

Practical JXTA

Cracking the P2P puzzle

Practical JXTA

Copyright © 2008 by DawningStreams, Inc. The Netherlands

First edition: July, 2008.

All right reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without written permission from the publisher, except for the inclusion of brief quotations in a review.

International Standard Book Number: 978-1-4092-1564-6

Library of Congress Control Number: 2008906696

Limit of Liability/Disclaimer of Warranty: While the publisher and the author have used their best effort in preparing this book, they make no representation or warranties with respect to the accuracy or completeness of the contents of this book.

The software code examples, advice, strategies herein are provided “as is” and any expressed or implied warranties, including, but not limited to, the implied warranties or merchantability and fitness for a particular purpose are disclaimed. No warranty may be created or extended by sales representatives or written sales materials. In no event, shall DawningStreams, Inc. be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profit; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of the software code examples, advice, strategies, even if advised of the possibility of such damages. You should consult with a professional where appropriate.

Trademarks: DawningStreams, Inc. is a registered trademarks of DawningStreams, Inc. in the United States and other countries and may not be be used without permission. All other trademarks are the property of their respective owners. DawningStreams, Inc. is not associated with any products or vendors mentioned in this book.

Publisher: Lulu Enterprises, Inc. (www.lulu.com)

Table of Contents

Foreword	11
Why This Book?.....	12
Who Should Read This Book?.....	13
Prerequisites.....	14
Introduction	17
Peer to Peer.....	17
A Bad Reputation?.....	17
A Quick Historical Review.....	19
Architectures & Design Principles.....	21
Client/Server.....	21
Three-tier & Multi-tier	22
Clustering & Load Balancing	22
Service Oriented Architecture & Middleware.....	23
Grid Computing.....	24
About Needs That Have Shaped IT Solutions.....	24
What is P2P?.....	25
ICQ.....	25
Napster.....	26
Gnutella.....	27
Kazaa.....	28
BitTorrent.....	29
Freenet.....	29
Pure P2P.....	29
Initial Objectives and Evolution.....	30
Benefits.....	30
Drawbacks.....	31
So, What Is The Use of P2P?.....	32
Towards Universality	33
JXTA.....	34
Introduction.....	34
The Three Layer Cake.....	36
The JXTA Project.....	36
JXTA Community & Bindings.....	37
Java.....	37
C/C++.....	38
JXME.....	38
Documentation, Forum & FAQ.....	38
Understanding JXTA	40
Introduction.....	40
A Tribes-In-Islands Metaphor.....	40
The Concepts.....	41
Overview.....	41
Peer.....	42
Peer Group.....	44
Service.....	44
Resources.....	46
Advertisement.....	46
Peer Advertisement.....	47
Peer Group Advertisement.....	47
Publication.....	47
ID.....	48
Content.....	48

Codats.....	49
Pipes.....	49
Module.....	49
The Protocols.....	51
Message.....	52
Endpoint Routing Protocol (ERP).....	52
Endpoint Service.....	52
Endpoint Address.....	52
Endpoint Routing Protocol.....	53
ERP Messages & Advertisements.....	55
Top-Down-Top.....	55
Rendezvous Protocol (RVP).....	55
Message Propagation Protocol.....	56
PeerView Protocol.....	56
Rendezvous Advertisement.....	58
Rendezvous Lease Protocol.....	58
Peer Connection to Rendezvous.....	58
Propagation Control.....	58
Top-Down-Top.....	59
Pipe Binding Protocol (PBP).....	59
Pipe Advertisement.....	60
Pipe Resolver Message.....	60
Propagate Pipe Message Header.....	61
Top-Down-Top.....	61
Peer Resolver Protocol (PRP).....	61
Resolver Query Message.....	62
Resolver Response Message.....	62
Shared Resource Distributed Index (SRDI).....	62
Resolver SRDI Message.....	63
Top-Down-Top.....	63
Peer Information Protocol (PIP).....	63
PIP Query Message.....	64
PIP Response Message.....	65
Top-Down-Top.....	65
Peer Discovery Protocol (PDP).....	65
Discovery Query Message.....	65
Discovery Query Response.....	66
Top-Down-Top.....	66
Access Control.....	66
Membership Service.....	66
Access Service.....	69
The Ignition Process.....	70
Module.....	70
Service.....	71
Peer Group As A Service – Part I.....	71
Well-known Module Class, Specification and Implementation IDs.....	72
Bootstrapping JXTA.....	72
Peer Group As A Service – Part II.....	76
Loading Other Services.....	77
Network Boundaries	79
Reminder.....	79
IP.....	80
Unicast, Broadcast & Multicast.....	81
IPv4 Versus IPv6.....	81
TCP.....	81
UDP.....	81
Datagram.....	81
Port.....	82

