1 Introduction to IM concepts

In this chapter

- The benefits of instant messaging technology for Java developers
- How the Jabber instant messaging protocols work
- The core concepts of instant messaging in Jabber
- The benefits and drawbacks of the Jabber protocols

Instant messaging (IM) systems using the Java programming language are poised to become a major part of both consumer and enterprise networking, and will play a core communication role similar to email.

Messaging of course has always been a core feature of the Internet. For example, one of the first and still most pervasive Internet technologies is email. It remains an Internet killer app. However, we all know that Internet communication can be even more interesting and powerful than "plain old email." We should be able to better exploit it as an inexpensive medium for transferring data almost instantaneously. And that is the aim of this book.

Most of the examples and IM issues discussed here are from the point of view of an enterprise developer interested in creating systems for medium or large businesses. This is primarily a bias on my part, as my projects tend to fall into this category. However, the same issues that face enterprises also will be important to any other developer who wants to create reliable, secure systems.

To make the discussion concrete, we'll implement an IM system in Java that complies with the Jabber protocols (which you can find at www.jabber.org). By examining the Jabber protocols in particular, we'll get a real sense for what makes a working IM system tick and why I think they may very well be the best ones to implement in the fragmented field of IM. Finally, the Jabber protocols have suffered from a lack of documentation that I hope this book begins to address.

In this chapter, we'll examine IM systems in general and Jabber in particular. The discussion will be largely nontechnical so we can concentrate on why and where we need IM. Discussion of the technical aspects of the Jabber protocols will occupy the remainder of the book and explain the *how* of Jabber. We will develop a basic Jabber server and client in Java throughout chapters 3–8 to help make the concepts concrete. The chapter closes with a look at the benefits and drawbacks associated with Jabber IM.

1.1 Background on messaging

The idea of instant messaging has been around for a long time. All of the visible IM features like one-on-one chat and group chats existed in other Internet applications long before IM entered the scene. For example, the classic Unix `talk` application allowed users to chat over the network years before IM ever appeared, and group chats have been carried out on Internet Relay Chat (IRC) systems almost as long as `talk` has been around.

The innovation in today's IM systems is the packaging of these separate systems into a managed messaging platform. Jabber takes this further and establishes a *universal messaging address* and the concept of *presence* that can be applied to that address for an even simpler communication experience. A common address and presence are relatively simple technologies that have existed in other forms. However, it's the packaging of the concepts into a single, easy to use, cohesive messaging system that has really caught on. The need for ease of use has increased as more people (many of whom have less technical knowledge or inclination than earlier Internet users) join the network community.

IM addresses are an extension of the well-established email addresses almost everyone has today. However, your email address can be used only to receive email. In contrast, IM ties many message types together using a single IM address (figure 1.1). In an IM system, you can receive a message, chat, or groupchat[1] with a person using one IM address.

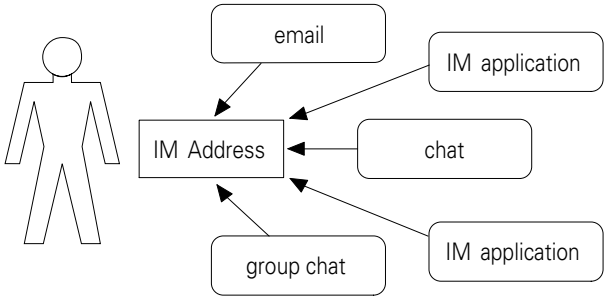

Figure 1.1 IM addresses serve as universal communication destinations. They aggregate many different messaging sources into a single user message endpoint.

Chat and group chat are conversational, and require you to be online at the same time as the person you are chatting with. To make the online rendezvous simple, your IM presence will constantly inform other users when you are online and available for chatting (figure 1.2). IM presence makes communication through IM

[1] IM messages are like memos or emails and tend to be complete like a letter. Chat and groupchat allows you to quickly exchange short text messages in a conversation. Each individual chat or groupchat message is like a sentence taken out of a telephone conversation; it doesn't make sense unless you know what has been said before. The difference between chat and groupchat is that chat is a one-on-one conversation while groupchat is a chat among more than two people.

systems similar to striking up conversations around the water cooler. You can easily see who is available to talk, and casually start conversations.

