Chapter 1

Introduction to Instant Messaging

Instant messaging (IM) is an Internet-based protocol application that allows one-to-one communication between users employing a variety of devices. The most popular form of IM is chatting, where short, text-based messages are exchanged among computers. With the advent of technologies such as Wireless Application Protocol (WAP) and the popularity of handheld devices such as mobile phones, Short Message Service (SMS) added a new dimension to instant messaging integration.

SMS refers to sending and receiving text messages to and from mobile telephones. The text may be composed of words or numbers or may be an alphanumeric combination. SMS was created as part of the GSM Phase 1 standard. We think the first short message was sent in December 1992 from a PC to a mobile phone on the Vodafone GSM network in the U.K. Each short message ranges between 70–160 characters. SMS has a store-forward capability; this means sending messages is possible even when the recipient is not available. The user is notified when a message is waiting, as with voicemail.

Integration of SMS and instant messaging allows you to deliver short messages by using the instant messaging facility over handheld devices. Today, instant messaging can be availed free of cost. Software giants such as Yahoo!, Microsoft, and AOL are already offering free instant messaging facilities.

A Brief History of Instant Messaging

Instant Messaging has been around for more than two decades. The first major player to enter the arena of Instant Messaging was AOL, which launched its own version of instant messenger with a component used for managing all the incoming and outgoing messages and the list of friends. This component is popularly known as *buddy list*. Soon, Microsoft and Yahoo! followed AOL's trail. As a result, MSN and Yahoo! messenger appeared on the market with a variety of impressive new services. In its early days, instant messaging uses were restricted to splashing messages on bulletin boards. Gradually, instant messaging became a major area of interest for youngsters. Society acknowledges instant messaging as the most common means by which people of varying age groups, especially youngsters, communicate with one another.

Until 1990, there was no significant change in the status of instant messaging from what it had been when initially conceived, mainly because instant messaging was not taken seriously till then. Subsequently, the corporate world changed its attitude toward instant messaging, thanks to the Internet revolution. With a sudden rise in the popularity of the Internet and the arrival of new techniques like voicemail, online transactions, and so on, the corporate world started taking instant messaging seriously. The Internet led not only to a substantial increase in the number of instant messaging users but to the realization of the potential of instant messaging and to earnest attempts to eliminate the limitations of instant messaging and exploit its possibilities fully.

The business community was first to explore the potential of instant messaging and expand its capabilities, making it the useful tool it is today. Instant Messaging is now in the mainstream of WAP technology and is being used with mobile devices such as cellular phones, laptops, and pagers.

The Advantages of Instant Messaging

Instant messaging, unlike e-mail, allows spontaneous interaction among users over the network. This feature of instant messaging is extremely helpful in accessing remotely located users. Entrepreneurs often avail this attribute of instant messaging for accessing and interacting with remotely located workers. Instant messaging meets their purpose quite well and helps save time by obviating the need for writing e-mail and waiting for delivery-confirmation messages.

With the introduction of new ideas such as notification in instant messaging, the user can now be kept on the alert round the clock. Such notification features range from informing the user about in-coming mails or mobile beeps to delivering the latest information such as stock quotes.

The role of instant messaging is crucial in the business world, as it allows for quick and cheap transmissions of information. Using instant messaging to convey information that contains strings of text is much cheaper and more flexible than conveying such information over the phone or arranging to meet face to face.

Another promising feature of instant messaging is the reliable security it offers, which ensures privacy and confidentiality. In addition, you can block an unwanted incoming message and prevent yourself from being distracted. Also, you can block instant messages while using handheld devices.

The Need for Instant Messaging

Over the years, instant messaging has proven itself to be a feasible technology not only to fun-loving people but to the world of commerce and trade, where quick responses to messages are crucial. With the wide acceptance it commands, instant messaging is looked upon by the business world as an ideal tool for promoting its interests. Instant messaging has

revolutionized business communication, as it is well suited for accessing remotely located workers. Also, crucial information can be delivered very quickly. Although some of you might consider e-mail the better option in this regard, e-mail lacks the spontaneous and direct person-to-person interaction instant messaging offers.