Firewalls.....	83
NAT.....	84
Some Limitations.....	85
Proxy.....	86
Router.....	86
Multicasting Versus Subnets.....	87
Natural Network Boundaries.....	87
Artificial Network Boundaries.....	88
JXTA Transportation Layer.....	89
JXSE Transportation Layer.....	89
Overcoming Firewalls.....	89
Relay Service.....	90
Overcoming NATs.....	90
Overcoming Proxies.....	91
Seeds.....	93
JXTA Community Seeds.....	94
Peer Accessibility.....	96
WAN.....	96
LAN.....	96
Same Subnet.....	96
Different Subnets.....	97
JXSE Cryptographic Layer	99
Cryptography Reminder.....	99
Alice, Bob & Eve.....	99
Introduction To Secure Communication.....	100
...And Vicious Circles!.....	103
Public Key Infrastructure (PKI).....	106
Who Certifies The Root Certificate Authority?.....	106
Duties Of A Certificate Authority.....	106
Web Of Trust.....	107
X.509.....	108
X.509 & Principal.....	109
Hash Function.....	109
The Collision Issue.....	110
Key Sizes.....	111
Assumptions.....	111
Calling Experts.....	112
Personal Security Environment (PSE).....	112
Java Cryptography Architecture.....	113
KeyStore.....	113
Creation.....	113
Automatic X.509 Certificate & Private Key Creation.....	114
Big Big Big Warning!.....	116
Registering Your Own X.509.....	117
PSE Configuration.....	118
Other Concepts.....	119
Secure Socket Layer (SSL).....	119
Transport Security Layer (TLS).....	120
Virtual Private Network (VPN).....	120
JXTA Security Layer.....	120
About Cipher Suites.....	120
About Encryption of Private Keys.....	121
PSE Configuration Advertisement Encryption.....	122
Conclusion.....	123
Architectural Considerations	125
Identity Issues.....	125
IP Addresses As Identities.....	125

Defining An Identity in JXTA.....	126
Creation Of Peer IDs & Importation Of IDs From Other Systems.....	127
Peer Group Creation & Identity.....	127
Configuration Modes.....	129
JXTA Peer Types.....	129
Minimal Edge Peer.....	129
Full-Featured Edge Peer.....	129
Rendezvous Peer.....	129
Relay Peer.....	129
JXSE Configuration Modes.....	130
ADHOC.....	130
EDGE.....	130
RENDEZVOUS.....	130
RELAY.....	130
PROXY.....	130
Peer Type versus Configuration Types.....	130
About Implementation Of Services.....	131
Network Scope.....	131
To Be Or Not To Be?.....	132
A Relay.....	132
A RendezVous.....	132
Number of Seeds.....	133
Network Administration.....	134
Default IP Ports.....	134
Default HTTP Port.....	134
Default Multicasting Port & IP Address.....	134
Implementing Seeds.....	135
About Subnets.....	135
Cryptography.....	135
Implementing Your Own Cryptography Layer.....	135
Exportation Limitations.....	136
Access Control Implementation.....	136
Using JXSE	139
Getting Started.....	139
Creating A Project.....	139
Eclipse Plugin.....	140
Javadoc.....	140
First Connection & Local Configuration.....	140
Example 100 - Starting And Stopping JXTA.....	140
Example 110 – Creating A Local Configuration.....	142
Example 120 – Retrieving, Modifying & Saving A Local Configuration.....	144
Example 130 – Setting Default Seeds And Waiting For A Rendezvous Connection.....	145
Miscellaneous.....	147
Understanding ConfigParam.....	147
Modifying Other Configuration Parameters.....	148
Loading A Configuration From A URI.....	148
Automatic Change Of IP Addresses.....	148
Local Configuration Directory Structure.....	148
Exploring Connectivity Issues.....	149
Running Multiple Peers On A Single Device.....	149
The Proper Angle To Strike A Match.....	149
Jack, The Rendezvous.....	149
Anna, The Edge.....	151
Testing Angles Between Jack And Anna.....	153
Angle I - Start Jack, Then Anna, Delete Configuration, Approve Seed & Enable Multicasting.....	153
Angle II - Start Jack, Then Anna, Delete Configuration, Approve Seed & Disable Multicasting.....	155
Angle III - Start Anna, Then Jack, Delete Configuration, Approve Seed & Enable Multicasting.....	155