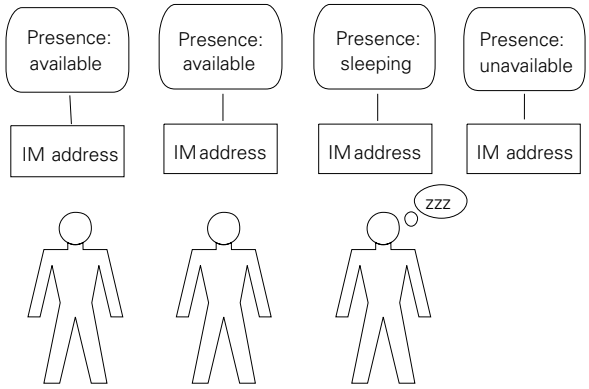

Figure 1.2
Presence shows at a glance who you can communicate with, and the best type of messaging to use (chat, messages, etc).

Older chat systems like Unix `talk` work similarly to the phone system in the sense that they lack presence. When attempting to chat, you have to make blind calls to the person with whom you want to converse, hoping he or she is available to answer the call. Unlike the phone system, though, most people are not available to talk online as much as they are with the phone so the chance of finding someone "at home" online is greatly reduced.

The simplicity and integration of IM systems first caught on with the consumer market. The most successful of these consumer systems is AIM (AOL Instant Messenger) that introduced IM to the mainstream consumer and encouraged its large membership to embrace the technology. The consumer market for IM continues to grow and many opportunities exist for the enterprising developer.

Despite the large consumer market for IM, most of the development community is interested in the opportunities for applying IM technologies in the enterprise (table 1.1). First, enterprises live or die by their ability to communicate within the company and with partners and customers. IM provides new communication channels that are well-suited to many messaging tasks.

Table 1.1 An IDC report's projections for IM usage in the enterprise and consumer markets.[2]

Year	Enterprise IM (messages/year)	Consumer IM (messages/year)
2001	145	262
2003	626	409
2005	1.2B	800

One of the most promising applications of IM in the enterprise is in the area of customer relationship management (CRM) or customer service. First, IM provides yet another way for a company to communicate with its customers. In addition, IM allows you to plug in to the customer's experience to provide better support.

Imagine a computer customer whose software application has just crashed. They start the Help utility that came with the application. The utility is actually a customized IM client. The client joins a chat group dedicated to users of the application. The user can ask anyone online for help. If there is no one available, the client contacts an automated chatbot[3] on the IM system. The chatbot can ask basic questions about the problem and use that information to route the user to the best technical support expert at the company.

Notice how the customer is instantly connected to support resources once a problem emerges. In addition, IM allows you to provide guided assistance starting with free online user groups. Automated chatbots can monitor the customer's progress, make suggestions, and eventually lead them to a company employee that can help. The filtering of simple problems by the free user group can eliminate many customer service incidents that drain profits and reduce customer satisfaction. Since IM is worldwide, your user community has a global reach that is very handy when customers want help outside of your business hours.

In addition to the added communication capabilities, IM provides an enterprise. It provides cost savings. Jabber Inc., a commercial vendor of Jabber software and services, reports significant savings when an IM system is properly integrated into the enterprise.[4]

---

[2] Jennifer DiSabatino, "Win XP to Include Instant Messaging," *Computer World*, June 11, 2001.

[3] Chatbots are special IM client applications that are completely automated. They typically provide services to IM users such as logging conversations, telling you the time, or providing online help. We'll investigate the full spectrum what you can do with chatbots in chapter 10.

[4] See Jabber.com Inc. promotional materials available at www.jabber.com/downloads.

Messaging between individuals is not the only benefit IM can provide to an enterprise. Businesses must also allow computers to communicate with each other. This is true whether the computers are internal to the company such as when accounting applications access customer service databases, or when the computer communication is between business partners in business-to-business (B2B) exchanges (figure 1.3).

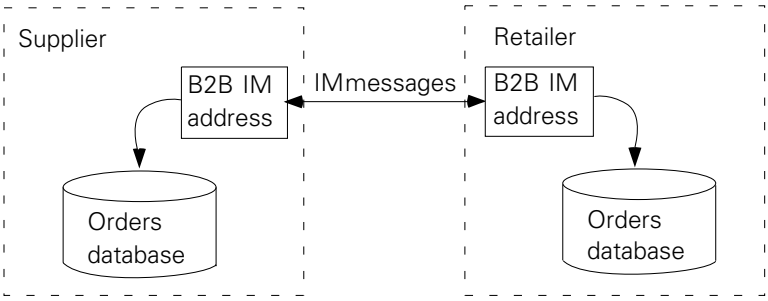

Figure 1.3 IM systems are being used for B2B data exchange including the heavily hyped web services initiatives such as Microsoft .NET.