The Future of Instant Messaging

Soon, people will stop considering instant messaging merely a convenient way of chatting over the Internet. Attempts are being made to convert instant messaging into a tool that will break all the barriers encountered in device-based networking such as managing mobile-phone calls. Today, you are likely to face problems such as you will miss all your important calls if you forget to carry your mobile phone. But in the near future, you will be able to monitor your mobile-phone calls from a distance, in other words, when you are not near to your mobile phone. We hope that in a short time from now the provision to handle phone calls with instant messaging will be realized, enabling you to sit at a distance from your mobile phone and handle in-coming calls.

With instant messaging, you will be able to send a message to your mobile phone, instructing it to make a call and to deliver the message to the recipient of the call, informing him or her of your availability. You will be able to receive a message in the same manner. The possibilities of instant messaging are unlimited, but innovations are needed for the implementation of these possibilities. Instant messaging has potential far beyond chatting; in fact, it has begun to find use in other areas. Given a little time and the efforts of big software buddies, instant messaging will bloom into its full splendor.

Jabber Instant Messaging Model

Jabber Instant Messaging system is different from other instant messaging systems, as it is based on XML. The most promising feature of Jabber Instant Messaging System is the open-source XML protocol, which makes Jabber more distributed and easily accessible to all who were formerly separated by cross platforms. The architecture of Jabber is almost like that of distributed servers, wherein a client connects to its host server and delivers a message. The host server, in turn, approaches the destination server and hands the message to its intended recipient.

The Java server opts for the open-source protocol because the instant messengers currently available cannot speak with one another. Consider the following analogy. Assume that you have four friends with whom you need to communicate at the same time; you have two choices: you can meet them under some common roof, which is quite cumbersome and expensive, or, if you're friends and you're Internet surfers, you can avail the instant messaging facility to communicate with them. However, to use the latter option, you must have all the instant messengers your friends use for communication. Also, in such a case, you have to open four windows on your computer and handle incoming messages individually, which requires a lot of time.

Introduction to Jabber server

The Jabber server enables its clients to communicate with a non-Jabber client such as MSN. The Jabber server uses a component named *transport* for communicating with the foreign client. Since the Jabber server protocol is completely XML oriented and since other instant messengers maintain their own standardized protocol, the difference in communication is evitable. To bridge this communication variation, the Jabber server uses the Transport. The Transport bridges the gap between these incompatible protocols.

The Jabber server is far more complex than those of the conventional Client/Server architecture, whose simplicity may be attributed to the fact that they often overlook the client priorities. With the Jabber server, making Jabber clients for different platforms is far simpler and involves fewer headaches than with the conventional servers.

Jabber client

Initially, you will probably think of a Jabber client as a common user availing the services of Jabber instant messaging just like the MSN or the Yahoo! instant messaging client. To a certain extent, this is right. But the Jabber client is something more than an ordinary client. In simple words, a Jabber client is something that uses Jabber's ideas and protocols to get a job done. You can write your own clients that utilize the Jabber protocols. For instance, you may develop Jabber-client software that uses the Jabber protocol for instant messaging communication, allowing users of different instant messaging communities to share their views and ideas.

The Jabber server involves two major components: Base and Transports.

Jabber server Base components

Base components handle the job of delivering messages between Jabber clients in their open XML protocol. For instance, Client A and Client B have their accounts on the Jabber server. Whenever Client A sends a message to Client B, the Jabber server manages the delivery of the message to the receiving Client B in its own XML protocol. Thus, the medium of communication between the clients is a protocol, which is known to the Jabber server.

Transports

Transports are the components that bridge the gap between the Jabber and non-Jabber protocol. When you access a non-Jabber system using the Jabber server, the transports hide the system variation from you, and you do not feel that you are visiting some foreign system. When you access a non-Jabber system for the first time, you need to register yourself with Jabber transports by providing information such as the user name and the foreign system you propose to use. Once this is done, messages will be sent and received on the foreign system. We have the transports available for the well-known instant messengers AOL, ICQ, MSN, Yahoo!, and so on. Serious work is underway in developing transports to support other services such as SMS, very popular in Europe.