Angle IV - Re-Start Jack, Then Anna, Keep Configuration, Approve Seed & Disable Multicasting.....	155
Angle V - Re-Start Jack, Then Anna, Keep Configuration, Disapprove Seed & Disable Multicasting.....	157
Angle VI - Re-Start Anna, Then Jack, Keep Configuration, Approve Seed & Disable Multicasting.....	157
Conclusions And Recommendations.....	157
Aminah, The Other RendezVous.....	159
Chihiro, The Other Edge.....	160
Automatic Reconnection To Other RendezVous.....	163
About PeerViews.....	164
Creating Peers And Peer Groups.....	165
Example 200 – Creating IDs.....	165
Example 210 – Creating A New Peer.....	167
Example 220 – Creating A Peer Group.....	168
Discovery Operations.....	171
Finding Advertisements.....	171
Example 300 - Retrieving And Flushing Local Advertisements.....	171
Example 310 – Remote Search For Advertisements.....	174
Local And Remote Publication.....	177
Messages & Advertisements.....	177
Example 400 – Creating An Empty Message.....	177
Example 410 – Add String, Long & Int Elements.....	178
Example 420 – Retrieving Message Elements.....	179
Example 430 – Add Byte Array Element And Retrieve InputStream.....	180
Example 440 – Adding An Advertisement In A Message.....	181
Example 500 – Customized Advertisement.....	182
Attributes.....	185
Methods.....	185
Instantiator.....	186
Example 510 – Registering A Customized Advertisement Instance.....	186
Private Keys, Certificates & KeyStores.....	187
Example 600 - Exporting & Importing Private Keys And X.509 Certificates.....	187
Example 610 – Working With A KeyStore.....	189
Simple Pipe Communication.....	192
Chandra, The Rendezvous Listening to Messages.....	192
Dimitri, The Edge Sending Messages.....	198
Running The Example.....	203
Bidirectional Pipe Communication.....	205
Adelaide, The RendezVous At One End.....	205
Quinisela, The Edge At The Other End.....	208
Running The Example.....	211
JXTA Socket & Socket Server.....	212
Lidong, The Rendezvous And JXTA Socket Server.....	212
Ayrton, The Edge And JXTA Socket.....	215
Running The Example.....	217
JXTA Multicast Socket.....	218
Hans, The Rendezvous, A Multicast Participant.....	218
Teyacapan, The Edge, Another Multicast Participant.....	220
Running The Example.....	222
Implementing A Customized Service.....	223
Introduction.....	223
Example 700 – Creating Module IDs.....	223
Example 710 – Astrology Service	224
Static And Non-Static Attributes.....	229
Static Methods.....	229
Static Class.....	229
Implementation Of Interfaces.....	229
RendezVous Joe, The Astrologer	230
Edge Jill, The Customer	232
Running The Example.....	234

Self-Service Weather Forecasting Method.....	235
Peer Information.....	236
Activation Of The Peer Information Service.....	236
Ping.....	236
Example 800 – Edge Ping.....	236
Example 810 – RendezVous Pong.....	239
Running the Example.....	240
Miscellaneous.....	241
Sending A Message With The Endpoint Service.....	241
Customized Queries.....	241
About Registering Handlers In Services.....	242
JXSE Shell.....	242
The Future of JXTA	245
Adoption Factors.....	245
Technological.....	245
Simplification Of Complexity - Universality.....	245
Security.....	245
Towards IPv6?.....	246
About Needs That Have Shaped IT Solutions – Part II.....	246
Educational.....	246
Documentation, Books And Tutorials.....	246
Code Examples.....	247
Economical.....	247
Costs And Return On Investment.....	247
Potential For Profit.....	249
New Applications.....	249
Accountability & Vendor's Credibility.....	250
Conclusion.....	251
Appendix	253
Frequently Asked Questions.....	253
Why Can't I Establish A Connection With Another Peer?.....	253
How Can I Detect Network Communication Failures?.....	255
How Does JXTA Guarantee Peer ID Uniqueness In The JXTA Network?.....	255
JXSE Generates Too Much Verbosity! How Can I Get Rid Of It?.....	256
I Think I Have Found A Bug, What Should I Do?.....	256
I Know I Have Found A Bug, What Should I Do?.....	257
How Is Data Propagated Between Subnets?.....	257
Why Do I Get The JXTA Shell Window When I Run My Programs?.....	257
Why Am I Getting Inconsistent Results When I Run My JXTA Tests?.....	257
Can I Start Two Peers On The Same Device Or Within The Same Application?.....	258
Must A Peer Implement All JXTA Protocols?.....	258
Does A Message Necessarily Navigate Through A RendezVous Peer?.....	259
Do I Need To Become A Peer Group Member Before Using/Joining It?.....	259
Do I Need To Implement My Own JXTA Services?.....	259
Protocol Messages Types.....	260
Advertisement Attributes.....	261
Peer Advertisement.....	261
Peer Group Advertisement.....	261
Module Class Advertisement.....	261
Module Specification Advertisement.....	261
Module Implementation Advertisement.....	262
Pipe Advertisement.....	262
Rendezvous Advertisement.....	263
Route Advertisement.....	263
Access Point Advertisement.....	263
Code Examples.....	264
KeyStore Creation Example.....	264