Using messaging frameworks for computer communication is not a new idea. Products like IBM MQSeries, Micosoft MSMQ, TIBCO Rendezvous, Open Horizon Ambrosia, and Modulus InterAgent have been in enterprise use for years. The benefits of messaging in the enterprise have been well-demonstrated.

In fact, the Java 2 Enterprise Edition (J2EE) includes the Java Message Service (JMS) standard libraries to provide a standard Java interface to messaging systems for enterprise computing. The power and flexibility of Java and enterprise messaging-oriented middleware (MOM) have made the adoption of JMS happen quickly. Most enterprise messaging system vendors support JMS.

For example, imagine you are a telecom company providing local telephone services to residences. You want to create a computer system that will help you handle service problems. Let's consider the scenario where a telephone line has been cut. An IM-based system can log the problem as an IM message and route it to a trouble desk. The trouble desk operator receives the message and knows that she must dispatch a work crew to the site. She can check the IM presence of the work crews, find one that's available, and send a trouble ticket to repair the line.

We can also add expert system functionality to the system by modifying an existing expert system so that it acts as an IM client and receives copies of incoming trouble desk IM messages. It recognizes that a "line down" problem activates

several service level agreement (SLA) contracts. In addition, there are several alternate phone lines that service the same area, and traffic can be switched to these if the problem will last longer than a few hours. The expert system sends more trouble tickets to the trouble desk operator alerting her to other actions she needs to take and offering suggestions on solutions. In most cases, the suggestions are routine and she can simply approve the expert system's suggestions allowing it to follow proper procedures.

This example is not far-fetched. Telecom companies spend millions every year developing and maintaining systems with these capabilities. A few innovative companies are experimenting with messaging technologies as the basis for next-generation versions of these systems that are more capable, and less expensive to create and maintain.

If the industry's predictions for the growth of IM in the enterprise prove true, it is obvious that IM systems will become a fundamental part of any enterprise system, just as web, database, and email servers are today. There are many opportunities for IM systems to expand out of simple messaging to meet enterprise needs, perhaps filling in the roles currently filled by JMS systems. Alternatively, existing enterprise messaging systems may wish to expand their capabilities to include IM. The question facing organizations that need to meet the IM needs of today and in the future is: What IM system should they use?

Jabber is a compelling IM solution that is well-suited to meet today's and tomorrow's IM needs. Jabber is not a particular piece of software. Instead, it is an open, freely available set of protocols for building IM systems. Existing messaging systems can implement the Jabber protocols to add IM to their list of features. Alternatively, new systems are being built from the ground up to support the Jabber protocols and prepare for the rapidly expanding responsibilities being assigned to IM systems.

Alternatively, you can use the existing Jabber network supported by standard Jabber servers as a foundation on which you can build specialized applications (figure 1.4). Jabber takes care of the details of messaging, leaving you free to concentrate on your application. An excellent example of this is the next generation of file-sharing applications[5] and games being built on top of existing IM systems.

[5] One of the most popular, next-generation file sharing systems is Madster (www.madster.com) which operates on top of AIM. You may be more familiar with its previous name: Aimster. These IM-based systems are picking up where Napster left off.

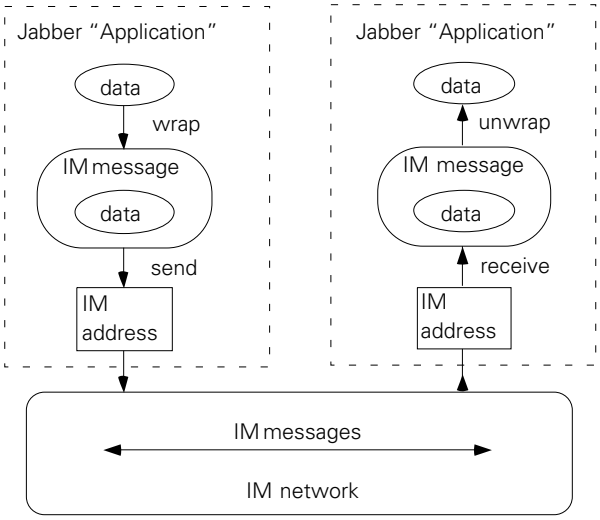

Figure 1.4
Jabber IM systems can serve as a generic messaging network for distributed or collaborative applications.

