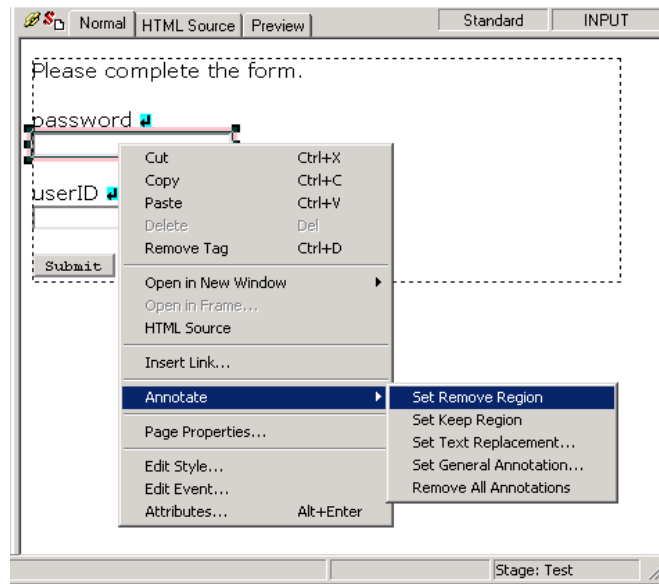


Figure 10-5 Annotation in WebSphere Studio

Java compiler

You can compile your Java source classes directly in the WebSphere Studio environment. As a default, WebSphere Studio provides the two JDK versions 1.1.8 and 1.2.2. The compilers can be customized via the menu option **Tools->Preferences**. Here, it is possible to specify the directories of the compiler as well as the appropriate classpath.

Hints and tips

The development of the Trade2 application is based on WebSphere Studio 3.5 and VisualAge for Java. We will give here a short overview of our recommendations regarding WebSphere Studio and the development of an application for multiple devices.

- ▶ Team development: You must place your project file on a share network drive so that everybody can access the assets and automatically use the editing management feature of WebSphere Studio.
- ▶ Interlink Studio / VisualAge for Java: By installing the required tools, you can easily access both resources and can exchange and control the Java class and source files of your project. Make sure that you enable the autostart option of the tool.

- We have found that the WML and VoiceXML markup languages support exhibits strange behavior under certain circumstances. We recommend that you switch off the parser option for these files, so that when the developer checks in the files, they will not be parsed. This means that the content will remain the same and no validation will be done; there are some trade-offs when using this method, because the links will not be handled by WebSphere Studio. On the other hand, if the files are valid, they have no errors and the paths are correct then they will be published correctly.

10.2.2 VisualAge for Java

The VisualAge for Java (VAJ) environment offers a development environment for the whole life cycle of a Java application. It is split up into four main parts: Code Editor, Repository Management, Debugger and Test Environment. The next paragraphs outline the most useful tools for developing an integrated Web application. We will discuss the WebSphere Test Environment, the team development support and the debugging feature. For information about using VisualAge for Java with WebSphere Studio, please refer to “Using WebSphere Studio with VisualAge for Java” on page 170.

For more information about VisualAge for Java and the integrated development tools, please refer to the following Redbooks:

- *Programming with VisualAge for Java 3.5*, SG24-5264
- *Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*, SG24-5755
- *How about Version 3.5? VisualAge for Java and WebSphere Studio Provide Great New Function*, SG24-6131
- *Version 3.5 Self Study Guide: VisualAge for Java and WebSphere Studio*, SG 246136

WebSphere Test Environment

The WebSphere Test Environment (WTE) is one of the most valuable features of VisualAge for Java 3.5. It provides an application and Web server environment on a developer machine. It can be used to test and debug all the resources for a Web-based application. The WTE is launched by selecting **Workspace -> Tools -> WebSphere Test Environment...** from the menu bar. It consists of two main parts, namely the Servlet Engine and the Persistence Name Server. The latter enables the environment to use JNDI services. Furthermore, you can change the settings for the JSP monitoring in the WTE Control Center and register new databases for JNDI. Figure 10-6 gives an overview of the different components in the environment.

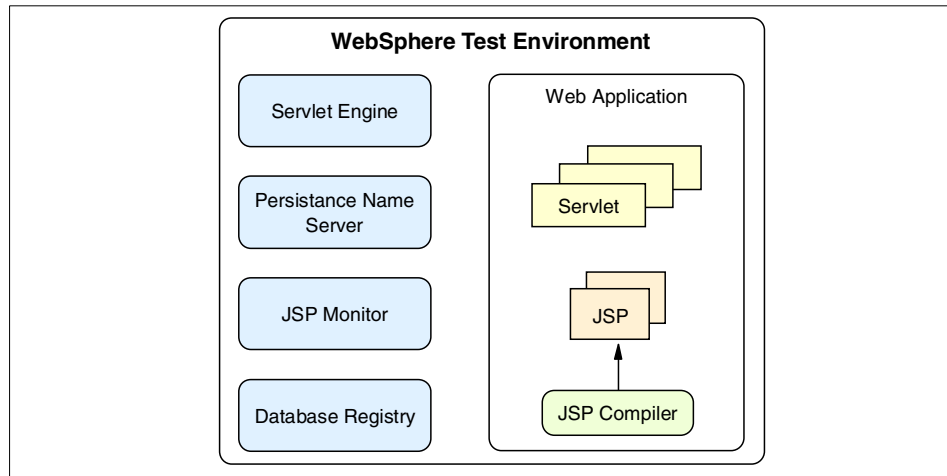


Figure 10-6 Overview of WTE

A major advantage of WTE is that any change in the source code is immediately compiled and hotlinked to the current application. Because of this, it is not necessary to restart the Servlet Engine. The only exception to this rule is the `init()` function of a Servlet, because it is only invoked once, during the setup of the runtime environment. It is also possible to access and interact with other Web resources and services such as LDAP directories or Web servers outside the WebSphere Test Environment.

As shown in Figure 10-6, the WTE also includes a JSP monitor. This feature allows you to monitor the execution of a JSP file and to debug it step by step. It is possible to set different breakpoints in the JSP code in order to control the document output.

The JSP execution monitor is enabled by selecting **JSP Execution Monitor Options -> Enable monitoring JSP execution** from the WTE Control Center. The compiled code of a JSP file is stored in the JSP Page Compiler Generated Code project folder of the VisualAge for Java environment.

Note: VisualAge for Java and WTE support the JSP.91 and JSP 1.0 standard. As a default, the JSP.91 version is used: To change the version, you have to update the `default_app.Webapp` properties file.

Team development support

The Team Repository is used for the version control for VisualAge for Java. It is a built-in software component and operates on an “edition, version and release” model. The developer is able to store multiple editions of project items in the repository. In order to grant the access to every developer, the Team Repository is typically stored on a network server.

VisualAge for Java also offers the possibility to define and set up different roles and permissions for the developers.

A Team Repository can be loaded and browsed by selecting **Window -> Repository Explorer** from the VisualAge for Java menu bar. From this window, the developer can compare different editions and versions. Furthermore, stored solutions, projects, packages, classes and methods can be added to the workspace; by selecting **Admin -> Change Repository** from the menu bar the repository can be changed. It is possible to work on different projects with a separate version control.

Please notice that deleted items from the workspace still exist in the Repository. For more information about the Team Repository, please refer to the Administration Guide of VisualAge for Java.

Debugger

The VisualAge for Java Debugger can be used to step through the code during runtime; it is therefore possible to control the flow of the application. All triggered methods and variables are shown. The developer is able to step through the different logical levels and views of the application code.

To activate the Debugger, the developer has to set breakpoints inside the code. When the breakpoint is reached during runtime, the Debugger feature appears. Please note that changes to the code in the Debugger window are reflected directly in the workspace.

The Debugger window is divided into a Debug page and a Breakpoint page. The Debug page gives you an overview of the code invoked, the visible variables, their value and their current state. The value and the state can be controlled and changed by the inspectors.

The Breakpoint page provides management for setting the breakpoints inside the code.

Important: Make sure that the breakpoints are enabled under the menu called by selecting **Window -> Debug -> Global Enable Breakpoints**.

10.2.3 Development tools in WTP

The WebSphere Transcoder Publisher contains the following four development tools:

- ▶ Transform Tool
- ▶ Request Viewer
- ▶ Profile Builder
- ▶ API for developing new transcoders

They can be used to customize and extend WTP in order to satisfy the customer's needs. We will discuss them in the next few sections. For more information about WTP, please refer to the following IBM Redbook: *New Capabilities in IBM WebSphere. Transcoding Publisher Version 3.5*, SG24-6233.

Transform Tool

The Transform Tool allows you to easily change the profiles in WTP and to preview the generated results and effects. It is useful in testing various transcoding options or new XSL StyleSheets.

The first step to start testing with the Transform Tool is to select an input source. The input source can be of one of the following types:

- ▶ HTML
- ▶ cHTML
- ▶ XML
- ▶ any XML defined language (WML, VoiceXML, etc.)
- ▶ image (GIF, JPEG)

In the second step, the Target Device and the Target Network have to be selected. The defined actions in those profiles will be used for the transcoding of the input source.

The transcoding process itself is invoked by clicking the **Transcode** button on the toolbar. The result is shown in the right panel of the application. The generated output can be saved for further purposes by selecting **File -> Save Transcoded Content**.

Request Viewer

The Request Viewer is a useful debugging tool for the WebSphere Transcoding Publisher. It monitors the request flow through the transcoding proxy. It also observes the whole operation during the flow from the triggering point on. The developer can see which plug-ins are invoked and when they are invoked. This can help to understand how WTP is working and also help to determine problems or find system bottlenecks.

The Request Viewer consists of two views: the Server Configuration panel and the Request Processing panel. The Server Configuration panel shows the current state of WTP with the registered components. The displayed plug-ins on the panel contain the different MEGlets installed in WTP.

The Request Processing panel monitors the dynamic process of an incoming request. It displays the triggered plug-ins and their associated MEGs. For every invoked MEG, the input source and the modified output source are shown. Figure 10-7 provides an example of an XML request of the Trade2 application processed by the Request Viewer.

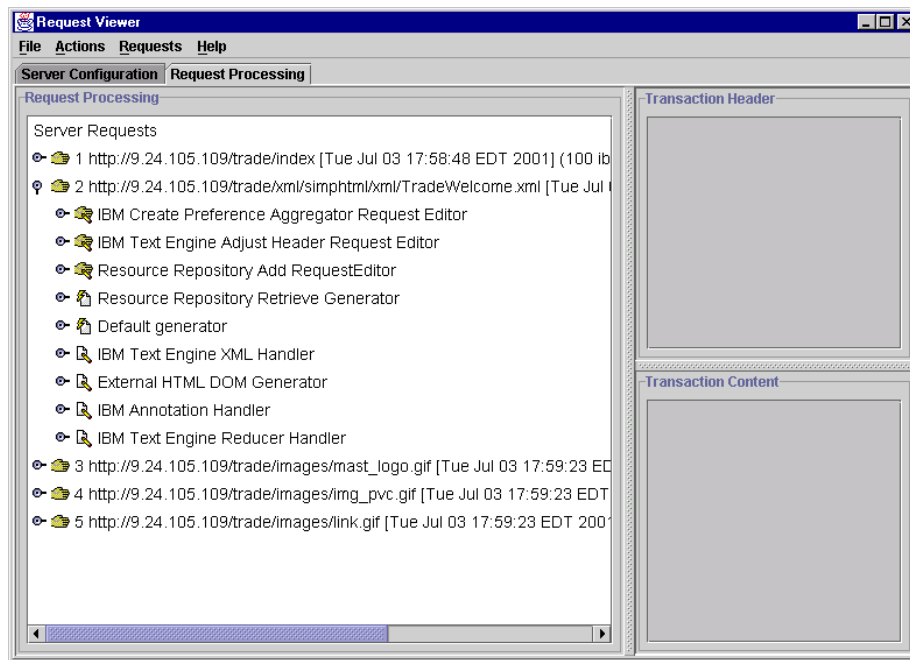


Figure 10-7 Example of a request processed by the Request Viewer

Note: The Request Viewer should only be used for development and debugging purposes, and not in a runtime environment. Furthermore, it is necessary to stop running WTP services before you can use the tool.

Profile Builder

WTP already provides preference profiles for the most commonly used devices. The Profile Builder helps the developer to extend this list and to create new profiles. It is possible to match new upcoming requirements, such as new mobile phones, PDAs or network types.

Note: The Profile Builder is not shipped with the WTP install package; you can download it for free from the following Web site:
<http://www.ibm.com/software/webservers/transcoding>

For details on the installation of the Profile Builder, please refer to the included instructions.

After launching the Profile Builder, the developer can choose between the following options:

- ▶ Create a new profile.
- ▶ Create a new profile using an existing profile as a template.
- ▶ Edit an existing profile.

A new profile is created in four steps. First, the developer has to decide between setting up a new device profile or a new network profile. Device profiles are used to adapt new devices to your application. They are used to determine which content should be delivered to the device. A network profile gives the developer the opportunity to set up rules in order to optimize network traffic.

During the second step, the developer has to specify conditions under which the profile should be used. To define the rules, the HTTP headers of a request are used, usually from the User-Agent field.

Tip: To combine several clauses, you can use a logical operator. WTP supports the following symbols: **!** for negation, **&** for logical AND, **|** for logical OR. Furthermore, it is possible to use the asterisk symbol at the beginning of the arguments.

The third step involves working with preferences. WTP provides various defined preferences that can be adapted to the specific needs of the profile. After selecting a preference, the developer can choose to mark it either as editable or view-only. In the editable state, the parameters can be changed in the Administrator Console of WTP. View-only parameters can only be changed through the Profile Builder.

One example for a preference you can adopt to your profile is the Fragmentation preference. With this preference, you can specify the maximum number of bytes you want to send to the device. This can be useful, for example, for WAP or i-mode applications with restrictions on the client browser side.

The last step is to save the profile and to register it at the Administrator Console of WTP by clicking **Register -> Preference Profile**. The new profile can also be supplied to other Web applications, customers or vendors.

Note: Please make sure not to store the file in the \etc directory of WTP. This can cause problems during the registration of the new profile.

The second way to create a new profile is to use an existing one as a template.

The third possibility is to edit an existing profile and modify the preferences. As in the second case, the developer is able to modify the editable as well as the view-only grouped parameters.

Note: An existing profile has to be registered again before changes to the profile can take effect.

Developing new transcoders

The Transcoder Publisher framework is a pluggable framework. Everybody has the opportunity to develop new transcoders to match new devices or new transcoder needs.

To conform to WTP, the new transcoders have to be written against either the Web Intermediaries (WBI) Version 4.5 API or the Java Servlet 2.1 API.

For more information about writing new transcoders, please refer to the WTP Developer's Guide (provided with the WTP package). More basic relevant information can be found in Section 15.2.3, "Text clipping" on page 284.

10.2.4 XML tools

At this time, the offering does not include an XML generating and editing tool. The default tool is Windows Notepad for editing. We can recommend other applications which we found useful during the development of this redbook.

XML Spy 3.5

XML Spy 3.5 is a third party product used to develop XML-based applications, or simply to edit XML documents. It provides several powerful and easy to use tools for XML, XSL and XSLT processing. The tool allows you to create, handle and process the different resources.

One example of the provided features is the array of different possible views for an XML document. The developer also has the opportunity to generate the appropriate DTD structure automatically.

Another example is the support of different XSLT processors. The individual modules can be downloaded from the Website and plugged into the installed product. Through the XSL option menu, it is possible to switch between the XSLT engines. Furthermore, XML Spy offers an enhanced scripting environment.

To evaluate the product, it is possible to obtain a 30 day trial edition key from the XML Spy Web site. For more information about the product please refer to <http://www.xmlspy.com>

Other XML and Web development tools

Other XML and Web development tools can be found on the IBM Alphaworks site: <http://www.alphaworks.ibm.com>

10.2.5 Voice SDK

The IBM WebSphere Voice SDK 1.5 is a toolkit used to test your VoiceXML application on a desktop system. As shown in Figure 10-8, the product itself consists of the following parts:

- ▶ **VoiceXML Browser:**
The VoiceXML Browser is used to interpret the VoiceXML files and to control the other modules inside the SDK. It also includes the DTMF simulator. This simulator can be used to recognize the input of numbers.
- ▶ **IBM ViaVoice Speech Recognition engine:**
The Speech Recognition engine is used to recognize the spoken input in order to match it to the defined grammar of a VoiceXML file.
- ▶ **Text-to-Speech engine:**
The Text-to-Speech engine (TTS engine) generates the synthesized speech output.

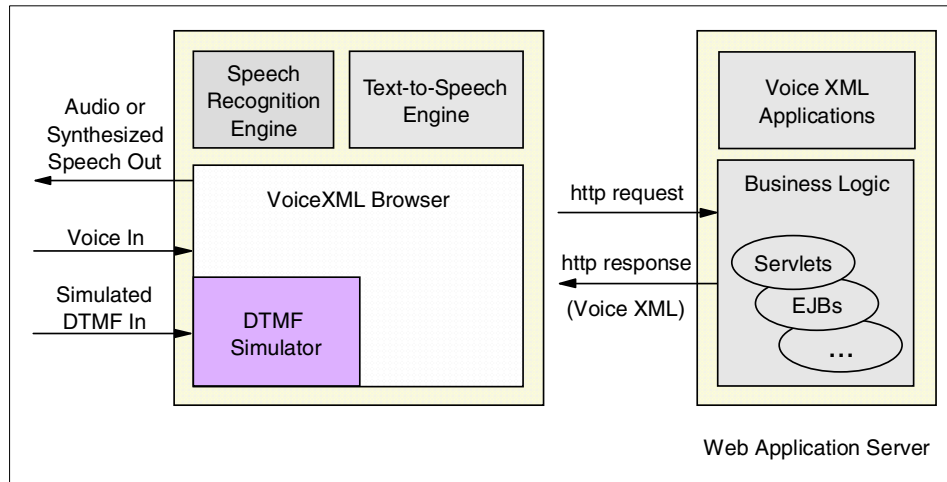


Figure 10-8 Overview of the IBM WebSphere Voice SDK

The SDK offers the developer a telephone acoustic model and an audio setup utility. The telephone acoustic model simulates the behavior of a telephone installation. It helps to adapt the application to a real telephone environment during development. The audio setup tool allows the developer to set the microphone and speech environment of the SDK.

For testing a VoiceXML application, the SDK provides two runtime modes:

1. Audio - plays the audio files, synthesizes speech, and recognizes speech. Input can be given via the microphone, the keyboard or the displayed DTMF keypad.
2. Text - writes the text to the console and replaces the audio files with the description. Input can be given via the keyboard or the displayed DTMF keypad.

Tip: If the DTMF simulator is closed, it can only be restarted by stopping and restarting the VoiceXML Browser.

The Voice Browser also handles cookies sent by the application. Please note that the cookies' information is only stored during the session. Unlike for set cookies on a desktop browser, the information is deleted after the session.

To develop the VoiceXML files, we recommend that you use the IBM WebSphere Studio 3.5 (see 10.2.1, "WebSphere Studio" on page 166).

Proxy settings for WebSphere Voice SDK

The launcher .bat files for WVS SDK can be found in the directory: <WVS SDK install directory>/bin; the files are: vsaudio.bat and vstext.bat.

In order to access the WebSphere Transcoding Publisher proxy server, you have to set up the WVS SDK to use the proxy for connection.

The proxy settings resemble the following piece of code:

```
-DproxySet=true -DproxyHost=proxyserver -DproxyPort=proxyport
```

Find the :Execute line in your .bat file, then add the parameters according to your setup after the Java runtime, but before the classpath settings. The line should look like this:

```
"%VSJREPATH%java" -DproxySet=true -DproxyHost=my.proxy.dom  
-DproxyPort=8088 ... - classpath ...
```

Important: Make sure that you include the dash before the parameters, and watch out for upper case and lower case characters.

For more information about the IBM WebSphere VoiceXML SDK 1.5 and other voice-related products from IBM, please refer to:
<http://www-4.ibm.com/software/speech/>

10.3 Tools for testing the application

In the development process, it is very important to test the application throughout the whole process.

Developing a user interface for Web applications is more like editing and testing repeatedly until the result is satisfactory. Unfortunately, the Web application development tools lack WYSIWYG editors which developers could use instead of testing the applications with real devices or emulators.

The best solution is to use emulators for testing the applications. These applications ensure the same user experience as the original applications.

10.3.1 Overview

Developing an application is an iterative process. At the end of every development cycle, you must test your application in order to satisfy the defined requirements. Typically, enhanced tools, including debugging facilities, are used.

This section gives an overview of recommended testing tools. We will discuss how to use them, and give you some additional tips.

10.3.2 Testing the HTML application

The Trade2 application can be tested on any desktop PC with a browser such as Internet Explorer or Netscape Navigator. If you are using WTP as a proxy, please make sure that the proxy settings of the browser are set properly. In Netscape Navigator, the proxy settings can be accessed by clicking **Edit -> Preferences -> Advanced -> Proxy**. Figure 10-9 shows the proxy server settings panel for Netscape Navigator.

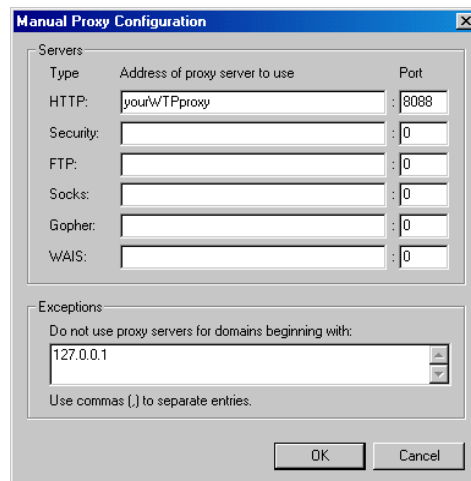


Figure 10-9 Proxy settings for Netscape Navigator

For Internet Explorer, choose **Tools -> Internet Options -> Connections -> LAN Settings**. Figure 10-10 on page 184 shows the proxy server settings panel for MS Internet Explorer.

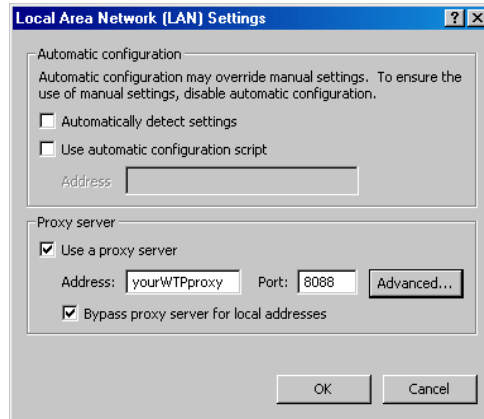


Figure 10-10 Proxy settings for Internet Explorer

If you are using WTP as a reverse proxy you do not have to modify your proxy settings.

10.3.3 Testing the simplified HTML application

The simplified HTML Web application runs on a simple desktop PC's browser, such as Netscape Navigator or MS Internet Explorer. The difference between the normal HTML and the simplified HTML is that, because of the low bandwidth, the client requires content that is less rich than the original. For example, the client is a laptop and it connects to the network using a wireless GSM modem.

In our sample application, we support the following two browser clients: Lynx and Opera.

Lynx

Lynx is a character-based Web browser (see Figure 10-11) running in the command window or terminal. This browser comes from the Linux community where it is still very popular, because it is fast and runs on a non-graphical terminal.

The browser runs on several platforms and the application, together with source code, is freely available on the Internet.

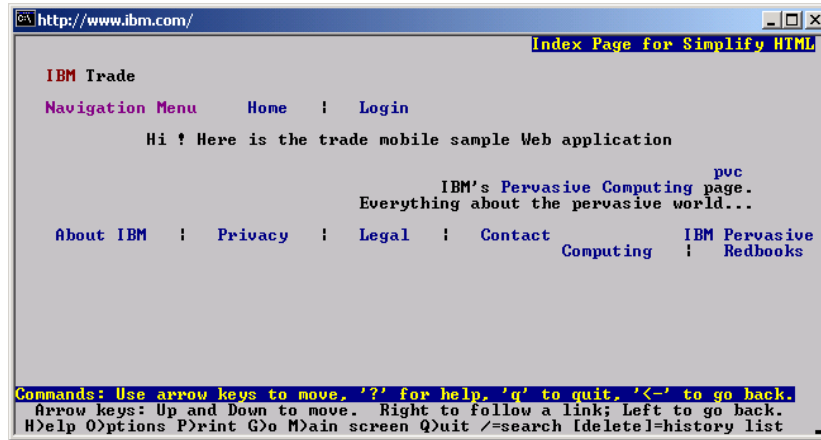


Figure 10-11 Lynx browser

How to set up

Lynx has a configuration file: `lynx.cfg`, where you can set up the proxy for transcoding. Pay attention to this line in the file:

```
#http_proxy:http://some.server.dom:port/
```

Remove the comment sign (#) and fill in the server name and port number parameters; the line should then look like this:

```
http_proxy:http://my.transcoder.dom:8088/
```

How to test

The sample application is looking for the Lynx token within the User-Agent field.

Open a command window or a terminal window, change the directory to the Lynx binary, then start Lynx.

The starting screen will come up; use the **G** (goto) key to type in the location. Type in the requested URL, then press the **Enter** key.

You can navigate the page using the spacebar and the Enter key. For help, use the **?** key.

Get the software

Lynx is available for download on the following site: <http://lynx.browser.org/>

Opera

Opera is one of the best Web browsers available for several different platforms. It has all the professional features of Netscape Navigator or MS Internet Explorer, and many other enhancements and new features.

In our sample application, we have used Opera to test the simplified HTML pages with colors and pictures (see Figure 10-12).

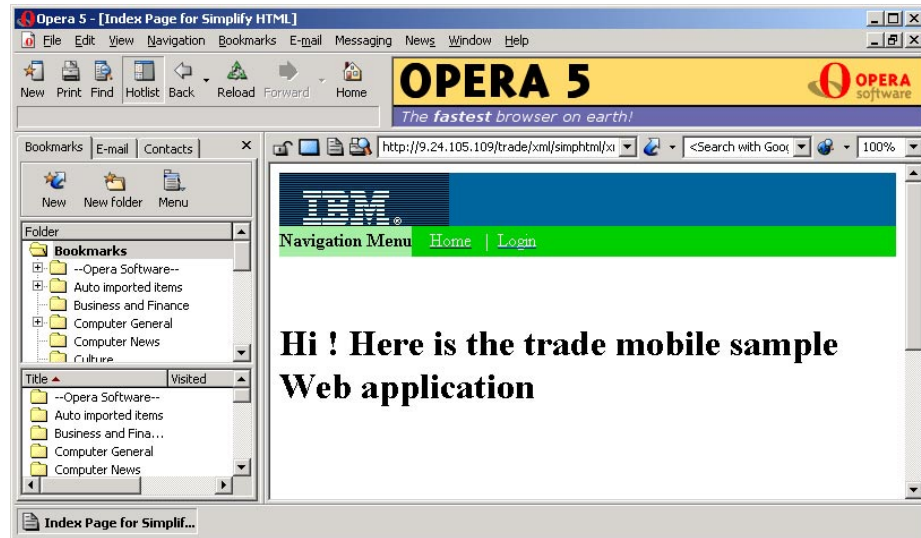


Figure 10-12 Opera Web browser

How to set up

Opera has a well-designed setup window to configure your browser. From the menu, choose **File -> Preferences**, or press **Alt-P**. In the left pane, select the **Connections** option; the connection properties will appear in the right pane. Click the **Proxy servers...** button, then set up your transcoder server as an HTTP proxy with the appropriate server name and port.

How to test

The sample application looks for the **Opera** token within the User-Agent field.

Start the browser on your machine; you will see a GUI similar to that of any other desktop browser. There is a text box for the URL below the main toolbars and above the viewer window.

Get the software

The software is available on several platforms: Windows 95/98/ME/NT4/2000, Windows 3.1 or NT 3.5, Solaris, OS/2, Macintosh (Power PC), Linux (x86), Linux (Sparc), Linux (Power PC), EPOC Release 5 & Release 3, BeOS.

The official Opera site can be found at <http://www.opera.com>

10.3.4 Testing the WML application

The WAP application runs on WAP-enabled phones. Since WAP is a specification, different manufacturers have different implementations for their browsers. Just in case, developers should test their application using different WAP emulators (for an example, see Figure 10-13).

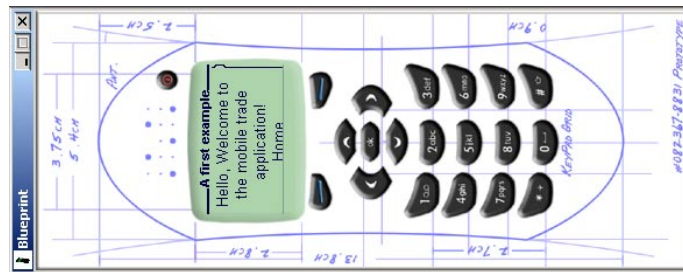


Figure 10-13 Nokia Toolkit WAP emulator

Nokia Toolkit 2.1

The Nokia Toolkit is a WAP development tool which has a built-in WAP browser. It is fully a Java application, and has several useful features:

- ▶ Multiple visual user interfaces (blueprint 6210, 7110)
- ▶ Use of customized HTTP connections or WAP Gateway
- ▶ Different code views and debugger

How to set up

From the menu, choose **Toolkit -> Device Settings** to access the configuration for the emulator. There you can set up the connection you have: HTTP or WAP Gateway. There is also a switch for cookies and authentication; if you want to use a proxy for your connection, there are the HTTP proxy settings as well.

How to test

After starting the emulator, two windows will appear; one is the workbench for development with several helpful views, the other is a phone with screen and the keypad.

Use the workbench to connect to the required site, type the URL in the location text box below the menu, then click **Enter**. The result will appear on the phone; you can then navigate on the page using the phone's keypad.

The advantage of the Nokia Toolkit is that, in the workbench, the developer can follow the recent actions, and can even debug the source code downloaded to the phone.

Get the software

The Nokia Toolkit is available for download on the Nokia corporate Web site. Go to the <http://www.nokia.com/corporate/wap/sdk.html> page, register (it is free); you will then be able to download the Toolkit.

UP.browser

UP.browser comes from a company called Openwave. The emulator is their own implementation of the WAP specification (see Figure 10-14).



Figure 10-14 UP.browser WAP emulator

How to set up

The UP.browser offers a configuration for the connections; open **Settings -> UP.Link Settings** in the menu. There you can set up the connection, whether going through an HTTP connection, using proxy servers if necessary, or using UP.Link servers for the connection.

How to test

Start the UP.browser; it is actually called UP.Simulator after installation. Two windows will appear as with the Nokia Toolkit: one for the terminal where the developer can follow the actions, the other for the phone where the test is occurring.

At the top of the phone's window, below the menu, there is a text box for the URL; type in the requested URL, then click **Enter**.

Get the software

UP.brower is available for download on the Openwave company's site:
<http://developer.openwave.com>

10.3.5 Testing the cHTML application

Waprofit.com i-mode emulator

Waprofit's i-mode emulator is an early application for i-mode emulation. It is very simple and works well with the i-mode sites. For development purposes, it is good enough to test the application.



Figure 10-15 i-mode emulator

How to set up

Unfortunately, the emulator does not provide any option for configuration. The solution is to use the transcoder as a reverse proxy, to avoid the proxy configuration on the client.

How to test

Since this i-mode emulator has a different User-Agent than the real devices, in our sample the transcoder is looking for a device with the following condition:

```
User_Agent\=*Microsoft URL Control*
```

To access a URL, type it in the text box at the top of the emulator window, then click the icon next to it.

For navigation, use the cursors below the phone's display and the numbers on the phone's keypad.

Get the software

Waprofit.com i-mode emulator is available for download on the official Waprofit site at <http://www.waprofit.com>

10.3.6 Testing the Voice application

A voice application can be best tested with the IBM WebSphere Voice SDK 1.5. It simulates a telephone interface on a desktop system by using the microphone and speaker of the system. Additionally, it provides a DTMF GUI to allow key input. For more details, please refer to 10.2.5, “Voice SDK” on page 180.

10.3.7 Other emulators

There are other emulators for several other mobile devices, not shown in this redbook. It was a difficult decision as to which one to choose, because there are several professional emulators with advanced features in existence.

PALM OS emulator

Palm OS is a well-know operating system for the Palm devices. This operating system (OS) was developed especially for handheld devices. It provides advanced functions and features for PDAs.

Palm OS provides several development environments to develop applications for this OS. This makes it easier to find the right application even for Web browsing, because the development community for Palm OS is very large and growing day by day.

Palm OS has many Web browsers with different presentations and features; it has WAP browsers as well, and it is worthwhile to mention that Palm has its own viewer which works together with the Web Clipping applications--the traditional Palm Web service.

The Palm OS emulator (see Figure 10-16 on page 191) runs the whole Palm platform in a Win32 environment, with additional features such as dynamic application upload, debugging, multiple ROM images, and so forth.

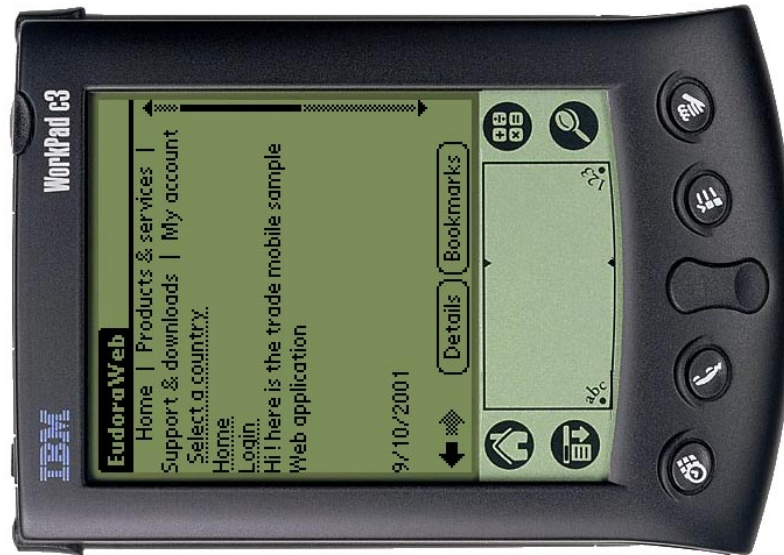


Figure 10-16 Palm emulator browsing an HTML site

Windows CE emulator

Microsoft has a Windows-like emulator for handheld devices called Windows CE, which runs on several devices. The OS provides the same functions and services as the Windows 95/98/ME.

It also runs several Web browsers, the most advanced of which is the Pocket IE, which is an Internet Explorer adaptation.

The WinCE emulator runs on Windows NT and brings the OS onto the desktop, with some additional features to help development and handle emulator functions.

EPOC emulator

EPOC is a young OS for handheld devices developed by Symbian. It now runs on several devices, such as all the PSION handhelds and the Nokia Communicators. The OS is specially developed for handheld devices and has capabilities to support the mobile applications.

The emulator runs in a Win32 environment as a fully functional OS.

Ericsson R380 emulator

The Ericsson R380 emulator looks like the other WAP phone emulators, but the difference is that the R380 originally runs EPOC. From the developer's standpoint, the built-in WAP browser is the most useful: it has the same behavior as the original browser on the phone. It is an ideal testing environment for this Ericsson phone with an advanced WAP browser.



Figure 10-17 Ericsson R380 emulator

This WAP emulator requires a WAP gateway set up on the network.

10.4 Best practices

The following sections discuss some of the best practices for certain situations, which we have found useful in our application.

10.4.1 Device management

When setting up a mobile Web application, you must deal with different devices and protocols. Examples are HTML, cHTML (i-mode), WML (WAP) and VoiceXML. Inside your application, you have to distinguish which device and which application to choose in order to select the appropriate markup language. The best way to do this is to take advantage of the User-Agent field from the request header and to use it as a selection criterion. The value for this field is unique for each browser model; the Trade2 application will also use this field.

The logic to distinguish between the devices can be placed in different modules of the application. One possibility is to use a controller servlet that automatically chooses the appropriate JSP file to produce the content. In this case, additional logic has to be added to this servlet. Another possibility is to use the WTP environment by setting up multiple profiles to match the different devices and

browsers. To select the correct markup language, these profiles can be used in combination with your View component (for example, XSL StyleSheets managed by the WTP). More details about the Profile Builder in WTP can be found in chapter 10.2.3, “Development tools in WTP” on page 176.

10.4.2 Session management

Session management is one of the key components of a Web application. It is used to identify a session between a client and the server. The server is able to store information about the session related to the client. This information can be, for example, the user ID of the client used for identification in every request. The most common way to implement session management is through the use of cookies. Cookies are small information units stored on the client side that can be accessed and modified by the server.

Other possibilities are URL rewriting and hidden fields. The URL rewriting technology stores the session information in the request. Typically, a session ID or identity is added, so the requested URL is unique for each session. The hidden field strategy uses hidden input fields to identify the session. Typically, as for URL rewriting, a session ID is added.

We recommend the use of cookies; it is the easiest approach and can be implemented with the least effort. For the most mobile devices, the support of cookies is available at the gateway. If the mobile device cannot handle cookies, the gateway can take over that function from the client, since the gateway handles the connection together with the session.

One example is the IBM WebSphere Voice Server. It allows storage and processing of cookies for the duration of the session. The most commonly used WAP gateways also provide cookie support. For the other solutions, additional implementation logic is needed.

10.4.3 Managing the content for different devices

The problems involved in developing a mobile solution are many. The most common problem will be that, depending on your application, you have to take care of the request and response flow between the client and the server. It may be that you have to add additional functionality to meet the requirements of the multiple devices.

One example is the registration forms. For HTML and WML browsers, it is just a form sent by one request to your device. After filling out the form, you easily send it back to the application using a Submit button or link. Implementing the same scenario for a voice application will produce a different flow, because you have to provide the grammar for the VoiceXML document in order to be able to recognize

the voice. The user friendly way would be to start with the city code and then provide the possible districts and streets in a new response. However, most voice applications do not offer a registration possibility. In order to use the service, the customer typically has to register on a Web page first, as in the Trade2 sample application.

Another problem with applications supporting different devices is the amount of data generated by the response. You can take the portfolio page of the Trade2 application as an example. The page contains a huge amount of data visually displayed as a table for an HTML browser. To match the capability of the mobile devices (maximum deck size as well as the screen size), you have to fragment the data. The best way to control this fragmentation is to add additional logic to the application. The output for the device will be generated in an appropriate form. This solution requires more effort during development.

Another possibility is to use the offered fragmentation feature of WTP. It helps to break down the amount of data and to adapt it to the browser's needs. All resulting fragments are stored locally in the cache of the WTP and are connected using links.

For fragmentation information, please refer to *New Capabilities in IBM WebSphere Transcoding Publisher Version 3.5 Extending Web Applications to the Pervasive World*, SG24-6233.

10.5 Development roles and responsibilities

The following roles are identified during mobile Web application development:

- ▶ Solution Architect - the person who designs the solution, including the architecture and the application.
- ▶ DataBase Designer - the person who designs and builds up the database, planning the database population, and planning for conversion or mass load.
- ▶ XML designer - the person who takes care of the XML documents, defines the grammars (DTD), matches the schema with the database structure, and ensures the interoperability between applications.
- ▶ Application developer - there are several type of application developers; in our case, they are Java developers. They provide the presentation and the business logic for the application.
- ▶ HTML Designer - mostly HTML coders, they take care of the ergonomic design, of implementing HTML files (JSP, XSL,...), and of coordinating the graphic designers.

- ▶ Other ML Designers - like the HTML designers, they have the same responsibilities, but their knowledge is based on other markup languages (WML, cHTML, ECMA Script, etc.).
- ▶ Voice Application Developer - this kind of developer is specialized in voice applications, and has a deep knowledge in VoiceXML.
- ▶ System engineer - this person builds up the infrastructure based on the architecture.
- ▶ Tester - this person tests the application and provides feedback to the developers and the designers.

10.6 More information

10.6.1 IBM related

- ▶ IBM Redbook: *The XML Files: Using XML and XSL with IBM WebSphere V3.0*, SG24-5479
- ▶ IBM Redbook: *Enterprise JavaBeans Development Using VisualAge for Java*, SG24-5429

10.6.2 Not IBM related

- ▶ More information on Java servlet, JSPs, JavaBeans and Enterprise Java Bean can be found on the following Web site: <http://java.sun.com/products>
- ▶ More information about XML, XSL and XSLT as well as HTML and cHTML can be found under the following Web address: <http://www.w3.org>



System management

This chapter will discuss system management for a mobile application based on WebSphere Everyplace Access V1R1.

System management is not a trivial task in any environment. This book covers the Web application part of the whole mobile solution.

11.1 General system management

Once the application development is complete and testing is done, it is time to start implementing the systems management phase. In this phase, application deployment into the runtime environment is the focus. It covers the set of post-implementation activities that have to be performed on a routine basis in system management.

System management typically involves the following:

- ▶ Application management
- ▶ Performance monitoring
- ▶ Availability management
- ▶ Security management
- ▶ Disaster recovery
- ▶ Operating system and network administration
- ▶ Asset management
- ▶ Software distribution
- ▶ Problem reporting
- ▶ Change management

Looking at this extensive list of activities, it is obvious that system management is a very important part of the whole process. In fact, each of these activities requires highly specific skills and professional experience to perform competently. System management also requires a set of tools to perform the tasks.

Incorporate system management considerations in the early phases of your design, since what you design will affect how you eventually manage it. Conversely, the tools available to manage your system also affect your application design. To explain each system management activity in detail is beyond the focus of this redbook. What we will focus on are the key system management activities related to applications based on WebSphere Everyplace Access V1R1.

11.2 IBM HTTP Server

IBM HTTP Server provides a Web browser-type administration client.

In order to access the Administration service, the IBM HTTP Administration service or daemon has to be started.

In Windows NT, open the service panel and navigate to the IBM HTTP Administration service; click push the **Start** button.

In AIX, open a terminal if you do not have one open, change the directory to `/usr/...`, then start the administrator daemon by typing in: `./adminctl start`.

Important: In some cases, the Administration service will not start, mostly because the configuration file needs some modification. Open the `admin.conf` file in the `conf` directory and add the following line:

```
ServerName <your server's fully distinguished name>
```

There is one more step: setting up the user name and password for the administrator. In order to do that, run the following commands:

In NT:

```
cd <IBM HTTP Server directory>
htpasswd -mb conf/admin.passwd <admin user name> <admin password>
```

In AIX:

```
cd /usr/HTTPServer/bin
htpasswd -mb ../conf/admin.passwd <admin user name> <admin password>
```

Once the service or the daemon is running, the administrator can access the Administration page on the server. To access the start page, use the following URL:

```
http://<your server address>:8008
```

Then type in the administrator user name and password which you set up previously.

The Administrator site is basically an interactive, graphical user interface for the configuration file (`httpd.conf`). You will find all the configurable options under this interface, as in the config file.

The advantage is that the configuration has a better view, via the GUI; furthermore, the administrator does not have to have access to the configuration file itself in the directory structure.

For more information about how to set up the IBM HTTP Server, use the help guide that comes with the server itself, accessible from the Administrator site.

11.3 WebSphere Application Server

WebSphere Application Server provides two different approaches to accessing administrative tasks.

- ▶ One is the WebSphere Administration console, which is a GUI for administrators.
- ▶ The other option is to use the XML configurator, where one can administrate the whole application server by writing plain XML files and feeding them into the XML configurator. The XML configurator will validate and process the config file and make the modifications to the application server.

To find out more about the XML configurator, please refer to the redbook *WebSphere V3.5 Handbook*, SG24-616.

11.3.1 Administration console

The Administration console for WebSphere Application Server is a GUI for administrating the application server. The administrator is able to access the server configurations and administrative tasks as well. The server configurations are structured as shown in Figure 11-1 on page 201. The structure is arranged into a hierarchy, and the navigation is an explorer-like navigation. Each level in the hierarchy represents an application server component; it is either a *container* for other components or a *resource* for other components.

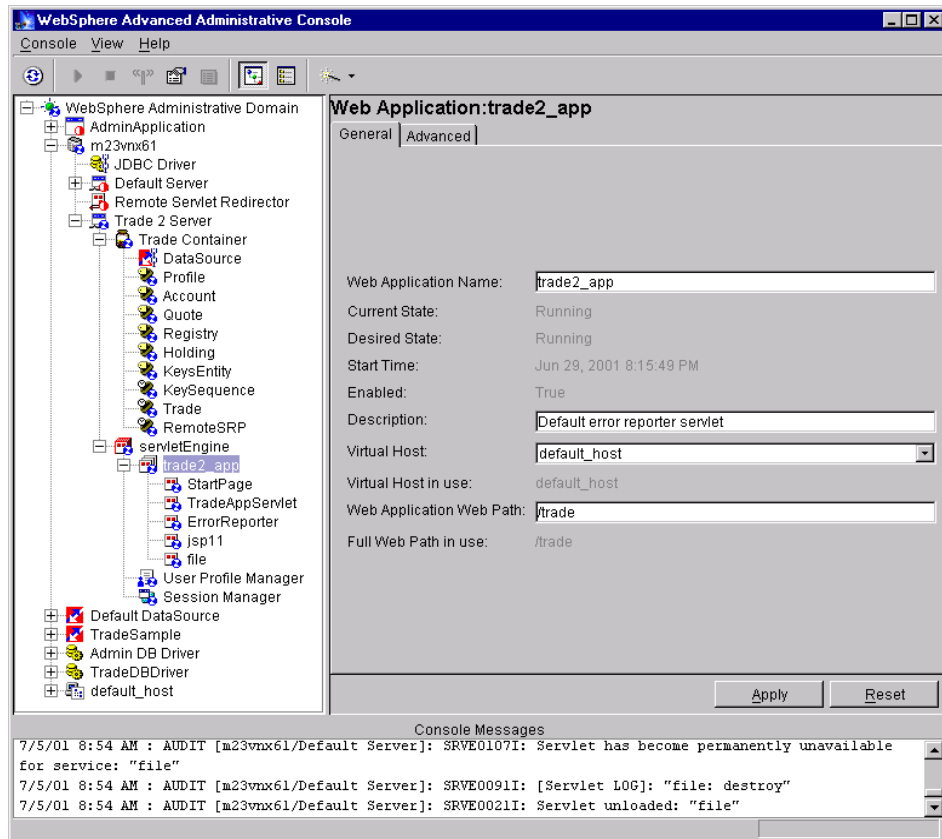


Figure 11-1 WebSphere server structure

The tasks are accessible from the menu or from the toolbar. Figure 11-2 on page 202 shows the tasks in the menu.

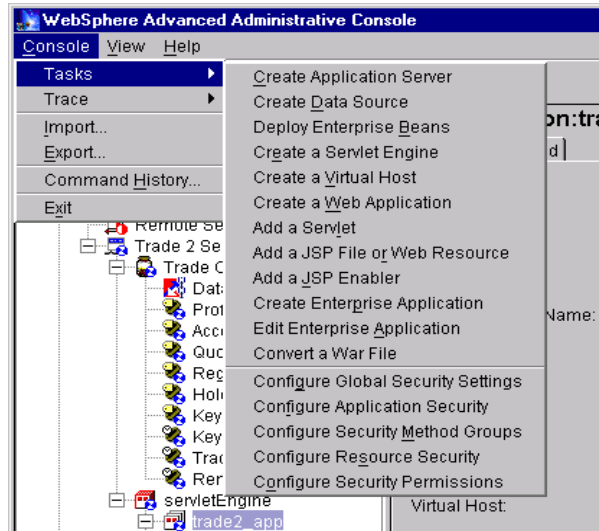


Figure 11-2 WebSphere tasks menu

The tasks, like wizards, guide you through windows and panels to do the job.

To find out more about the WebSphere Administration console, please refer to the redbook *WebSphere V3.5 Handbook* SG24-6161, page 811.

11.4 WebSphere Transcoding Publisher

WebSphere Transcoding Publisher, just like the WebSphere Application Server, has a GUI administration console. The purpose is the same: to administer the server, access the current configuration and modify it.

Since WTP can run on multiple nodes to increase performance and availability, there is a need for a centralized configuration. WTP allows you to store the configuration settings in a common directory, where they are shared by the WTP nodes.

11.4.1 Administration console

The primary tool for administering IBM WebSphere Transcoding Publisher is the Administration console. The console is a flexible program that adapts itself to your server configuration and the location of your configuration data.

The IBM WebSphere Transcoding Publisher Administration console provides the following functions:

- ▶ A tree view of your resources (preference profiles, annotators, transcoders, and StyleSheets)
- ▶ The ability to edit individual resources
- ▶ A menu structure to provide access to other tasks
- ▶ Help for each window, and a list of **How do I...** topics

To find out more about the WTP Administration console, please refer to the redbook *New Capabilities in IBM WebSphere Transcoding Publisher Version 3.5 Extending Web Applications to the Pervasive World*, SG-24-6233.

To start the Administration console in Windows NT and Windows 2000, select **Start -> Programs -> IBM Transcoding Publisher -> Administration Console**.

In AIX, Sun Solaris, and Linux, change the directory to the WTP's bin directory, then enter `AdminConsole.sh` in the command line.

The Administration console is shown in Figure 11-3:

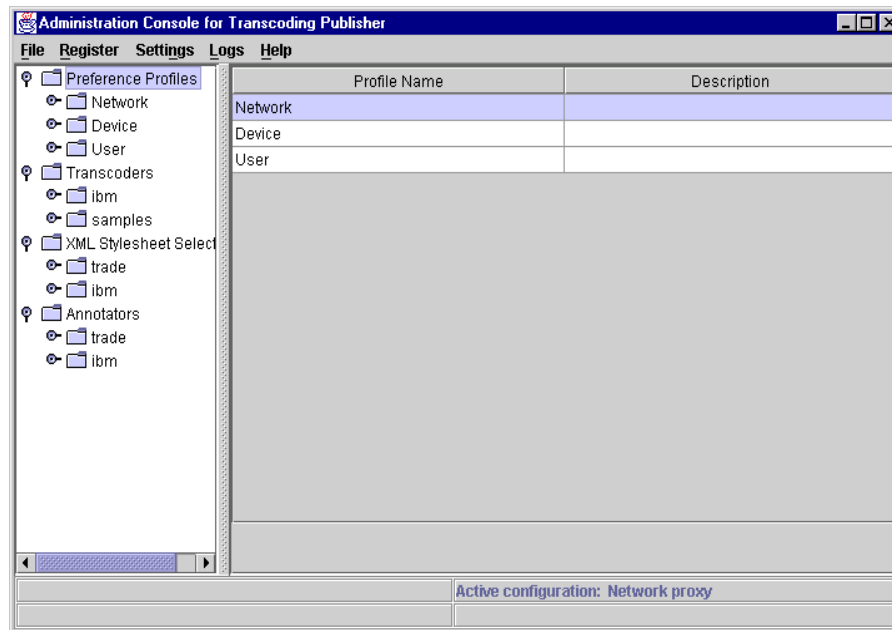


Figure 11-3 WTP Administration console

11.4.2 Common LDAP directory

Depending on the needs of the architecture, you can choose to store the configuration information for each IBM WebSphere Transcoding Publisher server on that server's machine, or you can use a central directory, IBM SecureWay Directory, to store one or more server models to be used by your IBM WebSphere Transcoding Publisher servers. Using a central directory enables you to define common configurations to be used by several servers and to maintain them from an Administration console anywhere in your network. Then, if you need to make a change to the shared information (for example, if you need to add a new StyleSheet or modify a device profile), you can make the change to the server model; all IBM WebSphere Transcoding Publisher servers that use that server model will recognize the change.

If you are using a central directory to store server models, you must log in when you start the Administration console. The user ID and password will be verified by the central directory. If your login fails, you cannot work with server models. However, you can work with local server settings without logging in.

Find out more about the common LDAP directory for WTP in the redbook *New Capabilities in IBM WebSphere Transcoding Publisher Version 3.5 Extending Web Applications to the Pervasive World*, SG24-6233, page 212.

11.5 WebSphere Voice Server

After a Voice Server is powered up, the software installation is complete, and the system is rebooted, the Windows NT Service starts the System Management component. The System Management component reads the sysmgmt.properties configuration file. The sysmgmt.properties file is the configuration file for the WebSphere Voice Server (WVS).

The WebSphere Voice Server does not have a GUI for administrative purposes, unless we consider the TIVOLI GEM management tool as a GUI for WVS. WVS offers a command line program: **vvsm**, which can perform administrative tasks.

For example you can start the VoiceXML browsers by issuing the **vvsm start** command.

To find out more about administrating the WebSphere Voice Server, please refer to the documentation included with IBM WebSphere Voice Server Version 1.5 Administrator's Guide.



Security

This chapter discusses the common security issues related to mobile Web applications.