Tools.....	265
Connected Peers And RendezVous.....	265
Check For Multicast, RendezVous Seed And Configuration Deletion.....	265
Pop Messages.....	266
Recursive Delete.....	266
Delete Configuration In Default Home.....	267
Display Message Content.....	267
Go To Sleep.....	268
Membership Service Example.....	268
Access Service Example.....	269
JXTA Bootstrapping.....	270
Index.....	273

Foreword

Acknowledgments

I thank my friend Jill Schmidig for her support. There are times in life when you need unconditional support from your friends, just because of the challenges of life. I am talking about the kind of support some people believe they will never need (or never have). I was one of these people. I am grateful to have had her during those days when life hit hard on me, my friends and my family. She was always there to listen and to talk when I needed to.

I also thank all those who have helped me answering questions and to clarify issues about JXTA on the JXTA forum. I also thank all those who have participated to the development of JXTA and the Open Source community in general.

About the Author

Jérôme Verstryngne is a graduate in computer science from the Université Libre de Bruxelles (Belgium). He also obtained a Master in Human Resource Management from that same institution and a Master of Business Administration in marketing from the Rotterdam School of Management (The Netherlands).

After working as a software developer for UCB, a pharmaceutical company, Jérôme worked as an IT consultant and lead projects at Cap Gemini Ernst & Young in both private and publicly owned companies. Then, he relocated to Amsterdam to join PeopleSoft (now Oracle). Since he finished his MBA, he has started his own company with a friend, in the New-York Metropolitan Area (New Jersey, USA). They are working on their own P2P related project.

Jérôme Verstryngne has recently joined the community of JXTA software developers. He frequently participates to the JXTA forum using the AdamMan71 pseudonym. He can be reached by email at adamman71@dev.java.net.

Why This Book?

The name of the book is inspired from another book called *Practical Cryptography*¹ by Niels Ferguson and Bruce Schneier. Practical Cryptography aimed at discussing cryptography implementation at an engineer's level, rather than at a mathematical level, which is often abstract and conceptual. It discussed “*how to apply the cryptographic functions in a real-world setting*” citing real software implementation issues.

Sun Microsystems introduced JXTA in 2001. JXTA is a set of protocol specifications for implementing peer-to-peer applications. Like cryptography, JXTA is very abstract and conceptual in its definition. It does not get involved with implementation issues (or very little). Although several books written about JXTA at the beginning of the Millennium included discussions on its implementation in the Java programming language, its evolution and the evolution of its implementation has made these books progressively obsolete.

The author started investigating JXTA in early 2006 and faced many issues trying to understand this technology. The Java implementation was undergoing a major overhaul, as it was migrated to version 1.5 of the Java platform. The available documentation became obsolete as new code was delivered by the open source community. The available software guides were becoming obsolete. This made learning JXTA very tedious.

Today, the latest available version of JXTA is 2.5. Although the corresponding version of the programmer's guide constitutes a tremendous improvement over the previous versions, it does not explain how JXTA concepts operate in details. Moreover, it does not describe the technical issues those who have implemented JXTA had to solve to enable smooth communication over computer networks. Understanding these issues is essential when designing, implementing and configuring software applications based on JXTA.

This book aims at shedding more light as to how JXTA operates behind the scenes, both from a conceptual and a practical point-of-view. In addition, we provide a reminder covering network concepts and basic cryptographic issues. This book is about bringing the core concepts of the JXTA puzzle together, in order to crack it and to make JXTA a *practical* technology to use.

1 Practical Cryptography, by Niels Ferguson and Bruce Schneier, published at Wiley Publishing, Inc. in 2003.

Who Should Read This Book?