Structure of Jabber XML protocol

As mentioned earlier, XML is the basis of the Jabber system. Communication between the client and the Jabber server takes place on port 5222. Two XML streams are involved in the exchange of data. One stream delivers the data packet from the client to the server, and the other stream delivers the data from the server to the client. The following code snippet is an example of XML exchange between the Jabber server and the client.

```
SEND:   <stream:stream
SEND:      to='jabber.org'
SEND:      xmlns='jabber:client'
SEND:      xmlns:stream='http://etherx.jabber.org/streams'>
RECV:   <stream:stream
RECV:      xmlns:stream='http://etherx.jabber.org/streams'
RECV:      id='39ABA7D2'
RECV:      xmlns='jabber:client'
RECV:      from='jabber.org'>
      (XML for user session goes here)
SEND:   </stream:stream>
RECV:   </stream:stream>
```

Jabber's Open XML protocol contains three top-level XML elements (also called tags).

- `<presence/>` — This element determines the status of the user. The structure of the `presence` element is as follows:

```
<presence
from='ericmurphy@jabber.org/notebook'
to='stpeter@jabber.com/Gabber'>
<status>Online</status>
</presence>
```

  Based on the status of the user to be communicated, the `<presence>` element can be evaluated on the basis of the following values:

  - `probe` — This value of the `presence` element is used to send a special request to the recipient of the message without waiting for the user's presence information. Notice that the sever, not the client, processes such a request.
  - `subscribe` — This sends a request that the recipient automatically send the presence information to the sender whenever the user changes its status.
  - `subscribed` — This sends a notice that the sender accepts the recipient's request for presence subscription. The server now sends the recipient the sender's presence information whenever it changes.
  - `unsubscribe` — If the value of the `presence` element is `unsubscribed`, the user sends the request to the recipient of the message to stop sending the messages of his/her presence.

- `unsubscribed` — In case the `presence` element holds this value, then it indicates that the user will not be able to communicate in any way with the sender of this message. In such a situation, the server no longer sends the sender's presence to the recipient of the message.
- `from` — This mentions the name or id of the sender of the message.
- `to` — This mentions the name of the recipient of the message.
- `show` — This displays the status of the user.
- `status` — This displays the description of the status.

- `<message/>` — This element is used for sending the messages between two Jabber users. JSM (Jabber Session Manager) is responsible for catering all messages regardless of the status of the target user. If the user is online, the JSM will instantly deliver the message; otherwise, the JSM will store the message and deliver it to the user no sooner than he or she comes online. The `<message>` element contains the following information:
  - `to` — This identifies the receiver of the message.
  - `from` — This mentions the name or id of the message's sender.
  - `text` — This element contains the message about to be delivered to the target user.

```
<message to='ericmurphy@jabber.org/notebook'
   type='chat'>
<body>hey, how's the weather today?</body>
</message>
```

- `<iq/>` element — This element manages the conversation between any two users on the Jabber server and allows them to pass XML-formatted queries and respond accordingly.

  The main attribute of the `<iq/>` element is `type`. The `type` attribute of the `<iq/>` element can carry the following values:
  - `get` — This attribute of the <iq/> element is used to retrieve the values of the fields present.
  - `set` — This attribute is responsible for setting or replacing the values of the fileds queried by the `get` attibute.
  - `result` — This attribute indictates the successful response to an earlier set/get type query.

```
<stream:stream from='jabber.org' id='1440203636' xmlns='jabber:client'
xmlns:stream='http://etherx.jabber.org/streams'><iq
from='gaurav@jabber.org/www.jabber.org' id='1440203636' type='result'>

<query xmlns='jabber:iq:roster'><item jid='msn.jabber.org/registered'
subscription='from'/><item jid='JohnSmith%hotmail.com@msn.jabber.org'
name='JohnSmith%hotmail.com'
```

```
subscription='both'><group>Friends</group></item><item
jid='Billy%hotmail.com@msn.jabber.org' name='Billy%hotmail.com'
subscription='both'><group>Friends</group></item></query></iq></stream
:stream>
```

Why integrate with the Jabber server?