The WebSphere Everyplace Access offering concentrates on the Web applications, leaving the system management and security issues for the underlying architecture.

The following sections will cover the basic security considerations which are applicable to any Web application. Furthermore, we will cover the issues specific to mobile applications.

12.1 Introduction

The constant growth of the wireless market has triggered a rapid increase in the number of services and applications for mobile devices such as cellular phones, PDAs or handheld PCs (HPCs). Consequently, there has been an increased interest in developing new architectures and protocols to meet the requirements of these applications. One of the requirements for wireless communications is security. The last years have registered a growing market for messaging, electronic payments and e-mails (to mention just a few) as well as a great demand for safe transactions.

Cryptography offers virtually unbreakable solutions (“virtually” because the algorithm to reverse the coding is known but is computationally unattractive at present), but the existing architectures need to adapt to this new form of communications. As a result, several extension packages have been created to support wireless communications on existing cryptography systems.

Wireless security is difficult to implement, because many different aspects need to be managed. Currently, there is no single standard for the security of wireless transactions; industries rely mainly on Secure Socket Layer (see 12.3.1, “Security Socket Layer (SSL)” on page 217) and Wireless Transport Layer Security (see 12.3.2, “Wireless Transport Layer Security (WTLS)” on page 218). Unfortunately, the two protocols are not compatible and the encoded data needs to be converted into an SSL format in order to be processed by a conventional network. The re-encryption is usually done inside the gateway by decoding the original WTLS data and coding it using SSL before passing it to the server. This feature creates vulnerability in the communications between the wireless device and the back-end application, because the data is available in plain format on the gateway for a short time. In some cases, the problem has been avoided by placing the gateway within the conventional network (hence relying on the security of transactions in the server domain). However, some security threats still exist.

Wireless devices, and in particular WAP phones, lack the CPU and memory power typical of standard systems; therefore, security algorithms need to be run on limited resources. For this reason, SSL cannot be implemented for communications between the wireless device and the gateway as the available computational capabilities are not sufficient to run the RSA encryption algorithm. A different approach is used instead. Wireless transactions have their own security protocol, WTLS, which provides only some of the same security features via a less resource-intensive encryption algorithm like ECC (see 12.3.4, “Elliptic Curve Cryptography (ECC)” on page 219).

Security on its own does not prevent malicious access to data on wireless devices.

Viruses are becoming a growing area of interest for mobile communications because of the current vulnerability of the devices. The widespread nature of wireless technology, combined with the continuous use of its associated devices, has increased the risk for virus attacks. Virtually, it would be possible for a virus to make transactions using somebody's else device without the owner realizing it (at least at the time the violation actually occurs).

12.2 Mobile versus conventional communications

This section gives a brief overview of the differences and the similarities of security issues and their solutions between HTTP communication, non-HTTP wireless communications and voice. We discuss the differences between them and refer you to relevant literature or Web sites for further details.

Security practices focus on five different areas:

- ▶ Authentication
- ▶ Confidentiality and message integrity
- ▶ Authorization
- ▶ Non-repudiation
- ▶ Secure boundary

Under these requirements, we will consider the following scenarios:

- ▶ HTTP clients
- ▶ Clients connecting to the network by modem
- ▶ Wireless gateway clients
- ▶ Non-HTTP wireless clients
- ▶ Voice clients

12.2.1 Authentication

We will now discuss the use of standards concerning client and server authentication. These can be complemented by authentication done at the application level.

HTTP clients

By HTTP clients, we mean traditional, desktop PC clients with a Web browser. The security used for this type of client is well known; we briefly recapitulate the main points of focus in this field.

Server authentication

The server authentication is routinely done with an X.509 server certificate in the context of an SSL or a TLS connection; these technologies use PKI (Public Key Infrastructure).

Client authentication

Typical alternatives for user authentication in standard HTTP communications include:

- ▶ An HTTP authentication challenge, as defined in the HTTP specification by the World Wide Web Consortium (refer to RFC 2617 on www.w3c.org). This can be a *basic* (only slightly protected) or *digest* (more protected) authentication. Such a challenge can be issued by an HTTP server (in which case it is called a 401 www Authentication request) or by a proxy (which corresponds to a 407 Proxy Authentication request).
- ▶ Client X.509 certificates signed by a trusted third party. Using client certificates is considered to be much more secure than HTTP authentication challenges, provided the trusted third party has followed stringent procedures to verify the physical identity of the user. This can be implemented within the SSL3 or TLS standards.

Clients connecting to the network by modem

Clients, either wireless or wired-connected through a modem, will typically be authenticated between the client and the entry point to the network by extensions to the PPP protocol. These protocols are CHAP (Challenge-Handshake Authentication Protocol) or PAP (Password Authentication Protocol); see RFC 1334 and RFC 1994 on www.ietf.org for more details.

Of course, within that link, SSL or TLS can be used. One could also use WLP (see below), which requires some software on the client's part.

Wireless Gateway clients

For the Wireless Gateway clients (a specific Wireless Client component of Everyplace Wireless Gateway for wireless devices using an IP link), the authentication is done through the Wireless Optimized Link Protocol (WLP), which is a modified version of PPP, optimized for wireless use.

WLP (formerly called the ArTour protocol) was created because PPP as such requires a high number of interactions before the link is actually established, which may be slow.

In WLP, the number of packets exchanged when the connection is established is brought down to six for the logon and to four for the subsequent calls, as long as the request comes from the same IP address. In addition to that, the IP headers are also encrypted. Additionally, the exchange of passwords is better protected.

Note: WLP was developed by IBM. It is currently not a generally accepted standard, but is available as part of the Wireless Gateway client software, and is heavily used, for example in the automotive industry and on different laptop systems.

Non-HTTP wireless clients

Non-HTTP wireless clients, such as WAP or i-mode clients, are authenticated using their specific protocol.

Server authentication

For WAP, within the WTLS specification (for a description of WTLS, see 12.3.2, “Wireless Transport Layer Security (WTLS)” on page 218), the server authentication via certificates is recommended. Possible certificates are X.509 or mini certificates.

Note that the term *server authentication* in practice here means that the WAP gateway is authenticated to the client, not the Web-server behind it. The latter can server-authenticate to the WAP gateway in the context of an SSL connection. This is illustrated in Figure 12-1.

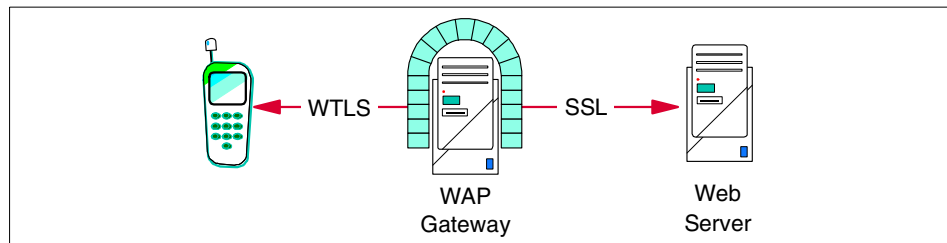


Figure 12-1 Current secure connections for WAP

A change in this setup has been announced with the implementation of the 3G networks, where, for securing connections, the use of SSL will be recommended between the device and the Web server. The server will directly authenticate to the client device, as illustrated in Figure 12-2 on page 210.

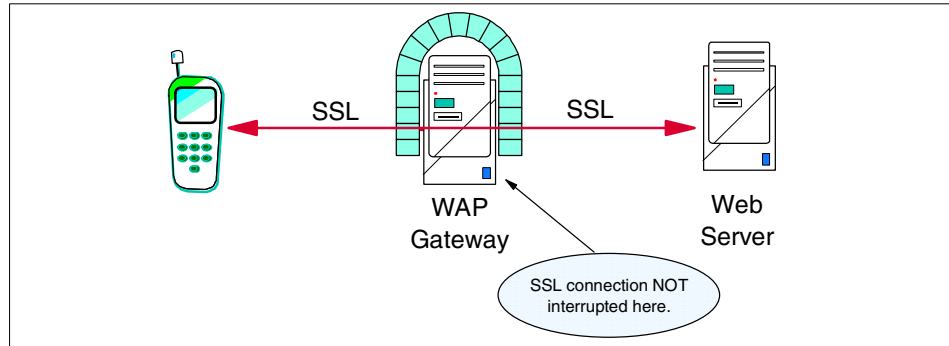


Figure 12-2 Possible secure connections for WAP

For i-mode, we make a distinction between the models predating the 503i-series and the 503i-series themselves.

For the models predating the 503i series, there is no security such as WTLS between the device and the gateway. Only SSL is possible with server authentication between the Web server and the gateway, if requested. This means that the Web server can authenticate to the gateway, nothing more (see Figure 12-3).

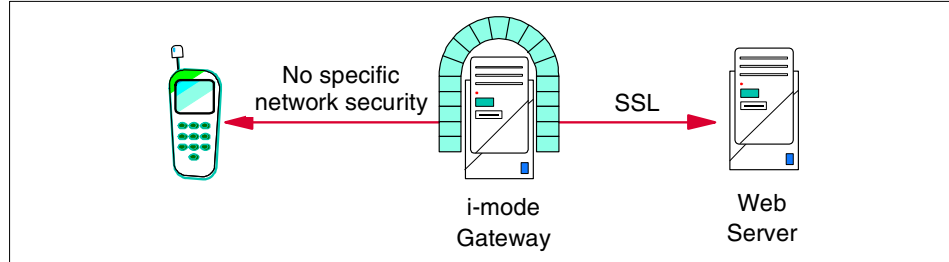


Figure 12-3 Current secure connections for i-mode, for models prior to the 503i-series

From the 503i-series on, SSL will be possible from the client device to the Web server, and so it will be possible for the Web server to authenticate directly to the client device. This is a similar situation to the one described for future (3G) WAP connections. Two certificates are stored in the 503i-series before shipment; they are issued by VeriSign and Baltimore (see Figure 12-4 on page 211). Please note that this is only an option. One can still use the device without specific data-level security, such as in models prior to the 503i series.

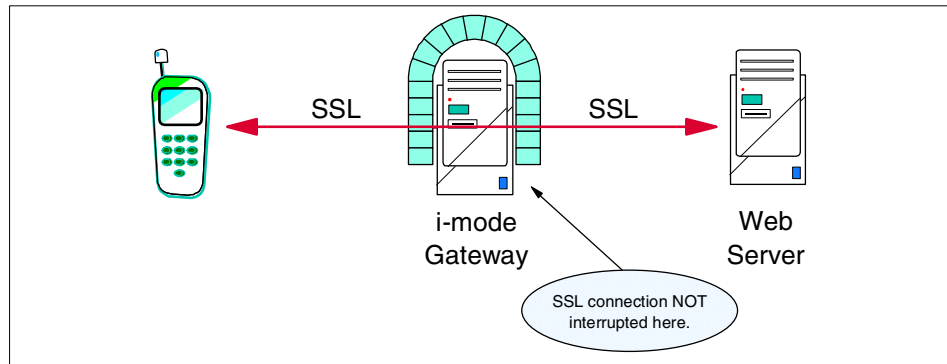


Figure 12-4 Possible secure connections for i-mode, for 503i models

Client authentication

In the case of WAP clients, the same authentication challenge mechanism as that of HTTP 1.1 applies - it is now a mandatory part of the WAP 1.2.1 specification - but alternative authentication methods are available. Still in the case of WAP clients, the implementation of client-side certificates, as described by the WTLS specification (see “Server authentication” on page 209), is still in a prototype phase at the time of writing this book.

In this area, the Wireless Identity Module (WIM) specification of the WAP forum is worth mentioning. This gives specifications about a device, possibly implemented on a smartcard, containing the private key of the client and the ability to calculate the encryption keys necessary to secure data transmission. Optionally, it also contains the CA-roots, the entire client certificate, or a URL pointing to a page where the client certificate can be found. It is technically possible to store the client certificate in a centralized directory, but the private key has to stay with the client device. It must not be exposed.

For i-mode, the *device identifier* can contribute to client identification. For models prior to the 503i series, this identifier had to be generated by the gateway and passed through to the content provider. In the case of the 503i-series devices, it has been announced that they will be able to transfer their own device identifier directly to the Web server; for more information, see the redbook *Mobile Commerce Solutions Guide, using WebSphere Commerce Suite V5.1*, SG24-6171.

Voice clients

To discuss voice communications, we start from the infrastructure shown in Figure 12-5.

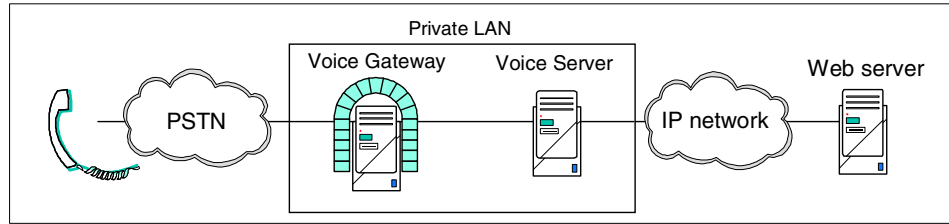


Figure 12-5 Schematic representation of voice infrastructure

If the VoiceXML browser, contained in the voice server, supports SSL with server authentication as a client, then authentication of the Web server to the voice server is possible.

This is the case of the IBM VoiceXML browser. It contains a number of predefined trusted roots.

Under this scheme, when a VoiceXML browser connects to a Web server providing the VoiceXML application, it will receive the certificate of the Web server and validate it against one of the voice browser's trusted roots. This is especially useful if the Web server is situated outside the private LAN and has to be accessed via the Internet.

There is no standardized scheme available for client authentication, so it will have to rely on application-level authentication. This can be a VoiceXML application requiring a user ID and password, but it should not exclude other alternatives in the future. For example, some recent evolutions in biometrics include speaker identification, which is a technique that contributes to user authentication based on the characteristic features of the speaker's voice.

12.2.2 Confidentiality and message integrity

This section discusses the standards for confidentiality and message integrity.

HTTP clients

To enable encryption of the dataflow between Web server and Web client, implementing SSL or TLS, even if it will only be used with server authentication, is the de facto standard procedure.

Note that encryption is also applied in the SSL connections that have only server authentication.

During the initial SSL connection, a symmetrical encryption key is constructed and made available on both sides of the connection; this key is used to encrypt the dataflow. Symmetrical (also called secret key) encryption technologies are used in SSL and in TLS.

Also in SSL and TLS, a hashed version of the data is sent together with the data itself to prevent message tampering, that is, modification of data during the transmission. Upon arrival of the data, the hashed version is compared to a hash of the received data, to check for message integrity.

Note: By hashing we mean the cryptographic function that transforms a piece of data into a short, fixed-length piece that represent the original data. Since two even slightly different pieces of data generate totally different hashes, the function is explicitly designed to make it impossible for a given hash to find the corresponding data.

Clients connecting to the network by modem

For clients connecting to the network via PPP, that standard does not, as such, provide encryption.

Of course, as was mentioned under “Clients connecting to the network by modem” on page 208, SSL or TLS can be used within that link.

Wireless Gateway clients

Wireless Gateway clients are clients that use an WLP connection, which we discussed earlier. WLP has an option to encrypt the data by using either DES or triple DES.

Please note that all this takes place at the link level. There is also an option to use SSL at the data level (socket layer). If, in addition to this, the option to encrypt the data using WLP was chosen, the data will be encrypted twice.

Non-HTTP wireless clients

Non-HTTP wireless clients, such as WAP or i-mode clients of models from the 503i series on, can also apply encryption.

For WAP connections using the WTLS specification, following encryption algorithms is possible (DES/TripleDES, IDEA, RC5). The same hashing algorithms as for SSL and TLS are mentioned in the specification.

Note: Currently, encryption stops at the WAP gateway (if a gateway is used). In other words, there is no end-to-end encryption between the WTLS client and the HTTP server. This problem and its solutions are mentioned in 12.4.1, “Broken secure connection in the WAP gateway” on page 222.

For i-mode clients, as mentioned in “Non-HTTP wireless clients” on page 209, the devices prior to the 503i-series have no specific network security (and hence no encryption like the one provided by SSL or WTLS) between the device and the gateway. Between the gateway and the Web server, SSL with 40 bit encryption is possible.

For the 503i series, SSL with 128 bit encryption is applicable as an option between the device and the Web server.

Voice clients

We only focus here on the segment between the voice browser and the Web server. If SSL is supported by the voice browser (which is the case, for example, for the IBM Voice browser in the Voice Server), then logically the packets will be encrypted there.

Encryption methodologies and standards also exist for audiovisual signals transmitted digitally between the Voice Over IP gateway and the Voice server (see for example the standards H.234 and H.235 of the International Telecommunications Union), but these topics are beyond the scope of this book.

12.2.3 Authorization

Authorization, also called *access control*, is used to restrict the user's authorized access to resources and applications.

A first authorization can be performed by the proxy or the Web server by defining a protection space and comparing it with user profiles and access control information stored in, for example, a directory using LDAP.

Authorization can further be refined at the application level.

In general, there are two fundamental approaches to control access: *permissions* and *capabilities*. The permission list stores a list of users, groups, or roles and what they are permitted to do. The capability list associates a list of resources and corresponding privileges with each user.

12.2.4 Non-repudiation

Non-repudiation means that a user, having launched and approved a transaction or sent a message, cannot deny this action afterwards (cannot “repudiate” it).

Non-repudiation can be implemented using digital signatures. When digitally signing something, we are actually using the elements that are part of the Public Key Infrastructure framework (see 12.3.5, “Public Key Infrastructure (PKI)” on page 219).

Digital signatures

This section is a short and basic reminder of the principle of digital signatures.

As an example, let us consider Figure 12-6; person A wants to send a signed document to person B. A hash is made from the document, and encrypted with the private key of person A. The signed hash (which is, in fact, the digital signature of the document) and the plain original document are sent to person B.

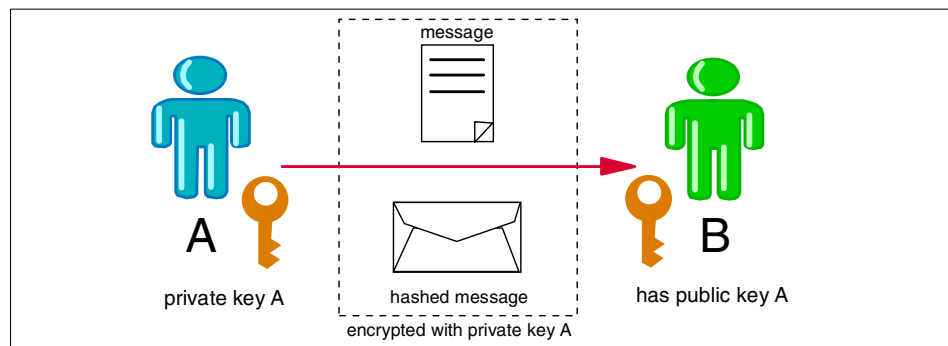


Figure 12-6 Creation of a digital signature

Figure 12-7 on page 216 shows the digital signature verification process.

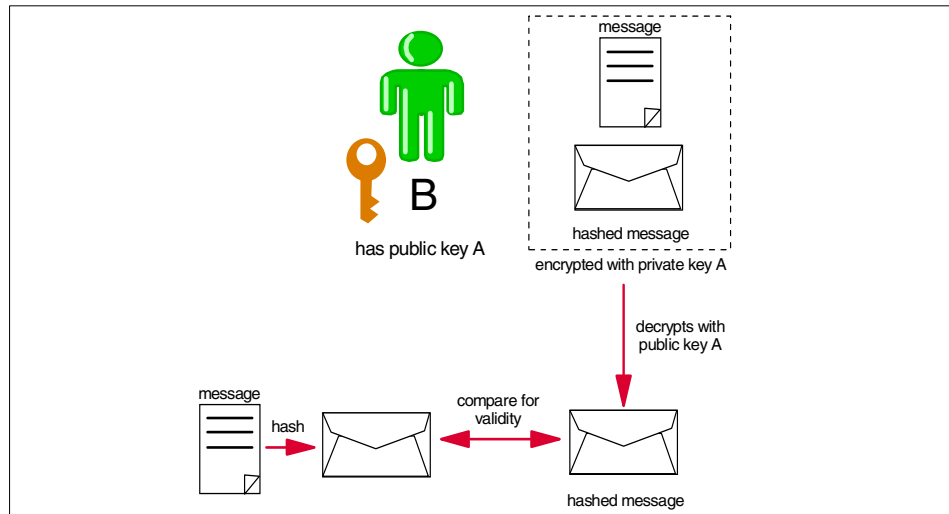


Figure 12-7 Digital signature verification

Person B uses the publicly available public key to decrypt the signed hash (the digital signature). After that, a hash is calculated from the plain original document and compared with the decrypted hash.

The validity of this relies on the fact that only A owns a private key and could have signed the hash. Person A cannot claim that the contents of the documents have been modified during the transition to person B, because otherwise the hash would not correspond to the document. In this example, we assume that the recipient is sure of the identity of the sender. This can be achieved via a valid certificate.

Applicability of digital signatures

As discussed above, a private/public key pair is essential for creating digital signatures. Protocols using PKI, such as SSL and TLS, work with such key pairs. Simply using SSL or TLS is not enough, however. Those standards only contain elements to construct electronic signature mechanisms, which is not done automatically.

For voice clients, digital signatures using PKI as such are not yet available, so validation of the transactions should be implemented by other means, such as application development, possibly combined with other authentication methods.

For more information about Public Key Infrastructure, read Section 12.3.5, “Public Key Infrastructure (PKI)” on page 219.

12.2.5 Secure boundary

Secure boundary is the ability to protect resources of a private network. This is achieved by firewall software, with or without proxies. Typical functionality in this respect includes the screening of incoming packets according to their IP-address and TCP port number, concealment of addresses from the outside world and tracking of activity so as to be notified of any suspicious behavior.

12.3 Security technologies for wireless transactions

In the following sections, we briefly discuss the algorithm of the most popular security system used for wireless transactions. Implementation details and other types of strictly technical information are not addressed in this book, and the reader can refer instead to the literature (books, articles or Web sites) mentioned.

We will first give a brief overview of wireless communications security. Security occurs between the wireless devices and the network, and between the network and the back-end applications.

12.3.1 Security Socket Layer (SSL)

SSL is a layered protocol that provides a private, reliable connection between parties where the peer's identity can be verified. It was developed by Netscape and submitted to the IETF (Internet Engineering Task Force) as a model for TLS (Transport Layer Security), while a different protocol named PCT (Private Communication Transport) was developed by Microsoft. However, SSL has become widely accepted and it is not only used to secure HTTP transactions with a HTTPS request. SSL's major components are the Record Protocol and Application Data Protocol. The former is used to encapsulate higher level protocols such as the handshake protocol. The latter is used to encrypt data with an algorithm agreed upon during the handshake process.

When a request for a connection is sent from a client to a server, a predetermined sequence of messages is exchanged between the two devices in order to allow further communications. Hence, after a session has been established, the handshake process is started.

SSL seeks to provide:

- ▶ Cryptographic security between two parties connections
- ▶ Interoperability to allow the exchange of information between parties without releasing any code knowledge

- ▶ Extensibility, to enable the addition of new encryption methods if necessary
- ▶ Efficiency, by minimizing the network connection via a caching scheme.

SSL performs authentication using certificates. A certificate contains, among other items, the issuer's name and the public key of the host for which the certificate has been issued. When a system needs to be verified, some data is sent to it and it must be correctly encoded using the certificate.

12.3.2 Wireless Transport Layer Security (WTLS)

WTLS is the wireless extension of the TLS protocol. TLS was developed to standardize the use of SSL. It mainly consists of two layers: the handshake protocol and the record protocol. The handshake protocol sets up a connection using a public key and negotiates the use of a symmetric key for the communications, as well as a compression algorithm, if necessary. The record protocol is directly connected to the transport layer and breaks up incoming messages into blocks. It encodes each block of data using the symmetric encryption algorithm that was agreed upon during the handshake process. The record layer also performs a data integrity check and finally sends the coded block to the transport layer.

WTLS provides security components for authentication, data integrity and confidentiality, while a PKI accounts for the missing ones. The wireless extension to TLS is needed to support datagrams in a low bandwidth and high latency environment, in order to ensure that the handshake process has been optimized with the introduction of dynamic key refreshing. This allows the encryption keys to be updated regularly, resulting in a higher level of security and a faster handshake process.

Note: A datagram is a self-contained piece of data that carries sufficient information to be transmitted between a source and a destination device (typically computers). The data is independent of network settings and any previous exchanges between the devices.

12.3.3 Public key cryptography

The RSA Public Key Cryptosystem was invented in 1977 at the Massachusetts Institute of Technology by Ronald Rivest, Adi Shamir, and Len Adleman.

Rather than using the same key to both encrypt and decrypt the data, the RSA system uses a matched pair of encryption and decryption keys. Each key performs a one-way transformation of the data and has the inverse function of the other; what one key does, only the other can undo.

The RSA Public Key is made available to the public by its owner, while the RSA Private Key is kept secret. To send a private message, an author scrambles the message with the intended recipient's Public Key. Once encrypted, the message can only be decoded with the recipient's Private Key. Inversely, the user can also scramble data using his/her Private Key; in other words, RSA keys work in either direction. This provides the basis for the digital signature, because if the user can unscramble a message with someone's Public Key, the other user must have used his/her Private Key to scramble it in the first place. Since only the owner can utilize his/her own private key, the scrambled message becomes a kind of electronic signature: a document that nobody else can produce.

For more information about RSA, refer to the following Web site:
http://www.rsa.com/rsalabs/rsa_algorithm/index.html

12.3.4 Elliptic Curve Cryptography (ECC)

The security of existing cryptosystems relies on two hard mathematical problems: the discrete logarithm problem (over a finite field) and integer factorization. Recently, researchers have introduced the use of elliptic curves in the definition of cryptosystems. Elliptic curves are mathematical structures that have been successfully used in a variety of applications and have recently attracted a significant interest in the field of security. The elliptic curve discrete logarithm problem requires extensive resources to be found and used for its solution and it is, therefore, practically impossible to break a cryptographic system based on this problem.

The algorithm is more complex than RSA and requires a significant number of system parameters. According to some recent studies, the time required to break an ECC system grows exponentially with the size of the key, while for an RSA modulus the growth is only logarithmic. For details, please refer to <http://www.certicom.com/research/wecc3.html>

12.3.5 Public Key Infrastructure (PKI)

Security for wireless communications can be achieved by using a dedicated infrastructure. This is the case of PKI, a system combining public key cryptography, certificates and a distributed server system to provide the following aspects of security:

- ▶ Identification
- ▶ Authentication
- ▶ Confidentiality
- ▶ Non-repudiation
- ▶ Data integrity

A distributed server system is simply an architecture in which servers distribute applications to different types of clients in a transparent way. The distribution takes place over a network. Typically, a root Certificate Authority (CA) issues certificates to different types of servers in different locations. The servers are then accessed remotely by clients over the Internet.

How are the different aspects of security implemented in a PKI system?

Authentication

Site authentication can take place in two different ways under PKI. In one case, the server proves its identity using a certificate. In the other case, the client is sent a certificate that needs to be installed and used for further connections. Certificates are issued by a CA upon request by the organization responsible for the servers. A representative of the organization needs to prove his/her identity, for example by going personally to the local registration authority for certificates. The CA, depending on the original request, provides a *server authentication certificate* or a *signing certificate*. In the former case, the certificate is used to identify the server when a client requests a connection. In the latter case, the server has been delegated by the CA to authenticate clients.

When a connection between two parties is established, the identity of the party requesting access must be verified. This is normally done using a certificate. A certificate implies an underlying form of trust that the person at the other end is in fact who he/she claims to be.

The threat that someone may step between the two communicating parties and captures the communication, is called man-in-the-middle. PKI provides the secure infrastructure to eliminate this threat. CAs play the role of the trusted third party, and the two communicating parties identify each other via the CA. Standing between two communicating parties is always a threat; either it is between two individuals, between two CAs or between a CA and an individual, but with PKI it is possible to decrease the security risk to almost zero.

Setting up a secure communication always requires a certain level of trust; PKI can reduce the risk by ensuring a high level of trust.

Authorization

Trusted connections from mobile client devices may have different permissions to access information on the server. That is why, when a request for a connection is made, the server verifies the authorizations of the clients to determine what resources and actions are permitted.

In a PKI, access to server information is only granted to authorized parties by means of certificates. These certificates contain two pieces of information: the ID of the keyholder and the server DNS, which are issued by CAs. Certificates can also provide information about authorization by enclosing information about the accessible resources for that particular certificate holder.

Non-repudiation

Non-repudiation is ensured by exercising the Digital signature technique. For more information, please refer to 12.2.4, “Non-repudiation” on page 215

Confidentiality

Data sent in a PKI is kept confidential by the use of symmetric keys for encryption and decryption. These keys are constructed at the beginning of each session, based on a secret message sent by one of the parties involved. That secret, previously encoded with the receiver’s public key, can only be decrypted using the receiver’s private key. Because the content is encrypted with the resulting symmetric key, the receiver is the only authorized person to have access to the information.

Data integrity

Information that is sent over the network can be corrupted and may not reach its destination in its original form. PKI provides a way to verify data integrity. The receiver applies the same hashing algorithm used by the sender to generate a hash of the data file. Then, the hash sent with the data file is compared with the one generated at the receiver’s end. If a mismatch is found, this is an indication that a data loss has occurred.

12.4 Apparent problems in wireless security

The following sections will shed light on a couple of specific security problems in mobile Web applications.

We will discuss the problem in depth and recommend solutions.

12.4.1 Broken secure connection in the WAP gateway

If there is a broken secure connection, the WAP gateway opens a security hole in the communication.

The problem

If we look at Figure 12-1 on page 209, we notice that there are two secure connections: WTLS and SSL. The encrypted data from WTLS has to be decrypted in order to be re-encrypted under SSL, and vice-versa. The reason is that SSL is defined at the transport level (the “TCP” in TCP/IP), and that transport level protocol is different from the one in WAP.

The consequence of this is that at one moment the data may pass through the gateway, while at other times or in some contexts this might cause a security issue.

The solutions

The following paragraphs are discussing the possible solutions and workarounds for the problem described above.

Current solution

The current fully secured solution is to have the data on a server that serves WAP requests directly, a so-called *WAP server* (the official term is “WSP-server”). In that case, there can be only one WTLS connection from the client device to the server.

Obviously, in the majority of cases one would like to reuse the resources of the Web server as well, and hence use a WAP gateway.

Short term recommendation

In the short term, a WTLS specification will come out (if it has not come out by the time this redbook is published), that will recommend redirection in this case. This means that requests, going to a WAP gateway (in this scenario, typically a public one) to a site that wants to establish a secure connection, are sent back to the client device for redirection to the destination site. At the destination site, a WTLS connection can be established with the WAP gateway on-site; this is considered to be more secure than the public connection. This is illustrated in Figure 12-8.

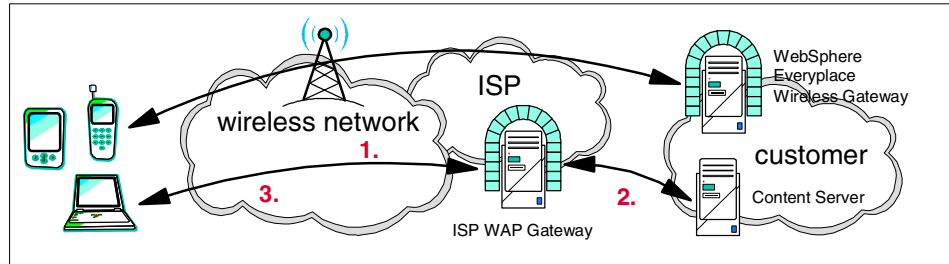


Figure 12-8 Recommendation in the short term for securing WTLS connections

1. The WAP client sends a request to the ISP WAP gateway for the customer's site.
2. The customer's content server responds that it wants a secure connection and gives the ISP gateway routing instructions.
3. The ISP gateway sends a navigation document to the client device for the customer's site.
4. The client redirects its call directly to the WAP gateway of the customer, so that a direct WTLS connection is established between the client device and the WAP gateway on the customer's site, which is considered to be more secure.

Solution for future networks

The WAP recommendation for future networks is expected to be a straight end-to-end SSL connection between the client device and the server, as shown in Figure 12-2 on page 210.

The reason for this is that in 3G networks, the challenges posed by poor network reliability and throughput will have at least partially disappeared; in other words, the connection will behave more like a normal network connection, so that the heavier SSL will be less of a problem.

12.4.2 Other gateways

The architecture introduced in Figure 12-1 on page 209 has some disadvantages in certain circumstances.

Development environment

Assume that the development environment within the company has its own network segment, which connects to the Internet via a firewall. This is a normal situation.

During development, in most cases developers use emulators to test their application. These emulators work like the original devices and require the same environment and settings to operate.

Problem

In some cases, the original devices and the emulators require a kind of proxy, simply called a *content proxy*. The reason why the client requires the proxy is that the client cannot use the cipher algorithms that are widely used on the Internet (for example SSL), because they do not have the computing power to perform the calculations. The client uses a different type of secure connection to a proxy, using other cipher algorithms (for example ECC). The other advantage is that the proxy and the client can use a different connection than TCP, and the UDP connection requires less energy from the clients, which saves the battery. In the case of PALM, the content proxy also does transcoding for the PALM browser.

The client connects to the content proxy instead of the target server; the proxy then makes the connection to the target server to retrieve the content and sends it back to the client.

The problem is that these proxy servers are outside of the internal network, on the Internet, and the client cannot connect through the firewall, because the required port and connection are filtered out.

Solution

There are several solutions to this problem, but each solution requires some changes in the network architecture or configuration.

Tip: If the internal network uses SOCKS servers, then the clients can be SOCKSified and can go through the SOCKS server out to the external network (Internet). This kind of proxy gives more freedom to the clients.

Server certificates

Based on the previous architecture, developers might want to connect securely to the Web application located within the internal network.

Problem

The problem is not only that the clients have to go through the content proxy, which does not have the right to access the server on the internal network; the secure connection involves another problem. Since the client uses a different security protocol than required, it cannot handle certificates, so the content proxy has to accept the target server's certificate. This means that either the content proxy has to know the server's certificate, or the target server has to have a certificate issued from a trusted CA.

The problem is that during development, servers have a dummy, self-signed certificate which is not acceptable to the content proxy.

Solution

The solution for this problem is to use real certificates, or to acquire a temporary certificate from a trusted CA.

Another solution is to have the content proxy in-house and register the self-signed certificates with it.

A third solution is to have the testing machines the clients and the servers out on the Internet. The deployment during the development can be very problematic.

12.4.3 The WebSphere Transcoding Publisher and encrypted content

WebSphere Transcoding Publisher as a Web intermediary brakes the secure connection between the application server and the client.

Problem

The WebSphere Transcoding publisher cannot handle encrypted content. This is logical, since in order to transcode content one has to be able to access the content as it is.

This means that if the transcoder resides on a separate node from the Web server, the data between the transcoding publisher and the Web server is not encrypted.

Solution

To solve this problem, WTP has to be configured as a MIME type filter.

When WTP runs as a servlet filter, an uninterrupted SSL connection can be established to the Web server, and so the data is encrypted end-to-end.

Note: Configuring WebSphere Transcoding Publisher as a filter is only possible with WebSphere Application Server.

To clarify this concept, see Figure 12-9 on page 226.

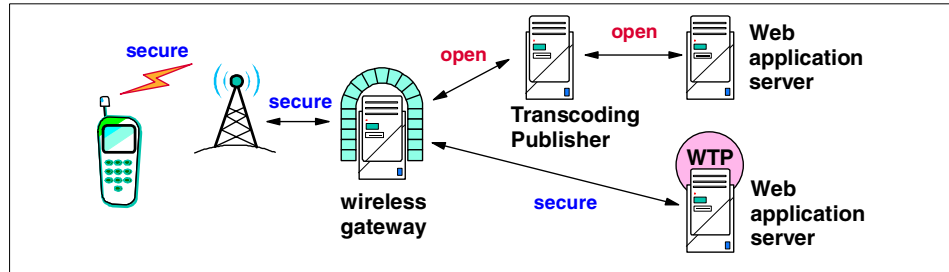


Figure 12-9 WTP security issues

12.5 More Information

- ▶ About Elliptic Curve Cryptography (ECC):
http://www.certicom.com/resources/ecc_tutorial/ecc_tutorial.html
- ▶ More on ECC:
http://www.certicom.com/resources/w_papers/w_papers.html



Performance

When it comes to handling numerous requests or high-volume sites such as `www.ibm.com`, the quality of services a Web server provides to end users depends on two parameters: network transfer speed and server response time. Network transfer speed is a matter of connection bandwidth, while server response time depends on resources such as CPU, RAM and I/O performance. What can you do when these resources are exhausted and the Web server is struggling against heavy traffic?

Many would first think of updating the hardware, for example, replacing the CPU with a faster one, using faster SCSI controllers or even a RAID system with a very large cache. The result of such an update is high cost and high maintenance.

Another tentative solution may be to tune the software by adjusting the operating system parameters, the Web server configuration, the application server configuration and/or the database parameters, but it is not realistic to expect a great performance improvement.

A third tentative solution would be to load balance the traffic and have two or more back-end servers running at the same time to process the requests. Large traffic loads must be handled properly or users will get slow response time or refused connections. The site may become unstable or even fail under critical load conditions. Frustrated users may not visit the site again.

A discussion of how to handle this problem is introduced in upcoming sections.

13.1 Load balancing

Load balancing refers to the distribution of the traffic load of one entity to many entities. A possible approach to load balancing is to increase the number of back-end servers for parallel processing. This increases parallel processing of requests as well as throughput. In order to have parallel back-end servers receiving requests at the same time, another component in the front end must be set up. This spreads incoming requests out to the available back-end servers.

The next section introduces three different approaches to handling load balancing.

- Network dispatcher approach
- DNS approach
- Reverse proxy approach

13.1.1 Network Dispatcher approach

Network Dispatcher has an advanced IP-level load balancing mechanism for any TCP or UDP protocol. Once installed, the dispatcher becomes the entry point of Web sites to which clients send packets. However, it still remains completely invisible to clients. A patented algorithm of Network Dispatcher guarantees that traffic is optimally balanced over the back-end servers. Another key issue is that the application server returns the response to the client directly without passing back through the dispatcher. The dispatcher automatically detects the availability of a back-end server before forwarding the request. This is achieved by issuing a simple command. Additionally, another machine can be set up as a backup machine in case of failure.

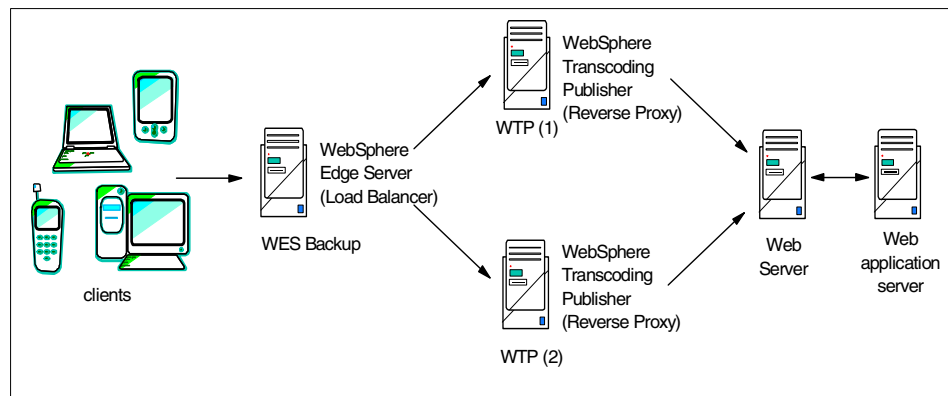


Figure 13-1 Sample load balancing scenario

Figure 13-1 depicts a sample scenario with IBM WebSphere Transcoding Publisher. Web servers, application servers and database servers must of course be clustered in a highly-available environment in order to serve the business logic and data for processing. For example, IBM HACMP can turn these servers into high-availability servers.

IBM has developed a product that can load balance the traffic; this product is called IBM WebSphere Edge Server and comes with advanced functions to meet the sites' scalability and availability needs. WebSphere Edge Server includes a caching proxy component (also known as Web Traffic Express) and a load balancing component (also known as Network Dispatcher). WebSphere Edge Server comes with a high availability daemon that can be used for setting up a backup machine. The features and impact of the caching proxy are discussed in Section 13.4, "Caching" on page 233.

13.1.2 DNS approach

Another approach to load balancing is the Domain Name System (DNS) approach, as shown in Figure 13-2.

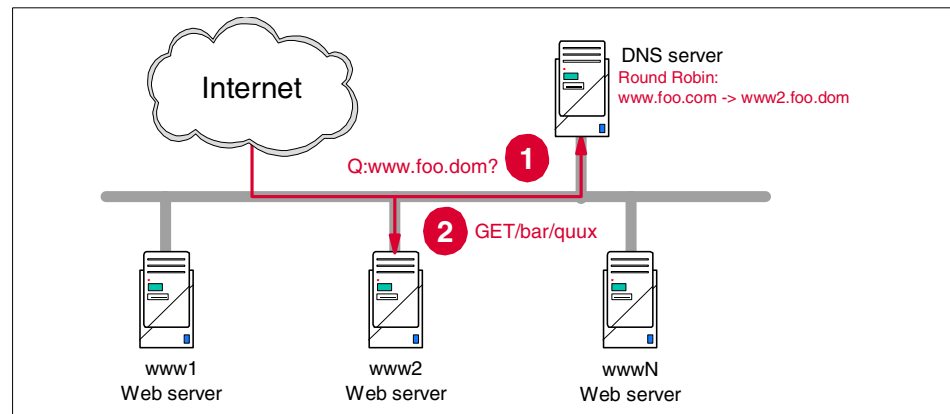


Figure 13-2 DNS approach

Every time a Web browser requests a URL, for example `www.ibm.com/bluepages`, the first step is to resolve the corresponding IP address. This is done simply through a passive resolver library, which calls a nearby DNS server familiar with the IP address. Rather than returning a static IP address, the DNS server returns one of the back-end servers. The decision of which IP address to choose depends very much on the traffic and technical possibilities.

The best in DNS server implementation is still the Berkeley Internet Name Daemon (BIND). BIND provides a feature called *round robin*, which maintains a particular pool of IP addresses and selects one if a request arrives. This pool of addresses is maintained sequentially, and if the pointer shows the last IP address, it simply points back to the beginning of the list (see Example 13-1)

Example 13-1 Pool of IP addresses

```
www.ibm.com IN CNAME www1.ibm.com
              IN CNAME www2.ibm.com
              IN CNAME www3.ibm.com
              IN CNAME www4.ibm.com
              IN CNAME www5.ibm.com
              IN CNAME www6.ibm.com
```

This approach seems very attractive, because the traffic is distributed over all the back-end servers, one by one. Unfortunately, in practice there are some considerations to be taken into account. To decrease the resolver traffic and increase the resolver speed, the DNS server caches the resolved data. Caching time is controlled by a time-to-live (TTL) value, which is appended to each piece of information. If the value of the TTL is too high, the traffic on DNS servers is automatically decreased, but on the other hand, the traffic is not distributed over all back-end servers. If the value of the TTL is too low, the traffic on DNS servers is increased, because the information expires more quickly, but DNS servers have to resolve the host name more often. However, better load balancing on the back-end servers is achieved. The setting of the TTL value depends on the traffic and the number of back-end servers. Hence, this parameter is system-dependent.

Once a particular back-end server is resolved, it remains the contact point for that visitor until the TTL expires. Another problem occurs when one of the back-end server crashes. In this case, no visitors to this back-end server can request documents from that server again until the TTL expires.

This approach is simple, but due to caching and the round robin feature, some restrictions do apply.

13.1.3 Reverse proxy approach

The last approach to load balancing is the reverse proxy approach. A reverse proxy simply operates in the direction opposite to that used by a normal proxy. Usually, proxies are used to bundle requests and to reduce bandwidth waste by caching data. A reverse proxy simply receives a request and translates the URL to the absolute URL of one of the available back-end servers (see Figure 13-3 on page 231).

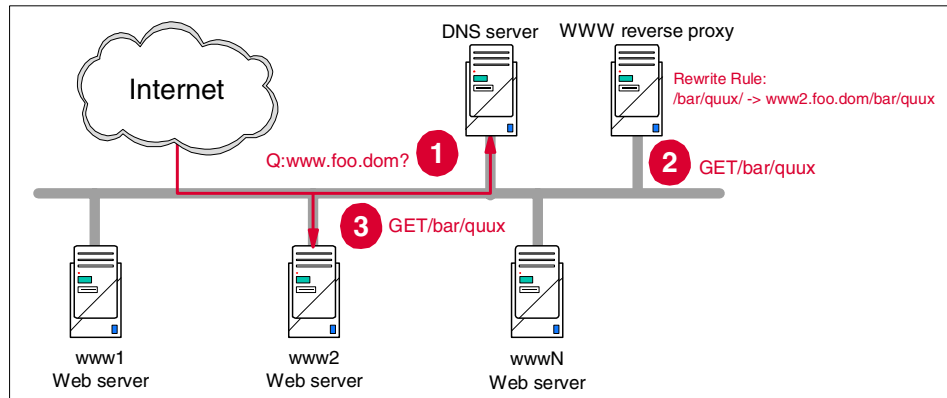


Figure 13-3 Reverse proxy approach

The reverse proxy itself does not serve any request, but determines a back-end server, forwards the request to it and sends the response back to the client. No DNS resolving is needed here, because all traffic must go through the reverse proxy, and there is no other problem with the TTL. For security reasons, the reverse proxy can be placed behind a firewall, and a firewall can also be installed behind a reverse proxy to secure the back-end servers.

In this approach, the reverse proxy is the only entry point to a Web site. This eases the need for logging and monitoring the Web site. If one of the back-end server crashes, the reverse proxy simply delegates the traffic to other back-end servers until the crashed back-end server is reactivated. The only drawback is that the reverse proxy must rewrite URLs on a massive scale. Hence, a powerful server is needed to run an URL-rewriting engine and a reverse proxy.

13.2 Scalability

Load balancing the traffic over two back-end servers increases the throughput of connections per second. This leads to better control of the scalability of the system. If two back-end servers are not able to handle the load, then another back-end server can be installed. This feature is known as *horizontal scalability*, which is in this case unlimited. Customers can put as many back-end servers on the same level as traffic requires. Hence, there is more flexibility to determine how many back-end servers are actually needed. It is worth mentioning that the back-end servers do not need to be large-scale machines. Medium scale hardware can be used instead and high costs to buy large-scale machines can be avoided.

It should be pointed out that trying to increase the scalability of one component often results in moving the bottleneck of the system from one component to another. The bottleneck still remains in the system; only the dynamics of the system have changed. A chain of reactions could be easily triggered. Therefore, scalability should be viewed with the *whole* system in mind.

Note: The term *bottleneck* is a simple way to refer to the most stressed component in the whole system.

Achieving maximum scalability means that the components must work in harmony with each other in order to cope with the demand. For example, a machine with one CPU and 512 MB of RAM running one process of an application has an average CPU utilization of about 92%. Increasing the number of processes to four may achieve better throughput, but the average CPU utilization increases to 100%. Hence, the bottleneck no longer involves the number of processes but has “moved” to the CPU of the machine. The next step is to increase the number of CPUs in the system (see Table 13-1).

Table 13-1 Sample statistics

Number of CPUs	Number of processes	Memory	Average throughput	Average CPU utilization
1	1	512 MB	2.57 trans/sec	92%
1	4	512 MB	2.59 trans/sec	100%
4	4	512 MB	7.46 trans/sec	90%

In this case, one back-end server with four CPUs running four processes can handle 7.46 transactions per second; if there are two or more back-end servers, the administrator might want to update the CPU of each one and have symmetry as well (to have the same type of machine in the cluster and not mix up different machines) within the cluster. A simple addition of processes involves no costs, but results in an upgrade of hardware which involves costs. Scalability must, therefore, be planned and investigated concretely before making any changes to the system.

13.3 Availability

When discussing load balancing, another issue to consider is availability. This is because the addition of extra back-end servers to serve requests involves higher availability as well. The more back-end servers are operating, the higher the availability of these servers. If one back-end server fails because of a hardware failure, another back-end server continues to process the requests. Of course,

the throughput is not as high as with two back-end servers working simultaneously, but this is only a temporary condition until the failed machine has been fixed. Additional machines can be added temporarily. 99.9% availability can be achieved by having two or more back-end servers running and serving requests. This also means that a complete back-end servers failure is impossible.

Network Dispatcher comes with a cluster daemon that can send and receive heartbeat from another backup network dispatcher set up on a different machine. If a network dispatcher failed on one machine, the network dispatcher on the backup machine can take over.

Note: *Heartbeat*, in a high availability environment, means that a server sends signals stating that it is still up and running.

It is also important to remember to implement a high availability solution to servers that hold the data source like a database server does. In a common architecture, there is only one machine running a database server, but there are many other Web servers to load balance and to ensure a better throughput. In this case, if the database server fails because of a hardware failure, then the whole system is down. The data source is gone, no matter how many Web servers there are to serve the requests. Here, it is a good idea to back up the database server through high availability to avoid this occurrence.

The more numerous the back-end servers, the more complicated it is to monitor these servers. Each server must be monitored for failures or machine load. Therefore, a tool must also be introduced to address this requirement in a clustered environment. For details on this topic, please refer to Chapter 11, “System management” on page 197.

13.4 Caching

E-business applications are face with a heavy load from the clients, since in most cases the client is a simple browser. There is nothing, or maybe just a few minor functions, running on the clients. The whole application is running on the server side, even the presentation logic itself, so that only the result documents are transferred to the clients.

Interactions and transactions take place via documents sent to the clients, then filled out and submitted by the user. This represents a very large number of documents, or, to be more precise, of sets of Web pages. These pages are most likely the same for each user and sometimes over several transactions. For example, the Web application starting page is the same for every user; browsing a catalog involves the same query during each session.

The repetition of pages during interactions makes it possible to speed up applications using the caching mechanism.

The most computation required from the server is to generate all sets of Web pages requested by the clients. Since there are pages that are the same, the application only has to store, or cache, the generated page; then, the next time a user requests the same document, the application can send the stored page through instead of generating it again.

Caching is a very important function for every Web application; it can speed up the applications significantly and dramatically decrease the load on the servers. That means faster operation and a greater number of clients with access to the same Web application.

There is nothing more annoying than waiting for a page to download from the server; companies can lose customers because of the long response time. Caching can help to reduce the response time.

From a performance standpoint, it is less expensive and more efficient to set up a cache server than to power up the application server.

Considerations

Since caching is very important, nowadays every application takes advantage of it. Application designers have to take caching into account at different levels, not only in runtime, but in development time as well. Caching can occur as in the following instances:

- ▶ The operating system usually takes charge of caching.
- ▶ The Web application server takes charge of caching.
- ▶ Sometimes, the Web application has a caching mechanism, for example WebSphere Commerce Suite v5.1.
- ▶ There are cache servers in the architecture for real caching.
- ▶ The clients are caching the content on their side.

The problem is that it is difficult to keep up with all the caching mechanisms, and to trace where the content came from.

It becomes very difficult for developers if they do not have influence on caching and cannot test the updated code because of the caching mechanism. Switching off the caching is the best solution; in the case of multilevel caching, it can help to disable just the right cache storage.

Tip: If all else fails, the best thing to do is to restart the application server and delete the cache on the disk.

Caching transcoded content

Differentiating between the different types of transcoded content can be a significant problem. The cache server differentiates the pages based on the requested URL or a pattern of the requested URL; it can be set up according to how the cache mechanism operates.

Transcoding occurs between the client and the Web application server; the client has no influence on the behavior of the transcoder, nor does the Web application. The following scenarios are possible:

1. Cache server **before** the transcoder: the server only caches the content from the Web application server, then the transcoder has to transcode every page each time, which is time consuming.
2. Cache server **after** the transcoder: the server will cache the transcoded pages coming from the transcoder. Since the cache server only checks for the URL, it will serve the same content to each device, so the first client accessing the site “chooses” the pages it needs, and the following devices can only receive the same pages as this first device.
3. Cache server **together** with the transcoder: this is the ideal solution, because the server will cache the transcoded pages, and can differentiate between devices. Each client has to go through the transcoder to take advantage of caching.

Note: There is an NOP (No Operation) transcoder which only passes through the content without doing anything. This is ideal for clients that do not require transcoding.

4. **Mixed** cache servers: caching on different levels is dangerous. Of course, it has the advantage of more precise and more efficient caching, but it requires more control and is difficult to set up.

For example, using WebSphere Transcoding Publisher as a servlet filter together with WebSphere Commerce Suite v5.1 can cause a problem. WCS v5.1 performs caching based on the URL, but cannot differentiate between clients. If a client comes with a request, the WCS server generates the content, the transcoder transcodes, then WCS caches it. The next time around, another kind of client will get the same content as the first because WCS v5.1 does not know about client types.