This book is intended for:

- **JXTA early adopters** – If you are one of those who adopted JXTA since its inception, this book will help keep you up-to-date with the recent evolution of the JXTA implementation. Contrary to many books published earlier, this book contains code examples using the most recent features of JXSE, such as the `NetworkManager` and the `NetworkConfigurator`, for example. We also discuss the Personal Secure Environment (PSE) service.
- **Functional analysts and software architects** – This book covers all the JXTA concepts (including their usage), that a functional analyst or a software architect needs to understand, in order to design JXTA-based P2P applications. The official JXTA specification document contains several abstract terms referencing each other, together with a lot of technical language, making it hard to understand by everyone. This book is solving this issue by using many metaphors from the real world.
- **Software developers** – This book will explain altogether the basic concepts of JXTA and how to use JXSE, the Java implementation of JXTA, to develop your own P2P applications. It contains several basic code examples explaining how to use JXTA, step-by-step. These were designed to be as didactic as possible, that is, without optimization, to facilitate the learning. You will be able to take inspiration from these building blocks for your own software implementations.
- **System and network administrators** – The JXTA technology is abstract in its concepts, but it relies heavily on network transportation. Many transportation protocols can be implemented with JXTA, although TCP/IP and HTTP are the most common. Nearly all implementations of JXTA require mechanisms to overcome firewalls, NATs and proxies boundaries. This book explains how administrators can improve the overall performance of JXTA-based applications using an adequate configuration of their systems, and without compromising security policies.
- **Managers and decision makers** – If you are a manager or a decision maker, you may have heard about peer-to-peer technologies. You may even have used them for chatting, phoning over the Internet, or on your mobile phone and hand-held devices, and would like to know more about them. This book explains the origin of P2P and demystifies the undeserved negative publicity it received at the beginning of the decade. It describes the bene-

fits of P2P, how to take advantage of it. Security and control access issues to copyright protected documents are also discussed here.

This book does not explain how to implement JXTA. However, it can be used as a base to understand JXTA. Developers willing to implement the JXTA protocols will have to refer to the official specification documentation available online at <https://jxta-spec.dev.java.net/> for technical details.

Prerequisites

This book assumes that software developers know about the Java programming language and concepts such as URI (Uniform Resource Identifiers), URL (Uniform Resource Locator) and URN (Uniform Resource Name). They should master the event firing and even listening mechanism. Functional analysts and software architects need to know about XML, TCP/IP and HTTP, in general. System and network administrators should be aware of how TCP/IP, firewalls, NATs, proxies, routers and other types of network transportation systems operate in general. Managers and decision makers do not need to know more than general IT concepts.

The code examples provided in this book are relying on version 2.5 of JXSE. Version 2.5 of JXSE requires JDK 1.5 (or a higher version).

Introduction

This chapter introduces peer-to-peer and JXTA in general. We will describe the motivations that led to the emergence of the peer-to-peer philosophy by revisiting parts of the IT history since the 1960's. We will compare it with other major IT design philosophies; describe its specificities, potential benefits and drawbacks. We will also explain why a universal set of protocols became necessary to implement open peer-to-peer functionalities in software applications, thus leading to the creation of JXTA.

Peer to Peer

Peer-to-Peer (or P2P for short) is defined as a concept allowing direct communication between individual computers by some, and as a set of networking design principles by others. The actual definition of what P2P is varies according to authors; but most agree that a peer-to-peer system traditionally rejects the client/server model and the underlying hierarchy it imposes between computers operating on a network.

Most of us heard about P2P for the first time when we used a file exchange application such as Napster, Kazaa or Gnutella, an instant messaging application such as ICQ or Yahoo! Messenger, or when we started “phoning” over the Internet with Skype. Others discovered P2P with Groove (now Microsoft Groove), a software application developed by Groove Networks in the 1990's.

A Bad Reputation?

In his July 15, 2007 cryptogram newsletter², Bruce Schneier, a renowned cryptography and security specialist, discusses the role of the *correspondent inference theory* on the human perception of terrorist group motivations. This theory, proposed by Edward E. Jones and Keith Davis, suggests that people are analyzing the consequences of other people's behavior and actions to guess their motivation, regardless of external or situational factors.

Bruce Schneider provides the example of someone violently hitting someone else. Most would assume that this person is inherently violent, until one tells them that he or she is role playing for a movie and that the camera can be seen in the distance. Information about the context is not taken into account and a false conclusion is drawn. Several television shows use this technique to trap guests and celebrities by hiding contextual information.

² <http://www.schneier.com/cryptogram-0707.html>

Since the year 2000, P2P technologies have boomed in a high number of file sharing and chatting applications. Unfortunately, some P2P applications have often been associated with illegal file transfers, copyright infringements on music and films, and all sorts of other illegal or unsafe activities. Those using P2P technologies have been accused of participating in a movement damaging the economy.

On the other side of the fence, others claimed that some companies, especially in the music industry, were abusing their dominating position and inadvertently created the context for these new Internet technologies to appear. They claim these technologies are good, since they allow us to buy songs and exchange movies online, to listen and view them at any time, improving the customer's overall experience. They claim that all of this should be accepted as part of the natural evolution of economies.