For example, imagine creating a massive online game where thousands of people participate in a single game universe. Ordinarily, you would create both game clients and a game server. The game server would be hosted on an online game hosting service. All communications between the clients and the game server would be proprietary and would require you to design and implement the various parts from the ground up.

However, if you use an IM system, your game clients can create proprietary data, but wrap them inside normal IM messages. The game can send these messages over existing IM networks like Jabber, to other players. Your game network can host an almost unlimited number of players because it is being hosted by the almost unlimited capacity of the underlying IM network. In addition, your game network avoids the expense and hassle of creating and maintaining custom game servers.

Your server is now another special IM client called a chatbot that maintains the state of the game universe. You don't have to worry about inventing a messaging or routing system, nor does your server have to do anything to support the massive number of connections that IM systems give you for free. You can concentrate on writing your game, leaving the network issues to the IM system.

Sound like something you're interested in? With all of the power that IM systems give you, it may be hard to believe that Jabber is a free, open system, whose inventors want you to use it for your own purposes. Let's take a look at how Jabber became the system it is today.

1.1.1 A brief history of Jabber

The Jabber project began in early 1998 as the brainchild of Jeremie Miller. The project quickly grew and evolved. It garnered wide public attention when it was discussed on the popular developer discussion website Slashdot (www.Slashdot.org) in January 1999.

The core Jabber protocols matured and the 1.0 release of the open source reference Jabber server was released in May 2000. The core Jabber protocols that were implemented in the 1.0 release of the reference server have remained relatively unchanged to this day.

From its beginnings, the Jabber development community has tried to create IM standards and encourage interoperability between IM systems. These cooperative efforts are in direct contrast to the behavior of other popular IM providers that actively work to keep their systems proprietary and isolated from other IM networks.

As part of the Jabber IM standards effort, in June 2000 the Jabber community submitted the Jabber protocols as a Request for Comments (RFC) to the Internet Engineering Task Force (IETF)[6] as part of its Instant Messaging and Presence Protocol (IMPP) standard. The IETF maintains some of the most important Internet standards including those for email and Internet addresses. Unfortunately, the IMPP effort bogged down and as of this writing appears to be going nowhere fast. Jabber has also tried to participate in other standardization efforts like IMUnified (www.imunified.org) and the Presence and Availability Management Forum (www.pamforum.org) with varying degrees of success.[7]

In May 2001 the Jabber community (www.jabber.org) and Jabber Inc. (www.jabber.com) created the Jabber Software Foundation (foundation.jabber.org).[8] The Jabber Software Foundation is an organization similar to the successful Apache Foundation. Its charter clearly shows the Jabber community's dedication to open standards and interoperability.

---

[6] The IETF website is at www.ietf.org.

[7] Jabber's success at creating Internet standards for IM have mirrored that of the IM community in general, which is to say its success has been "zero." I'm not sure if this is an indication of the immaturity and proprietary nature of IM today or a factor inherent to the IM community or its technology. I would hazard a guess that it's the former, but a cynic would claim the latter. In any case, standards are desperately needed and work continues within the Jabber community to promote and push for standardization.

JABBER SOFTWARE FOUNDATION CHARTER

The Jabber Software Foundation shall provide direct organizational assistance and indirect technical assistance to the Jabber Community in carrying out its mission. Direct organizational assistance will include brand management, logistical support, legal assistance, press relations, and communication facilities. Indirect technical assistance will include project management, protocol specification, standards activities, documentation support, and site development. The Jabber Software Foundation will not make technical decisions for Jabber but will help the Jabber Community to make technical decisions more effectively. All activities and proceedings will be openly accessible and published. Jabber Software Foundation Announcement: http://jabber.org/?oid=1309.

The Jabber Software Foundation is still in its infancy but I have great expectations for it. I'm particularly interested in using the Foundation to clarify and formalize the existing Jabber standards into a form that will allow Jabber technology to be rapidly adopted by the wider development community. This book is an effort toward that goal. This book at least begins to address the need for better documentation.