For more information about WCS 5.1 and mobile commerce, please refer to the redbook *Mobile Commerce Solutions Guide using WebSphere Commerce Suite V5.1*, SG24-6171.

Summary

Make sure that you are aware of all the caching functions in your Web application throughout the whole architecture. Work out the best solution for caching and take into consideration the different device types. In the case of mobile Web applications, the most efficient caching solution does not always work.

The next release of WebSphere Transcoding Publisher and WebSphere Edge Server will have tight integration, so more precise, more efficient and more intelligent caching will be possible for mobile Web applications.

13.5 Turning on transcoding

Using a transcoder in your solution will have an impact on the performance of your mobile Web application.

Performance

Transcoding works on the existing content.

1. First, the transcoder has to retrieve the content and parse for further processing.
2. Transcoding is done in several steps; for each step, the document is gone through and processed.
3. Finally, the transcoder transfers the document to the client.

As you see, transcoding requires computation, which takes time and CPU utilization.

The performance of transcoding depends on several points:

- ▶ The complexity of the page
- ▶ The transcoder chosen by the server to process the requested content

- ▶ The transcoded content: document, image or something else
- ▶ The logging level
- ▶ Caching
- ▶ The deployment mode

Caching

The performance of transcoding can only increase using multiple transcoding server or caching.

For details on caching, please refer to Section 13.4, “Caching” on page 233.

Proxy

The performance of the transcoder also depends on the type of deployment.

Using the transcoder as a proxy gives access to all the servers on the network without restrictions. The client has only to set up the transcoding server as a proxy, then go through it to access the target server.

For a Service Provider, this is the right deployment model for a service.

If the transcoder is running within the company and has to transcode documents from certain servers, then it is better to deploy it as a reverse proxy. In this case, the transcoder will not have to transcode other servers' content. This will optimize the load on the transcoder server.

For more information about the deployment of WebSphere Transcoding Publisher, refer to Chapter 7.3, “WebSphere Transcoding Publisher considerations” on page 97 or consult the redbook *New Capabilities in IBM WebSphere Transcoding Publisher Version 3.5 Extending Web Applications to the Pervasive World*, SG-24-6233.

13.6 Where to find more information

For more information, see the IBM white paper *WebSphere Edge Server Version 1.03*.



Part 4

Scenarios



Base sample overview

This chapter will introduce the base sample application for this book. The mobile Web application scenarios are based on this sample.

Each scenario provides the same functions and the same core modules.

This sample is optimized for desktop PC browsers to try to get the most out of the HTML specification.

14.1 The base sample

The base sample is based on the WebSphere Trade2 example application made for benchmark testing, and used in the book. The original code can be downloaded from the following URL:

http://www-4.ibm.com/software/webservers/appserv/wpbs_download.html

In this book, the whole sample is rewritten, a new look and feel applied, and the front end (presentation logic) is redesigned. Only the back-end (database application) and the functions remain.

Important: The application works together with several desktop Web browsers, such as Netscape Navigator, MS Internet Explorer, or Opera, but the presentation is optimized for MS Internet Explorer.

14.2 The scenarios

In this redbook, several different scenarios are implemented based on the Trade2 sample application.

We classify the approaches into scenarios, based on the type of client and the technology used during the implementation.

These scenarios are grouped into three main classes, and each scenario is listed under these groups:

1. Application for interactive mobile devices
 - Direct coding
 - Content transcoding
 - Universal transcoding
2. Application for voice
 - Direct coding
 - Content transcoding
 - Universal transcoding
 - Hybrid coding
3. Application for both interactive mobile devices and voice
 - Universal transcoding
 - Content transcoding
 - Multimodal applications

The scenarios feature different cases, for example WML, cHTML, simplified HTML, and VoiceXML.

14.3 The presentation logic

A mobile application has to serve many different kinds of clients with different content type, and some of the clients require a different application flow.

In the traditional situation, there is a site already running and serving content for traditional HTML browsers. In other cases, where the site is newly developed, the aim of the development process is to have a rich and elaborate HTML content.

The requirement is to create a site which has all the capabilities to serve the different clients not just with a different type of content, but also following different flows and using the same back-end.

This book gives one kind of solution for fulfilling all of these requirements.

14.3.1 The three tier servlet architecture

Figure 14-1 on page 244 shows the architecture for our base sample. It may look a bit sophisticated at first glance, but this chapter will discuss the concept behind this model.

First of all, it is important to note that this solution is intended for all our scenarios in this book. It must be flexible, in case we want to enhance it further or support new devices.

This sample also stands as a recommendation for designing a mobile Web application. It would perhaps not be used exactly as it is shown here, but considerations have been taken into account to ensure validity for all mobile e-business solutions.

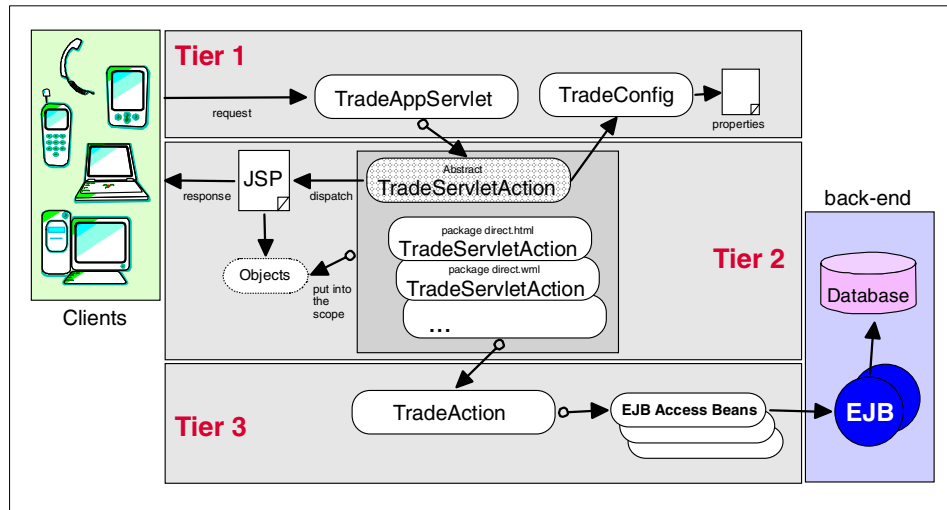


Figure 14-1 Three tier servlet architecture

The tree tier architecture separates the following three elements:

- ▶ User interface (UI) application flow: control
- ▶ Content generation: views and view control
- ▶ Database access: access to the model

The first tier (here TradeAppServlet) gets the request, then, according to the type of device, instantiates the appropriate object at the second tier (TradeServletAction here). The servlet captures the requested action from the client and passes to the second tier. The first tier therefore works as a router; it instantiates the device-specific object, which implements the specific application flow (control).

The second tier implements the actions for the application. It is device-dependent, controls the interaction flow according to the device type, and dispatches the request to the content generator.

The content generation also belongs to the second tier, which is obviously device-dependent. In order to represent dynamic data in the content, the data access beans are introduced in the third tier.

The third tier is device-independent like the first tier, so each device-dependent object can use the same Access Beans. Access Beans take over the difficult part of programming EJBs, acting as wrappers.

In summary, the three tier architecture provides a solution for different devices, separating the request routing, the application flow control and content generation, and the data access.

With this architecture, the device-specific elements at the front end are fully separated from the other application elements, and focused into one object only. This object is practically a class which can of course have other supplemental classes. Implementing access for a device requires only the development of the second tier and uses the access points from the first and third tiers.

The only device-specific element is the second tier, where each action takes place and the content is generated.

14.4 The business logic

We will now discuss the business logic behind the sample application. If we are to run a real-life application, it has to have real business logic running on the back-end.

14.4.1 Data model

Figure 14-2 shows the data model for the Trade2 application:

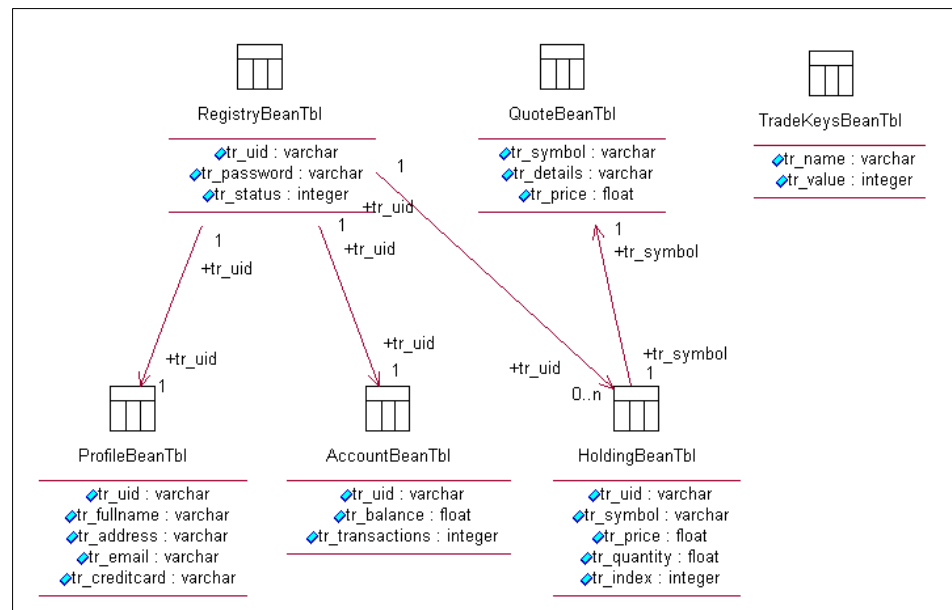


Figure 14-2 Data model for the Trade2 application

The following tables exist in the database:

- ▶ RegistryBeanTbl: this table holds the user ID, the password, and the status which tells whether the user is logged in or not.
- ▶ ProfileBeanTbl: this table includes the user profile; the full name, address, e-mail address, and credit card information is stored in this table.
- ▶ AccountBeanTbl: all account-related information is placed here.

The three tables above cover the user-related information; they are in a 1-1 connection, which means that each registered user has an entry (record) in each of the three tables.

- ▶ HoldingBeanTbl: this is a 1-0...n relation based on the tr_uid. This table stores the portfolio for all the users. Each entry is identified by a tr_index number. The tr_uid and tr_symbol fields refer to RegistryBeanTbl and QuoteBeanTbl. Each entry also has fields, which describe that holding (price and quantity).

The holdings are stored by transaction and are not accumulated. It could happen that one user has multiple holdings related to the same symbol, with different prices and quantities.

- ▶ QuoteBeanTbl: the current quote information is stored in this table; in our example, it holds static records, but in a real-life scenario the current values from the stock ticker would be here.
- ▶ TradeKeysBeanTbl: this is nothing more than a persistent common value to keep tracking the transactions; since each transaction has a unique transaction index, this table stores the latest available index for the next transaction.

14.4.2 Access Beans

In order to handle the EJBs more easily, the sample application uses Access Beans. The Access Bean wraps the EJB and handles the necessary procedures, such as context, JNDI lookup, and instantiating.

This method hides most of the more difficult coding portions related to EJBs, and makes the development easier.

You can create an Access Bean using the EJB framework in VisualAge for Java; please refer to the VisualAge for Java online help and search for **Access Beans**.

14.5 Model diagram for the application

Figure 14-3 shows the object model diagram for the base sample.

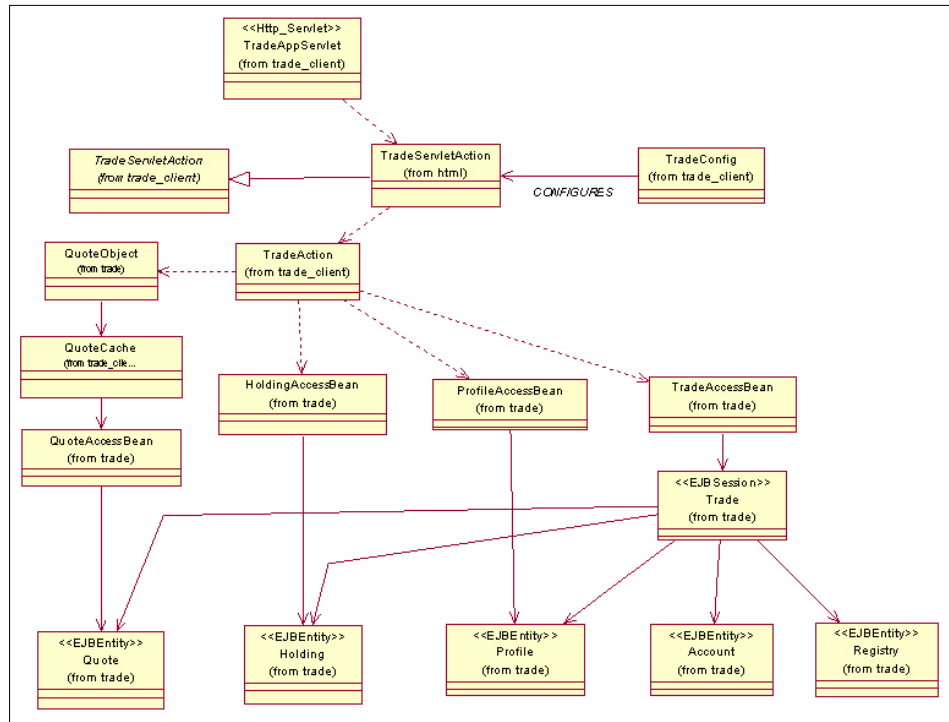


Figure 14-3 Object model for the Trade2 Web application

Let us start from the TradeAppServlet at the top of the diagram, where everything begins. This is the servlet class, which handles every interaction between the client and the application. The requests and responses, which do not go through this servlet, are not tightly related to the application.

For example, the starting page called “splash” will not use the servlet, since it has nothing to do with the application, but only provides content. On the other hand, the login session will always go through the servlet, since it is tightly related to the application.

The servlet instantiates the TradeServletAction supplemental class, which will handle all the device-related actions. This is a device-specific class. For example, this class controls the application flow for a specific device, which can be different for a WAP phone and for a wireless PC.

TradeServletAction has the TradeAction supplemental class, which is a device-independent class. This class handles all the back-end related services related to the actions. For example, during the login session, this class will talk to the Registry EJB via the registry Access Bean to find out whether or not the login was successful.

There is a utility class for the application, called TradeConfig. This is a static class; every method, every attribute is static, each object has the same static view of this class. It holds all the configuration information, retrieved at the initialization; it also provides utility methods, available for every other class during the runtime. For example, it has a hash table, which holds the device types with the names of the directories in which the appropriate content is stored for each device. It also provides a utility method, getPage, which will return the appropriate JavaServer Page URI depending on the parameters (device type, requested action) for the specific device.

There are Access Beans such as:

- ▶ QuoteAccessBean
- ▶ HoldingAccessBean
- ▶ ProfileAccessBean
- ▶ TradeAccessBean

They are wrapper classes for the EJBs. The EJBs are:

- ▶ Trade: a session bean
- ▶ Quote: an entity bean
- ▶ Holding: an entity bean
- ▶ Profile: an entity bean
- ▶ Account: an entity bean
- ▶ Registry: an entity bean

These are entity EJBs, and persist in the tables with similar names, for example: Account EJB in the AccountBeanTbl table.

The Access Bean for the quote EJB is not directly instantiated from the TradeAction. The reason for this is that each quote has a unique object, and that object handles the cache for quotes; the purpose of this cache is for the quotes to be delayed depending on the user.

14.6 Site map for the application

Figure 14-4 represents the site map for our Trade2 Web application. Each object represents either a Client Page, a Server Page or a Java class; the arrows represent links between pages when nothing else is indicated. Other relations between objects can be:

<<include>>: where the content of the pointing page is included into the parent page.

<<redirect>>: where the page redirects the request to the next page, without requiring any interaction.

In links, where an object points to the servlet, the caption of the arrow explains the action invoked within the servlet.

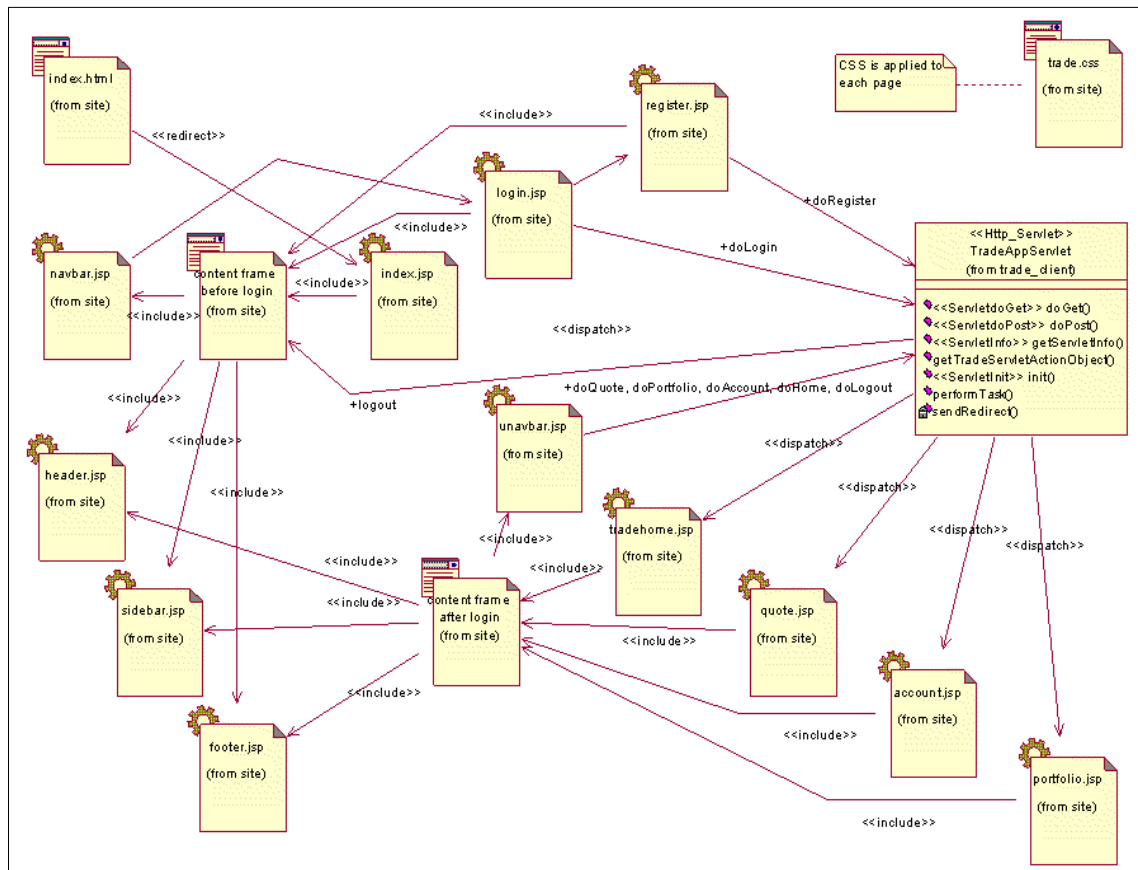


Figure 14-4 Site map for Trade2 application

Here are some notes to better understand the site map:

Figure 14-5 shows a legend for the diagram:

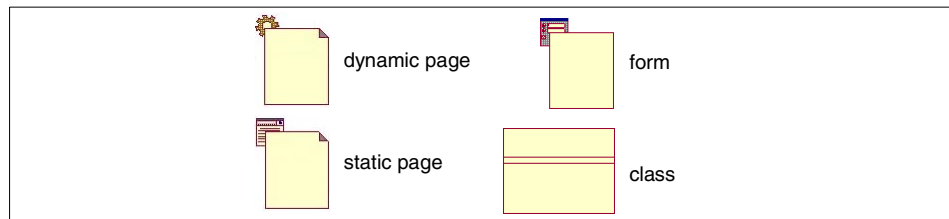


Figure 14-5 Legend for site maps

Dynamic pages represent the JavaServer Pages; those pages are rendered on the server side. They can include other pages, either static pages or other dynamic pages.

The static pages are served by the Web server and are written in a specific markup language, for example HTML.

Forms are document elements; they cannot stand alone and are always included in either a static or a dynamic page.

Classes are Java classes running under the server-side JVM.

The arrows between the objects represent the direction of the interaction. For example, if an arrow points from one page to another, this means that the page where the arrow starts has a link to the page where the arrow ends.

14.6.1 Publishing

The whole Web application is published under the application server, so the Web assets are placed under the <web application path>/Web directory.

The application requires one servlet, the TradeAppServlet, which can be found under the <web application path>/servlets directory.

In Figure 14-4 on page 249, you can follow the interaction between the pages, and the building elements as well.

Each page is structured as in Figure 14-6.

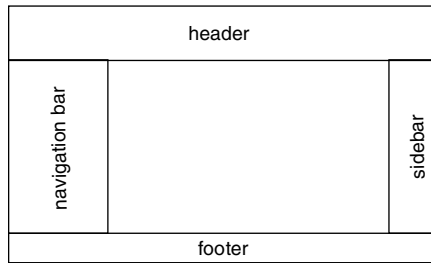


Figure 14-6 Page construction

The structure is based on tables. The header, footer, navigation bar and side bar are JavaServer Pages; these dynamic pages are included in every main JSP using the JSP include tag.

The header, the footer, and the sidebar are static pages for the whole site, although the sidebar is usually dynamic on a running site. The sidebar is useful in providing context-sensitive navigation for the content on the page, compared to the navigation bar, which controls the whole application flow and is less context-sensitive.

You might have noticed that there are two different navigation bars: navbar.jsp and unavbar.jsp; one appears before the login, the other after the login. The reason for this is that the menu for the application has to change after the user has logged in.

The two statuses are clearly separated on the site map as well; the bridge between the two sets of pages is the login page, where the login.jsp is on the "not logged in" side; then, after successful login, the TradeAppServlet dispatches the request to the tradehome.jsp, which is on the "logged in" side.

Here is a sample page from the application: the Welcome screen.

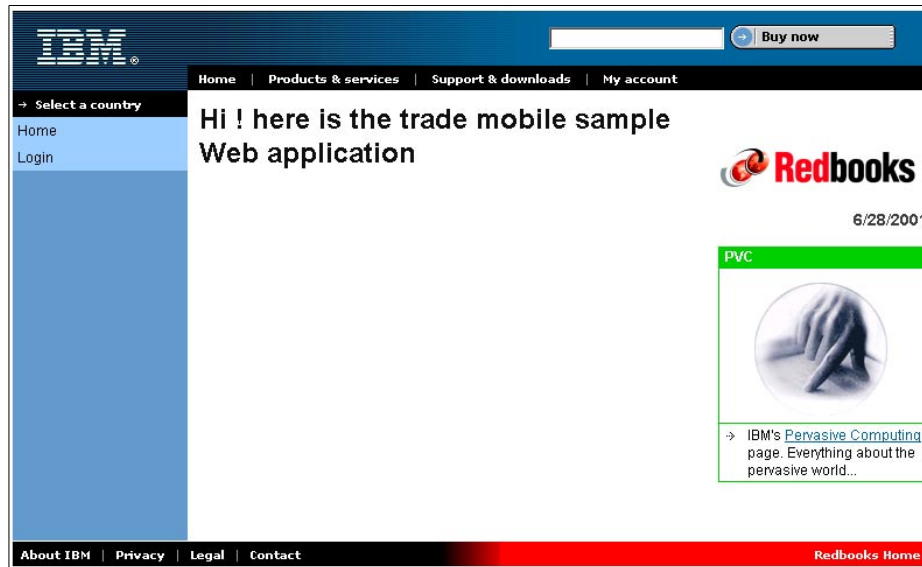


Figure 14-7 Welcome screen

14.7 Walkthrough

These walkthroughs are helpful in understanding and following the application flow.

14.7.1 Login scenario

From the starting page, the user can access the login page by using the login link.

On the login page, there are two common fields for logging in a user: the user name, and the password.

1. In our case, the user name and the password fields are already filled out; this helps to work with the application. By submitting the login form, the request goes to the TradeAppServlet with the following parameters:

```
uid=johnd  
passwd=johnd  
action=login
```

2. The servlet checks the User Agent field in the HTTP header, looking for the supported device type. Considering the device type, the servlet instantiates

the appropriate class which implements the available actions for that device. The instantiated class is an implementation of the `TradeServletAction` abstract class. Each implementation has the same name as the abstract class, but sits under a different package name; for example, the WML action handler class name is `TradeServletAction` under the `trade_client.direct.wml` package; the class for the HTML application has the same name under the `trade_client.direct.html` package.

In our case, the base application is tested with Internet Explorer v5.5, so the client is an HTML browser; the HTML action handler will be implemented by using `trade_client.direct.html.TradeServletAction`.

3. The `TradeServletAction`'s `doLogin()` method is called from the servlet. The method will immediately instantiate the `TradeAction` class, which is device-independent.
4. The `doLogin()` method within the `TradeAction` class is called. The method instantiates the `TradeAccessBean`, which is an Access Bean that wraps the EJBs related to the user (registry, profile, account, holdings).
5. `TradeAccessBean` will run the login through the `TradeEJB` and return with the result.
6. `TradeActionServlet` will handle the login based on the returned value from `TradeAction`. If the login was successful, it will call the `doHome` method to jump to the starting page after logging in.
7. The `doHome` method then dispatches the request to the appropriate page by using the `getPage` method from the `TradeConfig` utility class. The `getPage` method has two parameters, one for the device type and another for the page to dispatch to.

The information about mapping for the devices and which page to retrieve is stored in a `.properties` file. The `TradeConfig` class is initialized from the `TradeAppServlet` `init()` method before step 2, then the configuration is read from the file.

8. The `JavaServer Page` generates the result page, then the document is sent to the client.

After the user has logged in, every action goes through the `TradeAppServlet` until the user logs out.

14.7.2 Portfolio

The Account function is only available after the user has logged in. The navigation bar has changed after the login, and there is an option through which the user can check the stock portfolio. It is a list of stocks listed in the order they were acquired.

1. The user clicks the **Portfolio** link.
2. First, TradeAppServlet is called, using:
`action=portfolio`
3. The servlet's performTask method will instantiate the appropriate class for the TradeServletAction based on the User-Agent, just as in the login session previously. The servlet realizes that the action is requesting the portfolio, so it runs the doPortfolio method of the instantiated TradeServletAction.
4. The TradeServletAction's doPortfolio method builds up a vector with the portfolio elements using the TradeAction's doPortfolio method, which returns an array of HoldingObjects.
5. The objects come from the TradeAccessBean, which has a getPortfolio method to retrieve the records from the database using the Holdings EJB within the Trade EJB.
6. The TradeActionServlet populates a vector with the returned values from TradeAction, then puts the vector object into the request scope. The TradeAppServlet dispatches the request to the appropriate JavaServer Page returned by the getPage method from TradeConfig.
7. The portfolio JSP will acquire the vector object as a JavaBean from the request scope, and print out the elements through a "for" cycle.

14.8 Summary

The base sample, besides being a real life application, was built to show how to use a Web application as a starting point for a future mobile application.



Application for interactive mobile devices

This chapter discusses the application development for mobile devices. These devices can be PDAs, WAP phones or i-mode phones, wireless PCs, or any other device with a screen. This chapter will not include voice, which will be discussed in the next chapter.

There are three main scenarios for these devices:

- ▶ Direct approach, where the content is written directly in the specific language of the device, for example, WML for WAP phones.
- ▶ Content transcoding, where WebSphere Transcoding Publisher is applied to transcode the content from HTML to the required content, for example HTML-to-cHTML for i-mode.
- ▶ Universal transcoding, where WebSphere Transcoding Publisher applies the StyleSheets (XSL) to the XML content to produce the specific content. For example, we could use the XML data, applying specific XSLs for a wireless PC client to get a simplified HTML content.

In each scenario, we implement the Trade2 application for different devices: WML in the direct approach, cHTML for content transcoding, and simplified HTML for universal transcoding.

15.1 Direct approach

This section discusses the aspects of the direct development of a visual mobile Web application using a WAP example.

15.1.1 Design issues

The following section discusses some micro design issues encountered during the development of our directly coded content.

In this exercise, we took the existing Trade2 application described in Section 14.1, “The base sample” on page 242.

We focused on the screens that trigger a transaction and rewrote them in WML. The flow is basically as follows: the user first gets a welcome page (index.jsp). The user can either register (register.jsp) or log in (login.jsp), after which he/she receives the home page (tradehome.jsp). From there the options are:

- ▶ Home (back to tradehome.jsp)
- ▶ Account (account.jsp)
- ▶ Portfolio (portfolio.jsp and further quote.jsp)
- ▶ Logout (back to index.jsp)

MIME types

For the development environment and for the runtime environment, it is necessary to support the text/vnd.wap.wml MIME type and associated MIME types by the server. To find out how to set up the MIME types for the WebSphere Test Environment, refer to Section 18.6.2, “Adding MIME Types to WTE” on page 366, and for IBM HTTP Server and WebSphere Application Server refer to “Setting up MIME types” on page 387.

Large Forms

As we expected, on some HTML screens we were confronted with forms that were inadequate to display content on a WAP screen.

Registration

On the registration page, there were too many fields to fit easily into one WAP screen, as illustrated in Figure 15-1 on page 257:

Figure 15-1 Original HTML registration window

The first thing to consider is that there are alternative registration methods.

If the user already has, for example, an X.509 client certificate signed by a trusted third party, then the site will already have a certified proof of the user's identity, and one could consider eliminating a number of fields that are not directly contributing to the identification of the user and should already be available in public directories.

As we discussed in "Non-HTTP wireless clients" on page 209, many of the security technologies still have to be implemented.

We assume that the user registers by filling in all the above fields. We must consider two options:

1. List all the fields on one screen (one *card* in WML terms).
2. Construct a kind of wizard, with two sets of screens (*cards* in WML terms) with four fields in each screen, and implement navigation between the cards.

As illustrated in Figure 15-2, we chose the last option for our purposes. This was done in order to avoid too much scrolling through one screen, and to provide the user with a better orientation.

We can split up the registration form into two different cards and send it as one unit (deck) to the client, since they are strongly related.

Figure 15-2 illustrates this flow with a UML activity diagram:

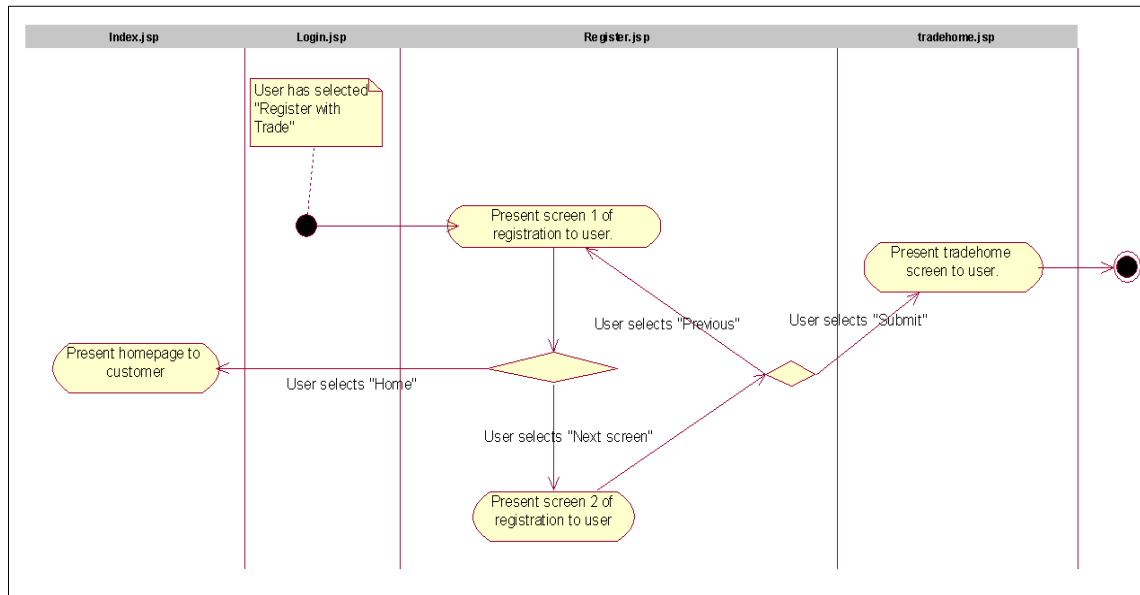


Figure 15-2 Login and registration flow for WML

The resulting screen flow can also be represented using the screenshots:

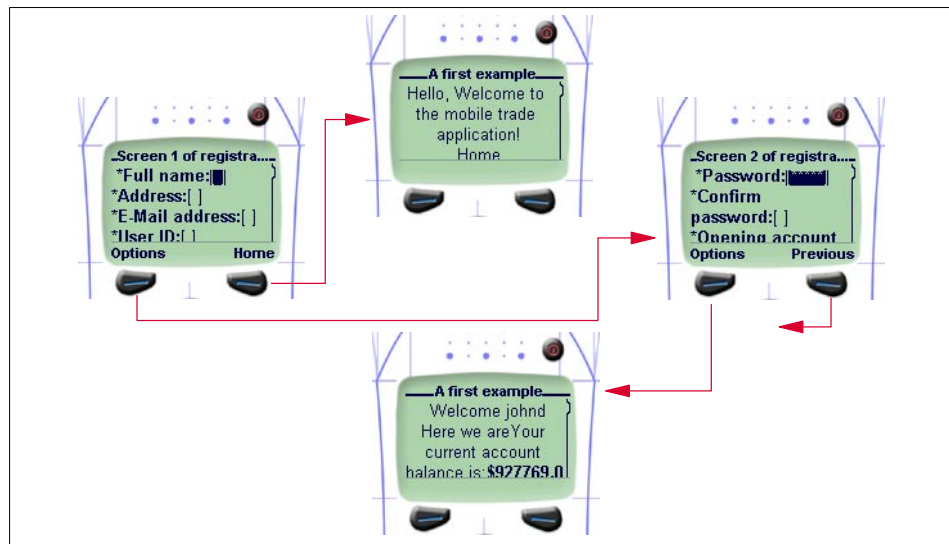


Figure 15-3 Registration wizard under WAP: screen shots

Update Account

In the HTML version, when the user clicks the **Account** link, the page as displayed in Figure 15-4 appears.

The screenshot shows the IBM Redbooks website's 'Account details for johnd' page. The top navigation bar includes links for Home, Products & services, Support & downloads, and My account. A sidebar on the left contains links for Home, Account, Portfolio, and Logout. The main content area displays the following account details:

- user ID: johnd
- full name: John Dow
- address: 113 Oak St.
- e-mail address: johnd@mail.d.com
- credit card number: 52-761-596-254

An 'update my account' button is located below the credit card number. On the right side, there is a Redbooks logo, the date 6/28/2001, and a section titled 'PVC' featuring a hand icon and a link to 'IBM's Pervasive Computing page. Everything about the pervasive world...'. The footer contains links for About IBM, Privacy, Legal, and Contact, along with a Redbooks Home link.

Figure 15-4 HTML screen with account details and option to update

This form contains a number of pre-filled fields that the user can change in order to update the information. When the user then clicks the button **Update my account**, the application returns the same screen, with the pre-filled fields.

For HTML, the result might be acceptable, in that the data has effectively been updated. For some device screens, for example the phone.com devices, input fields are shown one by one on separate screens and therefore we felt it more appropriate to split the screen into two parts:

1. Screen with the possibility to view the account data (data is only displayed, there are no fields). In this case, phone.com users can see all the account information at a glance. A Modify button brings up the screen described below (2).
2. Screen that displays fields with a pre-filled date (just as the original HTML form). An Update button effectively updates the account data, then brings back the screen described in (1).

The previous interaction is depicted in the UML activity diagram in Figure 15-5:

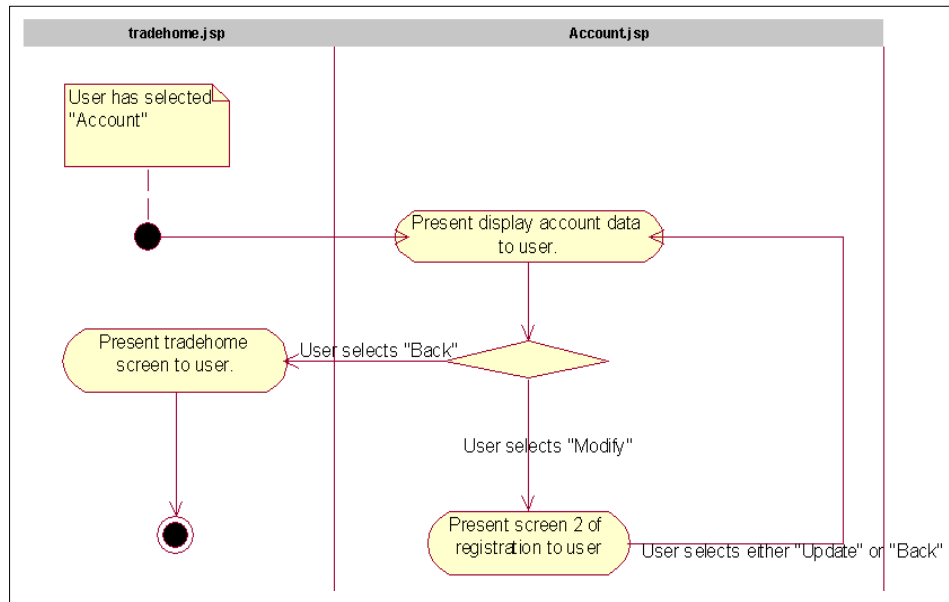


Figure 15-5 Account and update account flows UML activity diagram for WAP

A schematic representation of the screens is shown in Figure 15-6:

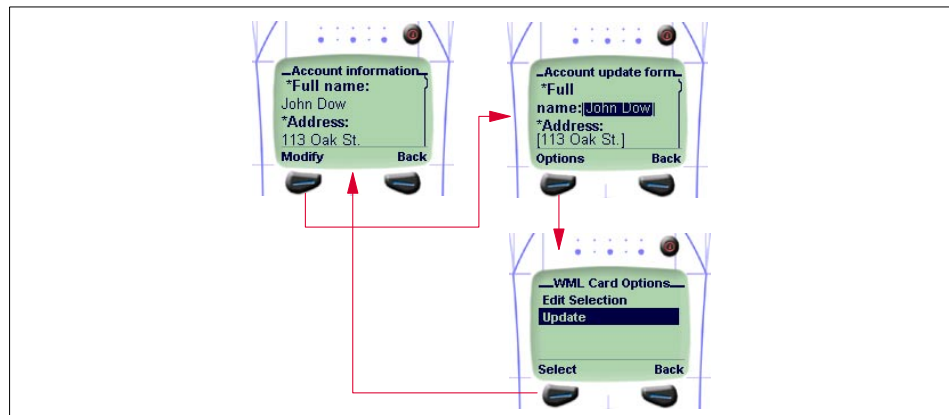


Figure 15-6 Screen flow when updating account information

Note that for usability, a Back button (or similar functionality) should always be provided. It can either bring the user to a previous card on the deck or to another screen.

Portfolio

To convert the portfolio screen to a WAP screen is another challenge (see Figure 15-7).

symbol	index	price	shares	value	buy	sell	quote
RHAT	8008	25.0	100.0	2500.0	buy 100	sell	quote
RHAT	8017	25.0	100.0	2500.0	buy 100	sell	quote
IBM	9001	178.0	100.0	17800.0	buy 100	sell	quote
IBM	14001	178.0	100.0	17800.0	buy 100	sell	quote
RHAT	15000	25.0	100.0	2500.0	buy 100	sell	quote
IBM	16000	178.0	100.0	17800.0	buy 100	sell	quote

Redbooks

6/28/2001

PVC

→ IBM's [Pervasive Computing](#) page. Everything about the pervasive world...

About IBM | Privacy | Legal | Contact

Redbooks Home

Figure 15-7 HTML version for the portfolio application screen

Obviously, we had to apply fragmentation to this screen, because of the limitation in screen size of WAP devices. This was done in two steps:

1. Split the table into one screen per row, with a Next button to navigate from one screen to another.
2. For each screen, provide a separate flow to either buy or sell, or to quote the portfolio item.

```

sequenceDiagram
    participant tradehome as tradehome.jsp
    participant portfolio as portfolio.jsp
    participant quote as quote.jsp

    Note over tradehome: User has selected "Portfolio"
    tradehome->>portfolio: 
    activate portfolio
    portfolio->>portfolio: Present first row of portfolio data to customer, in one card
    portfolio->>tradehome: User selects "Home"
    tradehome->>tradehome: Present tradehome screen to user.
    tradehome->>tradehome: 
    deactivate tradehome
    portfolio->>portfolio: User selects "Next"
    portfolio->>portfolio: Present current row of portfolio data, in one card
    portfolio->>tradehome: User selects "Next"
    tradehome->>tradehome: 
    deactivate tradehome
    portfolio->>portfolio: User presses "Back"
    portfolio->>portfolio: Present "Buy" screen to user
    portfolio->>tradehome: User selects "Buy"
    tradehome->>tradehome: Present "Buy" screen to user.
    tradehome->>tradehome: 
    deactivate tradehome
    portfolio->>portfolio: User presses "Sell"
    portfolio->>portfolio: Present "Sell" screen to user
    portfolio->>tradehome: User selects "Sell"
    tradehome->>tradehome: Present "Sell" screen to user.
    tradehome->>tradehome: 
    deactivate tradehome
    portfolio->>quote: User selects "Quote"
    quote->>quote: Present "Quote" screen to user
    deactivate quote
    portfolio->>tradehome: User selects "Back"
    tradehome->>tradehome: 
    deactivate tradehome
    deactivate portfolio
  
```

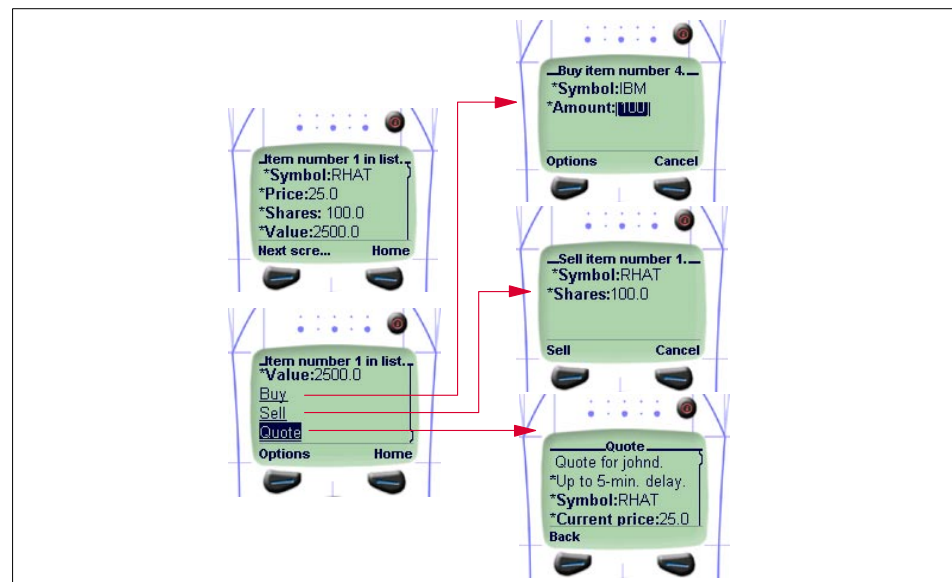
The diagram illustrates the workflow of the Portfolio Application across three JSP pages: **tradehome.jsp**, **portfolio.jsp**, and **quote.jsp**.

- tradehome.jsp** contains the initial state where the user selects "Portfolio". It presents the tradehome screen and handles navigation back to the home screen or forward to the portfolio screen.
- portfolio.jsp** manages the display of portfolio data. It starts by presenting the first row of data. Users can navigate between rows using "Next" or "Back", return to the home screen using "Home", or perform actions like "Buy" or "Sell".
- quote.jsp** is reached when the user selects "Quote" from the portfolio screen, where it presents the quote screen.

Key interactions include:

- Navigation from **tradehome.jsp** to **portfolio.jsp** upon selecting "Portfolio".
- Navigation from **portfolio.jsp** to **tradehome.jsp** upon selecting "Home".
- Navigation from **portfolio.jsp** to **tradehome.jsp** upon selecting "Next" (likely a typo for "Next" in the original diagram).
- Navigation from **portfolio.jsp** to **tradehome.jsp** upon selecting "Back".
- Navigation from **portfolio.jsp** to **tradehome.jsp** upon selecting "Buy" or "Sell".
- Navigation from **portfolio.jsp** to **quote.jsp** upon selecting "Quote".

The screen flow is further illustrated in Figure 15-9:



262 Mobile Applications with IBM WebSphere Everyplace Access Design and Development

The Buy and Sell buttons bring back the first row of the portfolio table, which in functionality corresponds to the HTML version.

Obviously, this can be improved; one should, for example, introduce an intermediary screen with a confirmation message, or insert the current price - for information only - on the Buy and Sell screens.

Tables

As mentioned above in “Portfolio” on page 261, the only table we encountered was the Portfolio table. We decided to remove some column values and present one row per WML card. Also, because the browsers on the mobile phone have only limited memory, only three cards are sent per deck. When the user arrives to the third card and clicks **Next**, another call is made to the servlet and a WML page (generated by a JSP) is returned with the next three cards.

Images

Although in this financial application the images did not present any added value, we introduced two: one graphic on the login page and one photograph on `tradehome.jsp`. We noticed a significant impact on response time after their introduction.

15.1.2 Test clients for development

During development, we used two test clients: the UP.browser and the browser that comes with the Nokia Development Toolkit. See also Section 10.3, “Tools for testing the application” on page 182 for more details.

Our experience was that the two clients differ on the following levels:

- ▶ The Nokia WAP toolkit shows tables, the UP.browser break them into lists.
- ▶ The Nokia WAP toolkit shows GIF image files, the UP.browser does not support this.
- ▶ The Nokia WAP toolkit shows different fields on a card at the same time, together with any additional text; the UP.browser shows the fields one by one, each on a different screen.

If the limitations of the UP.browser reflect limitations of existing devices, this is actually a reason to include tests using the UP.browser (together with the Nokia emulator).

We enabled cookie support on the emulators in order to avoid having to encode the URLs. This, of course, can be changed if cookies are not supported on the WAP gateway. See also Section 10.4, “Best practices” on page 192 for more information.

Note: The Everyplace Wireless Gateway, part of other WebSphere Everyplace Suite offerings, supports cookies on behalf of the connected WAP clients.

15.1.3 New and modified code

Our work consisted almost exclusively in developing new JSPs in WML. The only exceptions were the following:

- ▶ Since we did not use any PVC adapter, the `TradeAppServlet.performTask` method had to recognize the client using the User-Agent from the HTTP header.
- ▶ We had to set the content type of the header to **text/vnd.wap.wml**

This was done in the `requestDispatch()` method of the `TradeServletAction` class (subclass from the abstract `TradeServletAction` class), in the package `direct.wml`:

```
private void requestDispatch(  
    ServletContext ctx,  
    HttpServletRequest req,  
    HttpServletResponse resp,  
    String page)  
    throws ServletException, IOException {  
    resp.setContentType("text/vnd.wap.wml");  
    ctx.getRequestDispatcher(page).include(req, resp);  
}
```

- ▶ As mentioned above, in the portfolio application we sent the entries three by three. Therefore, the start and end indexes of the rows were computed, given the position of the current row in the corresponding HTML table. In order not to overload the JSPs with Java code, we computed these indexes in the `doPortfolio()` method of the `TradeServletAction` class.

Please note that we did not have to modify the underlying back-end components (Access Beans, EJBs, etc.). As we had assumed, we were able to use the whole existing back-end infrastructure for the new content.

15.1.4 Developing the content JSPs

We started the development of the new JavaServer Pages by taking copies of the HTML pages in WebSphere Studio.

In WebSphere Studio, we then removed any redundant information or data that would not fit on the small screen of a mobile device:

- ▶ Header and footer of the page.
- ▶ Any sidebar that only contains links to external sites.
- ▶ All images.

In this case, no image was really relevant for the application. In other circumstances, we might have kept images if they had presented some added value.

We also shortened the titles of some screens in order to save screen space.

Before actually starting to code WML, first make sure WebSphere Studio associates the extension .jsp with WML instead of the default HTML. To do this, perform the following steps:

1. In WebSphere Studio, select **Tools -> Registration**, then for extension type JSP. The window shown in Figure 15-10 appears.

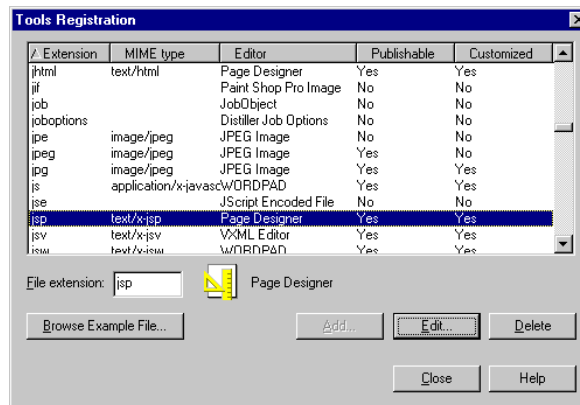


Figure 15-10 Tools registration window in WebSphere Studio

2. Select the **JSP** extension and click **Edit**. The following screen appears:

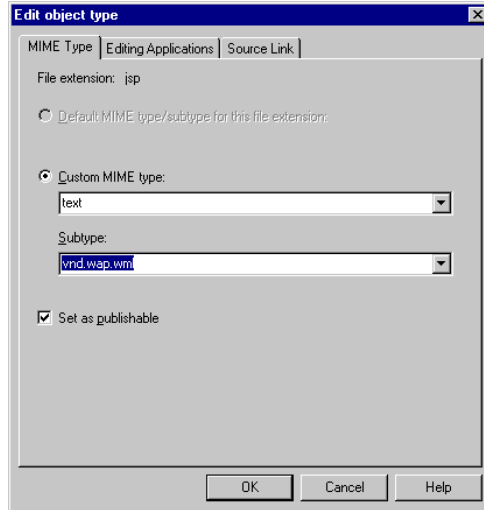


Figure 15-11 MIME-type editing window

3. Associate the JSP extension with text/vnd.wap.wml.

This does not change this setting for the existing JSPs, however. In order to associate existing JSPs with WML, edit the properties of each JSP and associate it with text/vnd.wap.wml, as shown in Figure 15-12.

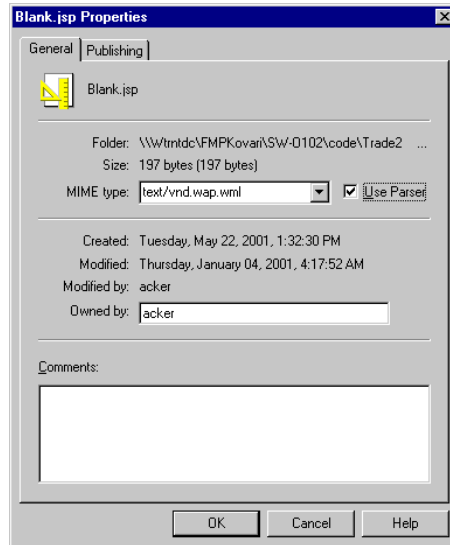


Figure 15-12 Modification of MIME-type for existing JSPs

Note that this only has an impact on the parser and validator mechanism of WebSphere Studio. During editing, the page designer will only support the source editor for WML.

Important: We noted that, when working with tables in WML, the error-checking mechanism gives false error alerts. We recommend that you ignore them in this case.

We also set the output characters for WML (in the Page Designer options) to lower case, as shown in Figure 15-13.

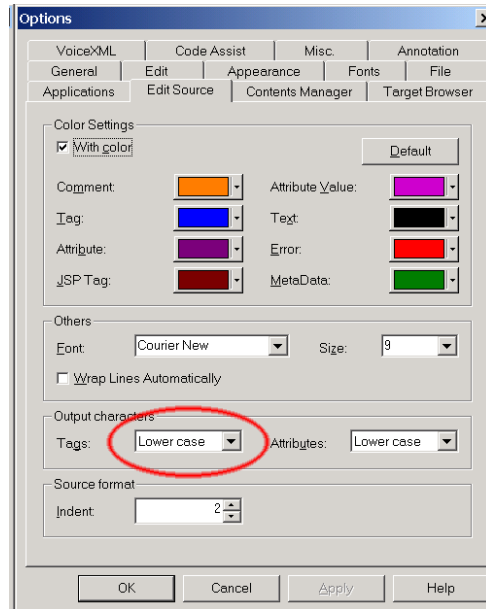


Figure 15-13 Setting the output characters to lower case

Now we are ready to code the pages in WML.