Can we really blame P2P technologies for bad behavior, illegal activities and exposure to security breaches? The first P2P application dates back to 1979 and many have been created since. The public started using P2P in the late 1990's as the Internet became more and more available to a wide audience. These applications were not initially designed by intention to perform any sort of illegal activity.

Although we can observe a correlation between the increasing number of P2P application users and some copyright infringement activities since the beginning of the Millennium, we cannot infer that these applications are the cause. The absence of access control to resources is not intrinsic to P2P applications. One needs to make a clear difference between access control to resources and P2P software design principles. They are not incompatible and can be implemented together harmoniously.

Several P2P applications have been called unsafe. The main argument being that since all sorts of files, including applications, were being exchanged, the risk of virus propagation was increased. Interestingly enough, email applications have never been called unsafe, although emails contribute to the propagation of viruses. Email content, not the application itself, has been blamed. Different perceptions have created different explanations for a unique cause requiring a unique solution: anti-virus software applications.

So, is the bad reputation attributed by some to P2P applications fair? Some developers have exploited key features of P2P applications and added anonymity to facilitate anonymous transactions. Some were claiming a right for privacy alibi, while de facto, their applications were about sharing copyright protected content, such as audio files and movies. Consider similar controversy:

should we blame knives and ban them because they have been used by some criminals? Or should we focus on the criminals? Would the elimination of knives prevent criminals from committing crimes? Aren't knives useful in other situations? P2P, itself, is not to blame for illegal activities and does not deserve the bad reputation.

P2P design principles have been instrumentalized by some to enable many activities, including illegal activities, but these should not be rejected as a whole. We will see that it is possible to take advantage of all the benefits of P2P design principles without suffering from the abuse of careless or malintentioned individuals, through the use of access control and cryptography, for example.

A Quick Historical Review

More than 40-years ago, the first mainframe computers appeared, allowing remote users to share its usage via a client/server model. These were expensive items. Users (clients) would access the mainframe (server) with terminal computers performing very basic operations: establishing a connection with the central server, collecting key strokes from users, sending them to the mainframe for processing and waiting for feedback, that is, characters to display on the screen. That's it! The communication model between the client and the server was very primitive and the client terminal was completely useless when not connected to the mainframe server. These were very different from today's personal computers which are capable of much more sophisticated operations.



ADM-3A terminal

As time went by, mainframes became more powerful and less expensive to acquire. Eventually the need for these computers to exchange information between themselves arose. The first connection was established in 1969 via the ancestor of the Internet, the ARPANET. This type of communication was very different from the *passive* client/server model described above. Both parties were acting individually, requesting, providing and processing information on their own. In the absence of a connection, each party was still capable of operating on its own. This was not the case with terminal computers. Technically speaking, we can consider this as the first P2P connection between two computers. In 1983, ARPANET evolved into what is now called TCP/IP, a set of protocols to exchange information between remote computers over the *Internet*.

Meanwhile, starting from the end of the 70's, computers became available to individual consumers: the *personal computer* was born. Some were equipped with the capacity to connect to mainframes or to other computers via telephones and modems. In technical terms, P2P connections could be established between peers. But this is still far from today's definition and the gener-

al understanding of P2P systems and applications.

The World Wide Web was introduced in the early 1990's and the usage of TCP/IP increased. More users connected to the Internet and for longer periods of time. Yet, users were still accessing web pages or databases located on a central server and sending their email via mail servers. This way of interacting between computers closely resembled to the 1960's client/server model. To some extent, having to connect to central servers and waiting for responses was nothing different than behaving like terminal computers of the 60's. PC's were left with a lot of underutilized power and resources.

One thing leading to another, some developers imagined the possibility of designing P2P-like applications operating directly from PC to PCs instead of going through central servers. This was facilitated by the fact that every PC was capable of establishing connections to other PC's via the Internet. Were these new applications the first P2P applications?

Not really. In 1979, Usenet, an Internet application allowing users to post messages, was created. Its true novelty was that it did not rely on a single server, but on a changing set of loosely connected servers. Users could post messages on a server and servers could exchange these messages between themselves in order to make them available to other users. There was no central server or central authority. *Decentralization* and *distributed communication* were born. The downside of this is they were only available to servers, but not yet to end users.

So, what is P2P precisely? Is the 'absence of central authority or central servers' required to claim that an application or a system is P2P? In that case, Napster, one the most popular early P2P applications would be disqualified for using a central server for indexing shared files. Does 'two peers exchanging files and communicating directly' say enough to claim that we are dealing with a P2P application? Some P2P applications are now used to process data on a large set of computers. This is more than just exchanging files and communicating; peers are providing services to each other. Should this be taken into account in the definition of P2P? P2P covers many facets, as we will see.