As Jabber moves forward, much work remains to be done. Further standardization work is still under way. In addition, new standards are being proposed to extend the Jabber protocols beyond simple IM to meet the wider needs for electronic messaging of all forms. These changes include better security, and support for enterprise requirements such as transactions, quality of service, and delivery guarantees.

1.1.2 Goals of the Jabber project

In many ways, the Jabber project's goal is simply to build a better IM system supporting real-time presence and messaging. From my discussions with other Jabber developers, the underlying goals seem to really be about defining what "better" means. For most Jabber developers, better means:

---

[8] There is a tight link between the Jabber community and Jabber Inc. (usually called Jabber.com or jc). Jabber Inc. employs most of the "core" members of the Jabber community and developers including Jeremie Miller, the creator of Jabber. However Jabber Inc. has been earnest in its efforts to keep the Jabber community and Jabber standards open. The Foundation is primarily an effort to formalize Jabber Inc.'s commitment to open Jabber standards and provide a legal entity to represent the Jabber community's interests (especially when they don't necessarily match that of Jabber Inc.). Many of the board members of the Foundation are Jabber Inc. employees; however there are enough outsiders on the board to keep Jabber commercial interests balanced against that of the community.

- *A completely open system and standards*—Unlike other systems, Jabber will be completely open, thus allowing anyone to create Jabber implementations at no cost and with no strings attached.
- *Open, XML-based technology*—Extensible Markup Language (XML is an enabling technology with many benefits with respect to flexibility, availability, and ease of use. The use of XML helps to "future proof" the Jabber protocols. At the same time, XML is a well-known technology that people are interested in using. There is also an extremely wide selection of tools for modeling, analyzing, programming, and authoring XML.
- *Interoperability with other* IM *systems*—The value of a communication system increases with the number of people with whom you can communicate. Optimally, a user of an IM system should be able to IM with any other user regardless of the underlying IM system. Jabber will maximize its value by working with as many other IM systems as possible.
- *Simple protocols*—Simple protocols are easier to design and implement. In addition, for most Jabber developers, simple things have an inherent appeal that we strive for constantly.
- *Simplifying the responsibilities of clients whenever possible*—There will be many clients and few servers in any Jabber system. It makes sense to design the Jabber system so that writing clients is as simple as possible. In addition, clients may run on resource-constrained systems so reducing their needs increases the possible types of clients.
- *Control over the system*—No organization, group, or service provider should be able to control the system. Jabber's design allows anyone to create a Jabber server and run their own Jabber network as they see fit.

These unwritten goals seem to be commonly held among Jabber community members. They represent an open, self-reliant community that sees the benefits in sharing technology and knowledge whether you are an open-source or commercial developer. From these basic goals, the Jabber community has produced a feature-rich IM framework.

1.2 What is Jabber?

Jabber IM means different things to different people. End users typically associate Jabber with the Jabber IM system as a whole, just as they consider the Web to mean the entire web system including web servers, web clients and the protocols and data structures that power the World Wide Web. Developers often confuse the Jabber

open source reference implementation of a Jabber server called jabberd[9] for the Jabber protocols that it supports. For the purposes of this book, we'll restrict the discussion of Jabber IM to the Jabber protocols and messaging model.

One of Jabber's most discussed advantages is its truly open nature. The Jabber protocols and data formats are all documented either directly in the Jabber standards documents or as source code in the freely available Jabber reference server. This is in stark contrast to all other major IM systems that use proprietary standards.

Jabber's advantages don't end there. Its use of XML-based data formats exploits the popularity and extensibility of XML. In addition, Jabber uses a simple, distributed client/server architecture. The combination of the simplicity of the basic client/server communication model with the scalability of distributed servers makes it well-suited to the dynamic environments that IM systems will be used in the future.

Let's break down Jabber technology to get an overview of how it works.

1.2.1 Jabber's XML-based data formats

If Jabber's open standards are its primary political and business advantage, its use of XML as its standard data format is its crucial technical advantage. XML is a World Wide Web Consortium (W3C) standard that defines a standard, generic data format for documents. Its primary technical advantages are its simplicity, extensibility, and a compromise between easy human and machine readability.