15.2 Content transcoding (HTML source)

Content transcoding is not an industry standard term, and is not used widely. In this book, we chose this term to define those solutions where the transcoding capabilities of WTP is utilized using an HTML source. The important factor is that HTML is the source document in this case.

The content of established Web sites today is written in HTML. Rewriting all the content to a markup-independent language like XML involves huge modifications to a Web site. To overcome this problem, Section 9.3, “Mobile architecture” on page 132 introduces a new technique called Web Intermediaries. This technique helps Web sites to have their content transcoded easily. To demonstrate this, a simple document originally written in HTML will be output in WML using the annotation and transcoding features of WebSphere Transcoding Publisher.

IBM WebSphere Transcoding Publisher 3.5 (WTP) implements the concept of Web Intermediaries. The following examples introduce the features and functions of WTP. The following issues will be addressed:

1. Preferences
2. Annotators
 - Example using external annotators
 - Example using internal annotators
3. Text clipping
4. Guidelines for content transcoding
5. HTML to cHTML

A prerequisite for successful content transcoding is that the data source written in HTML be syntactically correct. This issue not only addresses the structure of a document, but also the way in which the data is presented.

15.2.1 Preferences

WTP provides several preferences for representing specific networks and devices. These are called *profiles*. Each individual profile has some characteristics that give indications as to how a document that is delivered to that device or network should be treated.

Network profiles

The network profile is used to determine a specific network type and apply the registered characteristics to the document that is delivered to this network. The following additional preferences can be activated within a network profile:

1. File extensions that are not supported

This preference allows you to prevent large files of a specific type (for example *.mpeg or *.mp3) from being sent through a network.
2. Convert images into image links

Sending image links instead of images will give the user the option to view the images of his/her choice, thereby reducing the bandwidth to send over the network.
3. Remove images from HTML document

With this feature, images are not sent through the network, because due to display limitations of specific device types, images may not appear as originally intended.

4. Compress size of HTML text files

Compressing the size of an HTML output file results in the removal of non-required attributes that are intended to enhance the output format and the replacing of complex elements by simpler ones.

5. Image quality resolution

- a. Compromise between quality and size
- b. Favor high quality over reduced size
- c. Favor small size over quality

6. Enable image transcoding

Images can be transcoded for better display on the client side. The better the image on the client side, the bigger the size; hence the bandwidth is increased.

With these settings, the speed, bandwidth and load pertaining to a specific network can be decisively influenced. For example, on a wireless network, images should be converted to links for a faster page upload. See the *IBM WebSphere Transcoding Publisher Version 3.5 Administrator's Guide* for more information on working with preferences.

Device profiles

The device profile is used to determine a specific device type and apply the registered characteristics to the document that is delivered to this device. A device type is mostly determined by its User Agent in the HTTP header, hence each device type has a defined User Agent. If the User Agent is not sufficient to define a device, then additional criteria can be added on, such as the Accept value. The following is a list of available preferences:

1. Fragmentation preferences

A WML or HDML document is often too large and cannot be viewed on the client's device. Such a document should be fragmented into smaller pieces with links to navigate through each piece. This preference allows you to set a fixed number of bytes that should not be exceeded.

2. Output type preferences

Here, a desired content type and document type definition can be included. Some types of devices can support several content types, and these can be listed here.

3. Java/XML preferences

This preference allows the profile to support Java applets, JavaScript, Cascading StyleSheets and objects.

4. Image preferences

This preference determines whether images should be supported, scaled, removed, or converted to links. Also, specific types of images can be supported, such as *.mpeg or *.gif.

5. HTML/browser preferences

Some HTML/browser-specific properties can be specified here, such as compressing the document, converting tables to lists and frame support.

6. Device-specific preferences

This preference allows the support of device-specific features.

Using these settings, the result document can be fine-tuned in order to obtain the best viewing result on the client's device. This is also a good means of getting a better overview and makes it easier to administrate specific profiles. See the *IBM WebSphere Transcoding Publisher Version 3.5 Administrator's Guide* for more information on working with preferences.

15.2.2 Annotators

This section provides an example of annotators and their effects on documents. An HTML document will be transcoded to WML and the result compared to the result of using the direct approach discussed in Section 15.1, "Direct approach" on page 256. A simple WAP client (Nokia WAP Toolkit 2.1) will be used to access the requested document.

Note: The concept of *annotators* should be well understood before going further into this chapter. For more information, read the IBM Redbook *New Capabilities in IBM WebSphere Transcoding Publisher Version 3.5 Extending Web Applications to the Pervasive World*, SG24-6233.

Data flow

The following diagram depicts the annotator flow under WTP.

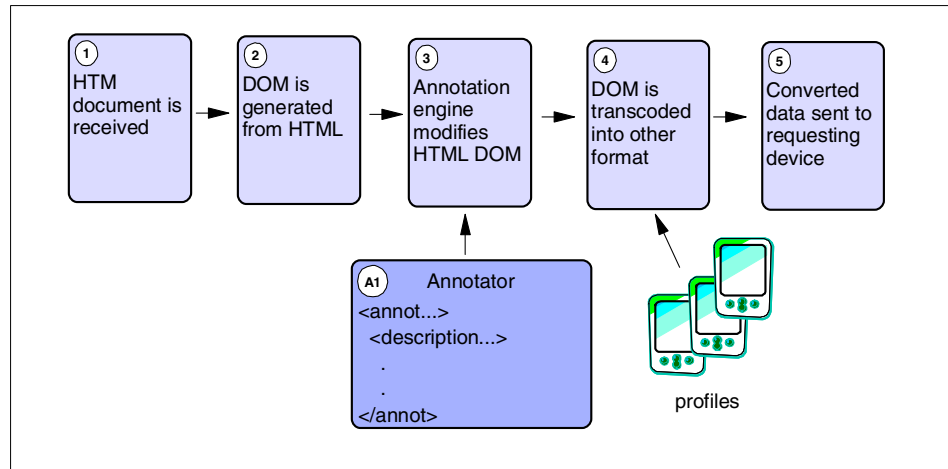


Figure 15-14 Annotator flow

The process of applying an annotator is as follows:

1. WTP receives a HTML document.
2. A DOM object is generated from the HTML document.
3. The annotation engine is triggered and the annotator file is used to merge into the HTML DOM object.
 - a. In case of external annotation, the registered annotation file is used.
 - b. In case of internal annotation, internal annotation instructions are used.
 - c. Internal and external are merged into the DOM object and the annotation engine sets the modifications as specified.
4. The DOM object is transcoded into WML.
5. The transcoded WML document is sent to the WAP client.

Before the HTML document is transcoded to WML, it has been clipped by the annotation engine using the registered annotator. This flow is not dependent on whether internal or external annotators are used.

Note: The HTML DOM generator must be enabled if using a desktop browser to view the applied annotation. Go to the Transcoding Publisher's Administrator Console and set the transcoder to **Enabled**.

Example using external annotators

This section focuses on using the external annotators with WTP, and provides an example for the Trade2 sample.

Annotation file

The annotation file is a very simple one. The syntax used in specifying where an annotation should be inserted uses the XML Path Language (XPath). Using the XPath expression, the node where annotation should be applied can be easily specified. The take-effect attribute of <description> indicates whether the annotation should be applied before or after the node. The inner tag then tell the annotation whether it should remove or keep the content. The following are examples pertaining to the used annotator.

Example 15-1 Annotation example 1

```
<description take-effect="before" target="/HTML[1]/BODY[1]/*[1]">
  <remove />
</description>
```

This annotation tag will be put *before* the *first* occurrence of a tag after the <body> tag. The action of this description tag is to *remove* everything that follows the <body> tag. Hence everything will be removed until the annotator encounters another <description> tag.

Example 15-2 Annotation example 2

```
<description take-effect="before" target="/descendant::TABLE[5]">
  <keep />
</description>
```

This annotation tag instructs to take effect *before* the *fifth* table of the document, and the action is to *keep* the content from the 5th table before the defining tag. Hence everything will be kept from the fifth table on until the annotator encounter another <description> tag.

Example 15-3 Annotation example 3

```
<description take-effect="after" target="/descendant::TABLE[6]">
  <remove />
</description>
```

This annotation tag instructs to take effect *after* the *sixth* table of the document, and the action is to *remove* the content. Hence everything will be removed after the 6th table tag.

These simple instructions are enough to extract the most important information from a fully written HTML document.

Using annotators with the Trade2 application

The following is the HTML output of the requested document.

IBM®

Home | Products & services | Support & downloads | My account

→ Select a country

Home

Login

Register

*Full name:

*Address:

*E-Mail address:

*User ID:

*Password:

*Confirm password:

*Opening account balance: \$

*Credit card number:

Submit registration

Redbooks

6/28/2001

PVC

→ IBM's [Pervasive Computing](#) page. Everything about the pervasive world...

About IBM | Privacy | Legal | Contact

Redbooks Home

Figure 15-15 HTML output

Without even showing the WML output, we can see that this output will result in many cluttered screens on a WAP client. This is not because the HTML source is not transcoded well into WML by WTP, but because transcoding does not involve any modification pertaining to presentation. WTP simply transcodes the content and forwards it to the client. Therefore, the content must be annotated or clipped before transcoding is applied.

The following is the annotator file that will be used in the example. Save this file as TradeRegister.ann for registering in the WTP administration console; you can also find the file within the additional materials provided with the book.

```
<?xml version="1.0" ?>
<annot version="1.0">
  <description take-effect="before" target="/HTML[1]/BODY[1]/*[1]">
    <remove />
  </description>
  <description take-effect="before" target="/descendant::TABLE[5]">
    <keep />
  </description>
```

```

<description take-effect="after" target="/descendant::TABLE[6]">
  <remove />
</description>
</annot>

```

See the *IBM WebSphere Transcoding Publisher Version 3.5 Administrator's Guide* on how to register an annotator. While registering, specify the URL to annotate as: `url~*/trade/register*`

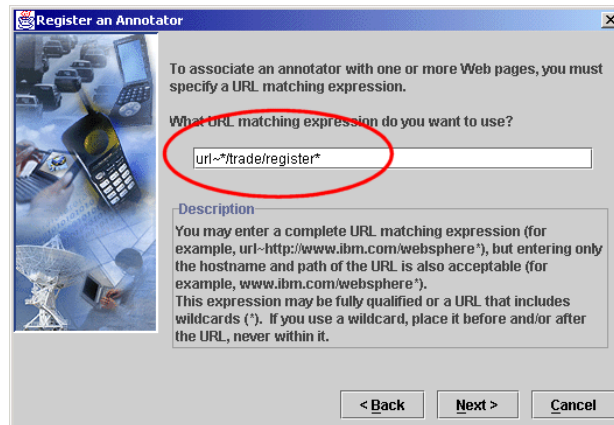


Figure 15-16 URL to annotate

Another advanced value must be added in order for the registered annotator to function. Use the Administration Console of WTP, click the registered annotator, click **Advance** and input the following key and value into the Criteria matching preferences fields:

Key=**deviceType**, Value=**WML Device**.

Then save and refresh the server.

This key/value restricts this annotation to WAP devices only.

Tip: Due to different display screen sizes on different mobile phones, a more accurate design for each display can be implemented by adding other key/value pairs and annotators.

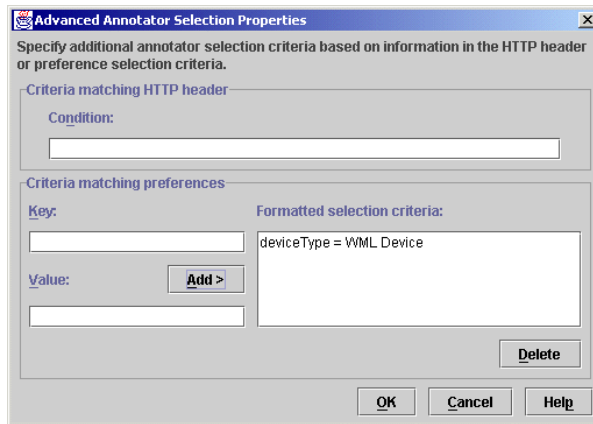


Figure 15-17 Criteria matching preferences

Important:

Some HTML designers like to code variable in two different tokens, for example

```
<input type="text" name="full name" value=""></input>.
```

If that is the case, WTP will simply keep the value of the **name** attribute and not change it, but according to the WML 1.1 specification, the value of a **name** attribute must be NMTOKEN. This results in an error at the WAP client when such a document is received. To overcome this, the HTML and servlet source needs to be changed.

Also, in HTML some tags can be set to **readonly**, but there is no such attribute in WML 1.1 and WTP will not delete this attribute from that tag during transcoding. This results again in a message from the WAP client stating that an unknown attribute has been encountered.

The annotator is now ready to be used. In this example, the Nokia WAP Toolkit 2.1 is used as a WAP emulator. The following are the screenshots of the resulting annotated and transcoded document.

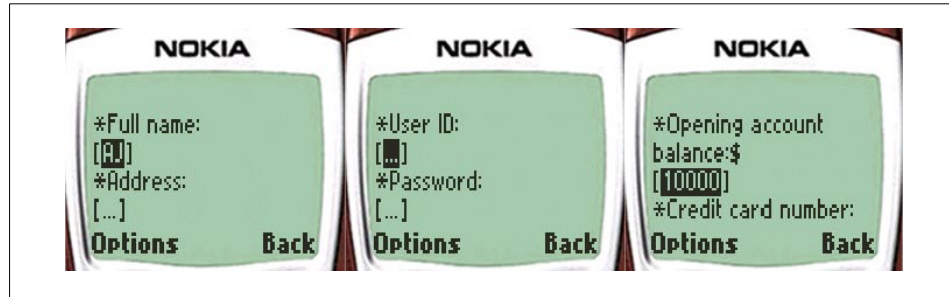


Figure 15-18 Externally annotated and transcoded document

Comparing the result to the result using the direct approach

See Figure 15-3 on page 258 for the result of direct approach. Actually, there is not much difference between the result using direct approach and the content transcoded result. From the developer's point of view, there should not be any differences using content transcoding. The most important information is extracted by the annotator, so superfluous text or pictures are removed from the document. The remaining document is also transcoded by WTP correctly. The user can enter the information the same way as on a normal Web browser. The resulting document is annotated to one WML deck, because the user can still scroll the document to input the information. No other communication is needed to request another fragment of the document. This very much simplifies the data flow and logic.

Note: Note that the UP.Browser behaves differently from the Nokia WAP browser. It has a different user experience, which is different in representing the content and navigating the pages. Refer to the direct approach regarding this in Section 15.1.2, "Test clients for development" on page 263.

Example using internal annotators

This section focuses on using the internal annotators with WTP, and provides an example for the Trade2 sample.

Internal annotators

The syntax of internal annotators is almost the same as for external annotators. Internal annotation tags only have to be included straight into the HTML file. XPath is actually not needed if using internal annotators, but can be included, allowing the user an easy migration from internal annotators to external annotators. The following example shows a simple internal annotator.

```
<html>
<head>
```

```

<meta name="GENERATOR" content="IBM WebSphere Page Designer V3.5.3 for
Windows">
<meta http-equiv="Content-Style-Type" content="text/css">
<meta name="Annotation_v1.0" content="remove">
<title>Websphere Everyplace Access Redbook</title>

```

At the beginning of every document using internal annotators, there must be a `<meta>` tag with attributes of name and content set. The value of the *content* attribute sets the default clipping state of this document. Subsequent `<annot>` tags are enclosed within the `<body></body>` tags.

Example 15-4 Internal annotators

```

...
<body bgcolor="#ffffff" onload="preloadImages();">
<!--
  <?xml version="1.0" encoding="ISO-8859-1"?>
  <annot version="1.0">
    <keep />
    <table majoraxis="row">
      <column index="1" clipping="keep" />
      <column index="2" clipping="remove" />
      <column index="*" clipping="keep" />
      <row index="*" clipping="keep" />
    </table>
  </annot>
-->
<b>Portfolio for johnd</b>
<p />
<table border="0" width="100%" class="tblu">
  <tbody>
    <tr>
      <td align="center" width="10%"><strong>symbol</strong></td>
      <td align="center" width="10%"><strong>index</strong></td>
      <td align="center" width="10%"><strong>price</strong></td>
      <td align="center" width="10%"><strong>shares</strong></td>
      <td align="center" width="10%"><strong>value</strong></td>
      <td align="center" width="20%"><strong>buy</strong></td>
      <td align="center" width="15%"><strong>sell</strong></td>
      <td align="center" width="15%"><strong>quote</strong></td>
    </tr>
    <tr>
      ...

```

The main difference of syntax between internal and external annotators is that each annotation instruction of internal annotation must start with the declaration of `<xml>` and the instruction of whether to keep or remove must be inside an `<annot>` tag. Inside the `<annot>` tag, any other annotation tags can be used. The instructions must be placed before the section where annotation occurs. Also, instructions must be placed inside a comment. In Example 15-5, column 2 of the table will be removed, column 1 and any rows will be kept.

Example 15-5 Internal annotators

```
...
<form action="/trade/servlet/TradeAppServlet" method="post">
  <td align="center" class="vlblu" valign="center">
    <input type="submit" value="quote">
    <input type="hidden" name="action" value="quote">
    <input type="hidden" name="symbol" value="SUNW">
  </td>
</form>
<tr>
  <td colspan="8">&nbsp;   </td>
</tr>
</tbody>
</table>
<!--
  <?xml version="1.0" encoding="ISO-8859-1"?>
  <annot version="1.0">
    <remove />
  </annot>
-->
...
```

The instruction used here is fairly simple. Everything after that instruction inside the document will be removed.

Using annotators with the Trade2 sample

This is the original HTML document that will be annotated.

IBM

Home | Products & services | Support & downloads | My account

Select a country

Home
Account
Portfolio
Logout

Portfolio for johnd

symbol	index	price	shares	value	buy	sell	quote
RHAT	8008	25.0	100.0	2500.0	buy 100	sell	quote
RHAT	8017	25.0	100.0	2500.0	buy 100	sell	quote
IBM	9001	178.0	100.0	17800.0	buy 100	sell	quote
IBM	14001	178.0	100.0	17800.0	buy 100	sell	quote
RHAT	15000	25.0	100.0	2500.0	buy 100	sell	quote
IBM	16000	178.0	100.0	17800.0	buy 100	sell	quote

Redbooks

6/28/2001

PVC

→ IBM's [Pervasive Computing](#) page. Everything about the pervasive world...

About IBM | Privacy | Legal | Contact

Redbooks Home

Figure 15-19 Account document

Again, this document contains too much in the way of text and pictures, which are not suitable for display on a WAP device. Hence, this document needs some annotation before it can be transcoded to a final document. The annotation technique introduced here uses internal annotation, which uses almost the same syntax as external annotation. As the name implies, all annotation tags are embedded inside a document which should be annotated, so there will be no registration of an internal annotation as for external annotations.

Important: If WTP encounters nested tables in an HTML document, it could be that the document is not transcoded correctly into WML. Use nested tables sparsely!

The document contains many tables, therefore in the device profile of WAP the feature **Convert tables to lists within lists** must be enabled (see Figure 15-20 on page 281).

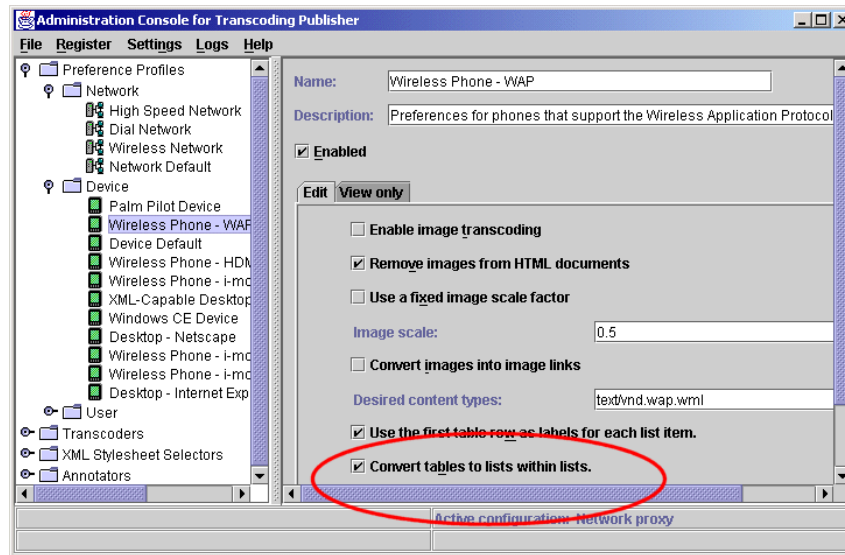


Figure 15-20 Convert tables to lists

This way, the elements will be sequentially shown on the screen. With annotator tags embedded inside the HTML document, the document is ready to annotate.

The following is the result of using internal annotators inside an HTML document.

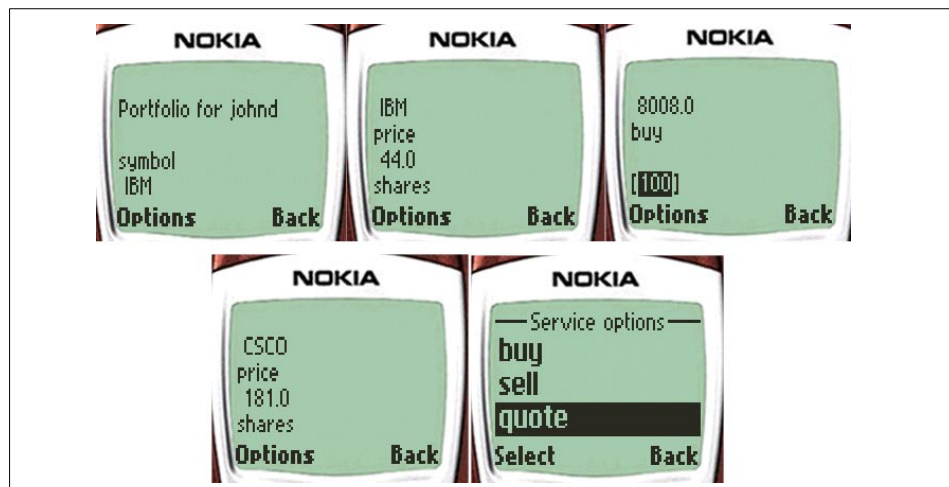


Figure 15-21 Internally annotated and transcoded document

The resulting document using internal annotators seems to achieve the desired result. Tables are converted to lists and the user can easily scroll up and down to reach the desired share. For each share, the user can buy, sell or quote as on a normal desktop browser. Because of the large amount of data, the document is broken down into fragments, so the user must choose to continue in order to request another fragment for the document.

The internal annotation feature in WTP 3.5 has just been introduced and more flexibility will be added soon. For now, once internal annotations are embedded into a document, the document will be annotated by WTP every time it is requested. There is no possibility of applying conditions to internal annotators.

It is not possible to create a condition to specify when internal annotations should be applied. Currently, with WTP 3.5, internal annotators support an all or nothing approach, that is, there is conditional annotation (not supported). With external annotation, it is possible to create an individual condition simply by adding a key/value pair to the registered annotator. This means that no matter which client is requesting an embedded document with internal annotations, this document will definitely be annotated. The reason for this is that the annotation engine (internal or external) is running before the transcoding engine (see the data flow in Section 15.2.2, “Annotators” on page 271). In the case of internal annotation, as soon as the annotation engine encounters an annotation tag in the DOM object, it will modify the object according to the annotation tags of the document.

Also, if annotation is not required, it must be deleted explicitly from the document. If using external annotation, the annotator can be easily disabled.

Internal or external annotators

Table 15-1 gives some pointers on how to decide between internal or external annotators for your application:

Table 15-1 Internal or external annotators

Internal annotator	External annotator
Must have “write” access to the source.	Reading the source is enough to develop annotators externally.
Makes the original code difficult to understand.	Does not make any modification on the original source code.
Very easy to develop, since you only have to put tags into the code, and you know exactly where to find the appropriate part of content.	Difficult to map the external annotators to the original source. XPath references are used for matching.

Internal annotator	External annotator
WebSphere Studio v3.5 Page Designer supports the internal annotators.	This time there is no such external annotator editor. WTP 4.0 will come with an editor.
Easy to apply to dynamic content, because the annotators are also created dynamically.	Very difficult to apply dynamic content, because the source is always changing even if the structure is not.

One reason not to choose an internal annotator is that nowadays most of the documents are created dynamically. This makes it difficult to decide when to include internal annotation. WTP is not in control of the annotation of a document. WTP has the function and ability to transcode, annotate and clip documents, but by using internal annotation, a part of this responsibility is taken away from WTP. Doing so makes the system more difficult to maintain.

Using external annotators is recommended, because WTP is still in control of the annotating and transcoding of documents. Also, annotators are easily disabled using the Administration Console. This makes it much easier for the developer to write annotation in a separate file than in the already fully crowded original source. If a document is generated using different frames and source files, then it is more difficult to write internal annotators, because a source file is used not just for one document, but for many other documents as well.

Future directions

Writing annotators requires a good understanding of each markup language and of XPath. Unfortunately, in WTP 3.5 there is no such editor for writing external annotators; this makes it easier to have the result presented directly in the editor. Such an editor will relieve much of the difficulty encountered by the developer. In the next release of WTP, such an editor will be shipped.

For internal annotator editing, WebSphere Studio v3.5 has tools which help the developer to insert annotation tags into the document not only under the source code editor but even in the visual editor mode.

Another alternative to using annotators is *text clipping*, which will be described in the next section.

15.2.3 Text clipping

WTP offers the possibility to clip a document using a custom transcoder called *text clipper*. This is a programmatic solution which must be written in Java. Remember that the concept of Web Intermediaries infers that it is possible to monitor, edit and generate a data stream. A text clipper is an editor that simply edits a data stream and puts it back into the stream after editing. A text clipper can edit a stream either as a DOM object or a text document.

Tip: See <http://www.w3.org/DOM> for more information about DOM

A text clipper is only a customized transcoder inside WTP which has Response Editors implemented. Like any other transcoder, it can be programmed using either servlet API or WBI API.

Tip: See <http://www.alphaworks.ibm.com> for more information about WBI.

Figure 15-22 depicts a sample data flow inside WTP using a text clipper. Due to the possibility, in the properties file, of a MEG specifying trigger conditions, it is possible that a MEG be triggered in the same flow multiple times.

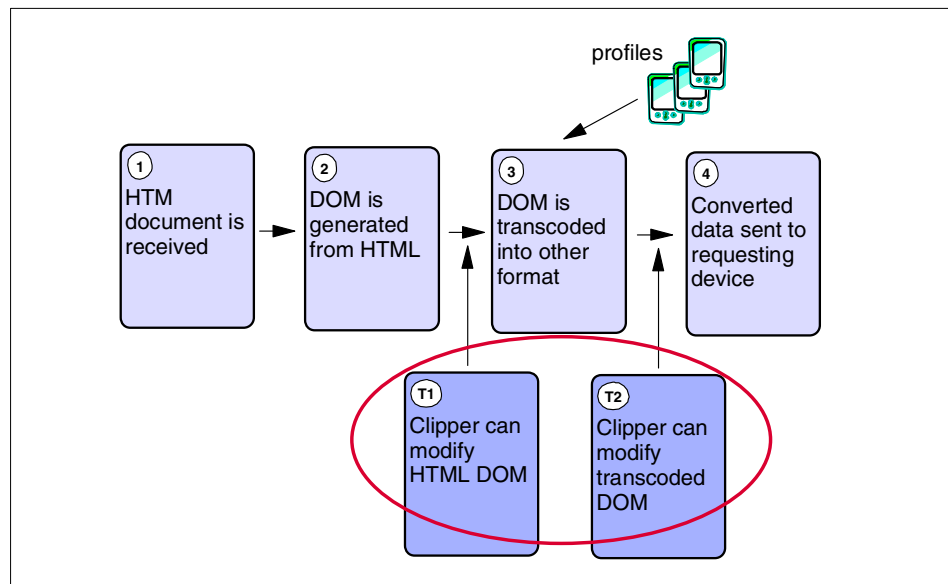


Figure 15-22 Sample data flow of a text clipper

1. An HTML document is received by WTP.
2. WTP generates a DOM object from the content of that HTML document.
T1: A registered clipper can be triggered and edit the DOM object.
3. Edited DOM object is then transcoded into another format.
T2: The same clipper can re-trigger and modify the transcoded DOM object.
4. The clipped and transcoded data is sent to the client.

The advantage of a clipper is that it is more flexible than annotators. A text clipper can be triggered at any time during the whole flow, as opposed to annotators, which can only be triggered before transcoding takes place.

The text clipper can access the document either as a text stream or as a DOM object. This depends on the document itself. In general, the DOM object is preferable. WTP already provides the functionality to access nodes inside a DOM object, another reason to use the DOM object.

The following table may help decide whether to use a DOM-based clipper or a text-based clipper:

Table 15-2 Text-based or DOM-based clipper

Text-based	DOM-based
Easy to understand and follow the content, since it is linear.	Until you understand the structure of the document, it is difficult to follow the content and find the elements in the hierarchy.
Very difficult to find the right element.	Very easy to find and access the right element.
Difficult to identify all the tags related to the element.	Easy to handle the element with all the tags.

Writing a text clipper

The following section is designed to give the developer a starting point in writing a text clipper. The development of a whole text clipper is beyond the scope of this book.

As stated above, a text clipper is a customized transcoder that can be programmed using Sun Java Servlet API v2.1 or WBI API v4.5. The example provided uses WBI API v4.5.

Tip: See <http://www.alphaworks.ibm.com> for API documentation of WBI.

In terms of Web Intermediaries, any transcoder is a plug-in into the framework. Before using any plug-in, it has to be registered in WTP using the Administration Console. A plug-in can be either enabled for use or disabled. Each plug-in can contain a set of MEGs (Monitor/Editor/Generator) which builds the whole transcoder. A MEG has the ability to monitor, edit or generate data. In the case of clipping, at least one MEG must be created as an editor. If a certain MEG is triggered because the registered condition matches, the `handleRequest()` method will be invoked by the framework. The input parameter of this method is an object containing all the information and data that this method will need to clip the text. After clipping the text, the object can either be forwarded to another MEG or sent to the client.

Creating a MEG first

Since a MEG is the smallest working unit, it should be thoroughly designed. In terms of WBI, a MEG must be inherited either from a monitor class, editor class or generator class. The MEG class that is going to be coded is inherited from the editor class. The WTP API comes with a Text clipper class that is of type *editor*, and that is the super class for other implementations.

Tip: Look in the WTP API to see whether a desired MEG is already written before writing your own MEG.

Since the `Textclipper` class is an abstract class with an abstract method `String getPropertiesName()`, this must be implemented. Also, the editor class is an abstract class as well with an abstract method `handleRequest(RequestEvent e)`; this must be overloaded as well. Hence, the MEG class looks like this:

Example 15-6 A MEG class

```
package com.itso.wea;

public class WEATradeEditor extends
com.ibm.transform.textengine.mutator.TextClipper {
    private static final String INIT_PROPERTIES = "plugin/WEATradeClipper";

    public WEATradeEditor() { }

    public String getPropertiesName() {
        return(INIT_PROPERTIES);
    }

    public void handleRequest(com.ibm.wbi.RequestEvent aRequestEvent) throws
com.ibm.wbi.RequestRejectedOperationException, java.io.IOException {
    }
}
```

The variable `INIT_PROPERTIES` indicates the class in which to find the properties file for this particular clipper. In this case, the properties file calls `WEATradeClipper.properties` and is stored in the directory `<WTP install dir>/plugin`. The `handleRequest()` method has an input parameter of type `com.ibm.wbi.RequestEvent`. This type of object contains the information and data this method needs to edit the data.

Accessing the data

The developer can either access the previously transcoded DOM object or the original HTML DOM object before transcoding has taken place. If some information is no longer available after transcoding, then the developer would have to access the original HTML DOM object, otherwise the transcoded DOM object would be used. The following examples show how to get the DOM object in different phases of transcoding.

- Accessing the original HTML DOM object:

```
org.w3c.dom.Document vHTMLDoc = getOriginalDOM(aRequestEvent);
```

- Accessing the transcoded DOM object:

```
org.w3c.dom.Document vDOMDoc = getTranscodedDOM(aRequestEvent);
```

After the DOM document has been retrieved, the nodes can be accessed using either DOM API or `com.ibm.transform.textengine.mutator.DOMUtilities`. For example, to search for a particular text node, use the method `org.w3c.dom.Node DOMUtilities.findChildWithNodeMatchingPattern(Node, "(NYSE: IBM*", true)`. This method simply searches for a text node with the content string "(NYSE: IBM*". In general, inside the `handleRequest()` method the developer has all the freedom to modify the data as needed, such as changing header values or replacing text. It is important that the developer should separate each MEG in a logical unit with a particular type of function. After the MEG is written, it can then be used in the plug-in.

Generating a plug-in

The following example shows a very simple plug-in for WTP:

```
public class WEATradeClipper extends com.ibm.wbi.Plugin {
    public void initialize() {
        WEATradeEditor vTEditorMeg = new WEATradeEditor();
        try {
            addMeg(vTEditorMeg);
        }
        catch(com.ibm.wbi.PluginError vPError) {
            vPError.printStackTrace();
        }
    }
}
```

WEATradeClipper extends the Plugin class of WBI. This is the first requirement of creating a transcoder. Every time WTP is started, the initialize() method will be invoked. The framework must know that there are some new MEGs, therefore inside the initialize() method all MEGs should be added using the addMeg() method. The enable() or disable() methods can also be used to carry out some work, in case the transcoder is enabled/disabled.

After that, the class files need to be packaged and a properties file for the WBI plug-in needs to be generated. Inside the properties file for the WBI plug-in, the following properties must be included:

```
Class=com.itso.wea.WEATradeClipper
Description=Meg clip a document
DescriptiveName=a clipper sample
Major=1
Minor=0
Name=WEATradeEditor
Condition = ( (path ~ */servlet/*) & !(content-type=text/vnd.wap.wml))
Priority = 1
```

Packaging a Web plug-in

The class files should be packaged according to the naming conventions of the classes. The properties files should be packaged into another directory to separate the classes from the properties files logically.

Tip: For more information about packaging, please refer to the following Web site: <http://java.sun.com/docs/books/tutorial/overview/packaging.html>

If there are two or more MEGlets, then each MEGlet has to be packaged into separate .jar file. It is also possible to provide a properties file for each MEG to add more flexibility to the transcoder.

See the *WebSphere Transcoding Publisher Administration Guide 3.5* on how to register a transcoder.

If registering using the Administration Console is successful, then the window shown in Figure 15-23 on page 289 should be displayed.

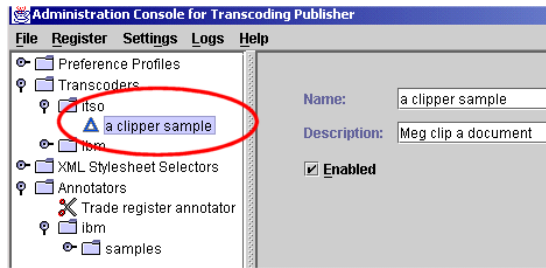


Figure 15-23 Administration Console with new clipper

The following result will be displayed if starting the request viewer:

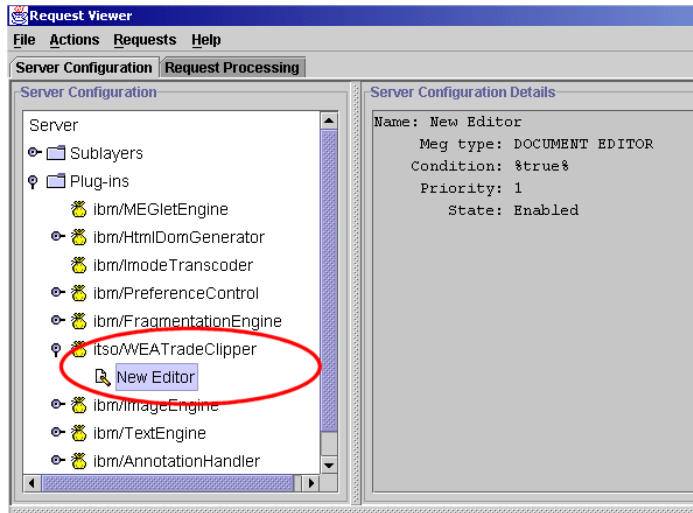


Figure 15-24 Request viewer with new clipper

Leaving this sample transcoder in the WTP environment does no good, because the method `public void handleRequest(com.ibm.wbi.RequestEvent)` is not implemented, but the developer now has the freedom to code anything to clip a document. See <http://<WTP install directory>/toolkit/apidoc/index.html> for more information on WTP API.

15.2.4 Guidelines for content transcoding

This section provides some information on how to write transcoder-friendly HTML documents. HTML documents do not have strict restrictions on syntax and structure. HTML tags do not need to be closed, tags can be written with different capitalization, etc. In general, HTML can be very error tolerant and this makes it difficult to parse, analyze and be presented. This may produce unpredictable results. To overcome such problems, here are some guidelines that should be followed to produce clean and transcoder-friendly HTML documents:

1. Always specify a closing tag to an opening tag.

```
<h1>heading</h1>
```

2. Specify end tags in the correct order.

```
<p>this is a paragraph<b>bold <i>bold italic</i> bold</b></p>
```

3. Always end a tag with "/"

```
<a href="#refs">anchor</a>
```

4. Either use capitalize letter or small letter for tags, but not both in the whole document.

```
<h2>heading</h2> and not <H2>heading</h2>
```

5. Put values of attributes into quoting marks.

```
IMG</img>
```

6. If a tag has no text, then end the tag with "/"

```
<hr width="80%" class="c4" />
```

7. Do not use nested tables, try using <th> and <td> instead.

Tip: There are many of utilities available to help developers write transcoder-friendly HTML document. See <http://www.w3c.org>.

15.2.5 HTML to cHTML

This section discusses content transcoding from HTML to cHTML, a subset of HTML which is used for i-mode. For demonstration purposes, the i-mode emulator from wapprofit.com i-mode emulator 1.1 is used. Since cHTML is a subset of HTML, not all HTML tags are supported. See Section 10.1.8, "HTML, cHTML, HDML, WML" on page 162 for more information on cHTML.

Still, these two markup languages are very much tied together and make transcoding easier than does WML. This does not mean that every HTML document can be transcoded into cHTML easily, however.

To some extent, annotations to the document are needed to clip the document and to lower the bandwidth. The Wapprofit i-mode emulator has a different User Agent than the i-mode browser in Japan; a new profile must therefore be created to show the example. Carry out the following steps before any transcoding can take place:

1. Create an explicit preference profile for Wapprofit's i-mode emulator.

Actually, if a new preference profile is not wanted, then any previously created i-mode profile can be used, but the User Agent value has to be modified. To have a better overview of what device types are supported to be more flexible, creating a device profile is recommended. Any profile can then be dynamically disabled.

The following example shows the source code for an i-mode profile.

Example 15-7 i-mode.prop: source code for the i-mode profile

```
#Preference Profile for i-mode emulator
Description=i-mode emulator profile for WebSphere Everyplace Access Redbook
ConfigurableProperties=desiredContentTypes{text} disposeImages{bool}
textLinksPreferredToImages{bool} fixedImageScale{bool}
propagateFirstTableRowData{bool} transcodeImages{bool}
disposeImages=false
NonConfigurableProperties=supportedImages{text} screenCapability{text}
colorSupported{bool} createCHTML{bool} imodeLevel{itext}
propagateFirstTableRowData=false
colorSupported=true
deviceRule=User_Agent\=*Microsoft URL Control*
fixedImageScale=true
transcodeImages=false
imodeLevel=2.0
DescriptiveName=i-mode emulator
textLinksPreferredToImages=false
createCHTML=true
screenCapability=low
desiredContentTypes=[text/html]
supportedImages=[gif]
```

Register this new generated profile; see the *Wtp Administration Guide* on how to register a profile.

2. Register the i-mode transcoder

By default, the i-mode transcoder is not registered; the administrator has to explicitly register this transcoder. It can be found in <WTP install directory>/plugins/ImodeTranscoder.jar. See the *WTP Administration Guide* on how to register a transcoder.

The following example uses the registration page as did the previous one.

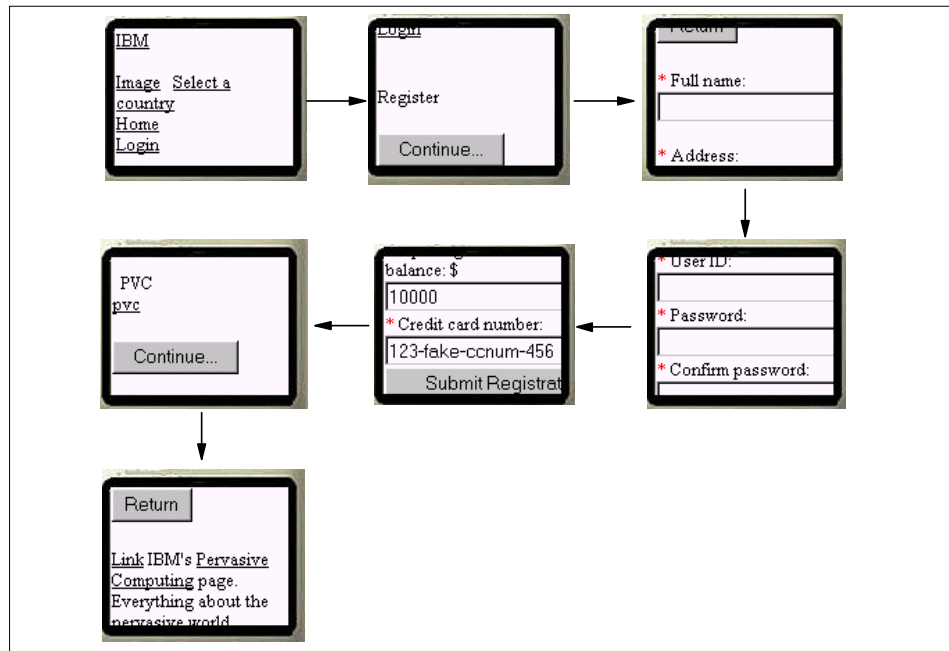


Figure 15-25 i-mode example

The result of this cHTML document is very similar to the original HTML document without any annotation or clipping. WTP has broken the whole document into many fragments. The user has to request each fragment, but the content is structured and the document can be used as it is. It is not recommended to keep this document, because too many fragments are generated and not all fragments are needed to gather the information. For example, the starting picture in Figure 15-25 which displays some information and alternative links is superfluous to this document's intent. It is therefore recommended to annotate this document to disregard some of the information. To do this, external annotations are recommended here, and the same annotation descriptions TradeRegister.ann from the previous example can be used. Simply register TradeRegister.ann again using the same configuration. If needed, special conditions can also be defined with additional key/value pairs. Having registered TradeRegister.ann, the following result will be shown when accessing the document (see Figure 15-26 on page 293).

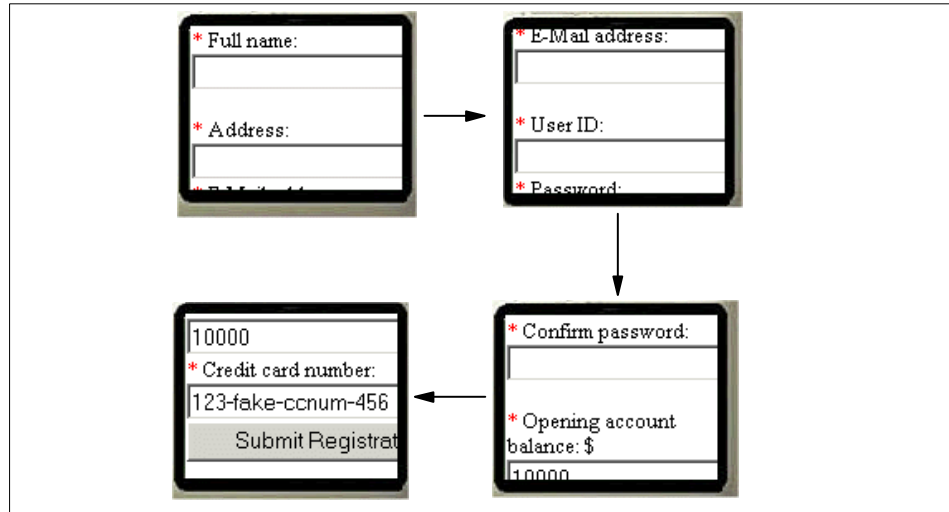


Figure 15-26 Annotated document while accessing using an i-mode client

The external annotator has clipped the document to just one fragment, keeping the most important information and disregarding the rest. In general, there are no differences between annotating to WAP or to cHTML, because the original HTML document is annotated, not the transcoded document. Hence, the same annotation file can be used to achieve the same result.

15.3 Universal transcoding (XML source)

Universal transcoding is not an industry standard term and is not commonly used. This term defines those solutions where the XSLT engine of WTP is utilized. It is basically merging XML and XSL documents on the server side, based on different conditions.

In this section, we will discuss the possibilities of using XML in the Trade2 application and how to handle it to produce multiple markup language output formats. To produce this output, XSL StyleSheets and Document Type Definitions (DTD) handled by the WebSphere Transcoding Publisher are used.

We outline problems we faced in our scenario and the solutions we assigned to them. Furthermore, we discuss the alternatives to the solutions and the best practice guidelines.

15.3.1 Converting XML to different markup languages

There are two different strategies to apply and handle an XML application in combination with the WTP.

1. The first and most obvious possibility is to use a direct XSL approach. In this case, an XSL StyleSheet is deployed to the WTP for every screen and for every device class. The major advantage of this solution is the customized specific user interface. Furthermore, the StyleSheet can access and display all information and data from the given XML stream. Also, the different look and feel of each markup language can be changed easily without impact to the other contents. The application therefore becomes more flexible. The disadvantage is the number of needed StyleSheets. For each device and for each XML document, a separate XSL document has to be created, deployed and maintained. An example: if there are M source XML documents and N devices, M x N StyleSheets must be created. More development and maintenance effort is required.

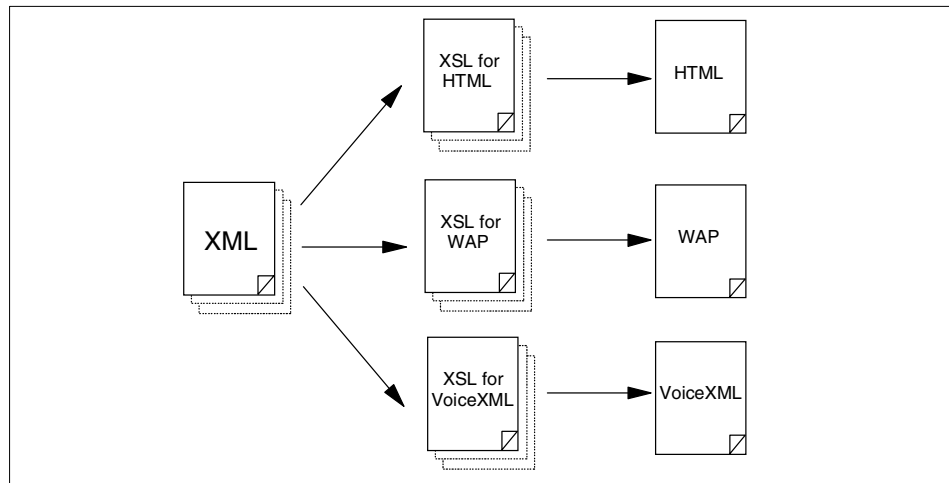


Figure 15-27 XML and XSL for transformation

2. There is the possibility of using HTML as a common intermediate format. In this case, only one StyleSheet per page will be applied to generate the HTML representation of the XML document. The result is rendered by the various HTML transcoders in WTP in order to receive the desired markup language, such as cHTML, WML or even VoiceXML (see Figure 15-28). The advantage is simplified implementation because only one StyleSheet per page is required, or even fewer if generalized StyleSheets can be used for multiple XML documents. Less development and maintenance effort is needed. This approach can speed up the development and later on, since the application is already based on XML, the StyleSheets for each device class can be

developed later. The disadvantage is the intermediate format. Changes in the HTML format could have an impact on the other derived device-specific content. Information from the original XML document can get lost during WTP processing. Furthermore, the look and feel and the workflow of the different markup languages depends on transcoder engines in WTP.

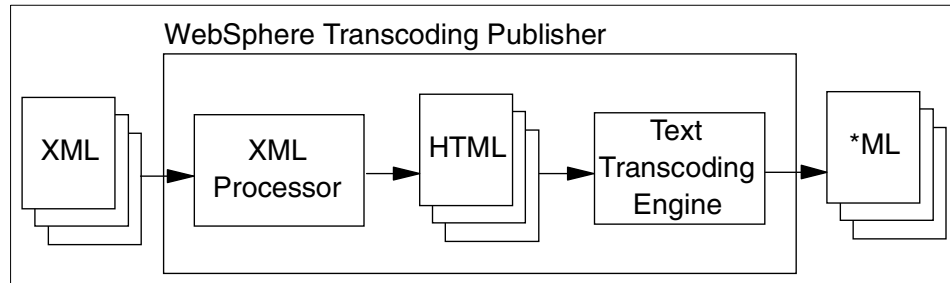


Figure 15-28 Using HTML as an intermediate format

You must decide whether it is better to have a separate StyleSheet for that particular device, or to have only one for HTML then to transcode the HTML content to the appropriate content format.

The best approach is always to use a hybrid solution, taking advantage of the newer technologies, but using primarily the reliable, proven ones. In this case, designers should count on using two sets of StyleSheets, one for the primary device, usually desktop PCs using Web browsers (for example Netscape), the second for the other devices. The second StyleSheet only produces an intermediary format, which is easily adaptable to the other documents formats required by the devices. The following diagram depicts this hybrid solution.

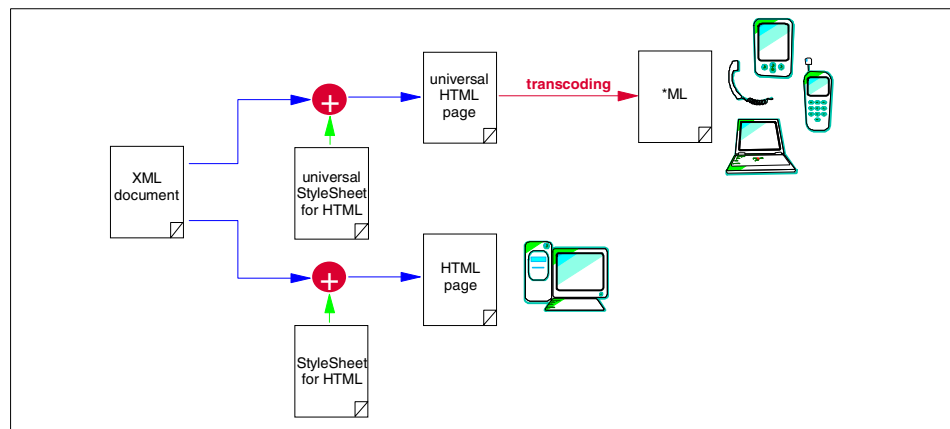


Figure 15-29 Hybrid XML and transcoder solution

15.3.2 Trade2 example

In our sample for this scenario, we chose the simplified HTML as the result content. Simplified HTML is normal HTML, the difference lies in the usage of HTML. Simplifying the content means avoiding fancy, sophisticated HTML formatting, and not using incompatible features.

15.3.3 Design decisions

First of all, we decided to modify the original Java classes to return the XML code we needed. For this, only the second tier should be changed by adding new functionality and code. Therefore, the inherited TradeAppServlet class has to be modified. The EJBs and the database (third tier) are kept untouched.

The alternative to this solution is to use JSP files to produce the XML output. The major advantage of this solution is that the code does not need to be modified at all. Unfortunately, the MVC model is becoming inconsistent. The JSPs are no longer only used to generate the view for the client (HTML, WML and so on). Therefore, we decided to produce the XML content inside the Java classes.

The following picture shows the two different approaches for generating the XML content.

1. The servlet writes the XML content directly into the response stream.
2. The servlet prepares the data, puts it into the scope (request, session or application), then the JSP generates the XML content using the objects from the scope.

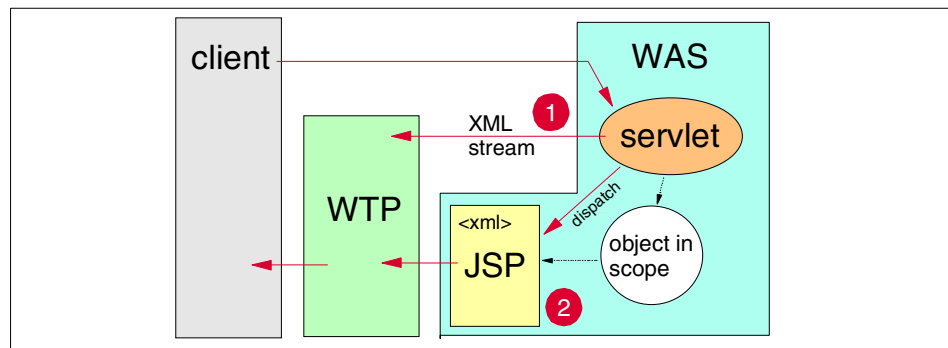


Figure 15-30 XML data generation

In our sample, we choose the first option, where the servlet generates the XML content.