First, we will explore different types of IT architectures and design principles to refine our definition and understanding of what P2P is all about..


```
}  
  
}
```

Example 510 – Registering a Customized Advertisement Instance

This operation must be performed on every peer that will use our customized advertisement, but not necessarily those who will transport or index a message containing our advertisement. Else, the advertisement will not be processed correctly by peers receiving it.

This mechanism, in combination with the `Initiator` static class created as part of our customized advertisement, will ensure that peers receiving our customized advertisement will receive it as a Java object of type `_500_Customized_Advertisement_Example`. This eliminates tricky code manipulations for developers; the received advertisement can safely be cast automatically into the proper Java object type.

Technically speaking, the JXTA protocols in JXSE will invoke an `Instantiator` for the advertisement type mentioned in the XML document they receive. Since the `AdvertisementFactory` will have been initialized properly, it will be able to return such an `Instantiator`. The protocols can then create an empty instance of our customized advertisement by invoking the `newInstance()` method. Then, they will be able to fill this new instance with the remaining data contained in the XML document by invoking the `ProcessElement(TextElement TheElement)` method in our customized advertisement.

Metaphorically: We have now learned how to write down messages on dried leaves. We have also learned how to create special dried leaves to describe/advertise our non-standard JXTA resources.

Private Keys, Certificates & KeyStores

Before we start describing how to use pipes to exchange messages, we need to describe how to work with private keys, certificates and keystores.

Example 600 - Exporting & Importing Private Keys And X.509 Certificates

The following code describes how to quickly generate a X509 certificate and its corresponding private key, plus how to export them into a `String` and re-import them from that same `String`:

```
import Examples.Z_Tools.Tools;  
import java.io.ByteArrayInputStream;  
import java.io.StringReader;  
import java.security.KeyFactory;
```

```

import java.security.PrivateKey;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.security.spec.PKCS8EncodedKeySpec;
import net.jxta.impl.membership.pse.PSEUtils;

public class _600_Exporting_And_Importing_Private_Keys_And_X509_Certificates {

    public static final String Name = "Importing and Exporting Private Keys and Certificates";

    public static void main(String[] args) {

        try {

            // Certificate and Private Key
            X509Certificate TheX509Certificate;
            PrivateKey ThePrivateKey;

            // Initialization
            PSEUtils.IssuerInfo ForPSE = PSEUtils.genCert(Name, null);

            TheX509Certificate = ForPSE.cert;
            ThePrivateKey = ForPSE.issuerPkey;

            // String encoded certificate & private key
            String Base64_X509Certificate = PSEUtils.base64Encode(TheX509Certificate.getEncoded());

            String Base64_ThePrivateKey = PSEUtils.base64Encode(ThePrivateKey.getEncoded());

            // Printing Results
            System.out.println("-----");
            System.out.println(Base64_X509Certificate);
            System.out.println("-----");
            System.out.println(Base64_ThePrivateKey);
            System.out.println(ThePrivateKey.getFormat());
            System.out.println("-----");

            // Recreating certificate & private key
            X509Certificate RecreateX509Certificate;
            PrivateKey RecreatePrivateKey;

            // Recreating the X509 certificate
            byte[] Temp = PSEUtils.base64Decode(new StringReader(Base64_X509Certificate));

            CertificateFactory TheCF = CertificateFactory.getInstance("X509");
            RecreateX509Certificate = (X509Certificate) TheCF.generateCertificate(
                new ByteArrayInputStream(Temp));

            System.out.println("-X509-Original-----");
            System.out.println(TheX509Certificate.toString());
            System.out.println("-X509-Recreated-----");
            System.out.println(RecreateX509Certificate.toString());
            System.out.println("-----");

            // Restoring the private key
            Temp = PSEUtils.base64Decode(new StringReader(Base64_ThePrivateKey));

            KeyFactory keyFactory = KeyFactory.getInstance("RSA");
            PKCS8EncodedKeySpec MyPKCS8EncodedKeySpec = new PKCS8EncodedKeySpec(Temp);
            RecreatePrivateKey = keyFactory.generatePrivate(MyPKCS8EncodedKeySpec);

            System.out.println("-Private-Key-Original-----");
            System.out.println(ThePrivateKey.toString());
            System.out.println("-Private-Key-Recreated-----");
            System.out.println(RecreatePrivateKey.toString());
            System.out.println("-----");

        } catch (Exception Ex) {

            Tools.PopErrorMessage(Name, Ex.toString());

        }

    }
}

```

```
}
```

Example 600 - Exporting & Importing Private Keys And Certificates

This example is performing the following:

1. We generate a pair of corresponding X509 certificate and private key using the `PSEUtils` tools provided as part of JXTA. The RSA key size is set to 1024 by default.