The greatest achievements of the Jabber server are its open-source protocol and its ability to speak with other instant messaging applications. Therefore, merger with the Jabber server provides us with the opportunity to bring users around the world under the net of our application, gaining a larger distribution.

Apart from open-source protocol, the services of the Jabber server are cost-free. Availing such benefits, anyone can build his or her own Jabber client to fulfill unique requirements. Any innovative IT personnel can develop his or her own version of the Jabber client. The only point that needs to be emphasized is that the Jabber services must be used rationally. In the near future, the Jabber camp might develop a protocol that handheld devices can use to access the Jabber server. If such a protocol emerges, we will be blessed with the ability to transfer our services from personal computers to embedded systems.

Introduction to the Instant Messaging Application

Over the years, instant messaging has undergone many changes, several new innovations in the field of information technology having been incorporated in instant messaging. One such innovation is Jabber instant messaging, based upon the open-source principle. Our instant messaging application avails the benefit of this novel concept and works hand in hand with the Jabber server.

We envision an application that works on behalf of clients to interact with the Jabber server. The primary aim of our application is to develop an instant messaging solution that is compatible with clients of different platforms. In other words, an additional feature of working for Jabber server is incorporated in our application. This feature of our application not only maintains communication between the client and Jabber but also enables the client to use the application whenever desired. Our application is complete in itself; apart from handling clients for the Jabber server, it has all the characteristics of an ideal instant messaging application. No compromise has been made of the normal functions required of any instant messaging application. This application works quite efficiently in spite of the incorporation of additional features. This instant messaging application is based upon the following three components:

- Client — The client is the origin of all requests. This component manages creating XML-formatted queries and sending them to the local server.
- Local server — This component satsifies queries the client sends. The local server can act as a complete service provider to the local client; on the other hand, it can act as an intermediate layer between the local client and Jabber server. For instance, when a client

query is meant for some external source (the Jabber server), the Local server acts as the middle layer between the client and the external source and manages a synchronized flow of messages between the client and the external source. On the contrary, if the client query can be resolved locally (the query is not meant for the Jabber server but for the local server), the local server does not act as a middle layer but resolves the client query without bothering about the Jabber server.

- Jabber server — This module acts as an external source in our application. The local server approaches the Jabber server only when client queries are meant for the Jabber server. All client queries forwarded by the local server are resolved by this component. Once a query is resolved, the Jabber server returns the results to the local server, which further returns the result of the query to the intended client.

Figure 1-1 shows the functions of the various components in our application.