15.3.4 Defining the class concept

As mentioned before, we decided to modify the second tier of the Trade2 application. This means that we create a new subclass inherited from the abstract TradeAppServlet. This subclass implements all the known methods from the original class like doLogin(), doAccount(), doPortfolio() and so on (see Section 14.3.1, “The three tier servlet architecture” on page 243).

In order to process the creation of the XML, we decided to add two classes (TradeXMLHandler, TradeXMLHanlderException) and two interfaces (ITradeXMLHandler, ITradeXMLHandlerException).

The first class TradeXMLHandler is defined as an abstract class. It implements the two fields oTable (hashtable) and oXML (String), as well as the three methods setInput(), getXML() and execute(). The first field, oTable, is used as a container to transfer the dynamic data to the TradeXMLHandler object. This dynamic data can be, for example, the user ID, account balance or a quote object. The hashtable is set to the TradeXMLHandle object by the setInput() method.

The execute() method is an abstract method. It is implemented by the subclasses TradeXMLWelcome, TradeXMLHome, TradeXMLAccount, TradeXMLPortfolio and TradeXMLQuote (see Figure 15-31 on page 297). The main purpose of this method is to generate the XML data stream by using the dynamic data from the set hashtable. This XML data stream is simply stored in the oXML field and can be returned via the getXML() function.

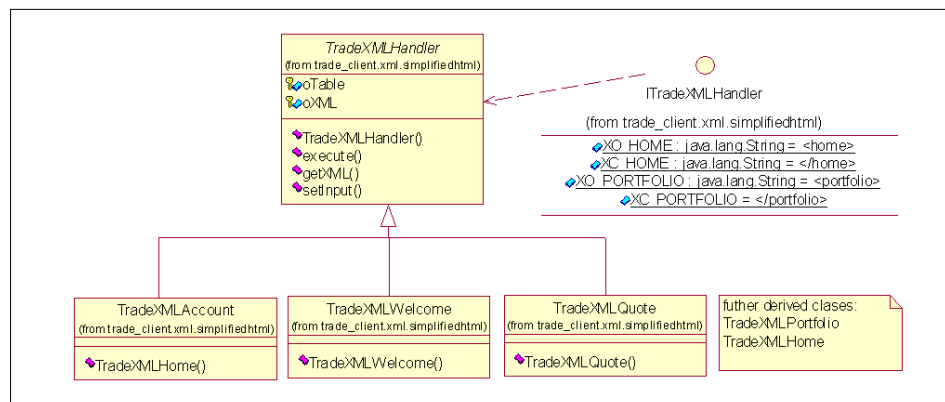


Figure 15-31 Class design for the TradeXMLHandler

In addition, the TradeXMLHandler class also implements the ITradeXMLHandler interface. The interface defines the used XML tagset for the Trade application and allows the developer to easily adapt the tagset to minor changes in the XML structure.

The second new class TradeXMLHandler manages the error handling of the application. As shown in Figure 15-32 on page 298, it contains a private field, StringBuffer XMLData and the three methods TradeXMLHandlerException(), TradeXMLHandlerException() and getString().

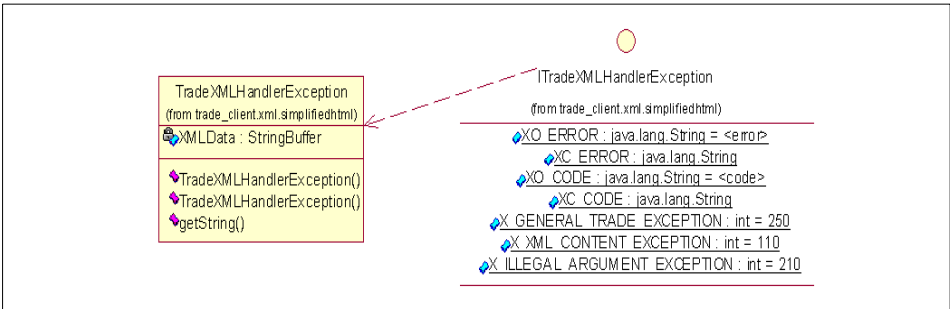


Figure 15-32 Class design for the TradeXMLHandlerException

The link between the TradeXMLHandler and TradeXMLHandlerException() is the oXML field. The variableXMLData is used to store the XML strings for errors. The getString() method returns this string. To build the stream, the TradeXMLHandlerException() is used. It wraps the given error code in the defined XML structure and stores it in the XMLData variable.

For a comfortable maintenance of the TradeXMLHandlerException class, the interface ITradeXMLHandler is added to the application. It contains the defined XML tagsets and error codes for the handling. In this way, minor changes in the XML structure and error code values can be dealt with by modifying the interface.

15.3.5 Defining XML structures

For the specification of the XML structures, we focused on the provided information of the Trade2 application and the workflow. As a template, we used the predefined dynamic data for the JSP files. A summary of all defined XML structures and their behavior is given in Table 15-3 on page 299.

Table 15-3 Overview of defined XML structure

XML structure name	Function	Includes dynamic data	Created by application
TradeWelcome	Used for the index page of the application	NO	YES
TradeLogin	Used for the logon page	NO	NO
TradeRegister	Used to register a new user	NO	NO
TradeHome	Used for the homepage for a logged on user (includes user ID and account balance)	YES	YES
TradeAccount	Used to display the account information of a user	YES	YES
TradePortfolio	Used to show the portfolio information of a specific user	YES	YES
TradeQuote	Used to show quote information for the stock symbol	YES	YES
TradeError	Used for the error page (includes error code)	YES	YES

Note: The last column indicates that *static XML* data does not automatically mean that the data is stored in a file; this column will indicate whether the XML is generated by the application or whether it is a simple XML file in the directory structure.

Note that the TradeWelcome XML file is the only XML structure that is provided static and dynamic. The reason for this is that the method for the logout of the application should not reference the static XML file. Therefore, a new method, doWelcome(), was implemented in the class (see Figure 15.3.8 on page 302 for details).

15.3.6 Creating XML files

In order to develop the XSL StyleSheets for the application, we have to provide the valid XML structure first. To create the XML documents, both tools, WebSphere Studio 3.5 and XML Spy 3.5, were used (refer to Chapter 10, “Application development” on page 153).

To store the new resources we define the following three new folders inside our project file:

- ▶ xml/simphtml/xml : to store the XML
- ▶ xml/simphtml/xsl : to store the XSL and DTD files
- ▶ xml/dtd : to store the DTD files

To create the documents, we used the XML Spy software (see “XML Spy 3.5” on page 179). In order to access it directly from the Studio Workbench, we registered it as the default editor for XML, XSL and DTD resources.

15.3.7 Developing the XSL files and the user interface

In order to produce the simplified HTML code, our main goal was to delete the nested tables and to reduce the number of pictures used in the original application. The navigation bar therefore appears as an additional header and below the original header at the top of the page. The link to the IBM Pervasive Computing Web Site was included in the footer bar. The sidebar with the picture and the link to the IBM PVC site is only used on very simple pages, for example the welcome page. However, to preserve an appealing user interface it was necessary to keep a number of tables (for example in the login page or portfolio page).

In the next development step, we created the XSL files to display the data in the user interface. We implemented the following XSL StyleSheets, according to the workflow and the defined XML structures:

Table 15-4 Overview of defined and implemented StyleSheets

StyleSheet name	Function	Dynamic data processed	Corresponding XML structure
TradeWelcome	To display the index page	None	TradeWelcome.xml
TradeLogin	To display the login panel	None	TradeLogin.xml
TradeRegister	To display the register page	None	TradeRegister.xml
TradeHome	To display the home page (after login)	User ID and balance	TradeHome.xml
TradeAccount	To display the account page	User account data	TradeAccount.xml

StyleSheet name	Function	Dynamic data processed	Corresponding XML structure
TradePortfolio	To display the portfolio information	User portfolio data	TradePortfolio.xml
TradeQuote	To display the quote information for the stock symbol	Quote data	TradeQuote.xml
TradeError	To display the error page	Error code	TradeError.xml
TradeTemplate	Template to display headers, navigation bars, sidebar and footers; imported in the other StyleSheets	None	None

As shown in Table 15-4, we added a TradeTemplate StyleSheet to handle the recurrent elements of the page. This template contains the HTML code for the different headers, footers and navigation bars as well as a sidebar with a link to the PVC home page. The TradeTemplate StyleSheet is included via the imported statement in the other StyleSheets.

Note: To run the StyleSheets in the WTP environment, you have to reference the imported StyleSheets with the full directory path (see XSL in the Studio project for an example). Otherwise, for relative referencing, the \etc directory of the WTP installation is used.

To create the XSL files, the following tools were used:

- IBM WebSphere Studio 3.5 to manage the resources and to bundle the application
- XML spy 3.5: to create the XSL StyleSheets and to test the results in combination with the defined XML structures

15.3.8 Implementing the XML solution

To implement the XML solution we created a new package: `trade_client.xml.simplifiedhtml` and a new subclass of the `TradeAppServlet`. Furthermore, the defined classes and interfaces from the class design section (see 15.3.4, “Defining the class concept” on page 297) were added. After these modifications, the project panel of the Visual Age for Java (VAJ) Workbench provides the following view:

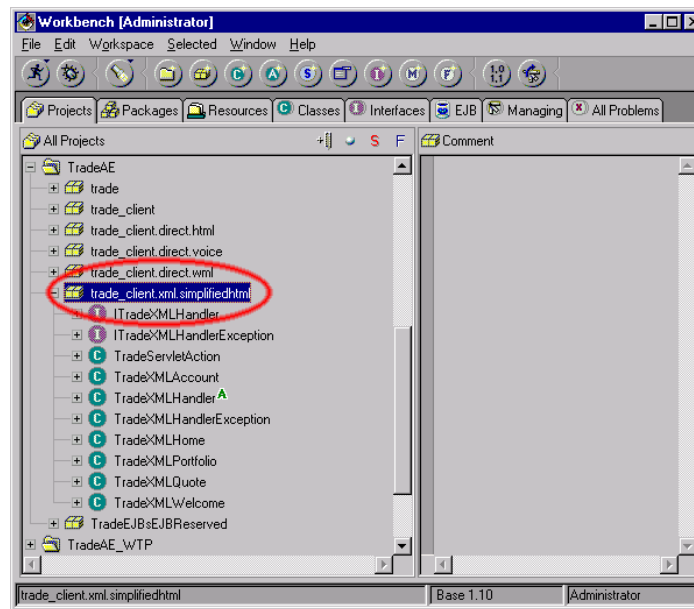


Figure 15-33 Created classes for the trade XML application in VisualAge for Java

In order to produce the XML content, we had to change the main methods of the inherited `TradeAppServlet` class. The following example presents a code example of the new `doHome()` method:

Example 15-8 Code example of the `doHome()` method

```
try
{
    //getting the balance from the TradeAction servlet
    balance = tAction.getBalance(userID);

    //step 1: initializing new TradeHome class
    TradeXMLHome tXMLHome = new TradeXMLHome();

    //step 2: initializing new hashable and setting dynamical data
    Hashtable hTable = new Hashtable();
```



```

hTable.put("userID", userID);
hTable.put("balance", String.valueOf(balance));
hTable.put("results", results);

//step 3: transferring the hashtable to the tXMLHome object
tXMLHome.setInput(hTable);

//step 4: calling the execute method from TradeXMLHome instance to generate
the XML stream
tXMLHome.execute();

//step 5: reading the XML stream from the TradeXMLHome instance
String xmlData = tXMLHome.getXML();

if (xmlData != null) {

    //printing out xmlData to the client
    requestDispatch(ctx, req, resp, xmlData);

} else {
    ...
}

//step 6: additional error handling
} catch ...

```

More precisely, the code is going through the following steps:

1. After retrieving the dynamic data from the existing functions, the application initializes a new object (tXMLHome) from the TradeXMLHome class. As a subclass, this class contains all specific fields and methods from the abstract parent TradeHome.
2. A new hashtable (hTable) is created. The dynamic data (the user ID, the balance and the result) are added to the hashtable.
3. The hashtable is transferred via the setInput() method to the tXMLHome object.
4. The execute() method of the tXMLHome is invoked. This method creates the XML stream by using the dynamic data from the hashtable.
5. The application retrieves the XML data from the tXMLHome object via the getXML() method and puts the stream to the output queue (done via the requestDispatch method).
6. Additional error handling is performed.

The last step listed (error handling) is performed by the `TradeXMLHandlerException` class. For every error, the application creates a new instance of the class and passes the error code to the constructor. The constructor then builds the XML structure for the error message. Finally, the `requestDispatcher` method prints out the new XML data stream to the HTTP response.

In addition to the discussed modifications, the two new methods `doWelcome()` and `doQuoteError()` were added to the servlet. The `doWelcome()` method must be implemented for the logout function of the application. Logout returns with an empty XML structure using the `TradeXMIWelcome` class. This XML data stream is needed to redirect the user to the welcome page.

To cover another feature of the original Trade2 application, the `doQuoteError()` method was added to the servlet. It does the error handling in the case of invalid input. In the original version, a new quote object with the error message was simply passed to the JSP file. For the XML version, the `doQuoteError()` wraps the error message in the original quote XML structure and puts it to the HTTP response.

Note: In the Trade2 application, you will find the variable `result` that is used to print out status messages to the user. This variable is not supported and used in the XML version of the Trade2 application.

Furthermore, a DTD reference is added to each error message created by the `TradeXMLHandler` class. This reference is needed to invoke the appropriate `StyleSheet` in the WTP environment (see 15.3.9, “Registering the StyleSheets in WTP” on page 304 for more details).

15.3.9 Registering the StyleSheets in WTP

To set up and register the `StyleSheets` in WTP, the following three steps have to be taken.

Publishing XSL files

The first step is to publish the files from the WebSphere Studio project to a defined target in the server directory. For more information, please refer to “Publishing” on page 169.

Register XSL files

In the second step, we first create a new folder (Trade/simplifiedhtml) in the WTP Administrator console. The StyleSheets themselves are registered by selecting **Register -> XML StyleSheets...** from the Administrator Console menu. A visual user interface guides you through the different steps to collect the necessary information, such as the location of the file or naming and directory issues. Please make sure to store the XSL documents in the newly created folder.

A detailed description of registering XSL documents with WTP can be found in Section 18.8.3, “Registering the StyleSheets” on page 374.

Setting rules and triggers

The third step is the most important one. Here we define the rules stating when a StyleSheet should be applied to a XML file. WTP offers two possibilities:

1. Defining rules based on the request headers

The rules can be set up under the Advanced option of the XSL registration page. Here you can define rules based on the header fields of the request. For legal expressions, the Accept field or the User Agent field of the header, for example, can be included. For the expressions, it is possible to use logical operators (such as “&” for logical AND). Furthermore, you can combine the rules with the defined profile preferences in the WTP environment. Please note that by using the rules, the selection of the StyleSheet is controlled from the client side.

2. Input DTD

This option can be used to control the selection of a StyleSheet from the application side. For this purpose, you must define a DTD reference in the output XML file as well as in the input field **Optional input DTD name or location** on the registration page of the XSL file in WTP. To match the StyleSheet, both entries must be equal. Please note that you must use absolute paths.

In the Trade2 XML example, we use both options to apply the defined StyleSheets to the application. In the first case, we are using the URL fields inside the header of the HTTP request to control the selection of the StyleSheet. An example is the portfolio page shown in Figure 15-34 on page 306. We define that the StyleSheet should be used whenever the URL includes one of the following strings:

TradeAppServlet?action=portfolio or

TradeAppServlet?portfolio

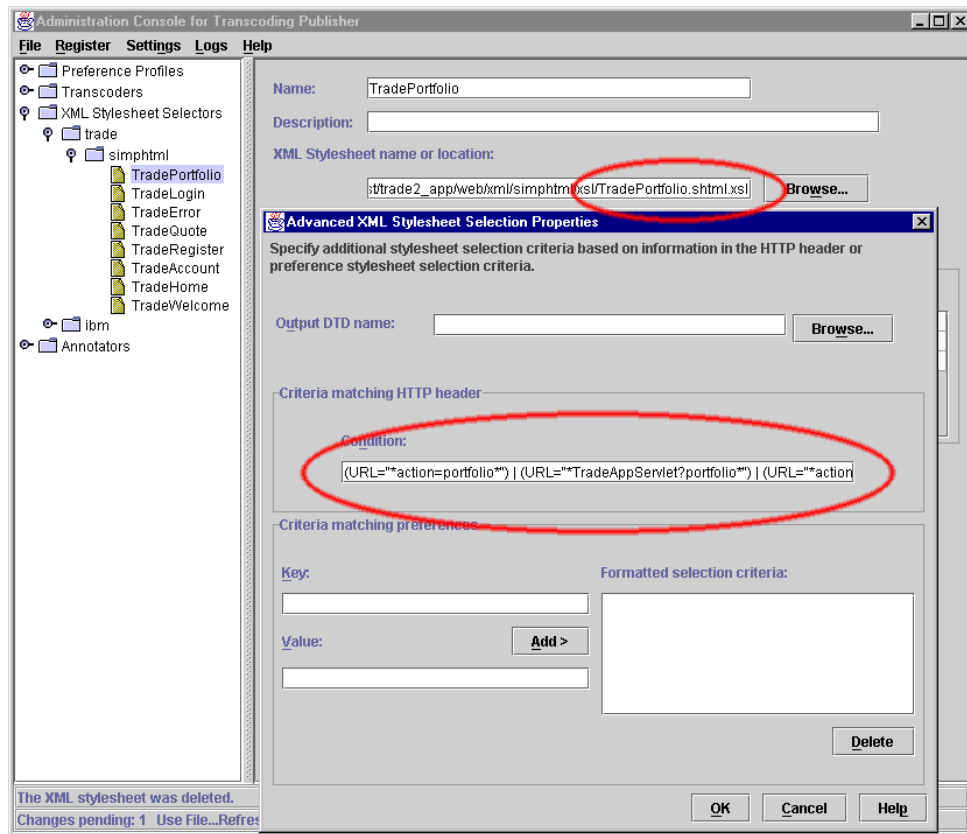


Figure 15-34 Settings for the TradePortfolio XSL in WTP

The second option is used for the error page. Here, we have to control the selection of the XSL file from the application because we do not know beforehand when an error will occur. To do this, we specify the DTD in the XML file as well as on the settings page of the StyleSheet in WTP. Figure 15-35 on page 307 shows the setup for WTP.

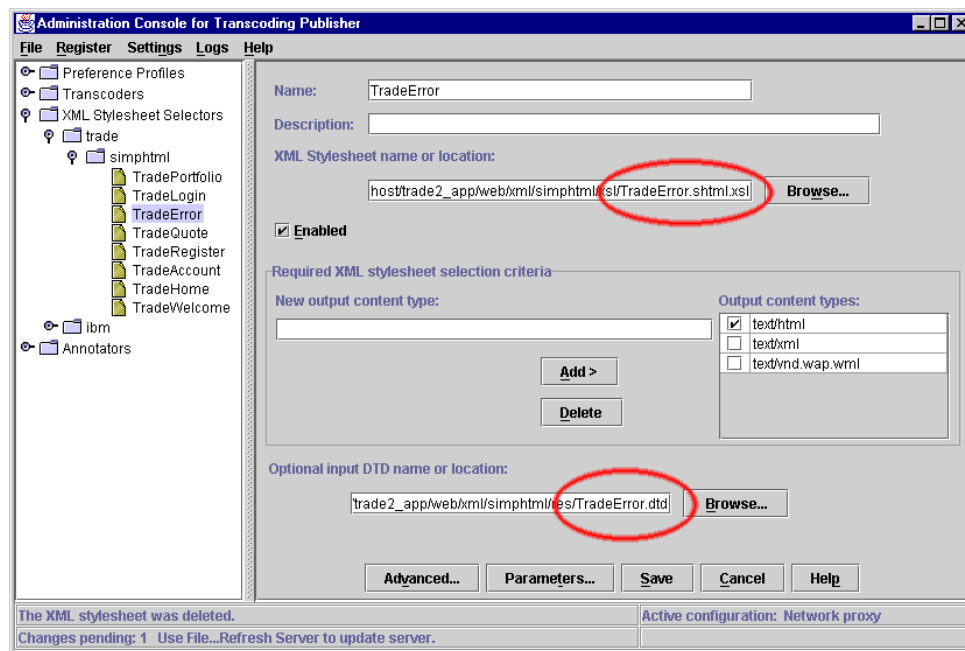


Figure 15-35 Settings for the TradeError XSL in WTP

The advantage of DTD matching is that the developer does not have to build unique URLs for the request.

The problem is that URL matching is only possible for different URLs or the same URLs with different URL parameters, for example:

TradeAppServlet?action=login and TradeAppServlet?action=buy. Using forms with the same URL, as in our case, where the application design is based on the command application design pattern, is a problem for URL matching conditions. For example, in the Trade2 application, we had to add unique identifiers to the request (for example *URL=TradeAppServlet?portfolio*) in order to build the rules.

Furthermore, it is not possible to trigger StyleSheets from the server side. If the developer chooses the DTD matching, then a DTD for every displayed page has to be defined and a reference must be placed in the generated XML structure.

15.3.10 Testing the application

The best way to test the XML application is to use a desktop Web browser. In our sample, we set up the application to recognize the two browsers Opera and Lynx.

We recommend that you use WTP in the proxy mode; you will have to modify your proxy preferences in the browser.

Note: The best way to run WTP for testing the application in combination with the WebSphere Test Environment is the proxy mode. Unfortunately, in the reverse proxy mode the WTP cannot reference images correctly. The WTE port (typically 8080) gets lost during the transformation, and the client cannot find the images because of the missing port number.

For more information about testing the application, please refer to Section 19.5, “Testing the application” on page 395.



Voice application

In this chapter, we explore the possibilities of delivering a voice-driven Web application.

The first section will give a clear picture of WebSphere Voice Server and its different editions, and the following sections will describe these scenarios:

- ▶ Direct VoiceXML coding to generate pages for the VoiceXML browser with the content.
- ▶ Using XML and XSL to generate the VoiceXML content.
- ▶ HTML-to-VoiceXML transcoder.
- ▶ How to leverage the existing HTML content and use the transcoder to merge with new VoiceXML content.

16.1 WebSphere Voice Server

First, a short introduction to WebSphere Voice Server is called for. After having a clear understanding of how the voice server works in the background, it will be easier to understand how the application works.

There are two different editions of WebSphere Voice Server:

- ▶ IBM WebSphere Voice Server with ViaVoice Technology
- ▶ IBM WebSphere Voice Server for DirectTalk

The development toolkit is IBM WebSphere Voice Server SDK.

IBM WebSphere Voice Server with ViaVoice Technology

This edition requires a VoIP gateway, which can switch between the PSTN and the IP network.

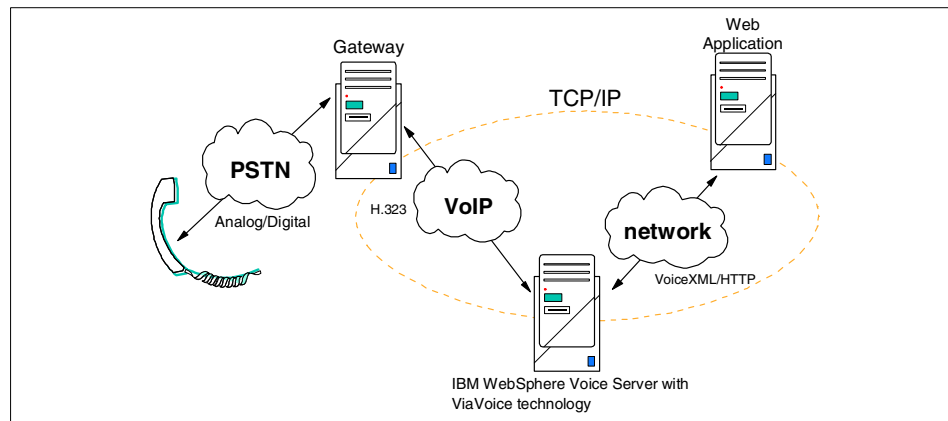


Figure 16-1 IBM WebSphere Voice Server with ViaVoice Technology 1.5

IBM WebSphere Voice Server for DirectTalk

With this edition, DirectTalk provides the connection to the PSTN and the voice server is a DirectTalk application.

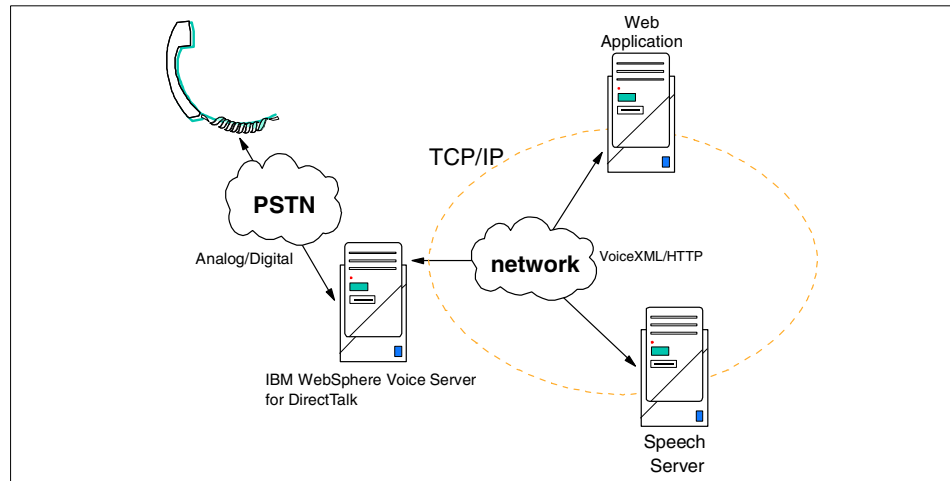


Figure 16-2 IBM WebSphere Voice Server for DirectTalk

IBM WebSphere Voice Server SDK

Additionally, there is the WebSphere Voice Server SDK, which is a development toolkit. However, the same VoiceXML browser runs under the SDK as the Voice server.

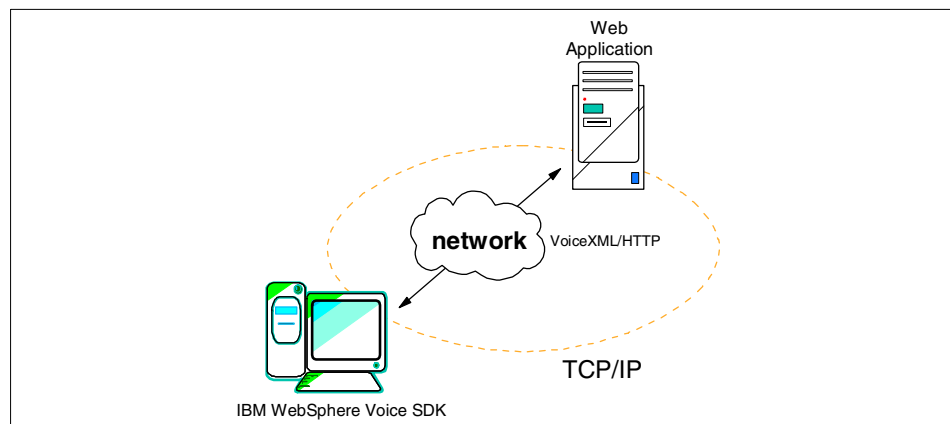


Figure 16-3 IBM WebSphere Voice Server SDK

To understand how the Voice Server works as a VoiceXML browser, we can compare it to a visual browser. Both a VoiceXML browser and a visual browser operate on documents based on a markup language.

The difference is that the browser for visual applications resides on the client side, while the browser for voice applications is on the server side. It is like having a regular Web browser running on a server and getting the picture on a display on the client side; that is what a VoiceXML browser is doing. The voice server is nothing more than a browser which runs on the server side, close to the Web application server. Figure 16-4 is an illustration of this concept.

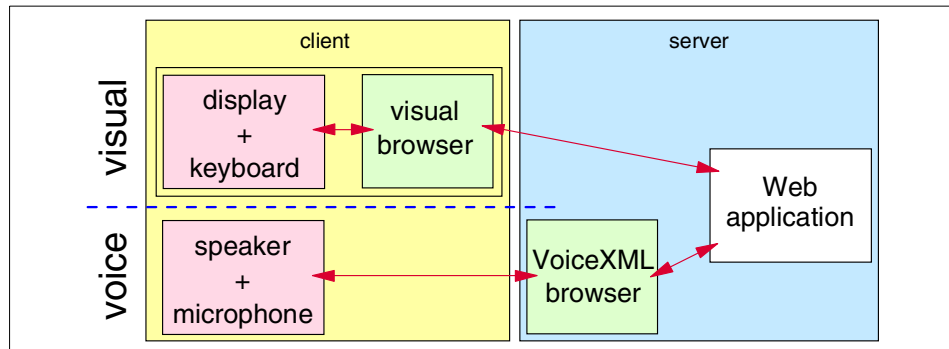


Figure 16-4 Understanding the VoiceXML browser

When you are developing VoiceXML applications, it is not necessary to set up a voice server with the whole environment as just described. The same VoiceXML browser runs within the voice server and the SDK. Using the SDK is the same as running a server on your machine and using the machine's peripherals (speaker and microphone) instead of having a whole runtime environment for voice.

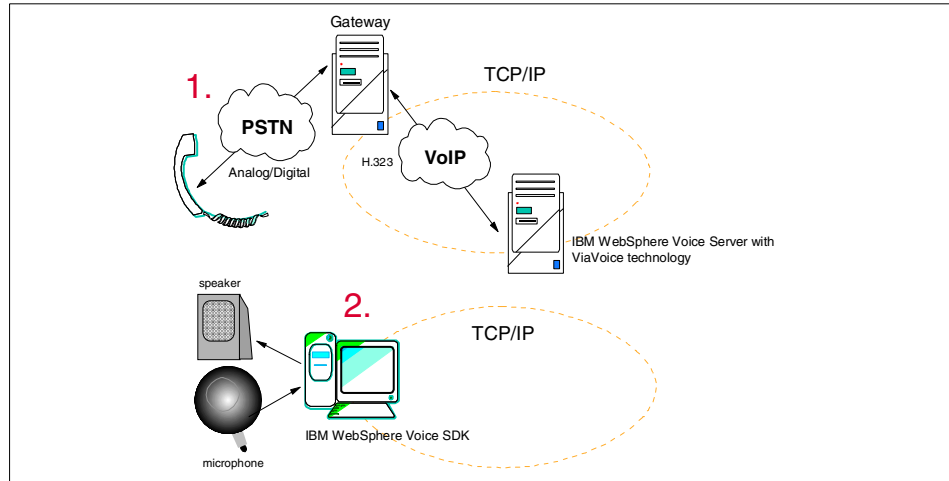


Figure 16-5 Two different voice runtime environments

From the voice application developer's standpoint, the two setups are the same; they provide the same result. However, the second one using the SDK is very useful during development, and much less expensive.

16.2 Direct approach

The direct approach means that the voice Web application is developed using the native language for voice applications, that is, VoiceXML. The final content can be produced using either static VoiceXML pages or JavaServer Pages to generate the VoiceXML documents.

Most of the voice-specific parts of the application appear at the front end, so the major part of the application follows common practices for the design and development of a Web application.

16.2.1 The voice site

Figure 16-7 on page 315 introduces the Trade2 voice application site map. You can follow the interaction between the Web application elements on the diagram.

There are differences between the voice application and the visual application implementations.

The most important difference is that there is no registration and no account modification in the voice implementation. These pages require more flexibility than a voice application can provide. In order to access these functions, the user has to transfer to a customer representative (agent), or go directly to the Web site.

Figure 16-6 describes the icons on the site map.

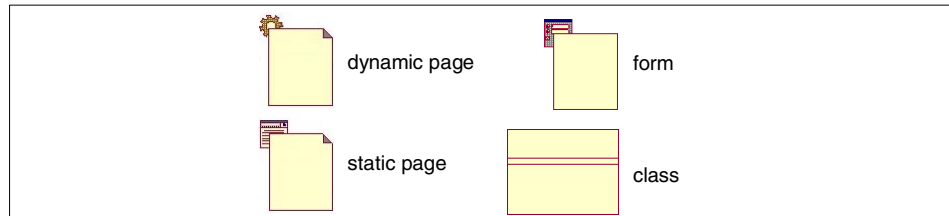


Figure 16-6 Legend for the site map

The dynamic pages are JavaServer Pages, the forms are VoiceXML forms, static pages are static documents written in VoiceXML, and classes represent Java classes.

You can see that one dynamic page can include several forms; this is shown by the chain of forms beside the dynamic page.

The arrows are links, and point from the source page to the destination. The destination can be either another dynamic or static page, or a servlet.

There is a special element on the site map, the EXIT. The link to the EXIT label means that the application ends at that point and the communication with the client closes. This feature is different from that of any other visual Web application.

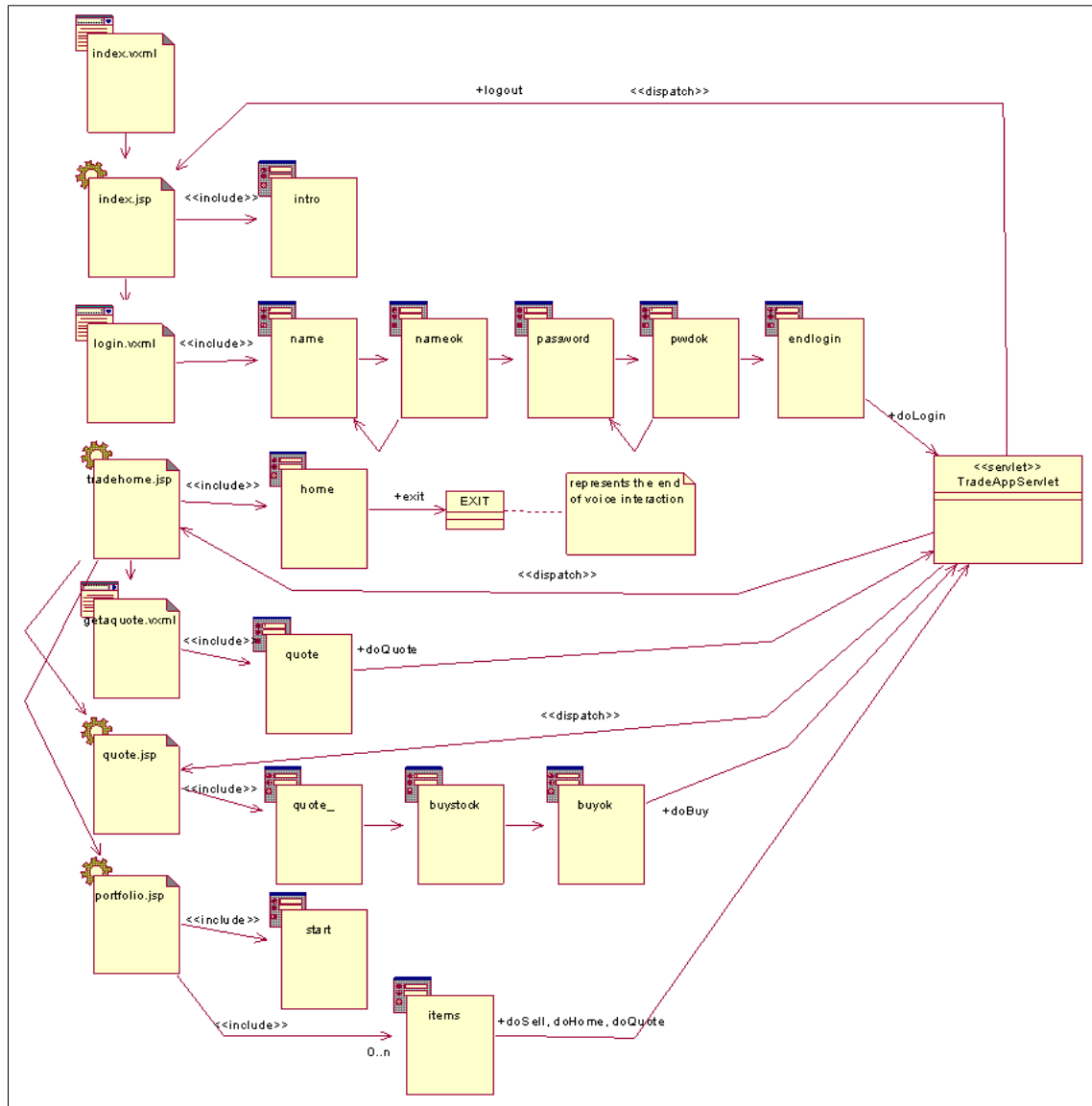


Figure 16-7 Trade2 Voice Application site map

About the sample application

The voice Web application is based on the original Web application for desktop browsers. It has the same functions, except for the two which are not implemented, and it has the same structure for navigation.

There are two interesting pages within the sample voice application:

- ▶ Login page
- ▶ Portfolio page

Login page

The login page has to collect only two pieces of information from the user: the user ID and the password, which need to be accurate; however, we need to avoid annoying the user with a long login process.

The login process can be broken down into two parts which use the same technique to collect the correct information.

The application asks for the value, and waits for only numbers. The user can either say the numbers one by one, or use the keypad to type the numbers in. Depending on the input, the user response can be terminated with a long break or the pound (#) key on the keypad.

After retrieving the result, the application repeats the value and asks if it is correct. If so, it will go on; if not, the process starts again with the same input field.

When both values are available (the user ID and the password), the application will check the user name and password for validity. In order to make the challenge, it sends a request to the TradeAppServlet with the necessary parameters:

```
<submit next="/trade/servlet/TradeAppServlet" namelist="action uid  
passwd"/>
```

Where action has the value of the login, and uid and password are the values collected from the user for the user ID and password. The <submit> tag ensures that the information is sent through using the POST method. If using the <goto> tag with the same parameterization, the GET method will be used.

Portfolio page

The portfolio is a dynamic VoiceXML page, where some of the information is dumped out dynamically and the whole list of the portfolio is generated.

Each element in the portfolio is a form which starts by playing the information about the element to the caller, then provides the following options: home, sell, quote, restart. If the caller does not choose an option within a time out period specified on the form, the next form is called. This process repeats until the last form, which calls the first, and so on until the caller breaks out or hangs up.

The **home** command will take the user to the welcome page, the **restart** command will restart reading the portfolio from the beginning. The **sell** and **quote** commands operate on the latest element read by the application. Whenever the user gives the **sell** or **quote** command, the TradeAppServlet is called with the actual parameters. In the case of the **sell** command, the following link makes the call:

```
<goto next="/trade/servlet/TradeAppServlet" namelist="action index symbol"/>
```

In our approach, the whole list of the portfolio is given in one dynamic VoiceXML document generated by a JavaServer Page.

We chose the list because it requires less change to the original page and application flow; however, it is only one solution and not the best approach for this function.

It is better to break up the whole document into a header, which starts the session, and a dynamic page which calls itself, instead of repeating it in one document, refreshing the content dynamically each time.

Figure 16-8 shows two different approaches for the portfolio process. The first one is implemented in our solution, where the whole process is implemented by one dynamically generated document. The second solution is more elegant; in it, the items are generated dynamically with a recursive loop.

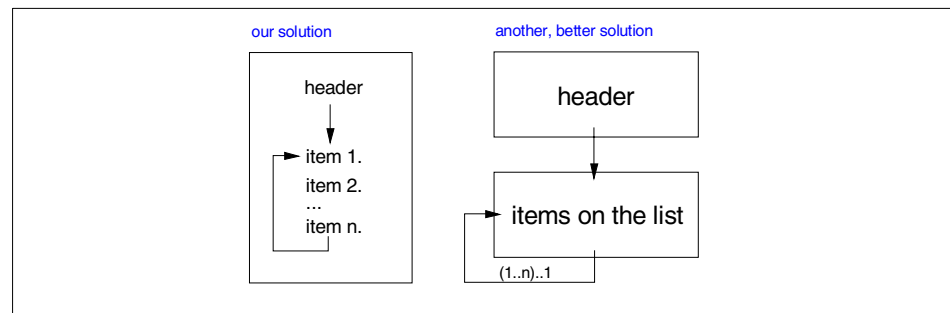


Figure 16-8 Solutions for the portfolio process

16.2.2 Application design

The following paragraphs will go through some useful guidelines in order to help design a voice application. We will start with general design techniques and tips, then turn to a VoiceXML-specific (voice Web application) discussion. At the end of this section, we will discuss some application-specific issues related to the Trade2 application, which is provided with the book.

Starting the design

1. Identify the main activities to be performed by the application with the users.
2. Together with users, design the dialog for each activity, concentrating on everything proceeding smoothly. You must take into consideration all the possible responses by the caller, remembering that they may have either keys, speech, or both with which to respond.
3. Decide whether you are going to allow both keys and speech throughout the call, or keys for some parts and speech for others.
4. As your dialog design progresses, make sure you document each of the branches of the application.
5. Identify all the things that could go wrong: for example, the caller could hang up in the middle of the call, the connection with a host computer could fail.
6. Again, with users, refine the dialog, taking into consideration all the things you have identified that could go wrong. The business departments will want to be assured that the caller is going to hear the right thing, whatever happens. They do not want to lose business because of a poorly designed voice application. If the voice application itself cannot continue, ideally, the caller should be transferred to a human agent.

User participation in dialog design

Before recording anything, act out the dialogs, using the “Wizard-of-Oz” technique, with one person reading out the application’s words out of sight (behind a screen or curtain), and another person acting the part of a caller.

When you are fairly happy with the dialog design, record the voice segments using the telephone, or synthesize them using text-to-speech. Start with company employees as the callers, and, when you are happy with the dialog, try it on potential customers or clients. If the service is aimed at company employees, start with experienced employees, but do not forget to try it out on newcomers. Only when you are satisfied with the dialog should you invest in professional recording.

Designing the dialog

Customer satisfaction with your voice response service will depend on a number of factors, including the voice you choose, the information you provide, and the ease with which callers can get the information they want. The sound and feel of the service is every bit as important as the look and feel of a visual computer application. With careful design, the caller’s experience will be pleasurable; with lack of attention to dialog design, the caller will get frustrated and hang up. At best, this will result in your staff handling just as many calls as they ever did; at worst, it may mean loss of business.

When designing the dialog, you need to be aware of some of the limitations of the medium. Compared with people, automated systems are inflexible, demanding, and intolerant of deviations from the expected conversation. They are best at handling very routine calls. Almost without exception, all voice response services should offer help in some way, for example by offering the choice of transferring to an operator or calling another number to speak to an operator. You should give human factors and usability a high priority when designing voice response services.

Design considerations

Compared with today's multiwindow screen-based computer applications, voice response applications have a number of potential limitations. With careful design, you can overcome these limitations and make your application a joy to use:

- ▶ Auditory output is sequential, and hard to keep in short-term memory: you have to take great care in wording the prompts.
- ▶ Auditory output can be slow: unfortunately, someone is paying for the call, and wants the whole transaction to take as little time as possible.
- ▶ The very ubiquity and availability of telephones means that callers do not have access to manuals and other supporting materials: the whole point is for the voice response service to be usable from anywhere, so the application itself must be completely self-documenting.
- ▶ Text-to-speech, while relatively acceptable for delivering frequently updated information, is not really suitable for a dialog with the caller, which should sound as natural as possible.
- ▶ Bear in mind that many callers may be using phones with an integral keypad and handset (for example: mobile phones): they will take longer to press the keys than callers using traditional phones with a separate keypad.
- ▶ Compared with the 100 or so keys on a computer keyboard, the standard telephone has twelve keys. The standard keypad is fine for numeric input, making simple choices, and answering yes-or-no questions, but what about alphabetic input? In some countries, there are no alphabetic characters on the keys at all; even in countries where telephones still have the alphabetic characters on the number keys, people do not use them enough to be familiar with their positions.
- ▶ Many telephones cannot transmit DTMF tones reliably or at all. In these cases, the only reasonable input device is voice.

Choosing a dialog style

No single dialog style is best for all applications, for all tasks within an application, or for all callers:

- ▶ Callers may be familiar or unfamiliar with the mechanics of driving the application.
- ▶ Callers may be familiar or unfamiliar with other voice response services.
- ▶ Callers may use the service frequently (and therefore acquire expertise with the mechanics, or the content, over time) or infrequently (never acquiring expertise).
- ▶ The content of the dialog may be familiar and predictable (for example, days of the week) or relatively unfamiliar and unpredictable (for example, toppings available for pizza).
- ▶ It may be appropriate to provide documentation or training sessions, for example if callers are employees or students, but it is unreasonable to base your design on the assumption that documentation will be read or training sessions attended.

There are three basic styles of voice response dialog, suited to different types of tasks:

1. Menu: suited to selecting one of a small number of options
2. List: suited to choosing multiple items, perhaps from a large number
3. Form: suited to providing input such as addresses and telephone numbers.

Within these basic styles, there are numerous variations. It is hard to provide rules for good dialog design, but you need to classify your application and your callers in these terms before considering the following questions:

1. Composite or separate actions? Composite actions may be simpler for the caller, but separate actions provide more flexibility.
2. Keys or speech recognition? The application uses speech to communicate with the caller, but should the caller be using speech or keys?
3. A mixture of key and speech input? You can mix key input and speech input in a single application (though it is not recommended that you allow both at the same time). For example, entering a Personal Identification Number (PIN) is easier using keys, whereas recording a street address is easier with speech.
4. Command-driven or prompted? You need to make a decision about whether to let callers interrupt the prompts or not. In general, you should let them interrupt. Once they learn the choices at each point, they can key ahead (or speak ahead), without waiting for the prompt to finish. There may, however, be some prompts that you want to force play to the end; there may even be whole applications in which you want all prompts to be force played. The type of

dialog that allows key-ahead or speak-ahead is sometimes known as a command-driven dialog, but you still need to play the prompts in a voice application, because of the absence of documentation.

5. A single selection key or option-specific selection keys? With a single selection key, you play each choice to the caller and give them a few seconds to press the chosen key (for example, 1); if they do not press a key, you play the next choice, and so on. The caller always presses 1 to select the current option. There is less for callers to learn if, at any point, they have only two choices: to select the current option or to proceed to the next; it can be useful for long lists of options (for example film titles). On the other hand, this kind of design tends to restrict the caller's ability to key ahead and bypass menus.

With option-specific selection keys, you allocate a different key to each option. This has the advantage of allowing callers to key ahead, but limits the options available at any one time. It can also be difficult to ensure consistency of key-allocation throughout the application (for example, on one menu, 3 may correspond to "Delete message" while on another menu, it corresponds to something else entirely). Again, infrequent use may point the designer toward the single selection key and frequent use toward the option-specific selection keys.

6. Passive or active advance through menus? Should each option drop through to the next option, or should the caller have to press a key to proceed? This becomes an issue if you provide a single selection key. Again, passive advance is probably most suitable for callers who use the application infrequently but, to avoid the problem of callers selecting an option just after the menu has moved on to the next option, you need to include a few seconds of silence between each option. Passive advance might seem easiest for the caller, but only at first. Later, callers may become frustrated at having to wait through entire menus.

Providing a Next key would seem to prevent this, but the drop-through behavior often results in callers never learning about the Next key, unless the menus mention it.

7. Long or short prompts? The more information you provide, the more likely the caller is to learn how to drive the application. However, long prompts slow down callers, particularly experienced ones. A choice of novice and expert prompts can help. The difference between them is not necessarily verbose versus terse: you could actually leave some information out of the expert prompts altogether.

Always provide essential information before information that is merely helpful. Provide Next and Previous keys, so that callers can skip information they do not want to hear.

Good things to do in voice applications

- ▶ Subdivide the recording of voice output to allow random access to it. Callers can skip and scan.
- ▶ Make sure that callers know they are using a machine; do not try to make them think they are talking to a human agent.
- ▶ Ask the caller whether they are using a phone that generates tones, by asking them to press a specific key.
- ▶ Allow the caller to interrupt prompts (key ahead) wherever possible (that is, do not force play the prompts).
- ▶ Refer to the # key as “pound” in the U.S. or “hash” in the U.K.
- ▶ Refer to the * key as “star”.
- ▶ Use the star key for the control menu.
- ▶ Refer to the 0 key as “zero”.
- ▶ Use the zero key to provide access to the operator.
- ▶ Always phrase your prompts so that the goal precedes the means of achieving it (for example, “to contact the operator, press 0”); if you mention the key first, the caller may forget which one it was before realizing that it was the one they wanted.
- ▶ Use questions and commands rather than statements, to encourage callers to take immediate action.
- ▶ Ask closed questions such as “Do you want a large, medium, or small pizza?” rather than “What size pizza would you like?”.
- ▶ Add pauses to encourage the caller to take immediate action.
- ▶ Always try to allocate similar functions to the same keys.
- ▶ Use directional metaphors where appropriate (use the relative position of the keys on the keypad to indicate some logical direction associated with the command, for example, 7 for back, 8 for pause, and 9 for forward).
- ▶ Try to limit menus to about four options, with a fifth option for more choices if necessary; this adheres to the rule of the “magic number 7 plus or minus 2” items that people can hold in short-term memory; also, it will not take too long to play.
- ▶ The order in which you play the options depends on the application: you might choose to start with the most frequently used option, or the least-frequently used, or play the options in ascending numerical sequence.
- ▶ Use simple, explicit language, for example: “Press” for single key entry (no delimiter) and “Enter” for multiple keys (which need a delimiter).
- ▶ Give feedback: repeat long data entries back to the caller.

- ▶ If the caller does not make a selection, repeat the menu.
- ▶ If the caller makes an error, explain the valid choices.
- ▶ Employ a professional to record the final version of the prompts.
- ▶ Do not mix prompts recorded in more than one voice, unless you do this consistently to convey information (for example, voice A for menus and voice B for data retrieved for playback to the caller). For this reason, you should get the system voice segments recorded by the same professional as your other voice segments.
- ▶ Correct wording of your prompts is particularly critical when you are using speech recognition. Your prompts must tell the caller what to say, when to say it, and how to say it.
- ▶ Ask for the same information no more than twice, to avoid irritating the caller and jeopardizing the business transaction. The second prompt should apologize to the caller, accept responsibility for the communication error, and repeat the request. You might change the wording in the second request to give additional clues or information.

Voice Web application design

Voice Web applications can be designed like normal Web applications, which is certainly a great advantage. The client (VoiceXML browser) is working with pages, static or dynamically generated on the server side. However, the construction of the pages and the interaction flow are different from those in any other visual Web application.

You can see on the site map that one page encapsulates one or more form.

Note: a form represents one interaction between the user and the application, where the application retrieves the answer for one input field. For example:

(Application) - What is your user ID?

(User) - 56463.

These forms are just like the forms in the HTML document. There could be more than one form in one document. The difference is that after a form is finished, it must call another form or a whole new document; if not, the interaction finishes, the application is ended and the call is terminated.

It is worth noting that unlike a visual Web application, a voice Web application only runs in a single thread. With a visual browser, it is possible to open many windows simultaneously, and switch between them. However, it is only possible to listen to one voice application over the phone.

Another important characteristic of the voice Web application is that the application ends after the conversation. It has a deterministic end, occurring when the browser exits. This exit can be programmatic (exit by request) or by event (the telephone connection is cut off).

It is up to the developer to decide whether the forms are separated into different files, or merged into one. It has to be decided at the beginning whether or not the forms are reusable in a different set of conversations.

A good practice to design the pages is using forms, then merging them according to the application flow and code reuse.

A good example of reusing forms instead of repeating them is the portfolio function. To find out more about this function, read “Portfolio page” on page 316, where we discuss two approaches, one using a list of the same forms, the other reusing the same form.

How to design VoiceXML pages

The first challenge is to design the site in as linear a fashion as possible; the site must provide only the main functions, and no other links. Wherever links to other pages are provided, they should be tightly related to the actual position of the connection. Every possible link on the site should not be listed on each page; this is confusing, and the user will easily become lost. Users can navigate backwards to previous pages or functions, but they should not jump to a different page.

To understand this better, try to imagine a tree, where at each node you can make a decision as to which branch you want to go to next, and if you have failed to find the right path, you can only go back to the previous node you encountered. This system will let you browse through the whole tree without the navigation being confusing.

Of course, designing the tree and putting the links in the right order and on the right node are the keys to designing the voice application. The related links should be listed or available on one page only.