All Jabber communications involve the exchange of Jabber packets, each being a well-formed XML fragment. These XML fragments can be considered XML subdocuments within the Jabber communication stream. For example, to send a message to iain@shigeoka.com, you would send a Jabber packet that looks like this:

```
<message to='iain@shigeoka.com'>
 <subject>How are you today?</subject>
 <body>Just thought I'd drop a line and see what you're up to.</body>
</message>
```

Even if you don't know anything about XML you can probably figure out what each part of this XML fragment does.[10] If you are XML-savvy, Jabber defines all its packets in standard XML Document Type Definitions (DTD) and uses XML namespaces extensively to define both its own standard packets, and provide

[9] The `Jabberd` reference server development is continuing along two separate paths. `Jabberd` will continue to improve and expand for the foreseeable future. It provides the definitive reference implementation of the current Jabber protocols that we'll cover in this book. A revised set of Jabber protocols, referred to as Jabber Next Generation (JNG), is in development. A new reference server named `Jabberd` will be written to implement these protocols when they are settled upon.

mechanisms for you to easily extend Jabber to accommodate your own custom packets while staying Jabber-compliant.

XML NAMESPACES AND JABBER

XML *namespaces* are designed to isolate XML document tag definitions. As the name implies, a namespace defines a separate space where names are unique. Thus, I can create two namespaces like `baking` and `money` that use the same tag `<bread>`.[11] The tag `<bread>` means different things within the two XML namespaces but there is no confusion between the reuse of the name because each is defined in the context of its namespace. You can fully qualify a particular name by prefixing the normal local tag name with the namespace. So we might use `<money:bread>` to buy `<baking:bread>`.

In the hierarchical XML data model, you can create default namespaces that apply to all child elements by using the `xmlns` attribute. An XML document for our bread example may look like the following:

```
<MyStuff>
 <item xmlns='money'>
   <bread>10</bread>
   <currency>dollar</currency>
 </item>
 <item xmlns='baking'>
   <bread>rye</bread>
   <size>loaf</size>
 </item>
</MyStuff>
```

As it is being parsed, the document creates a stream of data that can be loosely translated as:

- You've got some stuff.
- In your stuff, you've got an "item" entity. All its children will be in the money namespace.
- You have a "bread" entity. Its value is 10. Done with "currency."
- You've got a "currency" entity and its value is "dollar."
- Done with "item." You have left the money namespace.

---

[10] If you aren't familiar with XML, I highly recommend getting a basic understanding of it. You don't need to know anything about XML in particular to program Jabber clients or servers. However, knowledge of XML and its basic vocabulary will make it easier to understand discussions of the Jabber packet formats. A good place to start is *XML Family of Specifications* by Danny Vint (Manning) or *Java & XML* by Brett McLaughlin (O'Reilly).

[11] Bread is a slang term for money.

- You've got another "item" entity. Its children will be in the "baking" namespace.
- You have a "bread" entity. Its value is "rye." Done with "bread."
- You have a "size" entity and its value is "loaf." Done with "size."
- Done with "item."
- That's all the stuff you have.

Notice how namespaces avoid confusion between the uses of the `<bread>` tag. In addition, you can define the data format for `<mystuff>` as simply containing zero or more `<item>` entities. You don't need to know anything about the item namespaces or what they will contain.

Jabber cleverly plays with namespaces to create an extremely flexible protocol. It defines the basics of the Jabber protocol in three namespaces (`stream`, `jabber:client`, and `jabber:server`) containing only a handful of standard entities. However each of the core entities are extended by defining new namespaces. This lets the foundations of the Jabber protocols remain stable and well-defined, yet allows the community to constantly add new namespace extensions including custom namespaces that don't break the Jabber protocols.

In our example, we might define a "my stuff" standard that allows you to send me a list of your stuff. It will always start with `<mystuff>` and contain one or more `<item>` packets. I know about the money and baking namespaces so I can understand what is inside of those items. However, we may decide to add a `playstation2games` namespace in the future. If I get such a packet and don't know about the `playstation2games` namespace I just ignore whatever is inside of the `<item>` packet; nothing breaks. I know it is an `<item>`, but I don't know what it is (other than it's a `playstation2games`). However I can still properly handle it like an `<item>`.