Work flow of the IM application for external client request

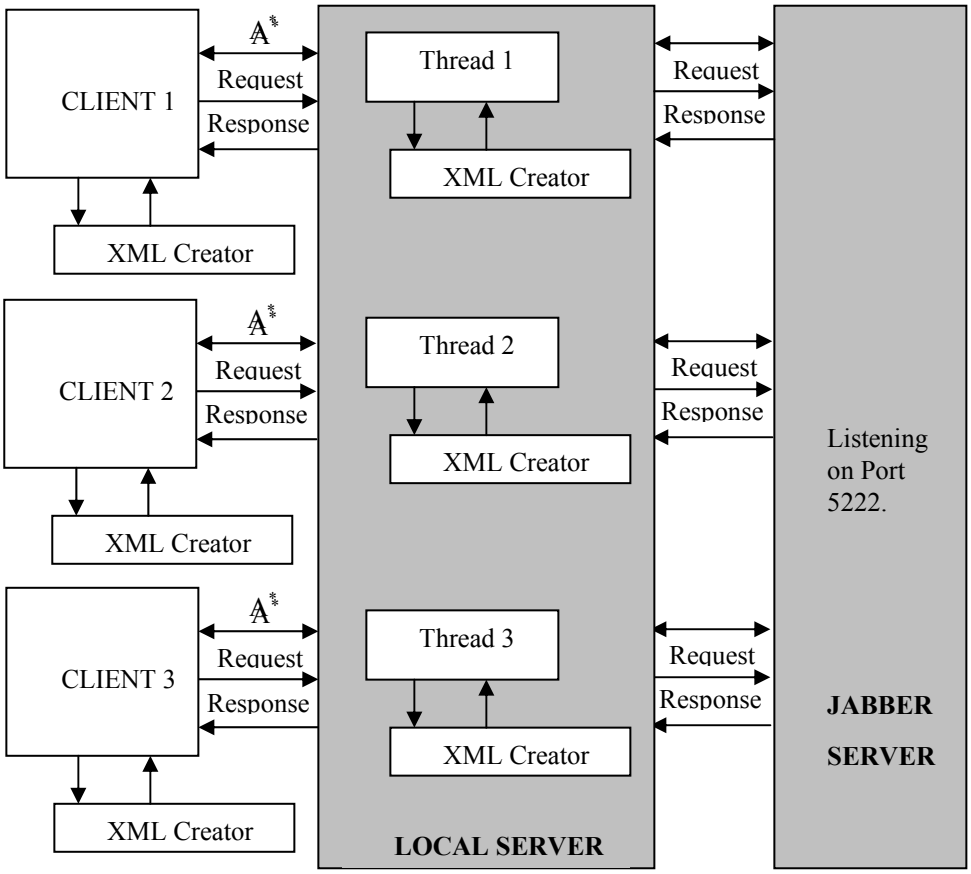
External client requests are generated by the client module for the Jabber server. Evident in Figure 1-1, our application, depicted as a local server, is handling communication from and to Jabber server for both the client and the Jabber server. The medium of communication among the client, local server, and Jabber server is XML driven. To cater to the client requests, the local server is listening on port 5555. Also, to handle multiple clients and to keep the session alive for the client and Jabber server, the local server maintains threads.

Throughout the life cycle of our application, the client and Jabber server never come face to face. The boundary of client requests is restricted to the local server, even when requests are meant for the Jabber server. To understand the workflow of our application, consider an example.

Assume a situation where Client A wishes to connect with the Jabber server. The client initiates the process by sending a query, created using a prewritten module named XML Creator. This prewritten module writes client queries in a format that the local server can understand.

Once the client successfully establishes connection with the local server, a thread is assigned by the local server for the client, which will synchronize the flow of messages to the client originating from the local server. The same thread is used up by the local server for a harmonious exchange of messages between itself and the Jabber server.

Client requests for the Jabber server are routed to the local server. Once a request is delivered to the local server, it parses the request, identifies its type, and forwards it to the Jabber server on behalf of Client A. Since the Jabber server maintains its own version of XML for communication, the local server, using its own XML Creator, creates a request that the Jabber server can recognize and hands it over to the Jabber server.



A* -- Socket Connection on Port 5555.
Medium of Communication XML.

Figure 1-1: Instant messaging application architecture

The Jabber Server listens for all incoming requests on port 5222. Once the request is delivered to it, the Jabber Server generates some response. The local server, on behalf of client A, handles this response. Since the response generated by the Jabber server will be in its own XML format, the local server first of all parses this response for its own understanding and makes use of the XML Creator to transform the Jabber server's response to an XML format that the client can understand.

You may notice that the request for Jabber server goes through two distinct stages of XML communication. First, the client request is converted into XML format for the local server, and the local server, after parsing the request, reconverts the client request in Jabber-server format. Similarly, the response the Jabber server returns also goes through two stages. In the first stage, the local server resolves and parses the Jabber server's response and once the local server understands and analyses the Jabber server response s in the second stage, the local server converts the response for the client.

Work flow of the IM application for local client requests

The client request might be meant for the local server itself. In such a situation, communication is limited to the client and the local server, and the Jabber server has no role. All requests the client makes are handled and managed by the local server itself. The request Client A makes is parsed and identified by the local server, and the local server returns the appropriate response.

To parse the requests and responses respective components generate, the appropriate XML parser is used. Thus, in the Java version of the application, the XERCES 1_2_2 XML parser is used. The C# version of our application uses the `XML Document` class to parse the XML that various components return.