Another important piece of advice for designing VoiceXML pages is that it is always worth repeating the user's answer for confirmation. This is standard practice in traditional voice applications, so the same precept needs to be applied in VoiceXML applications.

Login process

In our example, we wanted to keep the login session simple; that is why we decided to use numbers for the user ID and password. The problem is the limitations of an audio interface and the capabilities of the speech recognizer.

With today's speech recognizer engines, it is very difficult to implement the login process. *Login process* here means asking the user for a user ID and for a password. The best approach is to use numbers, either by using the keys on the phone or by saying the numbers.

Web applications usually do not use numbers for the user ID and password; instead, they use letters, numbers and signs mixed together. Using only numbers with a voice application usually requires some changes in the original Web application. The application has to be able to do the challenge using either the original user ID and password or the numerically encoded user ID and password. There are several possibilities to solve this issue.

- ▶ Use of numbers only for users' ID and password.
- ▶ Storing the user ID and password encoded in the database.
- ▶ Implementing the encoding process during the database query; a stored procedure will encode the user ID and the password during the query.

There are other solutions for logging in a user, of course. In this book, we have a dedicated user, who uses numbers for a user ID and password. This user is the only one able to log in via phone. The user ID is 56463, which is the encoded string for *johnd* (j=5, o=6, h=4, n=6, d=3); the password is the same: 56463.

Back-end modifications

Implementing a voice solution into an existing application requires changes. The question arises as to whether the back-end needs to be modified or not.

Unless the original application design supports major modifications or new implementations, the question should not even arise. In most cases, the back-end must be modified to support the functions required by the new voice application.

This does not mean that the back-end has to be rewritten. Voice applications require some special back-end functionalities, which are different from other Web applications. These functions are usually related to the dynamic grammar. The application has to quickly generate not only the document, which is common for any Web application, but the grammar for voice recognition.

Plan ahead during the application design, even if voice implementation is not certain or planned for later. For more information about voice application design, refer to Chapter 8, "Solution design" on page 103.

16.2.3 Application development

The WebSphere Voice Server SDK provides a VoiceXML browser. The browser has two runtime modes:

- ▶ Audio: plays the audio files, synthesizes and recognizes speech.
- ▶ Text: writes out the text, replaces the audio files with the description, receives the input via the keyboard or the simulated DTMF keypad.

Best practices for development

Using the Audio mode can be very slow, because of having to listen for the directions, then use voice for input. The problem is that even with the “barge-in” feature enabled, it takes time to interrupt the speech; also, false recognition could waste time.

In this case, the best choice is to use the Text mode at the beginning of development to create the functional site.

Once the functional site is available, the Audio mode becomes important because the text can be surprisingly different from the speech. Typically, additional information is required and some parts can sound strange.

The voice recognition depends very much on the dictionary, which the recognition engine works from. Dictionaries may vary based on the language. During application development, developers should further investigate the dictionary. For example, some abbreviations exist in the dictionaries and some of them do not. If “IBM” exists in the dictionary, then the speech synthesizer knows how to pronounce this term, letter-by-letter (“I-B-M”); if it does not exist, then the speech synthesizer will try to pronounce the term as one word, which will sound strange.

The speech synthesizer engine has the ability to change the voice of the speaker. This feature can help to organize the content more efficiently. Different speakers can be applied to different situations. This has the same effect as using bold, italic or underlined text with the visual browser. For example, information and error messages can have different speakers.

Coding

The VoiceXML browser uses a cache like other browsers, and during development this could be a problem, as with any other browser. The developer wants to see the most recent version, not the cached one.

To disable the cache, change the following in the `vsaudio.bat` or in the `vstext.bat`. Within the **Execute:** section, after the Java interpreter, but before the classpath settings, add the following parameter: `-Dvxml.cache=false`.

The other solution is to delete the cache folder under the <WV SDK install dir>/bin/cache directory.

The problem with disabling the cache is that the VoiceXML browser does not have a source viewer, so the cache is the only place where the final content can be found. This problem arises when the application is using dynamically generated pages, so the final content is very different from the source.

When JavaServer Pages are used to generate the content, the final documents are only available for debugging from the cache. The final document can have a variable amount of options or lists, dynamic grammar, and conditional elements, which makes it very difficult to trace the operation following only the original source code.

The other possibility to speed up the testing is to enable the “barge-in” feature in the forms, so the developer does not have to listen the whole text.

16.2.4 Trade2 voice application

This implementation of the Trade2 voice application exercises direct coding using the VoiceXML markup language. The voice application is also based on the base application introduced in Chapter 14, “Base sample overview” on page 241; however, it is a non-visual application.

The application flow is the same as with any visual implementation. The difference is the registration and the account update pages, at which points the user has to go to the Web site or speak to a customer representative, because these pages cannot be implemented using this technology, or the solution is too sophisticated.

Basically, during development the original HTML document has been encoded in VoiceXML format following the specification and the best practices from the product documentation.

16.3 Content transcoding (HTML source)

Within the transcoding option, we make a further distinction between transcoding full HTML pages and mining content from existing HTML pages to insert into directly coded VoiceXML pages.

16.3.1 Full HTML-to-VoiceXML transcoding

If you are a company who has already invested significant resources in building an e-commerce site or a customer self-service site, you can imagine that WebSphere Transcoding Publisher's (WTP) HTML-to-VoiceXML transcoder can be employed to voice-enable your entire site with little effort on your part. As we have already described, visual interfaces differ greatly from audio interfaces. If you intend to voice-enable a Web site, you will invariably need to either apply various WTP functions to simplify and mutate your content to make it easier to transcode into VoiceXML, rework the HTML and the Web site in general to make it easier to traverse using voice, or both.

WTP provides functions that can simplify an HTML page and make it more usable as a voice application, such as the ability to split the document into sections and to add annotations (see 10.2.3, "Development tools in WTP" on page 176), but these only work on a page-by-page basis (see Figure 16-9).

The HTML-to-VoiceXML transcoder and the rest of WTP can only help improve the page, not the entire site as a whole. Your Web site's navigation methods may be too complex for a voice medium, or based largely on navigation bars and graphic hot spots which cannot be associated well with the core content, or the navigation could get lost altogether. If you face this problem, it can only be solved if you have control over the original HTML source.

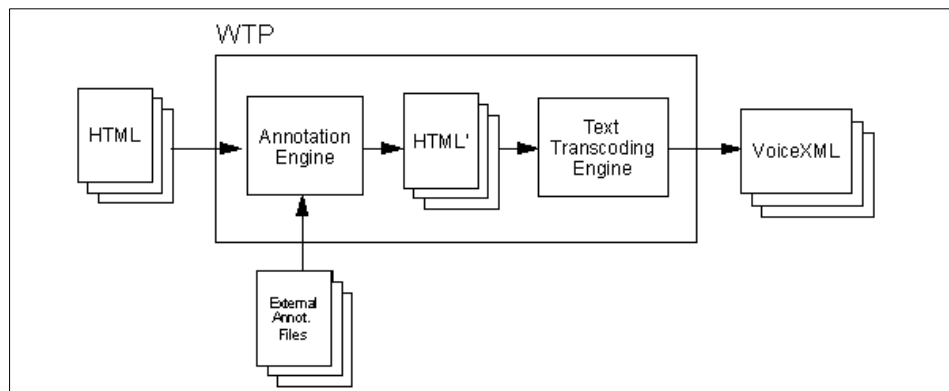


Figure 16-9 WTP uses annotator for VoiceXML transcoding

Figure 16-9 shows that WTP uses annotation to remove unwanted data from an HTML page before it is transcoded into VoiceXML.

Thus, when transcoding HTML-to-VoiceXML using WTP, extra care needs to be taken in reducing the complexity of the data.

Even so, how (and how much) data is converted to VoiceXML is under the control of the HTML-to-VoiceXML transcoder and of WTP in general?

Some of the navigational difficulties related to voice applications are accounted for by the HTML-to-VoiceXML transcoder as it keys off certain HTML tags to determine navigation points. It is up to the designer to ensure that the result is usable and the visual elements from which the transcoder builds the navigation are present in the source document. More information on this topic can be found in Section 8.4.2, “Voice applied to the decision tree” on page 119.

Unfortunately, most HTML form input is not as bounded. Because all visual Web browsers have some type of keypad or keyboard attached to them, almost anything can be presented as input. When transcoding HTML to VoiceXML, one of the most daunting tasks is to ensure that possible input is well bounded and defined so that accurate grammars can be produced and error tendency reduced. For example, if an HTML page requires that the user type in a US state name, you can use WTP to turn that text input into a select list of 50 entries, for which can have an associated grammar of 50 words and phrases. Better yet, if you have access to the HTML source, rework the original HTML page so as to have a select list from the start which improves both the VoiceXML produced by WTP and the usability of the original HTML page.

16.3.2 Annotators

Potentially the most useful application of WTP's HTML to VoiceXML transcoder is the ability to convert HTML content for use within a VoiceXML application. The HTML content is usually mined from existing pages hosting dynamic data such as weather updates, stock quotes, employee addresses and phone numbers, catalogue information, etc. Once you fully understand the differences between visual and audio interfaces, you might prefer to re-author the voice interface entirely in VoiceXML.

Be careful, though. Even putting aside the complexity of deriving a usable vocal interface from a visual one through transcoding, maintenance could be cumbersome. In fact, visual interfaces tend to change quite often, requiring modifications in how they are transcoded as a result. In any case, extensive usability testing should be conducted on the solution to determine the best mix of transcoded content versus static or native VoiceXML content and how to best blend that mix.

In this context, a VoiceXML application is maintained separately from its visual counterpart and is tailored to the unique requirements of a listening audience.

The challenge is now to incorporate dynamic data within the VoiceXML application. It would be duplicate work to try to get the dynamic data at the source, especially if that data already exists on a Web page somewhere. It would certainly be easier for the VoiceXML application designer to reuse existing HTML data rather than reproduce it from other sources. WTP serves this purpose. It not only provides the HTML-to-VoiceXML transcoder to render the HTML source in the appropriate markup, but also provides the necessary tools to extract information from a page with superfluous data (see 16.3.3, “Setting up the HTML-to-VoiceXML transcoder” on page 330). The HTML to VoiceXML transcoder can be used to derive either entire VoiceXML pages for use within the voice application, or portions of content which are to be placed within a VoiceXML page.

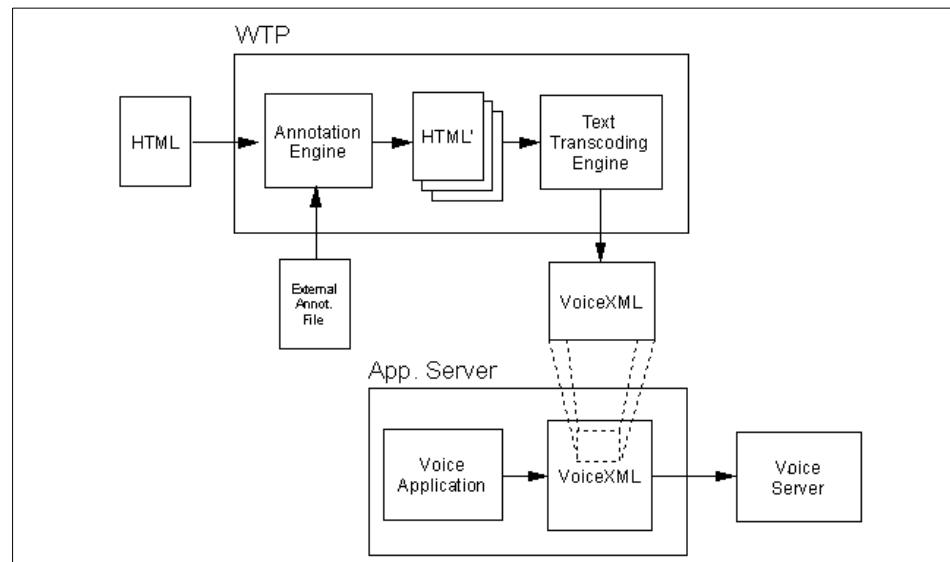


Figure 16-10 WTP uses annotation for VoiceXML transcoding

Figure 16-10 shows that WTP applies annotation to a page to derive the necessary data before it is included within the VoiceXML application.

16.3.3 Setting up the HTML-to-VoiceXML transcoder

This section describes how to register the HTML-to-VoiceXML transcoder, how to create a new device profile for the VoiceXML browser and how to register the new profile into WebSphere Transcoding Publisher.

Registering the transcoder

1. Start **IBM Transcoding Publisher->Administration Console**.
2. Select **Register->Transcoder** from the menu.
3. On the welcome screen, click **Next**.
4. Browse for the transcoder under the next screen. Select **ibm_VoiceXMLTranscoder.jar**; the file with the appropriate path appears in the text-box. Click **Next**.
5. In the following window, you have the opportunity to change the name of the transcoder and the description; leave them as is and click **Next**.
6. You can enable the transcoder immediately after setting up. Select **Yes**, then click **Finish**.
7. Select **File->Refresh Server** from the menu to refresh your configuration.

Now the HTML-to-VoiceXML transcoder is set up.

The next step is registering the VoiceXML device. You can either create your own profile for the voice browser or just create and register the file below (see Example 16-1 on page 333).

Creating a new device profile

First you must have the software called *WebSphere Transcoding Publisher Profile Builder*. You can download it from:

<http://www-4.ibm.com/software/webservers/transcoding/> ; select the **Profile Builder** on the navigation bar, then download the software.

1. After downloading, run **ProfileBuilder.bat**; this will execute a Java application, so you have to have JDK 1.2.2 or JRE 1.2.2 installed on your machine.
2. In the Welcome window, select **Create a new profile**, then click **Next**.
3. In the Choose Profile Type window, select **Device**; then click **Next**.
4. The next window allows you to identify the profile. Give a name to your profile and a short description, for example:
Name: Voice Browser
Description: Voice browser device profile for HTML-to-VoiceXML transcoder
Then click **Next**.
5. In the Specify a User-Agent Value window, you can set up the condition, which triggers the transcoder to transcode the content.
User-Agent value: (User_Agent=*Voice*)

Important: Make sure you have the parenthesis characters, and do not forget that the expression is case sensitive.

Click **Next**.

6. The Select Preferences window is a little bit sophisticated. You have to select the following preferences and add them to the configuration by clicking the **Add** button.

Add the following preferences to the Edit panel:

- Desired content types
- Convert tables to lists within lists

Add the following preferences to the View panel:

- Convert images into image links
- Use the first table row as labels for each list item

Click **Next**.

7. Set the preference values in the next window.

Desired content types: text/vxml, text/x-vxml, application/vxml, application/x-vxml

- Convert tables to lists within lists: disabled
- Convert images into image links: enabled
- Use the first table row as labels for each list item: enabled

Click **Next**.

8. In the Specify Additional Preferences window, add the following key and value, typing in:

- DeviceType
- VoiceXML Device

then click **Add**. Click **Next**.

9. Save the profile under the directory of your choice; you might want to consult it later, so save it to a safe directory, where it can be easily found; you can save it wherever you want, except in the <WTP install directory>/etc/... directories, because WTP will copy it under this directory. Give a name, for example: voicebrowser.prop; then click **Next**.

10. Click **Finish**.

Here is the example voicebrowser.prop properties file for the voice device.

Example 16-1 voicebrowser.prop

```
#Preference Profile for Voice browser
DescriptiveName=Voice browser
Description=Voice browser device profile for HTML-to-VoiceXML transcoder
desiredContentTypes=[text/vxml, text/x-vxml, application/xml,
application/x-vxml]
propagateFirstTableRowData=true
textLinksPreferredToImages=true
deviceType=VoiceXML Device
deviceRule=(User_Agent\=*Voice*)
convertTablesToUnorderedLists=false
ConfigurableProperties=desiredContentTypes{text}
convertTablesToUnorderedLists{bool}
NonConfigurableProperties=textLinksPreferredToImages{bool}
propagateFirstTableRowData{bool}
```

Registering the device profile

After creating the profile, you have to register it in order to handle the new device.

Open the Administration console for WTP, and follow the following steps:

1. Select **Register->Preference Profile...** from the menu.
2. Click **Next** in the Welcome window.
3. Browse for the preference profile you have; in our example, we select voicebrowser.prop; then click **Next**.
4. In the following window, you can change the name and the description of the device profile. Leave them as they are, then click **Next**.
5. Select **Yes** in the next window; your profile will be enabled immediately.
6. Click **Finish**.
7. Select **File->Refresh Server** from the menu to refresh your configuration.

16.3.4 Result of the HTML-to-VoiceXML transcoder

Before you decide to select the HTML-to-VoiceXML transcoder, you should check the results produced by the transcoder. It is always worth it to take the time and get some experience by playing with the transcoder and feeding it different documents from different sources. The following key elements are observed:

- ▶ Which part of the document remains and which part has been thrown out.
- ▶ How the options are organized.

- ▶ How the input fields are handled.
- ▶ Which are the non-handled elements.

The transcoder tries to figure out the main content and identify the links through the documents. Finally, the result is given with the following options:

- ▶ Main content: the collected content from the page.
- ▶ Links: the many links on the page.
- ▶ Exit: users always have this option.

Following this structure, the user can navigate through the whole Web site, as with a desktop browser.

Of course there are complex, sophisticated pages, which the transcoder cannot handle correctly; this is the reason why you have to get experience with the transcoder. In case you want to take advantage of the HTML-to-VoiceXML transcoder, you should design your HTML pages in a format such that the transcoder can handle it if need be.

The other option is to use the annotators together with the HTML-to-VoiceXML transcoder, then apply the annotator to the document to get the best result for the transcoder.

16.3.5 Trade2 application

We have tried the HTML-to-VoiceXML transcoder on the Trade2 sample application designed for mobile applications. The result was not worth mentioning.

Most pages are not transcoder-friendly for the HTML-to-VoiceXML transcoder. In several situations, the limitation was that the pages were designed for visual applications.

We have found that it is not worth transcoding the whole site as a voice portal because of the limitations of the transcoder.

The best approach is to enable certain services to be accessible using voice, for example the following services: quote, portfolio, buy, sell. A query page is required for these services to collect the necessary information using voice recognition. These pages should be developed directly for these queries. Then the result pages can come from the original application using the HTML-to-VoiceXML transcoder together with annotators.

16.4 Universal transcoding (XML source)

In this section, we discuss another universal transcoding scenario and how to develop a VoiceXML application using XML as a generic datasource. We outline possible problems solutions as well as practical guidelines.

The Trade2 sample voice Web application is also developed using the XML and XSL technologies. Figure 16-11 shows the application flow of how the VoiceXML document is produced from a single XML document.

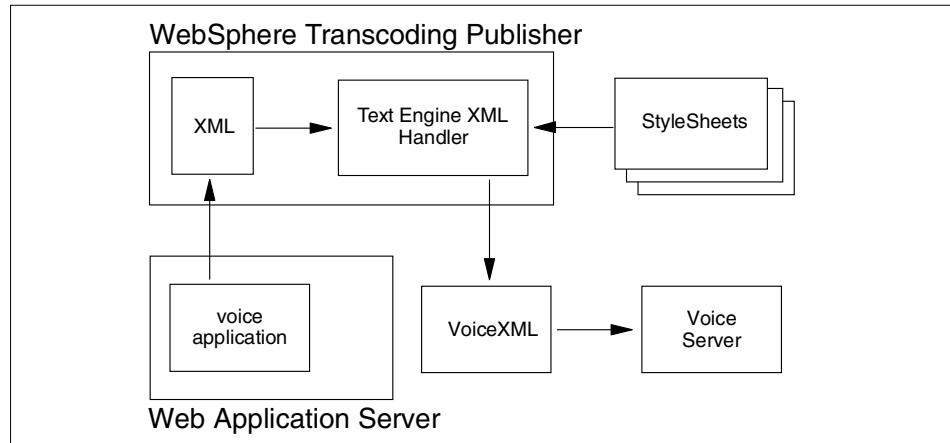


Figure 16-11 Converting XML to VoiceXML using StyleSheets

16.4.1 Design decisions

The first decision is to develop the new solution based on the implemented Trade2 XML application for simplified HTML. Many components can be reused from the existing solution, which results in quick development.

As seen in the direct approach, the workflow is kept untouched (see Section 16.2.1, “The voice site” on page 313). The functionality is reduced: the Register and the Account pages will be not available for the voice application, just as in the direct approach. However, the remaining workflow has the advantage that the class concept as well as the defined XML structures can be reused.

The third decision is to use the static login VoiceXML file from the direct approach to perform the login for the user. This should show you how you can reduce the number of StyleSheets by using other already implemented components.

We will use static grammar file for faster development.

16.4.2 Designing the class concept

As a consequence of the untouched workflow, the same class concept and the same XML data of the simplified HTML solution can be used for the voice application. In this case, the implemented TradeAppServlet as well as the additional classes (TradeXMLHandler, TradeXMLHandlerException) and the interfaces (ITradeXMLHandler, ITradeXMLHandlerException) do not have to be modified. Also, no changes are necessary to the first and third tier.

Note: All relevant classes and interfaces are stored in the `trade_client.xml.simplifiedhtml` package. For a better understanding you might rename the package with an appropriate name (for example `trade_client.xml.solution`).

For more information about the used class concept, please refer to 15.3.4, “Defining the class concept” on page 297 and 15.3.8, “Implementing the XML solution” on page 302.

Figure 16-12 depicts the similarities between the two different results, where one is produced for a Web browser using HTML documents, and the other is produced for a VoiceXML browser using VoiceXML documents. The diagram makes it obvious that the presentation logic is absolutely the same in both cases; only the StyleSheets, producing the final content, are different.

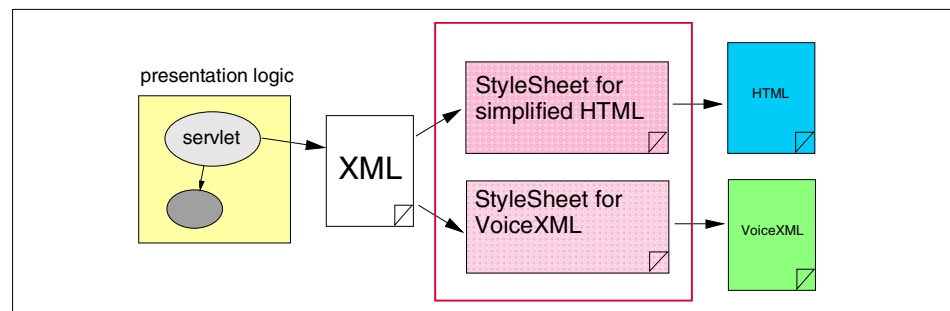


Figure 16-12 XML approach for simplified HTML and VoiceXML

16.4.3 Developing XML files

As mentioned in the discussion on design decisions above, we can use the same XML structure as in the simplified XML solutions. The only difference is that we do not need the TradeRegister and TradeAccount functions structure. Both functions are disabled for the voice application. Furthermore, no TradeLogin structure is needed because the static login.vxml is used (see 16.4.1, “Design decisions” on page 335). Table 16-1 gives an overview of the remaining structures.

Table 16-1 Overview of defined XML structures

XML structure name	Function	Includes dynamic data	Stored static	Created by application
TradeWelcome	Used for the index page of the application	NO	YES	YES
TradeHome	Used for the home page for a logged on user (includes user ID and account balance)	YES	NO	YES
TradePortfolio	Used to show the portfolio information for a specific user	YES	NO	YES
TradeQuote	Used to show quote information for a stock	YES	NO	YES
TradeError	Used for the error page (includes error code)	YES	NO	YES

Stored static in this context means that the XML file will be stored on the directory structure of the Web server and will be used to display the appropriate page. In this case, the XML files do not contain any dynamic data. Please note that the TradeWelcome XML file is the only XML structure that is provided static and dynamic. The reason for this is that the method to perform the logout of the application should not reference the static XML file. Therefore, a new method, doWelcome(), was added to the class (see 15.3.8, “Implementing the XML solution” on page 302).

In order to store the XML files for the voice application separately in the Studio Workbench, we created the following three new folders:

- ▶ xml/vxml/xml: to store the XML
- ▶ xml/vxml/xsl: to store the XSL
- ▶ xml/dtd: to store DTD files

The existing XML files can be copied from the /xml/simphtml/xml folder into this folder. The static XML structures should be set publishable in order to deploy them to the Web server later on.

16.4.4 Developing the XSL files

In the next step, create the XSL files to display the data on the user interface based on the direct VoiceXML pages. For this purpose, implement the XSL files as shown in the following table. In order to separate the VoiceXML StyleSheets from the simplified HTML ones, add the ending .vxml to the files.

Table 16-2 Overview of defined and implemented StyleSheets

Stylesheet name	Function	Processes dynamic data	Corresponding XML structure
TradeWelcome.vxml	Used to display the index page	No	TradeWelcome.xml
TradeHome.vxml	Used to display the home page (after login)	User ID and balance	TradeHome.xml
TradePortfolio.vxml	Used to display the portfolio information	User portfolio data	TradePortfolio.xml
TradeQuote.vxml	Used to display the quote information for a stock	Quote data	TradeQuote.xml
TradeError.vxml	Used to display the error page	Error code	TradeError.xml

In comparison with the simplified HTML solution, no StyleSheets for the registration, the login and the account information are needed. Also, no template XSL file is used in the voice application.

To create and handle the XSL files, the following tools were used:

- ▶ IBM WebSphere Studio 3.5 to manage the resources and to bundle the application.
- ▶ XML Spy 3.5 to create the XSL StyleSheets and to test the results in combination with the defined XML structures attaching the XML and XSL files together.

16.4.5 Register the StyleSheets in the WTP

To set up and register the StyleSheets in WTP, the following four steps have to be followed.

Adding voice profile

To access the application from the WTP, we have to set up and register a new voice profile first. Please refer to , “Registering the device profile” on page 333 for detailed guidelines.

Publishing XSL files

The second step is to publish the files from the WebSphere Studio project to a defined target in the server directory. For more information, please refer to 18.3.2, “Publishing a WebSphere Studio project” on page 357.

Register XSL files

In the third step, we first create a new folder, Trade/voicexml, in the WTP Administrator console. The StyleSheets themselves are registered by selecting **Register->XML Stylesheets...** from the Administrator Console menu. A visual use interface guides you through the different steps to collect the necessary information such as the location of the file or naming and directory issues. Please make sure to store the XSL documents in the newly created folder.

For more information on how to register the XSL files under WTP, refer to 18.8.3, “Registering the StyleSheets” on page 374.

Setting up rules and triggers

The fourth step is the most important one. Here we have to define the rules as to when a StyleSheet should be applied to the XML document, and which it should be. WTP offers two possibilities: *Defining rules based on the request headers* and *Input DTD* (see “Setting rules and triggers” on page 305). As in the simplified HTML version, we use the HTTP request rules for the application, except for the error pages (error pages are referenced via the DTD). All necessary rules for the voice applications are listed in “Configuring the StyleSheets for VoiceXML” on page 379.

16.4.6 Testing the application

The best way to test the XML application is to use the IBM Voice Server SDK. We recommend that you use the WTP in proxy mode.

For information on how to set up the WebSphere Voice Server SDK, please refer to 18.7, “WebSphere Voice Server SDK configuration” on page 371.

Note: The XML voice application uses static VoiceXML and grammar files of the direct approach. Please make sure to publish those files to your application server, too.

For more information about testing the application, refer to 19.5, “Testing the application” on page 395.

16.4.7 Further directions

The XML development shown was a first approach to enhancing the Trade2 application to support XML and XSL. The following list gives an overview of individual ideas that can improve the completeness of the application:

- ▶ Improve error handling and logging of the application to allow the user to trace exceptions.
- ▶ Enable the application so that it supports multiple languages. Multiple language support can be achieved by adding new StyleSheets to the application.
- ▶ Use only dynamic XML data streams. This means generating the static pages in the Java classes as well. Otherwise, the static pages require more effort to maintain.

16.5 Hybrid coding

The idea behind hybrid coding is to mix the new or recently developed application, using the direct approach or the XML transformation technologies, with the HTML-to-VoiceXML transcoder, which operates on existing pages.

Figure 16-13 on page 341 shows the concept of hybrid coding.

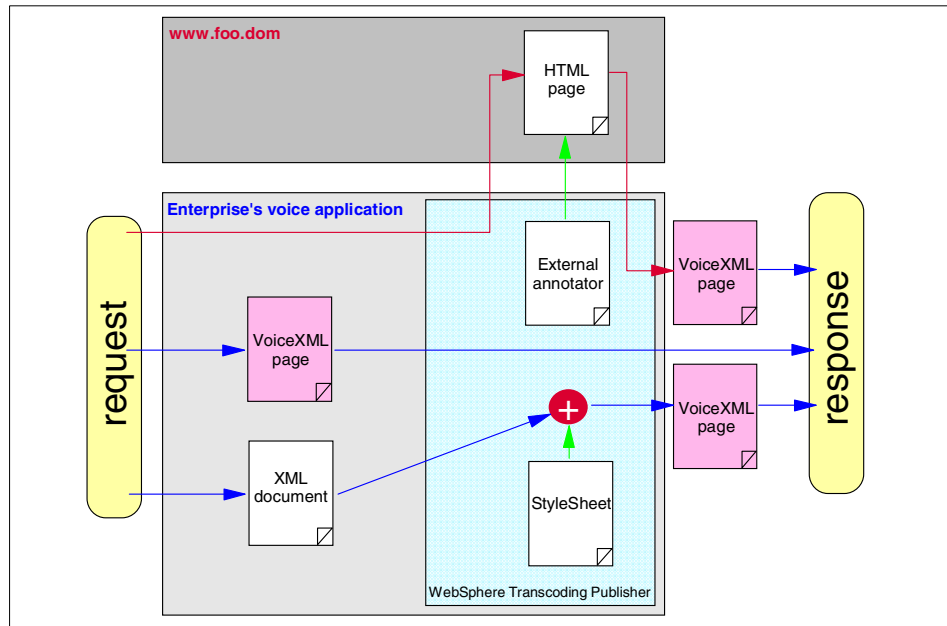


Figure 16-13 Hybrid coding concept

For example, in our sample, the quote result page can be replaced by a transcoded page from the Yahoo financial site, where the quotes are listed. Here is the theory:

1. Instead of calling the TradeAppServlet from the Trade2 application if a quote is requested (action="quote"), the application submits to a different server application:

`http://finance.yahoo.com/q?s=ibm`

where the URL responds with a page including the actual stock quote.

2. The request goes through the transcoder, and it realizes that there is a URL matching condition:

`URL=http://finance.yahoo.com/q?*`

The matching criteria triggers the transcoder to retrieve the page from the Yahoo site and apply the appropriate external annotation.

3. The external annotation will remove all the irrelevant information and keep only the stock symbol and the value.

The annotator also inserts the necessary tags for the following options: buy, refresh, home.

4. The WebSphere Transcoding Publisher sends back the result to the VoiceXML browser.

The previous example assumes that the form action for the quote is rewritten, WTP has been set up with the external annotator and the URL condition, and the external annotator exists under WTP.

Note: The application only works together with the registered stock symbols. If you want to buy a non-existing symbol, the application will report an error.

16.6 Where to find more information

- ▶ To find out more about VoiceXML refer to the documentation from the VoiceXML group: VoiceXML specification.
- ▶ The other very useful resource is the Programming Guide, comes with the WebSphere Voice SDK.



Application for both interactive mobile device and voice

This chapter will discuss those cases where both interactive mobile devices and voice are involved in the solution at the same time.

The very first section describes two different situations where both the interactive mobile device and voice are involved in one solution.

The following topics are then covered:

- ▶ Universal transcoding: using XML and XSL technologies together with WebSphere Transcoding Publisher (WTP)
- ▶ Content transcoding: using the transcoding capabilities of WTP
- ▶ Multimodal applications: a new wave of mobile applications

17.1 Introduction

There are two different approaches to using both communication modes in one application.

One approach involves the connections being independent and using separate sessions. This approach only makes sense when the different modes are using the same technology. For example, let us say that two different users are accessing the Web application; one is using a wireless PC client, and the other is using voice over a phone. In both cases, the application is using the same technology: XML and StyleSheets to produce the content.

The other approach involves the two modes following each other in one session and the application switching between them. It does not matter what kind of technology lies behind the modes.

17.2 Universal transcoding

Previously, Section 15.3, “Universal transcoding (XML source)” on page 293 and Section 16.4, “Universal transcoding (XML source)” on page 335 discussed how to use the XML, XSL and XSLT technologies for your mobile Web application.

In both cases, a scenario was developed using universal transcoding techniques. For interactive mobile devices, the simplified HTML pages are generated using a set of XSLs based on a set of XMLs. For voice, the XSLs for VoiceXML are based on the same set of XMLs as in the simplified HTML.

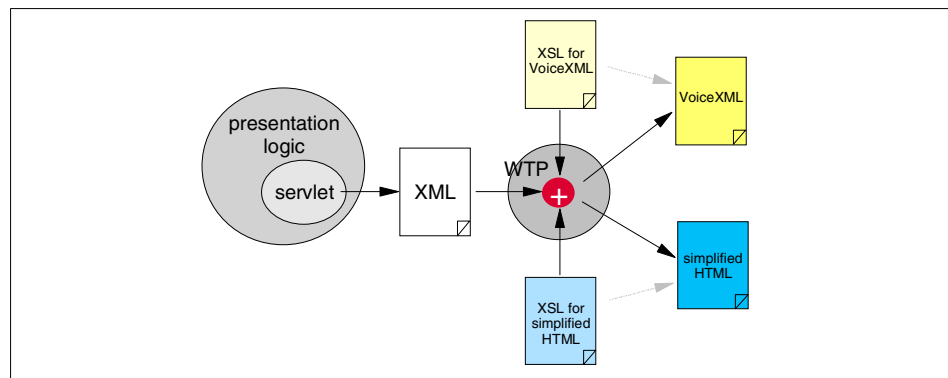


Figure 17-1 Generating simplified HTML and VoiceXML

Figure 17-1 shows two different scenarios for the solution developed for this book. The XML generated in the presentation logic is common for the two scenarios. The XML document is used for both generating the VoiceXML and the simplified HTML pages. The XML document and the corresponding XSL documents are merged by WTP, and the result is the final document in accordance with the request.

This technique has the advantage of separating the data from the content on the presentation level. Since the data is the common element for each form of presentation, only the content has to be customized for each type of client device.

Separating the content from the data makes the implementation easier when visual and non-visual (voice) applications are implemented into the same solution.

17.3 Content transcoding

Transcoding is the general means of translating one type of markup language into another type of markup language. Transcoders transform arbitrary content into a different form that can be displayed by the client device. This involves almost a complete transformation of the content.

The presented transcoding techniques use HTML as a base source and transform it into another markup language. Why not use XML as the base markup language? Because XML fits perfectly into this scenario, for several reasons:

Since XML contains no presentation, a StyleSheet must be applied every time a transformation is needed. If the data is presented in different markup languages, a StyleSheet must be created not only for each markup language, but also for every XML document. Hence, if there are M pieces of XML documents and N types of content, then there are $M \times N$ different StyleSheets. Maintaining these StyleSheets will require an enormous administration effort.

Transcoding has the advantage of implementing the transformation between the different type of contents with less effort. WTP is not concerned about the final content; it will do the transcoding based on the original HTML document.

The end result of the transcoder is questionable in some cases, but developers can improve it by making only small modifications to the original code.

There is also the possibility of using the XML, XSL and XSLT technologies to produce different sets of HTML pages, then using the transcoder's capabilities for transforming the HTML document into the required format. For more information about this option, read Section 15.3.1, "Converting XML to different markup languages" on page 294.

Summary

Since content transcoding is based on HTML documents, and from the WTP standpoint it does not matter what the final result is until WTP has the right transcoder for that transformation, content transcoding is an ideal solution for mobile Web applications where both interactive mobile devices and voice are involved. It is even better if the solution can take advantage of the hybrid solution and use the transcoding technology together with the XML, XSL and XSLT technologies.

17.4 Multimodal applications

This section describe how different user interfaces can interact in multimodal applications. Section 17.4.1, "Multimodal applications in WebSphere" on page 348 describes how this is applied in WebSphere Everyplace Access.

Multimodal applications are applications that use one or more input devices (for example keyboard, microphone, phone, etc.), recognize/understand the input, and communicate new information back to the user using one or several different output devices (for example screen loudspeaker, phone, etc.). This definition is derived from *Handbook of multimodal and spoken dialogue systems*, by D.Gibbon, I. Mertins, and R. Moore.

To see an illustration of this general concept of multimodal applications, see Figure 17-2 on page 347.

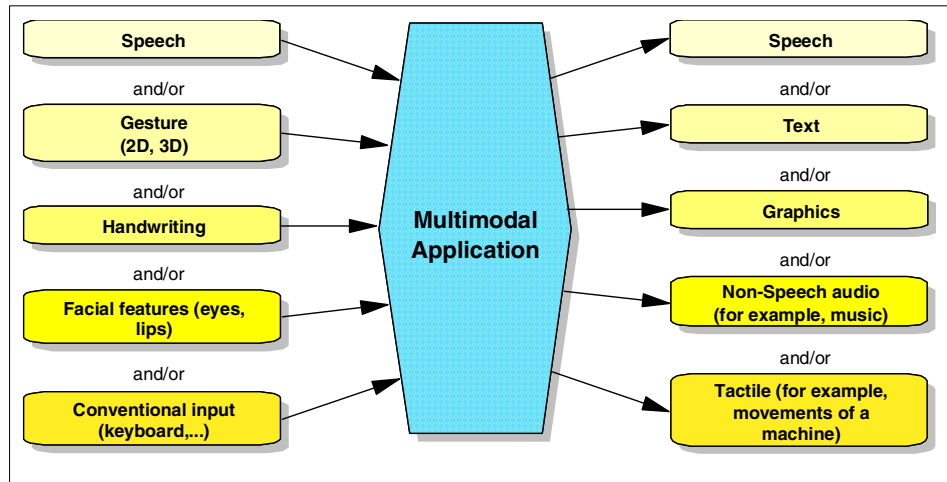


Figure 17-2 Scope of multimodal applications in general

On the input side, we can have:

- ▶ Speech, using the different types of speech recognition (see Chapter 3, “Overview of speech technology” on page 33).
- ▶ Gestures:
 - 2D gestures that, for example, point to an area on the map, on a screen.
 - 3D gestures with, for example, position trackers or cameras.
- ▶ Handwriting recognition, on palmpads, for example.
- ▶ Observation of facial features such as lip movements. This can help improve the speech recognition in some cases.
- ▶ Conventional input, like keyboards, mouse, etc.

On the output side, the possibilities are:

- ▶ Speech, using the text-to-speech technology.
- ▶ Text and graphics on a screen.
- ▶ Non-speech audio, for example sound clips, music, etc.
- ▶ Tactile, meaning the different kinds of movements a machine can produce (movements, vibrations, etc.).

Speech can further be combined in talking heads and talking agents (talking agents are images of whole bodies on the screen talking to the user).

In the following sections, we will concentrate on applications that use speech input and/or output on the one hand and screen/keyboard on the other.

It is obvious, also from the other chapters in this redbook, that one will have to choose carefully between those possibilities. The following guidelines appear to have been validated by empirical evidence:

Table 17-1 Voice or visual output?

Use voice preferably within multimodal applications when	Use visual output preferably when
The eyes of the user are focused on something else (for example while driving a car).	The eyes are visible.
Mobility is needed.	Mobility is not needed.
The graphical interface is already overloaded (for example in a plane).	Information can be added to the graphical interface.
No spatial manipulations are required.	Spatial manipulations are required.
Privacy is not an issue.	Privacy is an issue.

These are, of course, general guidelines to be used with sound judgement.

In the following paragraphs, we will concentrate on how current and future WebSphere products can be used for building multimodal applications.

17.4.1 Multimodal applications in WebSphere

In the context of WebSphere, the relevant multimodal applications are enterprise Web applications rather than document generation or command systems.

These multimodal Web applications use the same technologies and follow the same patterns as the other applications described in this redbook, but they have to address the additional challenge of combining the different user interfaces, rather than just allowing the different devices to access the same application. For example, one will have to decide how to combine the interactions described in VoiceXML with those in WML. Some of the choices to make are:

- ▶ Will the markup languages used be tightly coupled or loosely coupled?
 - Tight coupling in this context means using one markup language that contains, for example, parts written in VoiceXML and other parts written in WML. The synchronization is described in the markup document itself, as shown in Example 17-1.

Example 17-1 Code sample in the case of tight coupling

```
<HTML>
  <HEAD><TITLE>SBU Multi-modal Browser Demo</TITLE></HEAD>

  <vxml version="1.0">
    <form name="Welcome">
      <block> Welcome to the SBU Multi-modal Browser Demo.
        <goto next="#1"/>
      </block>
    </form>
    <menu id="1">
      <prompt> Say hello to authenticate user. </prompt>
      <grammar type="text/jskf"> hello </grammar>
      <choice next="main.vxml" conext="main.html"> hello </choice>
    </menu>
  </vxml>

  <BODY>
    <H2 ALIGN = "CENTER"> Welcome to the<BR/>
    SBU Multi-modal Browser Demo </H2>
    <P ALIGN="CENTER"> Release 1.00 <BR/><BR/>
    <A HREF="main.html" cohref="main.vxml">
      <IMG SRC="welcome.efrm" WIDTH="32" HEIGHT="32"/><BR/>
    </A>
  </P>
</BODY>

</HTML>
```

- Loose coupling means that two separate (typically standard) markup languages are used, in separate files. Synchronization has to be performed by an external tool.

The pros and cons of tight versus loose coupling are listed in following table:

Table 17-2 Pros and cons of tight versus loose coupling.

Tight coupling	Loose coupling
+ Easy synchronization within the markup language.	- Synchronization has to be done outside the markup language
- Non-standard markup language only.	+ Use of industry standard markup languages, easier to generate.
+ Application developer maintains a single file for voice and visual dialogues.	- Application developer has to maintain two separate files for voice and visual dialogues.

Tight coupling	Loose coupling
+ No need to send a parameter to the servlet about which markup to use, since the markup is the same.	- Need to pass a parameter to the servlet identifying which markup to send, or find another solution.
- Requires higher overhead since the entire markup stream, containing both the voice and the visual dialogue, is sent to both browsers.	+ Requires less overhead.
- Supports only a single embedded voice document per visual component.	+ Supports complex voice dialogues.

The approach that will be presented in the next two paragraphs will take the loose coupling as a starting point, and find ways to add the advantages of tight coupling.

- Should the synchronization be put on the client device or on the server side?

Although both options are possible, the technology discussed in the next section is put on the server; in other words, the client device is not supposed to be multimodal (although the ability of a client device to support voice and data simultaneously would be of great help, such devices do not yet exist on the market).

- Should the applications be sequential or simultaneous?

One has a sequential application when one first has to interact with one device, then close that communication, and then open the other connection to be able to retrieve the response from it.

Simultaneous applications operate on two simultaneous connections.

Obviously, simultaneous applications should be much more convenient for the user. But if one wants to run such an application on one device, that device should support two simultaneous connections. For example, a WAP phone should be able to handle voice and data transmission at the same time. As mentioned before, such devices are not yet commercialized (they are expected to come out in the next few years). For example, VIWO is a sequential application, so it can run on one existing device.

17.4.2 VIWO

A VIWO (Voice In WAP Out) is an application that uses speech recognition as input and WAP technology as output; the latter uses the push technology. The VIWO product that will be available under WebSphere will typically be used for sequential applications, that is, it will require the user to close the voice connection before being able to take advantage of the WAP response.

Pros and cons

The main benefit of this application is that it provides a solution for owners of one device to the painstaking problem of data input on a mobile phone. For the user, the advantage is that there is no need to press the buttons on the smartphone several times to type letters.

The current limitation is that the typical applications are sequential.

VIWO is therefore optimized for users having only one device, given the fact that current devices cannot handle simultaneous connections.

How it works

Figure 17-3 shows an example of a typical data flow with the VIWO product.

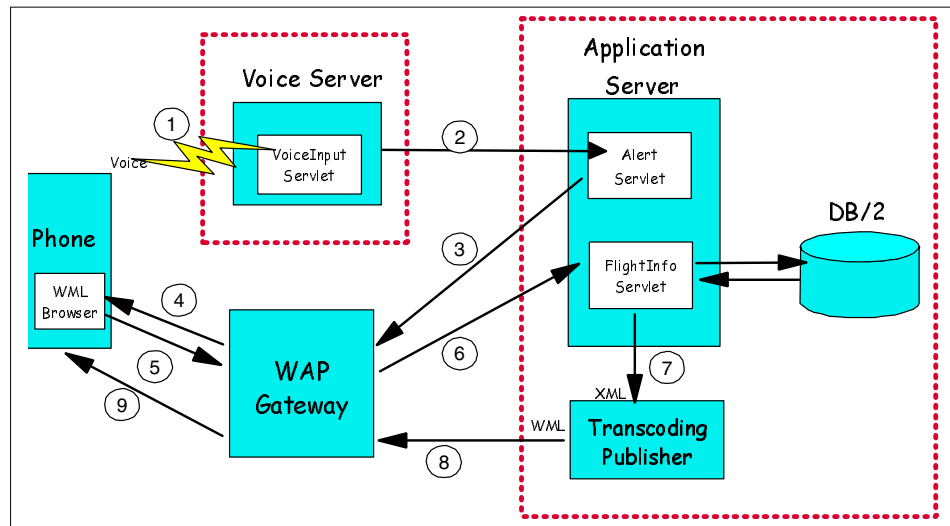


Figure 17-3 Example of usage scenario of the VIWO application

1. The user calls the voice server and gives input, recognized by the voice browser in the voice server.
2. The recognized input is sent as a parameter to the application server, in this case to the Alert servlet.
3. The Alert servlet initiates a push to the WAP gateway, typically using the PAP (Push Access Protocol). The information that is pushed is actually a URL with the parameters of the user.

4. The WAP gateway pushes the URL to the smartphone, using SMS messaging. In principle, WAP 1.2 Push could be used as well, but this protocol has yet to be implemented on client devices.

The user now must switch off the voice connection in order to be able to continue. This is because of the limitations of the current client devices.
5. The user accepts the URL, and the request is sent over WAP to the gateway.
6. The request is sent over HTTP to the servlet, which processes it.
7. The servlet sends XML output to the transcoder publisher (as will be clear from this redbook, other alternatives exist, such as JSPs directly coded in WML, or in transcoding from HTML to WML).
8. The transcoder produces WML output, based on the XSL StyleSheets, and sends the WML output to the WAP gateway as an HTTP response.
9. The WML output is passed over WAP to the client device.

The question is, how can the WAP browser get the URL out from SMS? This depends on the phone; some are capable of retrieving the URL for the WML page from the SMS. The message server only has to send a well-formed SMS to the user. When the user receives the message and opens it, it will initialize the WAP browser with the encapsulated URL from the message automatically (the user has to accept the action).

17.4.3 Future developments

In the coming years, we should see devices that handle voice and data simultaneously. For example, the GPRS standard defines three classes of devices, one of which is class A, meaning devices that can handle packet-switched and circuit-switched connections simultaneously. At this point, this is only a concept.

We should also see the use of that technology in the automotive industry, especially in combination with location-based lookups.



Part 5

Working example



Development environment for the sample application

This chapter addresses the installation of the Trade2 sample application into the pre-existing development environment consisting of:

- ▶ WebSphere Studio version 3.5.2 or newer
- ▶ Visual Age Java version 3.5.2 or newer
- ▶ WebSphere Voice Server SDK
- ▶ WebSphere Transcoding Publisher 3.5

For product installation information, refer to the documentation included with each product.

Keep in mind that the aforementioned products have many features and therefore many approaches are available to accomplish a task. In this chapter, we discuss the features we found most useful in our efforts.

18.1 The development environment

The development environment requires different settings and configurations for the application the developers are working on. This chapter will provide detailed information about how to set up the Trade2 sample application in this environment for further development purposes.

First, you must download the SG246259.zip file from `ftp://www.redbooks.ibm.com/redbooks/SG246259` (see Appendix A). Next, create a directory on your local hard drive (for example: `c:/weasamp` on NT or `/tmp/weasamp` for AIX) for the sample code, then unzip the file into the directory.

18.2 Application database

We assume that a DB2 database server is already set up in the development environment and accessible from the developer machine.

The application connects to a back-end database, which has to be created before you can start work on development.

Follow these steps to create your Trade2 database:

1. Log on as a database administrator on the database server.
2. Under the database directory, you can find the script which creates the database and the tables, and also populates the database. The content for the database is in the text (.txt) files.
3. Open a **Command prompt** window, change the directory to the **database** directory.
4. Run `init.bat`.
5. A new Command prompt window will appear where the database script is running.

After you are finished creating the database, close all the Command prompt windows. Now the Trade database is set.

18.3 WebSphere Studio

WebSphere Studio provides an integrated development environment for the Web applications. The following sections will describe how to import the studio archive with the Trade2 application, then how to publish the application within the development environment.

18.3.1 Importing the Studio Archive File

Open WebSphere Studio; if this is the first time you are opening Studio, you will be prompted with a welcome window that will allow you to create a new project, open an existing project or import a Web site and create a new project. For the sample application, we need to open an existing archive. This selection is not available from the welcome window, therefore we select **New** and create a dummy project.

1. From the menu bar, select **File->Open Archive...** to display the **Open Studio Archive File Panel**.
2. Select **trade.wsr** and click **Open**.
3. Leave **Create new project and extract entire archive** selected, switch to the **Destination** tab.
4. Select **Custom locations**.
5. Under the Project folder, set the directory for the Studio project, for example:
d:\wea\trade
6. Click **Extract**.

Now the Studio archive is extracted to the d:\wea\trade directory, and the project is opened in Studio.

18.3.2 Publishing a WebSphere Studio project

WebSphere Studio supports Publishing Stages. Publishing stages allow a team to test code in isolated stages. We found it beneficial to use as a model the traditional software development model: function, unit and system test. Therefore, three stages are included within the Studio archive. They are VAJ, Test and Production. For the purposes of this book we used the Production stage as the final testing environment. For real production environments, another stage should be considered.

Table 18-1 Publishing stages

Publishing stage	Environment	Purpose
VAJ	WebSphere Test Environment running within VAJ	Function test; mainly for Java code (for example JSP, servlet, JavaBean and EJBs)
Test	WebSphere Application Server	Component test
Production	WebSphere Application Server	Runtime

Make sure that you have different server names under each publishing stage, then you can apply different publishing targets to the directories. If the Studio project has the same server name under different publishing stages, the publishing stages will be the same for the servers and for the publishing stages.

To create a new server, follow these steps:

1. Select the top entry in the Publishing pane, for example **VAJ**; right-click the entry, select **Insert -> Server**.
2. Give a unique name to the server, for example `localhost_VAJ`, and set the server address to **http://localhost**.
3. Right click the server **localhost_VAJ**, select **Properties**; you will get the following window:

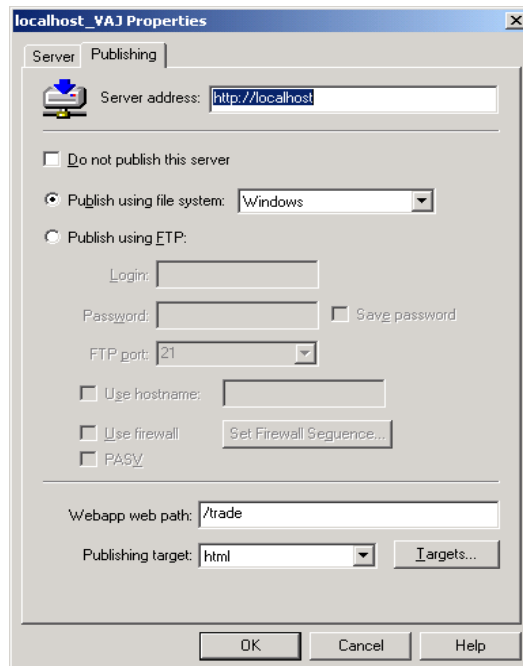


Figure 18-1 Server properties: WebSphere Studio

4. To test any changes you have made, you need to have mapped drives corresponding to the publishing targets we have established. To do this, share the appropriate folder (restrict it to your ID) and map it to a network drive.

Network drive mappings:

Table 18-2 Network drive mappings

drive	directory
V:	<VAJ install directory>\ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app
W:	<WebSphere Install Dir>\hosts\default_host\trade2_App

5. Select **Targets...** then set up the following publishing targets for all the publishing stages and the servers.

Publishing directories:

Table 18-3 Publishing directories for TEST

Test - localhost_TEST	
html	w:\web
servlet	w:\servlets

Table 18-4 Publishing directories for VAJ

VAJ - localhost_VAJ	
html	v:\web
servlet	v:\servlets

Table 18-5 Publishing directories for Production

Production - <your server>	
html	your Web directory on the production server
servlet	your servlet directory on the production server

6. After finishing the target setup, click **OK**.

18.4 VisualAge for Java configuration

After installing VisualAge for Java (VAJ), several features are required for the sample application to compile and run. Features can be added from the quick start menu (which has an **F2** shortcut key).