REM: If the reader is not satisfied with those settings, he or she can take a look at the code of `PSEUtils` to find out how to modify it or to write a new method inspired from it.

2. We export both the X509 certificate and the private key into base 64 strings and display the result. These strings can be easily exchanged, transported or stored by applications. Do not forget that the private key is not encrypted and has no protection.
3. We re-import the X509 certificate and the private key from the base 64 strings. Then, we display the original and the re-imported result for comparison.

Some readers will notice some difference between the display of the original X509 certificate and the recreated X509 certificate. In fact, the modulus and the exponent are expressed in decimals in one case and in hexadecimals in the other case, but the values are the same.

Example 610 – Working With A KeyStore

The following code example describes how to create a keystore and to interact with it:

```
import Examples.Z_Tools.Tools;
import java.io.File;
import java.io.IOException;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.PrivateKey;
import java.security.UnrecoverableKeyException;
import java.security.cert.X509Certificate;
import net.jxta.id.IDFactory;
import net.jxta.impl.membership.pse.FileKeyStoreManager;
import net.jxta.impl.membership.pse.PSEUtils;
import net.jxta.peer.PeerID;
import net.jxta.peergroup.PeerGroupID;

public class _610_Working_With_A_Keystore {

    public static final String Name = "Example 610";
    public static final PeerID PID = IDFactory.newPeerID(PeerGroupID.defaultNetPeerGroupID,
        Name.getBytes());
    public static final File ConfigurationFile = new File("." +
        System.getProperty("file.separator") + Name);

    public static final String MyPrincipalName = "Principal - " + Name;
    public static final String MyPrivateKeyPassword = "PrivateKey Password - " + Name;

    public static final String MyKeyStoreFileName = "MyKeyStoreFile";
```

```

public static final String MyKeyStoreLocation = "." + System.getProperty("file.separator")
    + Name + File.separator + "MyKeyStoreLocation";
public static final String MyKeyStorePassword = "KeyStore Password - " + Name;
public static final String MyKeyStoreProvider = "KeyStore Provider - " + Name;

public static final File MyKeyStoreDirectory = new File(MyKeyStoreLocation);
public static final File MyKeyStoreFile = new File(MyKeyStoreLocation + File.separator +
    MyKeyStoreFileName);

public static final X509Certificate TheX509Certificate;
public static final PrivateKey ThePrivateKey;

static {

    // Static initialization
    PSEUtils.IssuerInfo ForPSE = PSEUtils.genCert(Name, null);

    TheX509Certificate = ForPSE.cert;
    ThePrivateKey = ForPSE.issuerPkey;

}

public static void main(String[] args) {

    try {

        // Removing any existing configuration?
        Tools.CheckForExistingConfigurationDeletion(Name, ConfigurationFile);

        // Preparing data
        MyKeyStoreDirectory.mkdirs();

        // Creating the key store
        FileKeyStoreManager MyFileKeyStoreManager = new FileKeyStoreManager(
            (String)null, MyKeyStoreProvider, MyKeyStoreFile);

        MyFileKeyStoreManager.createKeyStore(MyKeyStorePassword.toCharArray());

        if (!MyFileKeyStoreManager.isInitialized()) {
            Tools.PopInformationMessage(Name, "Keystore is NOT initialized");
        } else {
            Tools.PopInformationMessage(Name, "Keystore is initialized");
        }

        // Loading the (empty) keystore
        KeyStore MyKeyStore =
            MyFileKeyStoreManager.loadKeyStore(MyKeyStorePassword.toCharArray());

        // Setting data
        X509Certificate[] Temp = { TheX509Certificate };
        MyKeyStore.setKeyEntry(PID.toString(), ThePrivateKey,
            MyPrivateKeyPassword.toCharArray(), Temp);

        // Saving the data
        MyFileKeyStoreManager.saveKeyStore(MyKeyStore, MyKeyStorePassword.toCharArray());

        // Reloading the KeyStore
        MyKeyStore = MyFileKeyStoreManager.loadKeyStore(MyKeyStorePassword.toCharArray());

        // Retrieving Certificate
        X509Certificate MyCertificate = (X509Certificate)
            MyKeyStore.getCertificate(PID.toString());

        if (MyCertificate==null) {
            Tools.PopInformationMessage(Name, "X509 Certificate CANNOT be retrieved");
        } else {
            Tools.PopInformationMessage(Name, "X509 Certificate can be retrieved");
            System.out.println(MyCertificate.toString());
        }

        // Retrieving private key
        PrivateKey MyPrivateKey = (PrivateKey) MyKeyStore.getKey(PID.toString(),
            MyPrivateKeyPassword.toCharArray());
    }
}

```