Required Programming Techniques

To appreciate this book, you must have a firm grip on a .NET application framework, preferably C# or Java. Besides, sound knowledge of XML is needed, as our application uses XML extensively. The communication messages among various components are in XML-based format.

Overview of programming techniques required for the C# client module

To develop a client module by using C#, you must be familiar with the following classes:

- `Network stream` — This class is used to manage the transportation of data between the client and local server. Whenever some data needs to be transferred, corresponding modules write the data on their network streams, which is picked up by the receiving component and is forwarded for reading.
- `XML Document` — This class reads the documents in XML format. The XML documents sent by the various modules to one another are nothing but requests and responses in XML format.
- `Thread` — The main objective of this class is to implement multitasking. The role of this class in the client module is to keep the session with the local server alive. The `Thread` class also sends and receives messages between the client and the local server.

Apart from the preceding classes, a separate lightweight module is required for transforming the client requests in XML format. In a nutshell, an XML Creator handles the job of creating XML-based requests.

An overview of programming techniques required for the C# server module

To develop the server module by using C#, you must have sound knowledge of the following classes:

- `TCP Client` — This class is used to connect the client to the host.
- `TCP Listener` — This class is used to create a host process that listens for connections from TCP clients. Application protocols such as FTP and HTTP are built on the `TCP Listener` class.
- `Sockets` — This class provides the core functionality of two-way communication between modules. The `Sockets` class in the server module of our application is used by the local server to maintain the two-way communication between clients.
- `Thread` — This class is used to accomplish multitasking. With regard to our application, the `Thread` class is used by the local server to handle the multiple in-coming client requests.
- Web services — Web services can be considered a transport mechanism to access a service. The service you access is the database of the application residing on the local server. A separate module acting as a Web service is built in our application. This module enables the client to access the database the application maintains on the local server to seek out the queries.
- `ADO.NET` — The services in the Web service module are built by using this class. The `ADO.NET` class provides various methods to access the database in order to view, modify, or add data.
- MS-SQL Server 2000 — To build the database required for the application, you must be familiar with some basic operations in MS-SQL, such as creating tables and defining primary and foreign keys in the database structure. Instead of using MS-SQL, you can use some other RDBMS, such as SQL7.0 or MS Access 2000.

Apart from the previously mentioned prerequisites for the client and server modules, MS XML parser 3.0 is required for parsing the XML requests and responses of the server and the client, respectively. Client and server modules can share this XML parser accordingly.

Why C#?

Instead of selecting C#, we could have selected some other language such as Visual C++ or C++, but we've decided to choose one of the latest and fastest-developing languages in the programming world. Besides, C# is an ideal tool for developing distributed applications, especially Web-based applications. Some advantages of C# are:

- Interoperability — C# can easily be used with any programming language and development tool. It can easily be incorporated with programming languages such as Visual Basic and Visual C++. Versioning support makes C# compatible with programming languages that were on the market prior to the launching of C#.
- Simplicity — C# is simple yet powerful. To understand C#, you do not require an elaborate background of the programming techniques involved, as it is a strongly typed language. Syntax-wise, C# provides the simplicity of Visual Basic along with excellent debugger and compilation tools. Various commands in C# are easy to manipulate, as they are hybrids of the ancestral languages, C++ and C, already known to most programmers and enterprises. In simple words, we can say that C# offers the functionality of C++ and C and the simplicity of Visual Basic. Programs written in C# are easier to understand and use than those in any other language. Chances of errors are reduced to a great extent, as C# has built-in functionality for detecting errors before the execution of an application.
- Web-service enabled — Applications built using C# can easily be converted into components and can act as Web services for other applications. It is not mandatory for the application to reside on the same network over which the host application is lying or to have features specifically meant for accessing the service-providing application. Accessing a C#-enabled application is almost like accessing an Internet site for which you must know just the name of the host site. The .NET framework provides an excellent environment for creating and developing reusable components for the Internet and other applications using C#.
- Supported by XML — C# has built-in classes that support XML-based documents. There is no need for incorporating any new component to effect support for XML. Another advantage of XML support is that any C# application that uses XML as a communication medium need not bother about how the consuming application handles data; XML supports a wide range of data types and is not bound to any restriction such as COM data types. Thus, when an application receives data, it can convert it into data types it supports. The only demand on the consuming application is the ability to read the XML.