1. From the quick start menu select **Features->Add features**.
2. A list of features to add to the workspace will appear. For the purposes of the sample application, add the following features.
 - IBM EJB Development Environment

- IBM Enterprise Toolkit for AS/400 3.5

(Do this even if you are not deploying on AS/400. The application is capable of being deployed to AS/400 and needs this feature.)

- IBM WebSphere Test Environment

After the features above have been added to the workspace, you can verify that the correct features have been added.

3. Select **Quick start->Features->Delete feature** and the following features (or more) should be installed.

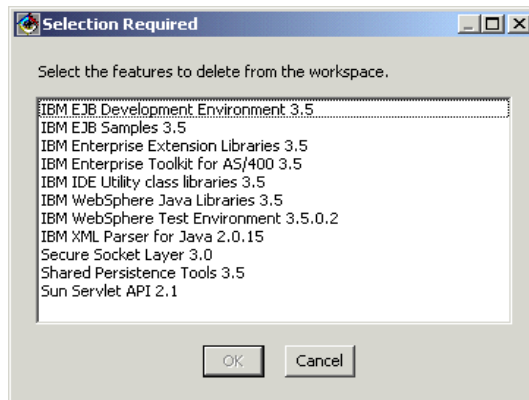


Figure 18-2 Listing installed features within VisualAge for Java

4. Press **Cancel** to return to the workspace.

18.4.1 Adding DB2 libraries to VisualAge for Java

In addition, the applications EJB container will need access to database drivers. For the purposes of this redbook, the application was tested with DB2; however, there is nothing inherent to the application that should prevent the use of other databases.

1. Select **Window -> Options** to bring up the Options panel.
2. Select **Resources** on the left side of the navigation bar.
3. Set the Workspace class path by adding the following library to it: <DB2 install directory>\java\db2java.zip.
4. Click **OK**.

18.5 Importing the VisualAge for Java repository file

The following procedure will guide you through importing a VisualAge for Java repository file, which contains the Trade2 application.

1. Select **File->Import...** from the VisualAge for Java menu.

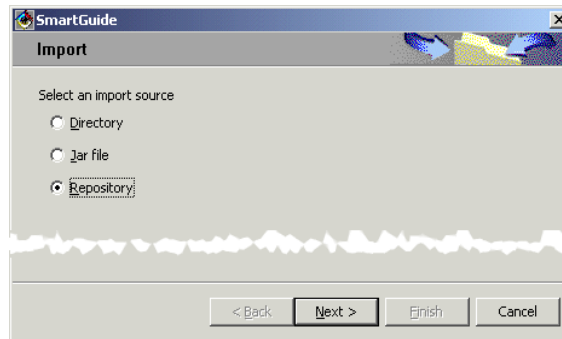


Figure 18-3 Import

2. Select **Repository** and click **Next**.

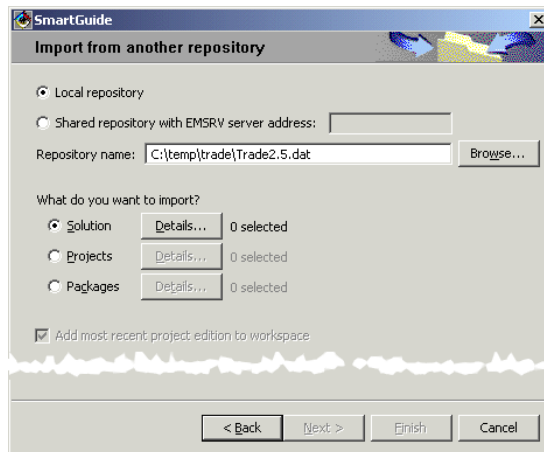


Figure 18-4 Import repository

3. Use **Browse** to navigate to the trade.dat file in the temporary directory created earlier.
4. Select the **Projects** radio button and click the adjacent **Details...** button. The Project Import window should be displayed.

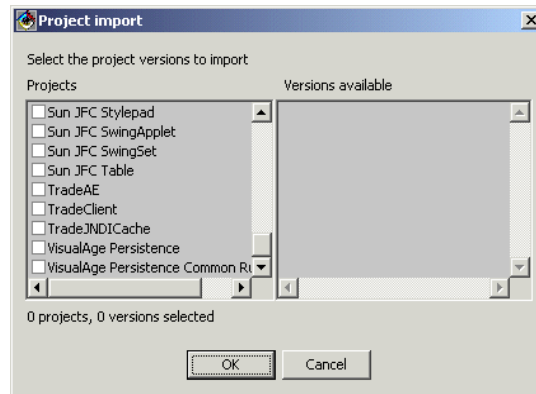


Figure 18-5 Project Import window

5. From the Project Import window, select the **Trade** project.
6. Select the latest version from the **Versions Available** pane, then click **OK**.
7. Ensure that **Add most recent project edition to workspace** is selected and click **Finish**.

18.5.1 Rebuilding the EJBs

The Trade2 application comes with several EJBs that must be mapped to database tables (see Figure 18-6 on page 363).

1. This can easily be accomplished by right-clicking the **TradeEJBs** EJB group, selecting **Add** then selecting **Schema and Map from EJB Group**.

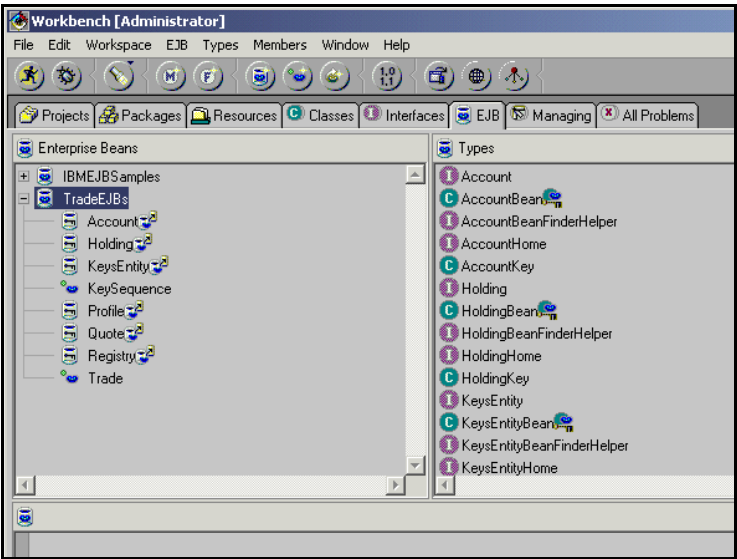


Figure 18-6 Trade2 EJBs

2. The Trade2 servlets need to be able to access the EJBs. VisualAge for Java greatly simplifies this task by generating *Access Beans*. These Access Beans contain the logic for the JNDI lookup, establishing the RMI connection and optionally caching the result.

Table 18-6 identifies the Trade2 application EJBs which require Access Beans and the type of Access Bean required:

Table 18-6 EJBs with AccessBeans

EJB	EJB Type	Access Bean Type
Holding	Entity	Copy Helper
Profile	Entity	Copy Helper
Quote	Entity	Copy Helper
Trade	Session	Java Bean Wrapper

3. To generate an Access Bean, right-click the EJB and select **Add->Access Bean...** The Create Access Bean SmartGuide dialog will be displayed.
4. The final step is to generate the deployed code. To do this, right-click the **EJB Group** and select **Generating Deployed Code**.

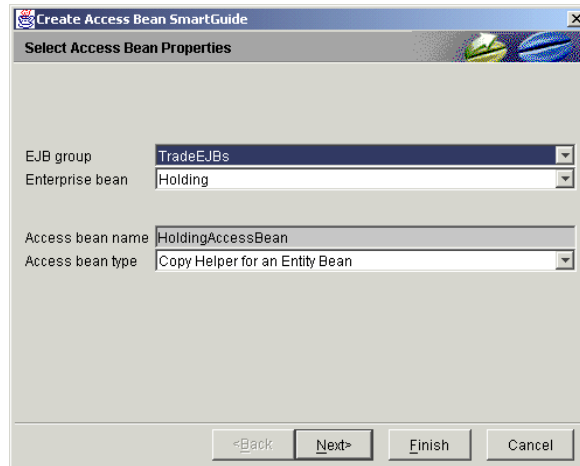


Figure 18-7 Create Access Bean SmartGuide

5. If DB2 conversion is not required and if using DB2, click **Finish**. For other databases, see the documentation for Java data types to determine if conversion is required.
6. Create the Access Beans for the required EJBs (see Table 18-6 on page 363).

18.5.2 Exporting the deployed EJBs

The EJBs for the Trade2 Web application handle the database tasks. You can find the EJBs under VisualAge for Java; they are shipped together with the example application.

In order to deploy the EJBs, the deployed EJB .jar file has to be produced.

1. Go to the EJB tab in VisualAge for Java, right-click the **TradeEJBs** EJB group, then select **Export->Deployed JAR....**
This function only exports the classes related to the EJBs by default; this application also has Access Beans, and other elements have to be packaged together with the EJBs.
2. In the .jar file textbox, type in the directory and the filename for the .jar file; if it is a temporary directory, then the .jar file has to be moved to the appropriate

directory on the application server node. Now type:
c:\temp\Trade2Beans.jar.

All the beans should be selected by default, so the beans checkbox is selected, and says 8 selected; below that, the .class checkbox is also selected, and indicates that 119 class have been selected for export. In order to select all the necessary class files, click the **Details** button beside the .class label, and select the following classes also:

- HoldingAccessBean :: trade
- ProfileAccessBean :: trade
- QuoteAccessBean :: trade
- QuoteAccessBeanDated :: trade
- TradeAccessBean :: trade

3. Click **OK**.
4. Now some of the necessary classes are selected, but some are still missing. Click the **Select referenced types and resources** button to automatically select the missing classes; now it says that 135 class files are selected.
5. Click **Finish** to export the.jar file.
6. Now the deployed EJBs are in the Trade2Beans.jar file. Copy the file to the <WAS install directory>\DeployedEJBs directory, then set them up under the WebSphere Application Server using the Administrator's Console.

For information on setting up EJBs in the runtime environment for WebSphere Application Server, refer to the *WebSphere V3.5 Handbook*, SG24-6161.

18.6 WebSphere Test Environment configuration

The following sections will guide you through the steps to set up and configure your WebSphere Test Environment (WTE) for the Trade2 application.

18.6.1 Setting up a new Web application under WTE

Creating a new Web application under WTE ensures that the application has the same environment during the development phase and the final production.

The following steps will guide you through the process of setting up the Trade2 Web application under WTE. You can also find the setup directions in the VisualAge for Java help.

1. Create the trade2_app directory under <VAJ install directory>\ide\project_resources\IBM WebSphere Test Environment\hosts\default_host.
2. Create the **servlets** and **Web** directory under trade2_app.
3. Copy the trade2_app.webapp file under the **servlets** directory from the unpacked additional material's directory. In the .webapp file, there is an entry where the TradeAppServlet is defined.
4. Modify the default.servlet_engine under the <VAJ install directory>\ide\project_resources\IBM WebSphere Test Environment\Properties, add the new **WebSphere-webgroup** definition to the file, after the default_app WebSphere-webgroup entry:

```
<WebSphere-webgroup name="trade2_app">
<description>Trade2 WebGroup</description>
<document-root>$approot$/web</document-root>
<classpath>$approot$/servlets$psep$$server_root$/servlets</classpath>
<root-uri>/trade/</root-uri>
<auto-reload enabled="true" polling-interval="3000"/>
<shared-context>false</shared-context>
</WebSphere-webgroup>
```

WTE is ready to run your new Web application under trade2_app, with the root URI: /trade/.

For further information or if you encounter any problems, refer to IBM VisualAge for Java Help (*Creating and configuring new Web applications*).

18.6.2 Adding MIME Types to WTE

In order to make the different kinds of pages work, you have to register the new MIME types for WTE.

On your Windows NT development machine, go to the directory: <VAJ_install_directory>\ide\project_resources\IBM WebSphere Test Environment\properties and open the file: default.servlet_engine with a text editor (Notepad).

Find the following pattern of code:

```
<mime type="...">
...
</mime>
```


There are many MIME type definitions. Go to the end of the definitions and add yours:

```
<mime type="text/vnd.wap.wml">
  <ext>wml</ext>
</mime>
<mime type="image/vnd.wap.wbmp">
  <ext>wbmp</ext>
</mime>
<mime type="application/vnd.wap.wmlc">
  <ext>wmlc</ext>
</mime>
<mime type="text/vnd.wap.wmlscript">
  <ext>wmls</ext>
</mime>
<mime type="application/vnd.wap.wmlscriptc">
  <ext>wmlsc</ext>
</mime>
<mime type="application/vnd.wap.wtls-ca-certificate">
  <ext>ca</ext>
</mime>
```

18.6.3 Publishing the Studio project into WTE

1. Select the VisualAge for Java publishing stage by selecting **Project -> Publishing Stage -> VAJ**.
2. Select the Server under the publishing stage name **localhost_VAJ**.
3. Right-click it, then select **Publish this server**.
4. Click **OK** in the window that appears, or set your publishing configuration beforehand.

The files and directories should be published under the <VAJ install directory>\ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\trade2_app\ directory. You should have the following structure under the Web directory (see Figure 18-8 on page 368).

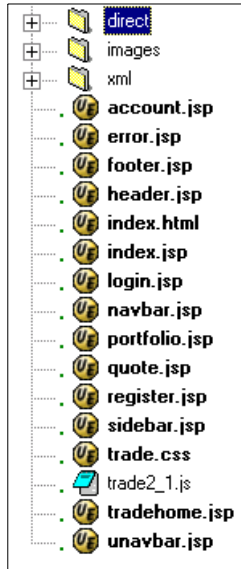


Figure 18-8 Trade2 directory content for the VisualAge for Java Web directory

Under the **client**, **images** and **xml** folders, there are other subfolders and files.

18.6.4 Starting the WebSphere Test Environment

Before you start WTE, make sure that the following services under Windows are stopped:

- ▶ IBM HTTP Administration
- ▶ IBM HTTP Server
- ▶ IBM WS AdminServer

On the other hand, make sure that the DB2 server is running.

To start WTE, follow these steps:

1. Start the WebSphere Test Environment by selecting **Workspace->Tools->WebSphere Test Environment...** from the menu.

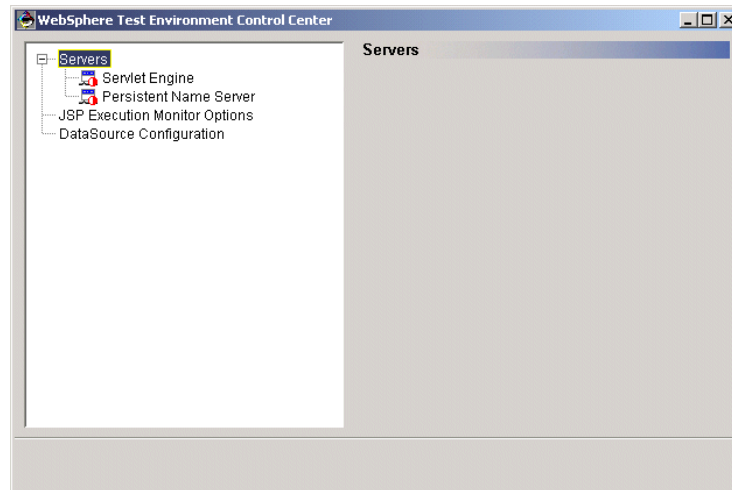


Figure 18-9 WebSphere Test Environment

2. Select **Persistent Name Server** and click **Start Name Server** in the adjacent pane.
3. Select **Servlet Engine** and click **Edit Class Path...** in the adjacent pane.

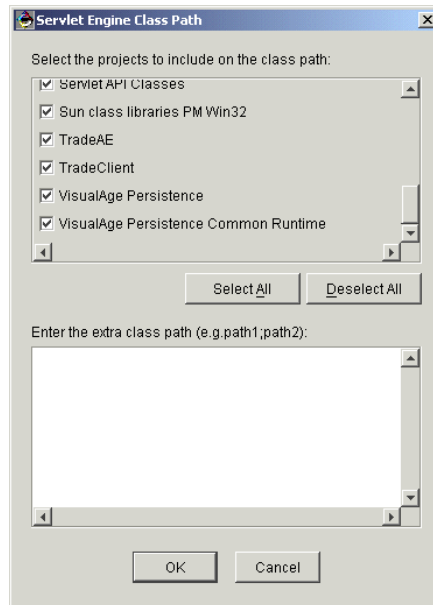


Figure 18-10 Class path pane

4. Click **Start Servlet Engine** in the adjacent pane.

Now that the servlet engine is running within the WebSphere Test Environment, it can serve the servlets and JSPs for Web clients.

18.6.5 Starting the EJB server

In order to access your EJBs, you have to start the EJB server, which manages the container for the Beans. Follow these steps:

1. In the workspace, select the **EJB** tab.
2. Right-click the Trade2 EJB group **TradeEJBs** and select **Add To ->Server Configuration**. If you have already added the group to the server configuration, choose **Open To ->Server Configuration**.
3. A new window appears with your EJB server. Choose the appropriate EJB server (you can have several) by selecting the EJB Group name in the right pane of the window.
4. Right-click the selected EJB Server, then choose **Start Server**.

It takes a while for the server to start; when the server is running, you can see a small running figure right next to the entry.

Now the EJB server is running within the WebSphere Test Environment and serving the EJBs under VisualAge for Java.

18.7 WebSphere Voice Server SDK configuration

WebSphere Voice SDK (WVS SDK) must undergo a post-installation process in order to make it work together with WTP.

In order to set up the WebSphere Voice Server to use a proxy server, refer to “Proxy settings for WebSphere Voice SDK” on page 182.

If you are using WTP as a reverse proxy, you do not have to go through the steps described above.

18.8 WebSphere Transcoding Publisher configuration

The following section will cover the configuration of WebSphere Transcoding Publisher for the Trade2 sample application. For more information about how to install and set up WTP, read the IBM Redbook *New Capabilities in IBM WebSphere Transcoding Publisher Version 3.5 Extending Web Applications to the Pervasive World*, SG24-6233.

The Trade2 XML example can be used to produce simplified HTML as well as VoiceXML output. The following sections will describe how to set up the example for WTP.

18.8.1 Setting up the Voice transcoder

The HTML-to-VoiceXML transcoder does not come with WTP v3.51; you must download it from the product's Web site, then install it under WTP. Follow the instructions in Section 16.3.3, “Setting up the HTML-to-VoiceXML transcoder” on page 330.

18.8.2 WTP preference profile for voice application

In order to run the voice application with WTP, you must set up and register a new device profile for the voice client, which will utilize the HTML-to-VoiceXML transcoder. This is done by using the Profile Builder from WTP, which has to be downloaded from the product's Web site as a separate component (see “Profile Builder” on page 177).

For more information about creating or modifying a device profile, please refer to “Creating a new device profile” on page 331.

Modifying the preference profile

In order to set up the profile correctly, the Desired content types have to be changed in the settings.

Please make sure that the value is set to text/vxml , as show in Figure 18-11.

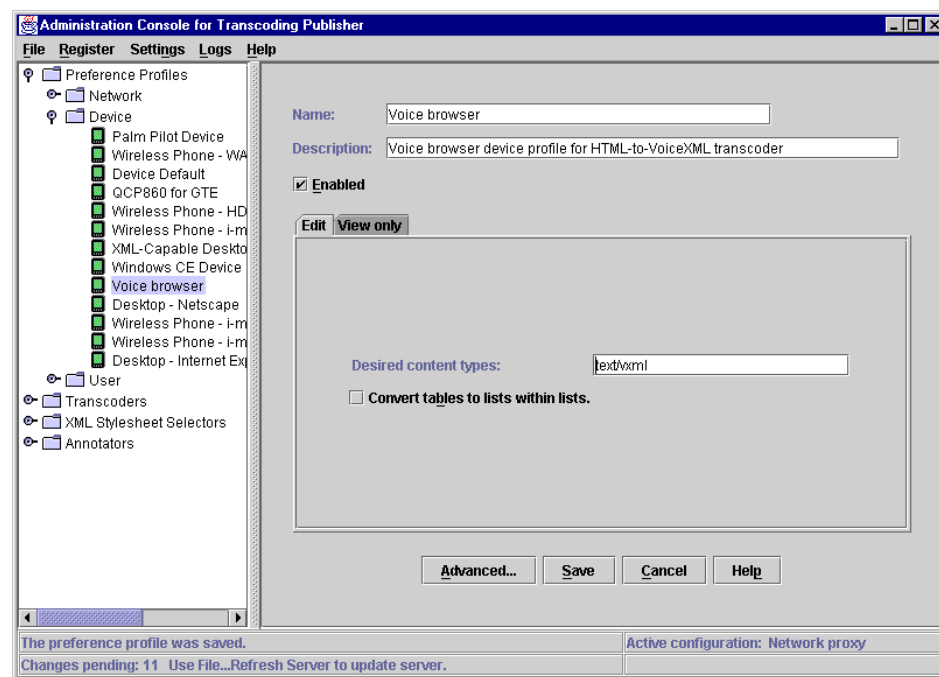


Figure 18-11 Main setting panel for the voice preference profile

Futhermore, to refer to the profile in our XSL settings we must define a a unique key. This is done under the Advanced options in the Additional Preferences. You should add here the key deviceType with the value VoiceProfile. See the following figure for an example:

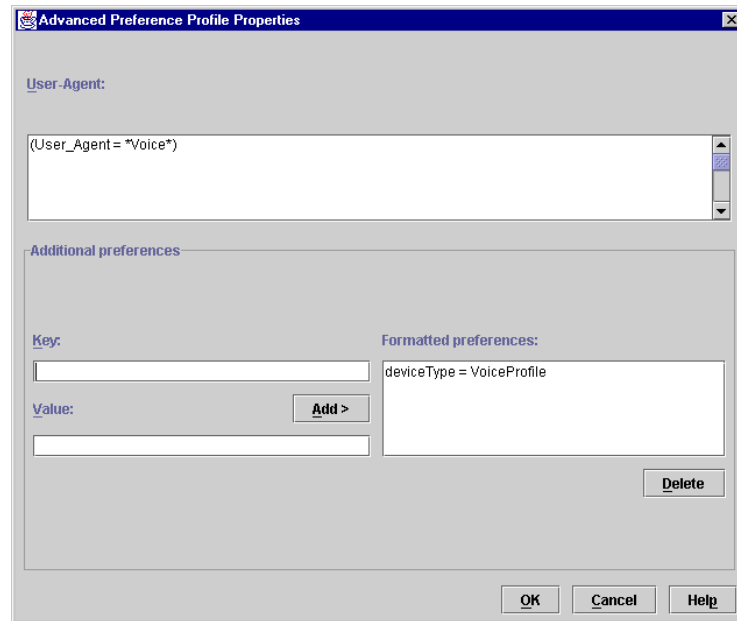


Figure 18-12 Advanced properties for the voice profile

In order to activate the changes made in the settings, you have to save the settings (using the **Save** button in the right window) and refresh the server (by selecting **File -> Refresh Server**). Otherwise, the changes do not take effect in the application.

Additional configuration of the device profiles (optional)

When you are developing, for example, the voice application, you should disable the other preference profiles. This is not a mandatory step; it will only help to avoid any interference with other profiles during development.

In the next step, you should disable all profiles except the Default and the new Voice profiles. To disable a profile, first select the profile, then click the **Enabled** checkbox, deselect it, and save the changes using the **Save** button at the bottom of the window.

Please make sure that the Desired content type of the Default profile is set to text/html only. The settings in the Administrator Console should look like those in Figure 18-13:

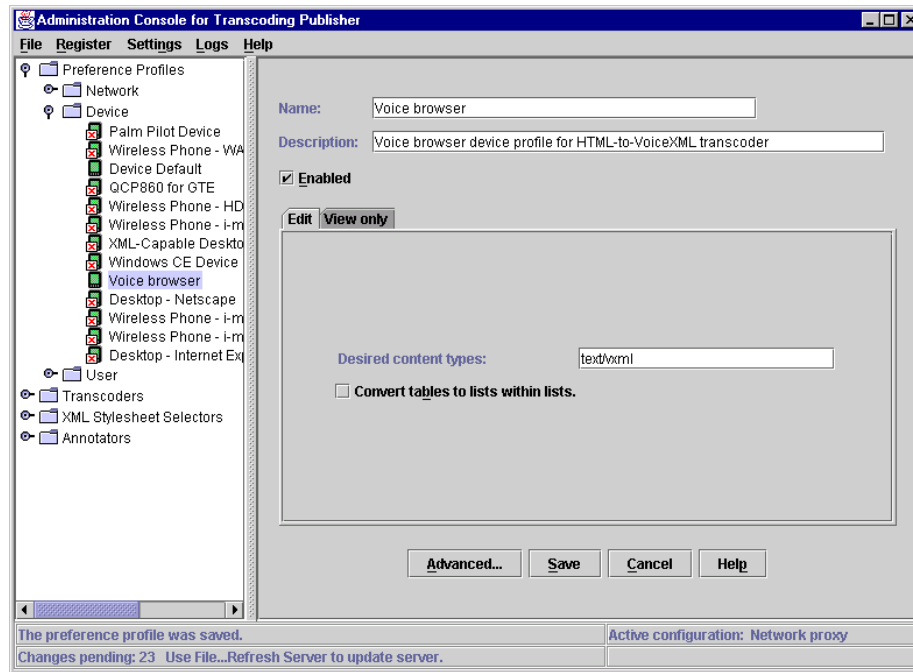


Figure 18-13 Overview of the device settings in WTP

18.8.3 Registering the StyleSheets

The next step is to register the StyleSheets in the WTP environment. This section will describe how to register and configure the Stylesheets for the simplified HTML and the VoiceXML (using universal transcoding) scenarios.

1. Open the WTP Administration Console if it is not running.
2. Create the following additional folders under the XML StyleSheet Selectors folder:
 - Trade/simplifiedHTML for the simplified HTML XSL documents
 - Trade/voiceXML for the VoiceXML XSL documents
3. Under the folders, the appropriate StyleSheet should be registered. This can be done by choosing **Register -> XML StyleSheet** from the menu bar.

4. In the upcoming guide, you can select the StyleSheet and the folder in which to store it in the WTP settings. After publishing the project (see 18.3.2, “Publishing a WebSphere Studio project” on page 357), you should find the XSL documents in the following folders:
 - **Simplified HTML XSL documents:** <VAJ_install_directory>\ide\project_resources\ IBM WebSphere Test Environment\hosts\default_host\trade2_app\web\xml\simphtml\xsl
 - **VoiceXML XSL documents:** <VAJ_install_directory>\ide\project_resources\ IBM WebSphere Test Environment\hosts\default_host\trade2_app\web\xml\vxml\xsl

The following steps will guide you through registering a StyleSheet with the WTP Administration Console:

1. Select **Register -> XML StyleSheet...** from the menu.
2. In the Welcome window, click **Next** to get to the next window.
3. Find the required StyleSheet using the **Browse** button; after selecting the file, the XSL file with the full path will appear in the text box. Leave the output content format as it is; later, you can change it in the Properties panel.
4. Click **Next**.
5. Give a name to this entry with the registered XSL; this name should be descriptive, and usually it is the same as the file name. You can add a description to the entry if you wish.
6. Click **Next**.
7. In the next window, a tree appears with folders, where the StyleSheets will be registered. Select your folder (for example: XML Stylesheet Selector\trade\simphtml) then click **Next**.
8. The next window gives you the opportunity to set up the input and the output DTD locations for your XML documents. Leave these fields empty unless you want to use them (later you can also change these fields from the Properties panel), then click **Next**.
9. The last window asks if the StyleSheet should be enabled or not after the registration. Leave the default (= yes); then the StyleSheet will be enabled.
10. If you want to add another StyleSheet, click **Another..**; if you have finished, click **Finish**.

The following tables list the StyleSheets for the Trade2 application.

Table 18-7 Simplified HTML StyleSheets

Name	File name
TradeAccount	TradeAccount.shtml.xsl
TradeError	TradeError.shtml.xsl
TradeHome	TradeHome.shtml.xsl
TradeLogin	TradeLogin.shtml.xsl
TradePortfolio	TradePortfolio.shtml.xsl
TradeQuote	TradeQuote.shtml.xsl
TradeRegister	TradeRegister.shtml.xsl
TradeWelcome	TradeWelcome.shtml.xsl

There is a TradeTemplate.shtml.xsl file published together with the StyleSheets for the simplified HTML scenario. This file is a template, which should not be configured under WTP; it is used by the other StyleSheets as an included file.

Table 18-8 VoiceXML StyleSheets

Name	File name
TradeError	TradeError.vxml.xsl
TradeHome	TradeHome.vxml.xsl
TradePortfolio	TradePortfolio.vxml.xsl
TradeQuote	TradeQuote.vxml.xsl
TradeWelcome	TradeWelcome.vxml.xsl

After registration, the window of the Administrator Console should look like this:

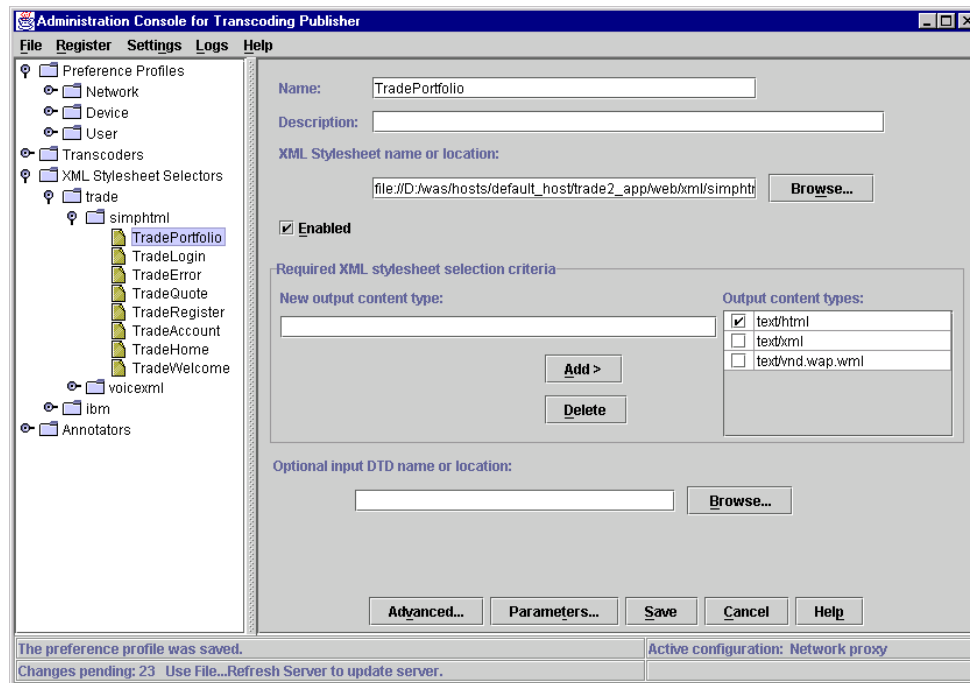


Figure 18-14 Overview of the registered XSL in the WTP

After any modifications to the profiles, click the **Save** button at the bottom of the window.

Configuring the StyleSheets for simplified HTML

In order to run the application, the different StyleSheets have to be configured. There are two main options to configure the StyleSheets: Input DTD or Condition for Criteria Matching HTTP header.

Input DTD (in the main panel of the settings) can be used to match the StyleSheet via a valid DTD entry. The referenced DTD has to be the same as in the XML document. This option is only used for the two error StyleSheets. The following table lists the values for the configuration:

Table 18-9 Overview of the configuration for the optional input DTD

Name of StyleSheet	Optional input DTD name or location
TradeError.shtml.xsl	file://C:/Program Files/IBM/VisualAge for Java/ide/project_resources/IBM WebSphere Test Environment/hosts/default_host/trade2_app/web/xml/simhtml/xml/TradeError.dtd (can be chosen from the Browse panel)

Condition for Criteria Matching HTTP header (Advanced XML StyleSheet Selection Properties) can be used to select the StyleSheet via defined criterias in the HTTP header. For example, in the Trade2 XML application, we used it to select StyleSheets with the help of the incoming URL. The following table lists the values for the configuration.

Table 18-10 Overview of the configuration of the header matching preferences

Name of StyleSheet	Condition for criteria matching HTTP header
TradeWelcome.shtml.xsl	(URL = "*Welcome.xml*") (URL = "*action=logout*")
TradeRegister.shtml.xsl	(URL = "*TradeRegister.xml*")
TradeLogin.shtml.xsl	(URL = "*TradeLogin.xml*")
TradeHome.shtml.xsl	(URL = "*action=gohome*") (URL = "*action=register*") (URL = "*TradeAppServlet?login*")
TradeAccount.shtml.xsl	(URL = "*TradeAppServlet?account*") (URL = "*TradeAppServlet?action=account*") (URL = "*action=updateAccount*")
TradeQuote.shtml.xsl	(URL = "*TradeAppServlet?quote*")
TradePortfolio.shtml.xsl	(URL = "*action=portfolio*") (URL = "*TradeAppServlet?portfolio*") (URL = "*action=buy*") (URL = "*action=sell*")

Configuring the StyleSheets for VoiceXML

The following table lists the TradeError XSL for the voice application, using the DTD for matching.

Table 18-11 Overview of the configuration for the optional input DTD

Name of StyleSheet	Optional input DTD name or location
TradeError.vxml.xsl	file://C:/Program Files/IBM/VisualAge for Java/ide/project_resources/IBM WebSphere Test Environment/hosts/default_host/trade2_app/web/xml/vxml/xml/TradeError.dtd (can be chosen from the Browse menu)

Table 18-12 lists the Stylesheets for the voice application with the HTTP header matching.

Table 18-12 Overview for the configuration of the header matching preferences

Name of StyleSheet	Condition for criteria matching HTTP header
TradeWelcome.vxml.xsl	(URL = "*TradeWelcome.xml*") (URL = "*action=logout*")
TradeHome.vxml.xsl	(URL = "*action=login*") (URL = "*action=gohome*")
TradeQuote.vxml.xsl	(URL = "*action=quote*")
TradePortfolio.vxml.xsl	(URL = "*action=portfolio*") (URL = "*action=buy*") (URL = "*action=sell*")

Criteria matching preferences can be used to select a StyleSheet via defined keys and values. This option is used in the Trade2 example to match the StyleSheets to the voice application. Table 18-13 lists the values for the configuration.

Table 18-13 Overview for the configuration of the criteria matching preferences

Name of StyleSheet	Criteria matching preferences
TradeWelcome.vxml.xsl	deviceType = VoiceProfile
TradeHome.vxml.xsl	deviceType = VoiceProfile
TradeQuote.vxml.xsl	deviceType = VoiceProfile
TradePortfolio.vxml.xsl	deviceType = VoiceProfile
TradeError.vxml.xsl	deviceType = VoiceProfile

Make sure that the content-type for the voice StyleSheets is text/vxml.

18.9 TradeAppServlet configuration for voice

The voice application is implemented in two different scenarios; one is the direct VoiceXML, the other is VoiceXML using universal transcoding.

In our example, these two scenarios cannot run at the same time. The developer has to decide between the two and modify the code accordingly to the chosen approach.

Below is the source code from the TradeAppServlet class' getTradeServletActionObject method. First, the User Agent will tell the servlet if the client is a voice client, then it will instantiate the necessary object, which can be:

- ▶ trade_client.direct.voice.TradeServletAction() for the direct approach.
- ▶ trade_client.xml.simplifiedhtml.TradeServletAction() for universal transcoding.

As you might noticed, for universal transcoding the same class is used as in the case of the simplified HTML. The reason is that the application flow and the XML data are the same in both cases; only the StyleSheets are different. The StyleSheets are handled by WTP, so in this particular case, there is no reason why we cannot use the same object.

Example 18-1 getTradeServletActionObject() method

```
1 // VoiceXML
2     else if (userAgent.indexOf("Voice") > -1){
3 // instantiating the object for the direct approach
4     tsAction=new trade_client.direct.voice.TradeServletAction();
5 // instantiating the object for the universal transcoding approach
6 //     tsAction=new trade_client.xml.simplifiedhtml.TradeServletAction();
7
8     session.setAttribute("devicetype","voice");
9     TradeLogging.logMessage("devicetype=voice");
10 }
```

The default setting for the Trade2 application is to use the direct approach. If you want to switch to universal transcoding, you have to comment the *fourth* line (where the object is instantiated) and uncomment the *sixth* line.

Note: In the case of the runtime environment with WebSphere Application Server, export the class from VisualAge for Java, then restart the Web application under WebSphere Application Server.

18.10 StyleSheet import

The StyleSheets for simplified HTML use the `<xsl:import=... />` tag. With this tag, the TradeTemplate.xsl file is imported into the XSL, which contains several supplemental content elements, such as header, navigation bar, and so on.

The `<xsl:import=... />` tag accepts only absolute URLs for the imported file. In this case, the tag should look like this:

```
<xsl:import="file:///C:/Visual Age for Java install  
path\ide\project_resources\IBM WebSphere Test  
Environment\hosts\default_host\trade2_app\web\xml\simphtml\xsl\TradeTem  
plate.shtml.xsl" />
```

where the install path is the fully qualified install directory with the drive letter, for example:

```
D:\Program Files\IBM\VisualAge for Java
```

The following StyleSheet files have the `<xsl:import=... />` tag:

- ▶ TradeAccount.shtml.xsl
- ▶ TradeRegister.shtml.xsl
- ▶ TradeQuote.shtml.xsl
- ▶ TradePortfolio.shtml.xsl
- ▶ TradeLogin.shtml.xsl
- ▶ TradeHome.shtml.xsl
- ▶ TradeError.shtml.xsl
- ▶ TradeWelcome.shtml.xsl

Make sure that you have updated the XSL files' import tags according to your directory settings. For further information about this, refer to "Developing the XSL files and the user interface" on page 300.



Runtime environment for the sample application

This chapter will describe how to install the Trade2 Web Application into the runtime environment.

The code for the Trade application, which is downloadable, includes the deployable code for runtime and source code for development.

19.1 Runtime enviroment for the sample application

The development environment for the Trade2 application was already introduced in the previous chapter. The same Trade2 application will be introduced here, but this time it will be deployed in the runtime environment.

19.1.1 Runtime environment

In this book, we provide a working example with several scenarios for mobile devices. The sample was run on the following architecture in our lab, shown in Figure 19-1:

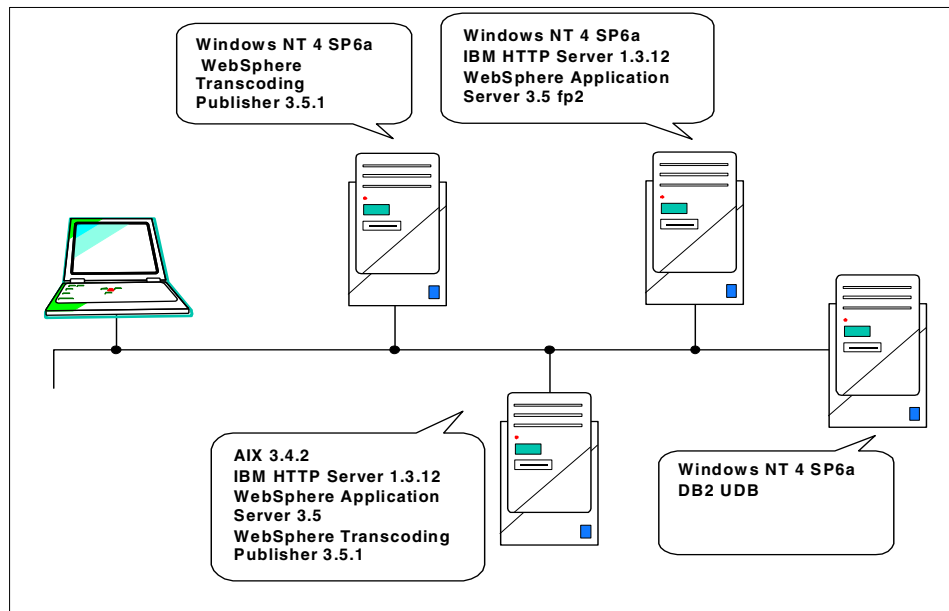


Figure 19-1 Runtime environment in the ITSO lab

The topology above depicts two separate cases, running in the same environment. The two cases are:

- ▶ Using WebSphere Transcoding Publisher as a proxy, either forward or reverse.
- ▶ Using WebSphere Transcoding Publisher as a filter servlet.

In the first case, a Windows NT machine has WTP running as a proxy, and another Windows NT server is running WebSphere Application Server with our Trade2 Web application.

In the second case, WTP and WAS are running on the same AIX machine, and WTP is deployed as a servlet filter.

In both cases, the same database server and the same database for the Trade2 application are used. Both WebSphere Application Server installations have their own database: wasnt and wasaix.

When the client accesses the site, it can reach one of the Web applications by using the right URL.

19.2 Installing and configuring the runtime environment

The following sections provide information about installing and configuring the runtime environment. This book will not go into details in terms of product installation, but refers to other books, and gives additional information on how to install the products properly.

19.2.1 Database node

The database node runs on an Intel-based machine with Windows NT 4 and Service Pack 6a. The database server is IBM DB2 UDB 6.1 with fixpack 4.

The node runs one instance of database server with three different databases:

Table 19-1 Databases

Database	Purpose
wasnt	WebSphere Application Server repository for the NT server
wasaix	WebSphere Application Server repository for the AIX server
trade	The Trade2 application sample database

In our runtime environment, the database server is running on a dedicated database node. The applications are accessing the server remotely, using a database client.

Setting up the WebSphere Application Server databases

The WebSphere Application Server has to connect to a database server, where the repository for the application server is stored. The WAS node contains the DB2 client, and the client connects to the DB2 Server remotely.

The documentation for installing and configuring the DB2 Server on Windows NT can be found in the *WebSphere V3.5 Handbook*, SG24-6161. Install DB2 server and create an administrative database on page 1054.

Create the two databases for the two WAS application server. Make sure that you are using the appropriate database names for your WebSphere installation, as shown on page 1056 of that book; **wasnt** for the Windows NT machine and **wasaix** for the AIX machine.

Setting up the Trade2 database

The Trade2 application uses a database to store user information and data from the stock market.

You must have the database and the tables under DB2, as well as some data to test the application.

To find how to create the database and populate it using the scripts provided within the additional materials, see Section 18.2, “Application database” on page 356.

Of course, it is possible to populate the database with your own data; in that case, use the sample text files under the *database* directory within the additional materials.

19.2.2 Web application node without Transcoder

The Web application node runs on a Windows NT 4 Server with Service Pack 6a.

The following products must be installed on this node:

- ▶ DB2 client
- ▶ WebSphere Application Server

Installing the DB2 client

The documentation for installing the DB2 client on the application server machine under Windows NT can be found in the *WebSphere V3.5 Handbook*, SG24-6161.

Installing WebSphere Application Server

The documentation necessary for installing WebSphere Application Server on Windows NT can be found in the *WebSphere V3.5 Handbook*, SG24-6161, A.2.4 “WebSphere installation” on page 1063.

After installation, make sure that you have the computer name set up under the default_host node in the WebSphere Administrator's Console.

1. Start the WebSphere Administrator's Console, then select the **default_host** node in the left pane.
2. Under the General tab, there is a list within the Aliases box. Add the server name, the server's fully distinguished name and the local host to the list, if they are missing. Optionally, you can also add the server's IP address to the list.
3. Click **Apply**.
4. You must restart the IBM WS AdminServer service for the changes to take effect.

Setting up MIME types

The Web server and the Web application server both have to handle the new mime types required by the mobile clients.

1. For the Web Server, the mime types are stored in the following file: <IBM http server installation directory>\conf\mime.types; open the file with a text editor such as Notepad.
2. Add the following lines to the end of the file:

text/vxml	vxml
text/vnd.wap.wml	wml
image/vnd.wap.wbmp	wbmp
application/vnd.wap.wmlc	wmlc
text/vnd.wap.wmlscript	wmls
application/vnd.wap.wmlscriptc	wmlsc
3. Save, then close the file.
4. Restart the IBM HTTP Server.

WebSphere Application Server also has to register the mime types.

1. Start the WebSphere Administrator's Console, then select the **default_host** node in the left pane.
2. Under the Advanced tab, there is a list within the MIME Types box. Add the same mime types to the list as above, typing in first the extension, then the type.
3. Click **Apply**.
4. Restart the application server.

19.2.3 Standalone Transcoder node

The standalone WebSphere Transcoding Publisher node runs on an NT 4 server with SP6a.

Installing and configuring WebSphere Transcoding Publisher

The installation steps for WTP 3.51 on Windows NT can be found in the IBM Redbook *New Capabilities in IBM WebSphere Transcoding Publisher Version 3.5 Extending Web Applications to the Pervasive World*, SG24-6233.

Important: Do not select the LDAP option when the installation asks for it.

To configure WTP as a network proxy, follow the instructions in the IBM Redbook *New Capabilities in IBM WebSphere Transcoding Publisher Version 3.5 Extending Web Applications to the Pervasive World*, SG24-6233.

19.2.4 Web application node with Transcoder

This node runs on AIX. First you must install WAS, then WTP as a servlet filter.

The following products should be installed on this node:

- ▶ DB2 client
- ▶ WebSphere Application Server
- ▶ WebSphere Transcoding Publisher

Installing the DB2 client

The installation steps for the DB2 client on AIX are the following:

1. Install the DB2 V6.1 client with fixpack 4. You can download the fixpack with the client from the following site:
`ftp://ftp.software.ibm.com/ps/products/db2/fixes/english-us/db2aixv61/client/runtime.`

Download the following file: FP4_U471241_rtcInt.tar to a temporary directory.

2. Go to the directory where the file has been downloaded.
3. Untar the file with `tar -xvf FP4_U471241_rtcInt.tar`.
4. Run the setup: `./db2setup`.
5. Check in the DB2 Runtime Client using the Space key. Make sure that the Java Support is checked in under **Customize...**
6. Select **OK**.
7. In the following screen, select **Create a DB2 Instance**.

8. In the next screen, you will find the settings for db2inst1. Set the password to **db2inst1**, then type it again to verify the password.
9. Select **OK**.
10. Select **OK** again.
11. Select **Continue**.
12. A warning window will appear; select **OK**. The DB2 client starts to install.
13. A Notice window will appear; select **OK** to acknowledge the Completed Successfully message.
14. A Status Report window will appear; select **OK**.
15. Click **Close**, then **OK**.

Check to see if the directory, behind the /home/db2inst1/sqllib/java12 link, exists. If not, you must install the Java support for DB2 manually.

1. Go to the directory where the DB2 client has been downloaded and unpacked, then change the directory to db2/aix.
2. Run the **smitty** tool, then install **db2_06_01.jdbc** (Java Support).

The DB2 client has been installed.

The configuration steps for the WAS database connectivity with DB2 for AIX can be found in the WebSphere InfoCenter, at the following URL:
<http://www-4.ibm.com/software/webservers/appserv/doc/v35/ae/infocenter/index.html>

1. Select the **Application Server AE** item on the left side of the navigation bar.
2. Click the **Planning and Installation** item under **Operating Systems...** in the right pane.
3. In the tree, select **Components in a WebSphere environment->Planning and skills for databases->Database skills->Configuring the DB2 Client for UNIX**.
4. Follow the installation steps.

Now the database for WAS running on AIX is ready.

Installing WebSphere Application Server

The installation steps for the DB2 client under AIX can be found at the WebSphere InfoCenter, at the following URL:

<http://www-4.ibm.com/software/webservers/appserv/doc/v35/ae/infocenter/index.html>.

1. Select **Application Server AE** on the left side of the navigation bar.

2. Click **AIX** under **Operating Systems...** in the right pane.
3. Select the **Using IBM HTTP Server and IBM DB2 UDB** link.
4. In the new page, select **Installing WebSphere Application Server 3.5**.
5. Follow the installation steps.

At step 10d, make sure you are using the **jdbc:db2:wasaix** DB URL instead of **jdbc:db2:was**.

After the installation, you should apply Fixpack 4.

1. Download Fixpack 4 to a temporary directory, then unpack it.
2. Stop the WebSphere Application Server and the IBM HTTP Server.
3. Run the script from the fixpack directory: `./install.sh`.

When the installation is completed, Fixpack 4 is applied to WebSphere Application Server.

Installing WTP as a servlet filter

For information about how to install WTP on AIX, see the product documentation on the WTP product CD, under the docs\en directory **tpdgmst.pdf**. Select the **Installing** link from the first page, then go to the **Installing Transcoding Publisher on AIX or Sun Solaris** section. Follow the steps.

After finishing the installation, go to the next step under the **Configuring Transcoding Publisher as a WebSphere filter** section. Select the **trade2_app** application for deploying the filter.

Note: The installation is very similar to the previous WTP installation in “Installing and configuring WebSphere Transcoding Publisher” on page 388, except that WTP is deployed as a servlet filter.

19.2.5 Voice Server node

The WebSphere Voice Server node was not installed in our runtime environment. Instead of the server, the WebSphere Voice SDK was running on the client with the VoiceXML browser, using a speaker and a microphone and simulating the telephony connection.

For information on how to use the WVS SDK, refer to Section 18.7, “WebSphere Voice Server SDK configuration” on page 371.

For more information about the WebSphere Voice Server, refer to the product documentation that comes with the WebSphere Voice Server.

19.3 Deploying the sample application

There is a setup facility for the sample code which will install the Trade2 Web application. There are prerequisites to be satisfied before you install the application.

19.3.1 Prerequisites

The prerequisites for the Trade2 applications are:

- ▶ DB2 6.1 with Fixpack 4
- ▶ DB2 6.1 client
- ▶ WebSphere Application Server with Fixpack 4
- ▶ WebSphere Transcoding Publisher 3.51
- ▶ WebSphere Voice Server 1.5 or WebSphere Voice Server SDK 1.5

At the beginning of this chapter, you can find an explanation of the sample runtime environment installation for the Trade2 application. There are other topologies and configurations on different platforms where the application might run. Of course, we cannot introduce them all, but do make sure that the listed products above are installed and configured correctly in your runtime environment.

19.3.2 NT

You must have administrator privileges to install the sample code under Windows NT.

1. Download the SG246259.ZIP file from the redbook site (<http://www.redbooks.ibm.com>) and follow the Additional Materials link. Create a directory on your local hard drive (for example: c:\weasamp) for the sample code, then unzip the file into the directory.
2. Open a command window.
3. Change the directory to the sample code directory: c:\weasamp.
4. The setup.bat file is in the root directory of the sample. Run the file.
5. The setup sequence will run automatically, when it is done, close the command window.

The setup script copies the necessary files temporarily under WAS, then runs a couple of XML-based configuration scripts. First, it creates the Application server and the other components under WAS, then it deploys the Web application, and finally it starts the Application server. In the end, the script cleans up the temporary files from the WAS directory.

19.3.3 AIX

You must log on as a root to run the installation under AIX (<http://www.redbooks.ibm.com>).

1. Download the SG246259.zip file from the redbook site and follow the Additional Materials link. Create a directory on your local hard drive (for example: /tmp/weasamp) for the sample code, then unzip the file into the directory.
2. Open a terminal window.
3. First you must catalog the trade database. Change the user identity to the database user: su - db2inst1
4. Type in: db2 catalog node trade at node <your database server name>
5. You can try the connection: db2 connect to trade user <user name> using <password>, where username and password are the database user name and password on the database server node, respectively.
6. Change the directory to the sample code directory: /tmp/weasamp.
7. Change setup.sh to be executable, chmod +x setup.sh.

8. The setup.sh file is in the root directory of the sample. Run the setup by entering `./setup.sh`.
9. The setup sequence will run automatically, when it is done **close** the terminal window.

The setup script for AIX does exactly the same thing as the one for Windows NT.

19.3.4 Configuring WTP for the Trade2 application

The Trade2 application requires you to install the HTML-to-VoiceXML transcoder, then to register new device profiles for clients and the XSLs (StyleSheets) for the universal transcoding approach.

HTML-to-VoiceXML transcoder

The installation steps for setting up the HTML-to-VoiceXML transcoder can be found in Section 18.8.1, “Setting up the Voice transcoder” on page 371. Follow the instructions there to install the transcoder in the runtime environment.

Device profile

The sample code distributed with the redbook contains the device profile file ready to register. You can find the files under `c:\weasamp\Wtp\profiles\`. There are two profiles:

- ▶ `voicebrowser.prop`
- ▶ `Waprofit_imode.prop`

To install them under WTP, follow the instructions from Section 18.8.2, “WTP preference profile for voice application” on page 372.