An overview of programming techniques required for the Java client module

To build the client module in Java, you must be familiar with Java's following classes:

- `Socket` — This class works like the `Socket` class of C#. The `Socket` class provides the functionality to establish two-way communication between the client and the local server.
- `Thread` — the `Thread` class in client module is responsible for handling continuous and uniform responses coming from the local server.

An overview of programming techniques required for the Java server module

To build the server module in Java, you must be familiar with Java's following classes:

- `ServerSocket` — This class is responsible for establishing the connection process between the server and the client. The connection requests that clients reach the server via some port number; in other words, the server listens for in-coming client requests on some specified port number. In our application, the local server module uses this class for catering to client requests for connection on port 5555. The `ServerSocket` class maintains the input/output stream for clients to receive requests and to send appropriate responses to the client.
- `Socket` — The local server module uses this class for maintaining connection with the Jabber server, which listens for all incoming requests on the specified port number (5222). Like the `ServerSocket` class, this class maintains an input/output stream. This occurs so that the local server can establish a connection with the Jabber server, enabling any client request meant for the Jabber server to be transferred over this socket connection; all responses that the Jabber server returns are collected by the local server.
- `Thread` — The `Thread` class that Java provides handles the job of multitasking and synchronizing the message flow occurring between respective components. With regard to our application, the local server module uses this class to handle multiple in-coming client requests for connection. Also, this class synchronizes the messages (requests and responses) exchanged between sessions that the local server for the client and the Jabber server run.
- `Vector` — This class works like a dynamic array for message queuing.

Why Java?

By itself, Java offers solutions to many problems you encounter while developing applications in the dynamically changing information-technology environment of today. Although almost all programming languages possess some unique advantages, Java, with its consistent performance, proves itself the ideal choice for developers. Java is highly efficient in managing time and cost factors, which makes it popular among entrepreneurs. Several other features of Java contribute to the fact that developers favor it. Some of these features are as follows:

- Platform-independent — Java is built to run on various platforms and easily mingles with networking distinctions. This feature of Java eliminates the complexity encountered by applications while running on different platforms. An application built in Java seldom requires platform configurations and networking topologies, making its reach wider and more distributive for the users.
- Simplicity — Among the popular object-oriented languages, Java is far simpler and easier to use than C and C++. Complex features of the C++ language, such as pointers, are eliminated in Java. Multiple inheritance is replaced by the simple structure known as

interface. Memory allocation and garbage collection in Java occur automatically, whereas in C++ you need to allocate memory and collect garbage on your own.

- Distributed and platform independent — Java eliminates the need to write lengthy coding, as modules can be built separately and integrated later to carve out the final application. Besides, once the code is written, there is no need for compiling it over and over again; applications built in Java are compiled into byte-code.
- Security — So far, no programming language has succeeded in providing absolute security. Java is no exception. But Java maintains an upper edge over its rival programming languages as it is equipped with a series of security measures deployed at various stages in consideration of the requirements of the enterprise world and also those of the end user.
- Robust — Java is robust, which means it provides reliability. Java, in the true sense, is a reliable language; it can easily detect possible coding errors well in advance of execution.We know Java does not support pointers and thereby avoids complexities. The reason behind not incorporating pointers is to restrict memory from being overwritten, providing more reliability in data management.

Summary

In this chapter, we explore the history of instant messaging and discuss its potential trends. In the overview, we discuss the requirements for using instant messaging and the advantages of instant messaging. The overview of instant messaging is followed by the most discussed form of instant messaging — Jabber. We briefly outline the working of the Jabber instant messaging server, along with details of some of the major technical aspects of the Jabber server: the main elements involved in XML-based, open-source protocol of the Jabber server and the purpose of integrating an application with the Jabber server. A glimpse of the working of our application and the programming techniques required for developing it is also presented in this chapter.

Toward the end of the chapter, the reasons for choosing some of the programming languages for developing this application are explained, and the merits of these languages are discussed.