StyleSheets

The StyleSheets are published under WAS; they can be found in the directory `<WAS install directory>\hosts\default_host\trade2_app\web\xml\`. There are three subdirectories:

- ▶ `dtd`
- ▶ `simphtml`
- ▶ `vxml`

To register the StyleSheets with WTP, follow the instructions from Section 18.8.3, “Registering the StyleSheets” on page 374.

Make sure that you are using the runtime environment directories for the StyleSheets, instead of using the VisualAge for Java directories. Change the TradeTemplate <...include...> tags in the simplified HTML XSL files; to do that, refer to Section 15.3.7, “Developing the XSL files and the user interface” on page 300.

The simplified HTML XSL pages have an `<xsl:include ...>` tag in the third line, where the included StyleSheet is defined. There you have to change the path for the included StyleSheet, for example:

```
<WebSphere install
path>\hosts\default_host\trade2_app\web\xml\simhtml\xsl\TradeTemplate.
shtml.xsl
```

Important: This tag cannot handle a relative path, but only an absolute path.

19.4 Entry point for the Trade2 Web application

Handling all the different devices poses a small problem when users want to access the first page of the application: each device requires an entry point which conforms to the device type. In other words, a WAP phone requires a WML page where the Web application can start, an i-mode phone requires a cHTML page, a VoiceXML browser is looking for a VoiceXML document.

There are several solutions to the starting page problem.

- ▶ Each device can have a different base URL; for example, WAP has `wap.mysite.dom`, voice has `voice.mysite.dom`, and so on.
- ▶ Each device can have a separate page for starting; for example, WAP has `www.mysite.dom/index.wml`, voice has `www.mysite.dom/index.vxml`, and so on.
- ▶ Other solutions are based on URL separation.

The Trade2 sample application utilizes a servlet, which is the common entry point for each device. The servlet uses the same technique as the TradeAppServlet to redirect the client to the right starting page based on the User-Agent.

The servlet is called StartPage, and it is deployed under WebSphere Application Server using the following URI: `/trade/index`.

Having the StartPage servlet, each device can access directly the `http://<your server>/trade/index` URL; the servlet will then redirect the device to the right page.

19.5 Testing the application

Finally, when everything is set and the system is running together with the sample application, there must be a test performed to find out if it is working, and how it is working.

For information about the test clients and testing software, please refer to Section 10.3, “Tools for testing the application” on page 182. There you can find a list of the different testing clients, where you can download them, and how to use them.

19.5.1 Test sequence

Here we provide a short test sequence which will guide you through the whole Trade2 Web application. The walkthrough is based on the base sample application for HTML browser running on a desktop PC.

1. Access the site at the very first URL, using the common entry point at `http://<server name>/trade/index`.
2. The first page is a general welcome page; select the **Login** link.
3. The login page asks for a user ID and password (johnd/johnd); fill out the fields then submit the form. The voice application will ask for the user ID and the password (56463/56463), then do the challenge.
4. If the login was successful, the first page will be a personalized welcome page for the user.
5. Access the Account page using the **Account** link, where you can update the account information.
6. Change any of the fields you wish, then submit the form.
7. The Account page reappears with the modified fields.
8. Access the Portfolio page by clicking the **Portfolio** link. The stocks held by the account are listed on the page.
9. Try buying one. Select the first one, modify the amount from 100 to 22, then click **Buy**.
10. The new list of portfolio appears, including a new item at the end with 22 shares.
11. Now try selling stock. Select the first one again then click **Sell**. The updated portfolio will show that the item has been sold.
12. Get quote for a stock. Choose one stock, then click the **Quote** button. The page will show the most recent details of the requested stock.

13. When you are done, log out from the application using the **Logout** link.
14. The very first general welcome page reappears (index page).

19.5.2 Direct

To access the Trade application directly for testing, use the following address:

`http://<server name>/trade/index`

Note: There is no extension after `index`, because the URL calls a servlet, which is mapped to that alias. The original servlet is the `StartPage` class; it receives the first request, checks the User-Agent, then dispatches the request to the appropriate URL.

It is also possible to access the individual pages for each type of device:

- ▶ Base sample: `http://<server name>/trade/index.html`
- ▶ WAP emulator: `http://<server name>/trade/direct/wml/index.wml`
- ▶ WebSphere Voice SDK:
`http://<servername>/trade/direct/voice/index.vxml`

19.5.3 Content transcoding

Here, the client goes through the WebSphere Transcoding Publisher, meaning that either the client has to be configured to use the proxy server, or the proxy server has to run as a reverse proxy.

Wapprofit's i-mode emulator can be found at `http://<WTP server name>/trade/`

Note: In the case of the Wapprofit's i-mode emulator, WebSphere Transcoding Publisher must run as a reverse proxy, because the emulator does not provide proxy settings for the client.

19.5.4 Universal transcoding

In this case, as in the previous case, the client must go through WebSphere Transcoding Publisher. You can set up the proxy settings for the client or use WebSphere Transcoding Publisher as a reverse proxy.

- ▶ Simplified HTML:
`http://<servername>/trade/xml/simhtml/xml/TradeWelcome.xml`
- ▶ WebSphere Voice SDK:
`http://<servername>/trade/xml/vxml/xml/TradeWelcome.xml`

19.6 Notes for other platforms

In this example, the Trade2 application is deployed under Windows NT 4 SP6a and AIX 4.3.2. Since this application is a pure Java Web application, it can be deployed under any platform running WebSphere Application Server v3.5. WebSphere Transcoding Publisher v3.51 is also available on several platforms besides Windows NT and AIX.



Part 6

IBM Web and wireless solutions



Introduction to WebSphere Everyplace Suite

In this chapter, other offerings of WebSphere Everyplace Suite will be introduced, which provide runtime environment components for mobile Web applications.

In this book, only features related to the WebSphere Everyplace Access offering will be discussed. We will try to answer the question of what to do once the WebSphere Everyplace Access offering has been “outgrown.”

For more information about the WebSphere Everyplace Suite, please refer to the IBM Redbook *An Introduction to IBM WebSphere Everyplace Suite Version 1.1 Accessing Web and Enterprise Applications*, SG24-5995.

20.1 What is it for?

WebSphere Everyplace Suite is a comprehensive service solution for mobile service providers. It includes components for subscriber management, data synchronization, notification, and optional wireless gateway.

20.2 Extending capabilities

There are customer demands which are not met with the capabilities of the WebSphere Everyplace Access offering. WebSphere Everyplace Suite offers three different editions for mobile e-businesses, in which WebSphere Everyplace Access represents only the first step in enabling businesses for mobile clients.

Whenever a function is required which is not provided by WebSphere Everyplace Access, it will be found within one of the other two offerings. The next section will introduce the new functions provided by the other editions.

20.2.1 Editions

WebSphere Everyplace Suite boasted three editions at the time this book was written:

- ▶ WebSphere Everyplace Access offering Version 1.1
- ▶ WebSphere Everyplace Server Enable offering Version 1.1
- ▶ WebSphere Everyplace Server Service Provider offering Version 2.1

These editions have different capabilities and features. From one edition to the next, in the order shown above, the services provided become more numerous. WebSphere Everyplace Access is the entry level offering for those who want to implement mobile e-business solutions.

New functions in the WebSphere Everyplace Server Enable offering are:

- ▶ Synchronization services
- ▶ Device management
- ▶ Support for third-party subscriber databases and authentication
- ▶ Transaction queueing

New functions in the WebSphere Everyplace Server Service Provider offering are:

- ▶ Instant messaging
- ▶ Advanced notification

- ▶ Location-based services
- ▶ Subscriber administration and management
- ▶ User and device authentication
- ▶ Encryption and VPN services

20.3 Connectivity services

As mentioned in Section 8.1, “The different modes of pervasive computing” on page 104, there are several modes of communication available for mobile applications. Some of these modes are not provided by the WebSphere Everyplace Access offering.

These services require other elements’ existence in the architecture and also require support from the underlying infrastructure. The two following communication modes are examples of other services provided in a different offering in WebSphere Everyplace Suite.

20.3.1 Notification

In most cases, clients are not connected online to the network, and may not, in fact, even be switched on. The user expects real-time, up-to-date information in certain circumstances. Notification makes the server-to-client direction of the communication possible, where the server pushes the information (with the approval of the client) without requesting it from the client.

This technology requires additional services and elements in the infrastructure.

20.3.2 Asynchronous communication

Messaging and synchronization services between the client and another client or a server is provided through asynchronous communication. It is a different form of communication and requires support for protocols other than the synchronous communication protocols.

20.4 Security

Security is one of the most essential parts of e-business solutions. The WebSphere Everyplace Suite family provides a security infrastructure for the applications. However, the WebSphere Everyplace Access offering does not propose any particular solution to security demands; it relies on the underlying security infrastructure.

Since most e-business solutions require enhanced security, the base network infrastructure or the application tier security usually cannot ensure the required security.

The IBM WebSphere Everyplace Suite is designed to create a safe environment to support pervasive computing. It is designed to have centralized user authentication from limited points of entry. It provides single sign-on for credential sharing across the services hosted by the Suite. The Edge Server relies on a set of industry standard security solutions, such as TLS/SSL and WTLS, to achieve the security objectives for the service domain. The Suite uses the proxy technology in conjunction with a firewall to define the secure boundary for the service domain.

For more information about WebSphere Everyplace Security, please refer to the IBM Redbook *An Introduction to IBM WebSphere Everyplace Suite Version 1.1 Accessing Web and Enterprise Applications*, SG24-5995.

20.5 Summary

The other WebSphere Everyplace Suite offerings provide new services and components for mobile e-business applications. These require additional elements in the infrastructure, either hardware or software components.

The main requirements which the other offerings can satisfy with more or better solutions are:

- ▶ New connectivity services, such as notification and asynchronous communication
- ▶ Advanced security infrastructure above the network layer
- ▶ Centralized application and solution management



Other products

The WebSphere family covers a wide set of products. Nowadays we can find a solution to any of the various IT demands among the WebSphere products.

The products introduced here are just a set of the WebSphere product family, related or close to mobile Web applications.

For more information about the WebSphere product family, please refer to the IBM Web site: <http://www.software.ibm.com/>

21.1 WebSphere Portal Server

The IBM WebSphere Portal Server allows companies to build their own custom portal Web site to serve the needs of employees, business partners and customers. Users can sign on to the portal and receive personalized Web pages providing access to the information, people, and applications they need. This personalized single point of access to all necessary resources reduces information overload, accelerates productivity, and increases Web site usage.

WebSphere Portal Server allows you to:

- ▶ Build multiple types of portals on a single integrated infrastructure based on the WebSphere Portal Architecture
- ▶ Provide a scalable, single point of access for data, people, and applications
- ▶ Deliver an easy to use graphical interface suitable for both occasional and expert users
- ▶ Crawl and categorize intranet and Internet repositories
- ▶ Execute a federated search against all forms of data, structured and unstructured
- ▶ Aggregate and summarize information content for users
- ▶ Customize the look and content of home page displays by user
- ▶ Build rules-based and collaborative filtering personalization
- ▶ Integrate applications and workflow systems into the portal
- ▶ Add collaborative services such as e-mail, shared places, and instant messaging
- ▶ Add pervasive wireless device support for remote and mobile users
- ▶ Provide multiple levels of security and authentication services
- ▶ Acquire syndicated information from over 50,000 databases for news and research
- ▶ Add modules from Independent Software Vendors or custom-developed modules
- ▶ Acquire Web site tools for JSP page building, performance monitoring, caching, etc.
- ▶ Build next generation Web sites with standards such as XML, SOAP, CORBA, and LDAP
- ▶ Manage users as individuals or within groups
- ▶ Access control at the Portlet level
- ▶ Use Portlets to access Lotus and Microsoft Office applications
- ▶ Use WebSphere Personalization as an integrated component
- ▶ Run distributed, heterogeneous searches across disparate data sources
- ▶ Implement a flexible architecture that enables integration with your current Directory, Database, and Security infrastructure

Figure 21-1 on page 407 gives a high-level overview of the WebSphere Portal architecture.

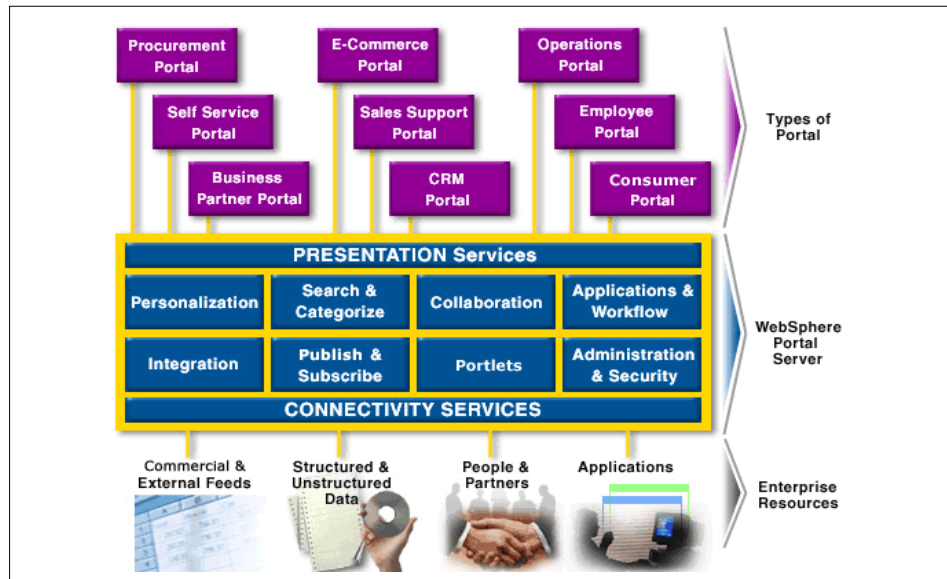


Figure 21-1 WPS architecture

Nowadays, the demand for mobile Web portals is increasing. Mobile Web portals are capable of serving not only desktop browsers, but other mobile devices such as WAP-enabled phones, different PDAs and even phones using voice.

Find more information at <http://www-4.ibm.com/software/webservers/portal/>

21.2 WebSphere Personalization Server

IBM WebSphere Personalization Server for Multiplatforms, Version 3.5 provides users of WebSphere Application Server Advanced and Enterprise Editions and WebSphere Studio Advanced Edition with the capabilities to build an intranet or extranet Web site which delivers Web pages customized to the interests and needs of each site visitor.

Personalization targets Web content to meet a user's needs and preferences. The WebSphere Personalization package contains support for two personalization technologies:

1. Rules-based personalization

For a rules-based personalization solution, the business manager defines a set of business rules that determine what Web content is displayed for a particular user. Developed by IBM, the WebSphere Personalization rules-based technology is tightly integrated into the WebSphere Application Server programming model and scalability architecture. Rules-based personalization is used across a wide spectrum of Web applications, such as employee self-service sites, business partner extranets and customer self-service sites.

2. Collaborative filtering powered by LikeMinds Personalization Server 5.2.1

The collaborative filtering technology employs recommendation engines that use advanced statistical models and other forms of intelligent software to extract trends from the behavior of Web site visitors. This approach adapts to changing trends in visitors' interests without creating new business rules. The Macromedia LikeMinds collaborative filtering technology is highly scalable and can be exploited by WebSphere applications using the LikeMinds Personalization Server Java APIs.

WebSphere Personalization for Multiplatforms includes three main engines:

1. Rules Engine: executes the business rules that determine which content is displayed to each site visitor.
2. Resource Engine: enables Web site owners to optimize their personalization strategy by calling upon content and profile information from multiple sources.
3. Recommendation Engine: uses collaborative filtering to offer content and product recommendations to site visitors, enabling cross-selling and up-selling.

The following schematic diagram (Figure 21-2) depicts the runtime environment for the personalization engine:

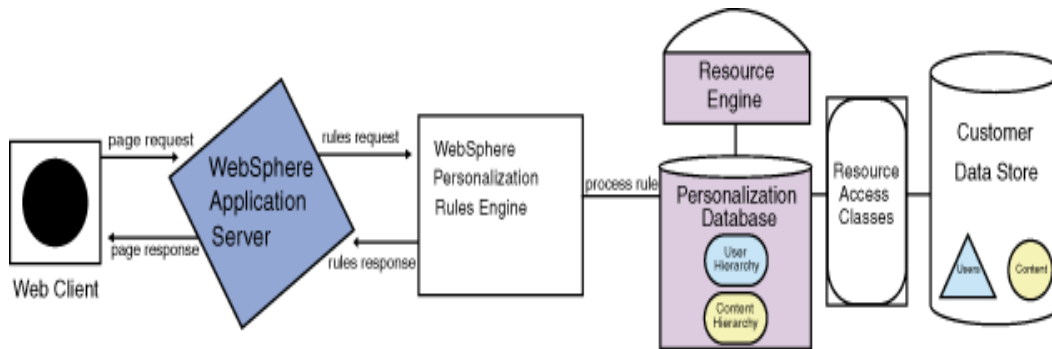


Figure 21-2 Personalization engine runtime environment

Personalization for mobile Web applications has the same purpose and advantages as for any HTML-based Web application. Since the personalization engine operates behind the scenes, any mobile Web application can acquire the result and format it into any type of content.

21.3 SecureWay Wireless Gateway

IBM SecureWay Wireless Gateway and Client extend Internet Protocol (IP) connectivity across a diverse set of wireless and wireline networks to enable TCP/IP applications to seamlessly access enterprise networks. SecureWay Wireless Gateway and Client, sometimes referred to as *middleware*, are components of SecureWay Wireless Software. These components work together to provide security for communications and improve the performance of mobile computing applications. The SecureWay Wireless Client supports the mobile worker's need for real-time access to the same information that is available in the office.

SecureWay Wireless Client resides on a user's mobile PC or handheld device and communicates with the Wireless Gateway over any IP-based connection, including wireless or dial-up networks. SecureWay Wireless Gateway integrates the mobile networks and provides the connection to the enterprise network.

Mobile users can use the same Wireless Gateway and access the same enterprise applications. SecureWay Wireless Gateway also enables mobile users to connect to multiple applications, as they can do using desktop computers within the enterprise network.

Companies can support multiple networks, which enables mobile users to use the network that best meets their individual needs and cost objectives.

SecureWay supports many worldwide protocols, including DataTac, Mobitex, AMPS, GSM, PSTN, Satellite, and Japanese networks. It supports applications using industry-standard protocol sockets, so users do not need to learn special interfaces or proprietary tools and protocols.

This comprehensive network access security solution features bidirectional user authentication and data encryption.

Find more information at

<http://www-3.ibm.com/pvc/products/secureway/index.shtml>

21.4 WebSphere Translation Server

The IBM WebSphere Translation Server for Multiplatforms is a machine translation (MT) offering that can help companies remove language barriers to global communication and e-commerce. WebSphere Translation Server (WTS) enables enterprises to provide Web pages, e-mail messages and chat conversations in multiple languages and in real time. Specifically designed for enterprise use, the WebSphere Translation Server allows companies to use their existing Web infrastructure to provide content to users in their native language, at a fraction of the cost of professional translation.

Based on IBM machine translation technology, WebSphere Translation Server is designed for scalability on multiple platforms.

IBM WebSphere Translation Server for Multiplatforms Version 1.0 consists of:

- ▶ MT engines for translating text from one language into another
- ▶ User Dictionary Manager tools, that allow specific words to be added to a domain
- ▶ WebSphere Translation Server, which makes the engines available for Web page translation on the fly
- ▶ HTTP server support, providing interfaces for WebSphere, Domino, Netscape, and Microsoft IIS

Supported languages include the following:

- ▶ English-to-French, French-to-English
- ▶ English-to-Italian, Italian-to-English
- ▶ English-to-German, German-to-English
- ▶ English-to-Spanish, Spanish-to-English

- ▶ English-to-Chinese (simplified)
- ▶ English-to-Chinese (traditional)
- ▶ English-to-Japanese
- ▶ English-to-Korean

IBM has expanded WebSphere Translation Server to incorporate bundled regional translation software licenses and services in the WebSphere Translation offering.

Find more information at

http://www-4.ibm.com/software/speech/enterprise/ep_8.html

21.5 Where to find more information

- ▶ For IBM software products, please refer to the following URL:
<http://www.software.ibm.com>



Part 7

Appendixes



A

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG246259>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select **Additional materials** and open the directory that corresponds to the redbook form number, SG246259.

Using the Web material

The additional Web material that accompanies this redbook includes the following file:

<i>File name</i>	<i>Description</i>
SG246259.zip	The Trade2 mobile enabled sample application

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	10 MB minimum
Operating System:	Windows NT or AIX
Processor:	500MHz or higher
Memory:	512 MB or higher

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder. The contents of the unzipped file are used in many of the examples throughout this redbook.

Refer to Chapter 18, “Development environment for the sample application” on page 355 and Chapter 19, “Runtime environment for the sample application” on page 383 for instructions on setting up the sample application for the development environment and for the runtime environment.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 419.

- ▶ *Mobile Commerce Solutions Guide using WebSphere Commerce Suite V5.1*, SG24-6171
- ▶ *New Capabilities in IBM WebSphere Transcoding Publisher Version 3.5 Extending Web Applications to the Pervasive World*, SG24-6233
- ▶ *Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*, SG24-5755
- ▶ *The Front of IBM WebSphere Building e-business User Interfaces*, SG24-5488
- ▶ *Programming with VisualAge for Java Version 3.5*, SG24-5264
- ▶ *How about Version 3.5? VisualAge for Java and WebSphere Studio Provide Great New Function*, SG24-6131
- ▶ *WebSphere Personalization Solutions Guide*, SG24-6214
- ▶ *Version 3.5 Self Study Guide: VisualAge for Java and WebSphere Studio*, SG24-6136
- ▶ *The XML Files: Using XML and XSL with IBM WebSphere V3.0*, SG24-5479
- ▶ *Enterprise JavaBeans Development Using VisualAge for Java*, SG24-5429
- ▶ *WebSphere V3.5 Handbook*, SG24-6161
- ▶ *Extending e-Business to Pervasive Computing Devices Using IBM WebSphere Everyplace Suite Version 1.1.2*, SG24-5996
- ▶ *An introduction to IBM WebSphere Everyplace Suite Version 1.1 Accessing Web and Enterprise Applications*, SG24-5995

Other resources

These publications are also relevant as further information sources:

- ▶ Avrahm Leff and James T. Rayfried, *IBM Research Report Web-Application Development Using Model/View/Controller Design Pattern*
- ▶ E.Gamma, R.Helm, R.Johnson and J.Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Pub.Co., ISBN: 0201633612
- ▶ D.Gibbon, I. Mertins and R.Moore, *Handbook of Multimodal and Spoken Dialogue Systems: Resources, Terminology and Product Evaluation*, Kluwer Academic Publishers, ISBN: 0792379047.
- ▶ B. Balentine, D. P. Morgan and W. Meisel, *How to Build a Speech Recognition Application*, Enterprise Integration Group, Inc., ISBN: 0967127815.
- ▶ M. Schroeder (ed.), *Speech and Speaker Recognition*, S.Karger Publishing, ISBN: 3805540124.
- ▶ A. Waibel and K.-F. Lee (ed.), *Readings in Speech Recognition*, Morgan Kaufmann Publishers, ISBN: 1558601244.

Referenced Web sites

These Web sites are also relevant as further information sources:

- ▶ <http://www.software.ibm.com/> IBM's Software Web site
- ▶ <http://www.openwave.com> Openwave's Web site
- ▶ <http://www.syncml.com> SyncML specification Web site
- ▶ <http://www.voicexml.org> VoiceXML specification Web site
- ▶ <http://www-4.ibm.com/software/webservers/transcoding/> WebSphere Transcoding Publisher Web site
- ▶ <http://java.sun.com/docs/books/tutorial/java/concepts/object.html> Object oriented programming concepts in Java
- ▶ <http://java.sun.com/products/ejb/> Sun's EJB Web site
- ▶ <http://java.sun.com/products/javabeans/> Sun's JavaBeans Web site
- ▶ <http://www.w3.org/XML> W3C's Web site about XML
- ▶ <http://www.w3.org/Style/XSL> W3C's Web site about XSL
- ▶ <http://www.w3.org/TR/xslt> W3C's Web site about XSLT
- ▶ <http://www.w3.org/Markup/> W3C's Web site about markup languages
- ▶ <http://xml.apache.org> Apache's Web site, XML related projects

- ▶ <http://www.w3.org/TR/1998/NOTE-compactHTML-19980209/> W3C's Web site about compact HTML
- ▶ <http://www.w3.org/DOM> W3C's Web site about DOM
- ▶ <http://www.nttdocomo.com/i/> NTT DoCoMo's i-mode Web site
- ▶ <http://www.wapforum.org> Wapforum Web sites
- ▶ <http://voicexml.org> VoiceXML Web site
- ▶ <http://www.xmlspy.com> XML Spy's Web site
- ▶ <http://www.alphaworks.ibm.com> IBM's Open Source development site
- ▶ <http://www-4.ibm.com/software/speech/> IBM's speech products site
- ▶ <http://lynx.browser.org/> Lynx Web browser's site
- ▶ <http://www.opera.com> Opera Web browser's site
- ▶ <http://www.nokia.com/corporate/wap/sdk.html> Nokia's WAP SDK site
- ▶ <http://developer.openwave.com> Openwave's developer site
- ▶ <http://www.waprofit.com> Waprofit's Web site
- ▶ <http://www.ietf.org> The Internet Engineering Task Force's Web site
- ▶ http://www.rsa.com/rsalabs/rsa_algorithm/index.html RSA's Web site about the RSA algorithm
- ▶ <http://www.certicom.com/research/wecc3.html> Certicom's Web site about ECC

How to get IBM Redbooks

Search for additional Redbooks or redpieces, view, download, or order hardcopy from the Redbooks Web site:

ibm.com/redbooks

Also 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.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Special notices

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 other companies:

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.

Other company, product, and service names may be trademarks or service marks of others.

Glossary

AMPS. Advanced Mobile Phone Services. A common analog cellular telephone service standard.

Applet. A Java applet is a small application program that is downloaded to and executed on a Web browser or network computer. A Java applet typically performs the type of operations that client code would perform in a client/server architecture. It edits input, controls the screen, and communicates transactions to a server, which in turn performs the data or database operations.

API. Application Program Interface.

Bean Managed Persistence. Bean Managed Persistence (BMP) is a term used to describe a type of entity Enterprise JavaBean where the bean developer specifies how the bean is to be persisted to a database by writing Java code in the appropriate methods to perform the tasks required.

Bluetooth. A short range (10 to 100 m.) wireless radio transport.

Cache. A cache stores cachable responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests. Any client or server may include a cache, though a cache cannot be used by a server while it is acting as a tunnel.

Cache server. Some networks use a cache server to store Web pages and other data, so that if the same pages are requested frequently, they can be served from the cache rather than repeatedly retrieved from external Web servers. The external cache is an HTTP proxy such as IBM Web Traffic Express. IBM WebSphere Transcoding Publisher can use it to store and retrieve transcoded Web pages and intermediate results to avoid repeating the transcoding of frequently accessed pages, delivering better performance.

CDMA. Code Division Multiple Access. A second generation digital cellular network standard.

CDPD. Cellular Digital Packet Data. Designed to work as an overlay on analog cellular networks.

CGI. Common Gateway Interface. A standard way of communicating between different processes.

Cell phone. CELLular telePHONE is the first ubiquitous wireless telephone. Originally analog, all new cellular systems are now digital. This has enabled the cell phone to turn into a smart phone that has access to the Internet.

Clustering. Clustering is a technique used to provide scalability through the use of multiple copies of an application on the same machine or on separate machines. Careful management of the different applications is necessary to ensure that they work together effectively. WebSphere has limited clustering support in Version 2.x and more support in Version 3.0.

cHTML. Compact HTML is a more efficient variation of HTML specifically designed for use by the i-mode wireless service.

Container Managed Persistence. Container Managed Persistence (CMP) is a term used to describe a type of entity Enterprise JavaBean where the code to persist the bean to a database is generated at deployment time by the EJB container.

Dictionary. In this context the word 'vocabulary' can be used as well, although it would generally have a slightly different meaning. In the book we will use both words to refer to the set of words that a speech recognition engine is capable of identifying.

e-business. e-business is a term used by IBM to describe the use of Internet technologies to transform business processes. What this means in practice is using Internet clients such as Web browsers as front ends for applications that access back-end legacy systems to allow greater access. See <http://www.software.ibm.com/ebusiness> for more information.

Enterprise Java Beans. Despite the name, Enterprise Java Beans (EJBs) are not Java Beans. Enterprise Java Beans are server-side Java components that are designed for distributed environments. They do not exist in isolation but rather are deployed in containers that provide services such as security, naming and directory services, and persistent storage. WebSphere Application Server is just such a container. See <http://java.sun.com/products/ejb/> for more information.

EPOC. A 32-bit operating system for handheld devices from Symbian Ltd. Used in Psion and other handheld computers, it supports Java applications, e-mail, fax, infrared exchange, data synchronization with PCs and includes a suite of PIM and productivity applications. See <http://www.symbian.com> for more information.

ESS. Enterprise Solution Structure defines a set of technical reference architectures that are included in SIMethod.

FIR/IIR. Finite and Infinite Impulse Response filters, respectively.

Fricatives (or fricated sound). Sounds produced by light contractions of the mouth, lips and tongue, which forces the air through narrow gaps producing audible friction.

Gateway. A server which acts as an intermediary for some other server. Unlike a proxy, a gateway receives requests as if it were the origin server for the requested resource; the requesting client may not be aware that it is communicating with a gateway. Gateways are often used as server-side portals through network firewalls and as protocol translators for access to resources stored on non-HTTP systems.

GPRS. General Packet Radio Service or GPRS is an enhancement to the GSM mobile communications system that supports data packets. GPRS enables continuous flow of IP data packets over the system for such applications as Web browsing and file transfer. GPRS differs from GSM's short messaging service (GSM-SMS) which is limited to messages of 160 bytes in length.

GSM. Global System for Mobile Communications is a digital cellular phone technology based on TDMA that is the predominant system in Europe, but is also used around the world. Developed in the 1980s, GSM was first deployed in seven European countries in 1992. Operating in the 900 MHz and 1.8 GHz bands in Europe and the 1.9 GHz PCS band in the U.S., GSM defines the entire cellular system, not just the air interface (TDMA, CDMA, etc.). As of 2000, there were more than 250 million GSM users, which is more than half of the world's mobile phone population.

Grapheme. The smallest part of written language that can be analyzed. It consists of one or more symbols used to represent a phoneme.

GPS. Geographical Positioning System.

Grammar. a set of rules to determine valid concatenations of phonemes or words. In VoiceXML, a grammar also includes the set of recognizable words for an ASR engine.

Hidden Markov chain. A computational model where the transition between two states A and B is measured by the probability of reaching the state B from the state A, that is $p(B|A)$. The states are not directly observable (hence, hidden) but each produces observable outputs with a given probability.

HDML. Handheld Device Markup Language is a specialized version of HTML designed to enable wireless pagers, cell phones, mobile phones and other handheld devices to obtain information from Web pages. HDML was developed by Phone.com (formerly Unwired Planet) before the WAP specification was standardized. It is a subset of WAP with some features that were not included in

WAP. AT&T Wireless launched the first HDML-based service in 1996.

HTML. Hyper Text Markup Language is a document format used on the World Wide Web. Web pages are built with HTML tags, or codes, embedded in the text. HTML defines the page layout, fonts and graphic elements as well as the hypertext links to other documents on the Web. Each link contains the URL, or address, of a Web page residing on the same server or any server worldwide, hence the term “World Wide” Web.

HTTP proxy. An HTTP proxy is a program that acts as an intermediary between a client and a server. It receives requests from clients, and forwards those requests to the intended servers. The responses pass back through it in the same way. Thus, a proxy has functions of both a client and a server. Proxies are commonly used in firewalls, caching and transcoding machines.

IBM WebSphere Transcoding Publisher. IBM WebSphere Transcoding Publisher is network software that modifies content presented to users based on the information associated with the request, such as device constraints, network constraints, user preferences, and organizational policies. Transforming content can reduce or eliminate the need to maintain multiple versions of data or applications for different device types and network service levels.

Image Transcoder. Image Transcoder is a transcoder that can scale, modify quality, and modify color levels in JPEG and GIF images. Additionally, the Image Transcoder can convert JPEGs to GIFs for devices that do not render JPEGs.

i-mode. A packet-based information service for mobile phones from NTT DoCoMo (Japan). i-mode provides Web browsing, e-mail, a calendar, chat rooms, games, and customized news. It was the first smart phone system for Web browsing and its popularity grew very quickly after its introduction in 1999. i-mode is a proprietary system that uses a subset of HTML, known as cHTML, in contrast to

the global WAP standard that uses a variation of HTML, known as WML. The i-mode transfer rate is 9600 bps, but is expected to increase to 384 kbps in 2001, using W-CDMA.

IrDA. The Infrared Data Association develops standards for wireless, infrared transmission systems between computers. With IrDA ports, a laptop or PDA can exchange data with a desktop computer or use a printer without a cable connection. IrDA requires line-of-sight transmission like a TV remote control. IrDA products began to appear in 1995. See <http://www.irda.org> for more information.

JavaBeans. JavaBeans are Java components designed to be used on client systems. They are Java classes that conform to certain coding standards. They can be described in terms of their properties, methods and events. JavaBeans may be packaged with a special descriptor class called a BeanInfo class and special property editor classes in a JAR file. Java Beans may or may not be visual components. See <http://www.javasoft.com/beans/docs> for more information.

JavaServer Pages (JSP) . JSPs provide a simplified, fast way to create dynamic Web content. JSP technology enables rapid development of Web-based applications that are server and platform independent. JavaServer Pages are compiled into servlets before deployment.

JDBC. JDBC is a Java API that allows Java programs to communicate with different database management systems in a platform-independent manner. Database vendors provide JDBC drivers with their platforms that implement the API for their database, allowing the Java developer to write applications to a consistent API no matter which database is used.

JNDI. Java Naming and Directory Interface (JNDI) is an API that allows Java programs to interface and query naming and directory services in order to find information about network resources. JNDI is used in WebSphere to provide a directory of Enterprise Java Beans.

See <http://java.sun.com/products/jndi/index.html> for more information.

JSP. See JavaServer Pages.

Language model. A mathematical description of language-dependent features.

m-commerce. Mobile commerce refers to the use of mobile devices to partially or completely perform a transaction electronically from a commerce Web site for the exchange of goods or services for monetary consideration. Simply put, m-commerce is electronic commerce using a mobile device such as a mobile phone or PDA.

Mobile device. A mobile device is a portable, generally small, wireless device that can be used to access the Internet via a browser. It includes a wide range of capability and functionality. Mobile devices include mobile phones, wireless PDAs, and wireless laptops.

Mobile phone. A mobile phone is a wireless smart phone that has a microbrowser to access Internet content. Other names for a mobile phone include cell phone and wireless phone.

MQe. See MQ Everyplace.

MQ Everyplace. It is designed to satisfy the messaging needs of lightweight devices and the requirements that arise from the use of fragile communication networks.

PDA. Personal Digital Assistant.

PCS. Personal Communications Services (PCS) are wireless services that emerged after the U.S. Government auctioned commercial licenses in 1994 and 1995. This radio spectrum in the 1.8-2 GHz range is typically used for digital cellular transmission that competes with analog and digital services in the 800 MHz and 900 MHz bands.

Persistence. Persistence is a term used to describe the storage of objects in a database to allow them to persist over time rather than being destroyed when the application containing them terminates. Enterprise Java Bean containers such as WebSphere provide persistence services for EJBs deployed within them.

Phoneme. The smallest linguistic unit that convey a meaning distinction between words. Hence, words can be split in units of individual sounds, each of which is a phoneme.

PKI. Public Key Infrastructure.

PvC. Popular short form within IBM for pervasive computing.

Proxy. Transcoding Publisher connects through a proxy server that is configured with a firewall to manage network traffic and to protect your network from outside intrusion.

Pulse. A signal of short duration.

Push. Push refers to a technology that sends data to a program without the program's request (unsolicited).

RMI. Remote Method Invocation (RMI) is a lightweight distributed object protocol that allows Java objects to call each other across a network. RMI is part of the core Java specification. See <http://java.sun.com/products/jdk/rmi/index.html> for more information.

Scalability. Scalability is an abstract attribute of software that refers to its ability to handle increased data throughput without modification. WebSphere handles scalability by allowing execution on a variety of hardware platforms that allow increased performance and clustering.

Servlets. Servlets are Java classes that run on Web servers to provide dynamic HTML content to clients. The servlets take as input the HTTP request from the client and output dynamically generated HTML. For more information, see <http://www.software.ibm.com/ebusiness/pm.html#Servlets>.

SMS. Short Message Service or SMS is a text message service that enables short messages of generally no more than 140-160 characters in length to be sent and transmitted from a cell phone. SMS is supported by GSM and other mobile communications systems. Unlike paging, short messages are stored and forwarded in SMS centers.

SOCKS. A SOCKS server is a proxy server that uses a special protocol, sockets, to forward

requests. Transcoding Publisher connects through a SOCKS server that is configured with a firewall to manage network traffic and to protect your network from outside intrusion (it supports Versions 4 and 5 SOCKS servers).

SSL. Secure Sockets Layer. A secure protocol used for authentication and encryption. SSL can be used over HTTP, RMI, Telnet and other protocols.

Stand-alone Network Proxy. When the IBM WebSphere Transcoding Publisher is used as a normal proxy in a browser, the data that flows from the original source will be transcoded in the proxy according to the device and network profile needed.

StyleSheet Transcoder. StyleSheet Transcoder is a transcoder that selects the style sheet and applies it to an input Extensible Markup Language (XML) document to produce a version that is appropriate for the target device.

TCP/IP. TCP/IP is a set of protocols developed to allow cooperating computers to share resources across a network.

TDMA. Time Division Multiple Access. A second-generation digital cellular network standard.

Text Transcoder. Text Transcoder is a transcoder that can modify elements of a text document based on device, network and, potentially, user preference information. The primary use of this Text Transcoder is to modify Hypertext Markup Language (HTML) documents to remove unsupported elements, reduce space usage, replace features such as images or frames with links, and otherwise tailor documents to allow for their better display on devices with screen limitations.

TLS. Transport Layer Security. The standard (IETF) security protocol on the Internet. It is expected to eventually supersede SSL.

Transcoder. A transcoder is a program that modifies the content of a document.

Transcoding. Transcoding is a new technology that gives you the ability to make Web-based information available on handheld and other new type devices economically and efficiently, or on

the slow network connections like a dial up modem connection. With transcoding, users receive information (text and images) tailored to the capabilities of the devices they are using and also tailored to the capacity of the network being used.

Transcoding is also the process whereby the MEGs modify the request and generate the original resource and all of the document (or resource) editing (or transcoding).

Tunnel. A tunnel is an intermediary program which acts as a blind relay between two connections.

UMTS. Universal Mobile Telecommunications System is the European implementation of the 3G wireless phone system. UMTS, which is part of IMT-2000, provides service in the 2 GHz band and offers global roaming and personalized features. Designed as an evolutionary system for GSM network operators, multimedia data rates of up to 2 Mbps are expected using the W-CDMA technology. In the meantime, GPRS and EDGE are interim steps that will speed up wireless data for GSM. For more information, visit <http://www.umts-forum.org>.

Understanding. In this context, we refer to the possibility of identifying idiomatic expressions or sentences where the meaning is affected by the feelings of the speaker.

URL. Uniform Resource Locator. The URL specifies the Internet address of a file stored on a host computer connected to the Internet.

Voiced sound. A sound is said to be voiced when vocal cords vibration is involved. Voiced sounds are usually vowels, semi-vowels and nasals.

VXML. Voice XML is an extension of XML that defines voice segments and enables access to the Internet via telephones and other voice-activated devices. AT&T, Lucent and Motorola created the Voice XML Forum to support this development. For more information, visit <http://www.vxml.org>.

Voice XML. See VXML.

VRU. A common synonym for VRU is Interactive Voice Response, which historically refers to DTMF applications.

WAP. Wireless Application Protocol. The point of this standard is to serve Internet contents and Internet services to wireless clients and WAP devices, such as mobile phones and terminals. The authoritative source for WAP is <http://www.wapforum.org>.

Waveform. A graphical representation of periodic signals.

Web application servers. A Web application server is a software program designed to manage applications at the second tier of three-tier computing, that is, the business logic components. A Web application server manages applications that use data from back-end systems, such as databases and transaction systems, and provides output to a Web browser on a client. For more information see <http://www.software.ibm.com/ebusiness/appsrsvw.html>

Web browser. To access the World Wide Web, you must use a Web browser. A browser is a software program that allows users to access and navigate the World Wide Web.

Wireless network. Used to transmit data between wireless devices such as a mobile phone, PDA, or personal computer without the use of a physical cable or wire.

Wireless service provider. An organization that provides wireless services, including cellular services, satellite services and ISPs.

Wireless LAN. A wireless LAN is a local area network that transmits over the air, typically using an unlicensed frequency such as the 2.4 GHz band. A wireless LAN does not require lining up devices for line of sight transmission, as IrDA does. Wireless access points (base stations) are connected to an Ethernet hub or server and transmit a radio frequency over an area of several hundred to a 1000 feet, which can penetrate walls and other non-metal barriers. Roaming users can be handed off from one access point to another like a cellular phone system. Laptops use wireless

modems that plug into an existing Ethernet port or that are self contained on PC cards, while stand-alone desktops and servers use plug-in cards (ISA, PCI, etc.).

WLP. A modified version of the Point-to-Point Protocol (PPP) used by the IBM Wireless Gateway to support wireless (non-WAP) client devices.

WML. Wireless Markup Language. XML-based, WML tags are used to mark up content in decks for WAP-enabled devices.

WTE. Web Traffic Express. An IBM caching proxy.

WTLS. Wireless Transport Layer Security. A simplified version of TLS designed specifically for WAP devices. It uses mini-certificates.

WTP. WebSphere Transcoding Publisher.

WWW. The World Wide Web (known as the Web) is a system of Internet servers that supports hypertext to access several Internet protocols on a single interface.

X.509. A digital certificate specification used by SSL and TLS. Mini-certificates are used by WTLS.

XML. XML, or Extensible Markup Language, is a platform-independent and application-independent way of describing data using tags. XML (a subset of SGML) is similar to HTML in that it uses tags to describe document elements, but different in that the tags describe the structure of the data rather than how the data is to be presented to a client. XML has the ability to allow data providers to define new tags as needed to better describe the data domain being represented. For more information see <http://www.software.ibm.com/xml>.

XSL. Extensible Style Language. XSL stylesheets are documents that describe a mapping between XML documents and visual data that can be presented to a client in a browser or mini-browser.

Index

Symbols

“Wizard-of-Oz” technique 318

Numerics

4thpass Kbrowser 25

A

Access Beans 246

Access Integration pattern 69

Advanced notification 402

AMPS 15

annotators 271

Application Integration patterns 69

Application patterns 74

application server node 81, 84, 85

Articulation based synthesis 52

asynchronous 105

Asynchronous communication 403

Automatic Speech Recognition (ASR) 36

AvantGo 25

B

back-end 242

barge-in 326

Bean Management Persistence 158

Behavior 148

Behavioral patterns 147

Bluetooth 17

BMP 158

Business patterns 67

C

card 257

Cascading StyleSheets 161

cdmaOne 17

Cellular Digital Packet Data (CDPD) 15

cellular radio networks 11

CGI 130

cHTML 163, 290

circuit-switched 11

Classification and Regression Trees (CART) 50

CMP 158

Code Division Multiple Access (CDMA) 17

Collaboration business pattern 68

Collaborative filtering 408

Command pattern 149

Compact HTML 163

Compaq iPaq 26

Composite actions 320

Composite patterns 70

Computer Telephony Interface (CTI) 58

Computing modes 104

Concatenation based synthesis 52

Condition for Criteria Matching HTTP header 378

Container Management Persistence 158

Content transcoding 345

Continuous speech recognition 38

Controller 127, 139

Correct action 47

Correct rejection 45

Creational patterns 147

Customer Relationship Management (CRM) 34

D

database server node 82

DB2 Client 385

DB2 Server 385

DCS1900 13

Decision points 108

decision tree 107

Deletion 45

Design patterns 146

device class 107

Device management 402

device profiles 114

DHTML 80

Dialog Management (DM) 43

Dictation 38

Dictionaries 40

direct 256

Direct Spread - Code Division Multiple Access (DS-CDMA) 16

directory and security services node 82, 84

directory dialer 56

Discrete speech recognition 37
DMZ 81, 83, 84, 85
Document Style Semantics and Specification Language 161
Document Type Definition (DTD) 159
DOM object 284
domain firewall 82, 83
Dual Tone Multi Frequency (DTMF) 34

E

EdgeMatrix WAPman 25
EDI (Electronic Data Interchange) 160
Editor agents 135
Editors 142
Emulator 29
Encryption and VPN services 403
Enterprise Java Beans (EJBs) 157
Enterprise JavaBeans 128
Entity Beans 158
EPOC 27
Ericsson 27
Existing 109
Extended Enterprise business pattern 68
Extensible HTML 164
eXtensible Markup Language 159
Extensible Stylesheet Language 160
Extensible Stylesheet Language Transformation 161
Ezos's EzWAP 27

F

Factory pattern 150
False rejection 45
filter 98
filter servlet 384
footer 251
forms 256
Forward-proxy 99
front-end 242

G

Gateway 88
General Packet Radio Service(GPRS) 14
Generator agents 135
Grammars 40
GSM 13

H

Handheld Device Markup Language 163
Handspring Blazer 24
HDML 163
header 251
Hewlett-Packard Jornada 26
Hidden Markov Models (HMMs) 41
Hosting Service Providers 6
HTML 162
HTML-to-VoiceXML transcoder 328
HTTP 134
HTTP protocol 18
hybrid 295, 340
Hypertext Markup Language 162
Hyper-Text Transfer Protocol 134

I

IBM framework and Open Standards 67
IBM HTTP Server 96
IBM Universal Messaging 58
IBM ViaVoice 54
IBM WebSphere Voice Server for DirectTalk 310
IBM WebSphere Voice Server SDK 310
IBM WebSphere Voice Server with ViaVoice Technology 310
Identity 148
IEEE 802.11b 17
IETF (Internet Engineering Task Force) 217
i-mode 19
Information Aggregation business pattern 68
Infrared 10
Input DTD 305, 378
Insertion 45
Instant messaging 402
Integration patterns 69
Intellisync Browse-it 24
International Mobile Equipment Identifier 14
International Mobile Telecommunications-2000 (IMT-2000) 16
Internet Service Providers 30
Inter-word rejection 45

J

JavaBeans 156
JavaServer Pages 128
JavaServer Pages (JSPs) 154
J-PHONE 19

K

KDDI 19

L

load balancer node 81
Locaton based services 403
Loose coupling 349

M

machine translation (MT) 410
MEG (Monitor, Editor, Generator) 135
MIME types 256, 366
Mobile applications 4
Mobile Service Providers 6
Mobile Services (M-services) 19
Mobitex 14
Model 127
Model-View-Controller (MVC) 122
Monitor agents 134
Multi Carrier - Code Division Multiple Access (MS-CDMA) 17
Multi-modal 107
multimodal applications 346
multimodal Web applications 348

N

Natural Language Understanding (NLU) 42
navigation bar 251
neural networks (NNs) 42
Nokia 27
Nokia WAP toolkit 263
Non-existing 109
non-IP network 86
Non-rewriteable 110
non-visual 345
Notification 403
notification 105
NTT DoCoMo 19

O

object 148
object-oriented components 146
Official Site 20
OmniSky OmniSky 25
Out-of-dictionary/out-of-grammar rejection 45

P

packet-switched 11
Palm 23
 HTTP browser 24
 HTTP browsers
 OmniSky 25
 Qualcomm EudoraWeb 25
 WAP browsers
 4thpass Kbrowser 25
 EdgeMatrix WAPman 25
 Web Clipping
 browser 26
PALM OS 23
PAP (Push Access Protocol) 351
Pattern matching algorithm 41
Patterns for e-business 66
PCT (Private Communication Transport) 217
Perfect word match 46
Personal Communications Services (PCS) 13
Personal Digital Assistant (PDA) 23
Personal Digital Cellular (PDC) 16
Platform 90
PocketIE 26
PocketPC 26
 WAP browser 27
protocol firewall 82
proxy 384
Proxy pattern 151
PSION 27
Public Key Cryptography 218
Publishing Stages 357

Q

Qualcomm EudoraWeb 25

R

Recommendation Engine 408
Redbooks Web site 419
 Contact us xvi
request headers 305
Resource Engine 408
Reverse-proxy 99
Rewriteable 110
RSA 218
Rule based synthesis 52
Rules Engine 408
Rules-based personalization 408
Runtime pattern 80

S

- scenarios 242
- Secureway Firewall 97
- SecureWay Wireless Gateway 409
- Security 403
- Self-Service business pattern 68
- separate actions 320
- sequential applications 350
- Service Level Characteristics 80
- Servlet 128, 154
- session beans 158
- Short Message Services (SMS) 105
- side bar 251
- simplified HTML 335
- Simulator 29
- simultaneous applications 350
- SMS 352
- Speaker dependent system 37
- Speaker independent system 37
- spectral features 41
- speech-to-text 36
- SSL 80, 217
- State 148
- stateful session bean 158
- Stateless session beans 158
- Structural patterns 147
- StyleSheet 294
- Subscriber administration and management 403
- Subscriber Identity Module (SIM) 13
- Substitution 46
- Synchronization 10
- Synchronization services 402
- synchronous mode 104
- SyncML 106

T

- Template pattern 149
- text document 284
- Tight coupling 348
- third-party subscriber databases and authentication 402
- three tier servlet architecture 243
- TLS (Transport Layer Security) 217
- Trade2 242
- Trade2 Web application 249
- Transaction queueing 402
- transcoder friendly 290
- Tweaking 113

U

- UML activity diagram 260
- Universal Mobile Telecommunications System (UMTS) 16
- universal transcoding 335, 344
- UP.browser 263
- User and device authentication 403
- User interface (UI) 244
- user node 80
- Utterances 39

V

- View 128, 140
- visual 345
- Visual Age for Java 8
- VIWO (Voice In WAP Out) 350
- Voice applications 106
- Voice Extensible Markup Language 164
- VoiceXML 164
- VoiceXML forms 323
- VoIP gateway 310
- Voluntary Site 20

W

- WAP 256
 - browser 25
- WAP 1.2 Push 352
- WAP gateway 351
- WAP push 105
- waveform 41
- WBI API v4.5 285
- Web application 365
- Web application server node 81, 83, 84, 85
- Web Clipping 20
 - proxy server 26
- Web Clipping Application 20
- Web Clipping proxy 20
- Web Intermediaries 115, 132, 269
- Web server node 81, 84, 85
- Web server redirector node 81
- WebSphere Application Server 7, 95
- WebSphere Edge Server 96
- WebSphere Everyplace Access Offering Version 1.1 402
- WebSphere Everyplace Access V1R1 Offering 5
- WebSphere Everyplace Server Enable Offering Version 1.1 402
- WebSphere Everyplace Server Service Provider Of-

- fering Version 2.1 402
- WebSphere Everyplace Suite 402
- WebSphere Personalization Server 407
- WebSphere Portal Architecture 406
- WebSphere Portal Server 406
- WebSphere Studio 8
- WebSphere Test Environment (WTE) 365
- WebSphere Transcoding Publisher 7, 95
- WebSphere Translation Server 410
- WebSphere Voice Server 7, 95
- Windows CE (WinCE) 26
- Wireless Application Protocol (WAP) 18
- wireless LAN 10, 17
- Wireless Markup Language 163
- Wireless networks
 - generations 12
- Wireless Personal Area Networks 10
- Wireless Service Providers 6
- WML 163
- WTLS 218

X

- XHTML 164
- XML 159, 293
- XSL 160, 293
- XSLT 161, 293



Mobile Applications with IBM WebSphere Everyplace Access

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



Mobile Applications with IBM WebSphere Everyplace Access Design and Development

Mobile Web applications using Application Server and Transcoding Publisher

Voice applications using Voice Server

Mobile extensions to Patterns for e-business

This redbook provides application designers and developers with a broad overview of mobile e-business application design and development using the WebSphere Everyplace Access V1R1 offering.

The book gives an overview of the Patterns for e-business and shows how to use the Patterns in the mobile e-business environment.

It also discusses the design and development guidelines for mobile e-business applications using the products bundled in the WebSphere Studio and Visual Age for Java offerings.

This book provides detailed information about the sample application by discussing scenarios and implementing mobile applications exercising different techniques for several type of clients.

It also provides detailed instructions for setting up the development and runtime environment for WebSphere Application Server, WebSphere Transcoding Publisher and WebSphere Voice Server together with the sample shipped with the redbook.

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