This feature is especially important for Jabber servers who may relay packets with contents they don't understand. Clients are free to invent their own protocols, protect them with custom namespaces, and still send them over generic Jabber networks. XML namespaces also provide freedom to the Jabber community to improve the Jabber protocols while preserving the integrity of the Jabber network.

Unless you've been sleeping for the past year or two, I'm sure I don't need to tell you that XML is one of the current hot technologies. It is a core web standard and is being adopted by enterprise programmers as the lingua franca of corporate data exchange. As with all hot technologies, XML is heavily buzzword-compliant.

In the case of XML, the most touted XML buzzwords are:

- *Open*—As a W3C standard, XML is on track to replace HTML on the web and will be used extensively in future web standards such as messaging (SOAP—Simple Object Access Protocol), and graphics (SVG—Scalable Vector Graphics). The open XML standard meshes well with the open design and philosophy of Jabber.
- *Simple*—One of XML's primary design goals is to maximize the simplicity of creating and reading (parsing) XML formatted data. This simplicity translates into simpler software making it easier to build and support software that uses XML data formats. Jabber exploits XML's simplicity to make it easier to write Jabber software. XML also allows the Jabber protocols to target a wider variety of platforms such as embedded systems.
- *Flexible*—XML is a generic data formatting language. It provides mechanisms like schemas, DTDs, and namespaces to allow users to create customized definitions of XML documents for their own uses. Jabber heavily exploits DTDs and namespaces to harness this flexibility as well as preserve flexibility so users can further extend Jabber while staying compliant.
- *Portable*—XML documents are simple, marked-up text files that can be sent over the network and read on pretty much any platform. XML support transcends programming languages and operating systems. Most common programming environments such as Java, C/C++, Delphi/Pascal, Perl, and so forth support XML with standard libraries.

This book focuses on Java Jabber programming. It is logical to ask if Java and XML are a good combination. The answer is a resounding "Yes!" In fact, a common joke among Java enthusiasts is that "XML gives Java something to do," referring to the fact that before XML many considered Java a solution in search of a problem.

Sun Microsystems (www.sun.com)is aware of the use of Java and XML and created a standard Java XML library package with all Java XML technologies packaged together for easy use. XML parsing support is being added to the standard Java libraries in the 1.4 release of the Java Development Kit (JDK).

In fact, the buzz surrounding XML and Java is a stronger advantage for Jabber and its XML protocols than many of its other advantages. As with many software-related technologies, building a critical mass in the development community is more important than the technical merits of the technology itself. XML and Java is a perfect example of this.

The strong interest in XML and Java has created a huge market of tools, books, programming libraries, and third-party experts (consultants, contract programming firms, etc.). This market legitimizes both XML and Java, and helps to ensure that anything based on Java and XML will be compatible with future technologies. It appears that we'll be using XML for all sorts of programming problems for a some time to come. However, even if it doesn't work out that way, enough people are committed to XML that new technologies will almost certainly continue to offer XML support or conversion tools.

The use of XML in the Jabber protocols is an excellent start in creating an accessible, flexible IM system. However XML alone is not enough to make Jabber truly interesting. The complementary breakthrough with Jabber is its simple, scalable architecture.

1.2.2 Jabber's simple architecture

The Jabber messaging model follows the well-understood client/server architecture. Jabber clients only communicate with Jabber servers in their Jabber domain.[12] Jabber domains break the entire IM universe into separate zones of control, each supported and managed by a separate Jabber server. This is in contrast with most IM systems that use one centralized server for the whole IM system.[13] In Jabber, messages pass from the sender's client, to the sender's server, to the recipient's server,[14] to the recipient's client.

Jabber is client/server

The basic Jabber communication model follows the simple and well-understood client/server architecture model. In client/server systems, the client displays information to the end user and handles user requests. Information is passed from the client to a server that offers well-defined services (figure 1.5).

---

[12] This statement is an over-simplification. Normal Jabber messaging occurs through the server. However, clients are always free to send data directly to each other through any means at their disposal. How to do so, though, is not covered by the Jabber standards. Jabber does provide an out-of-band packet to help clients arrange for these direct, client-to-client communications. However the client-to-client exchange does not occur within the Jabber system.

[13] A central server is the IM system architecture for services like AIM, Microsoft Messenger, and Yahoo! Messenger.

[14] If the sender and receiver both use the same Jabber server, this step is not necessary.

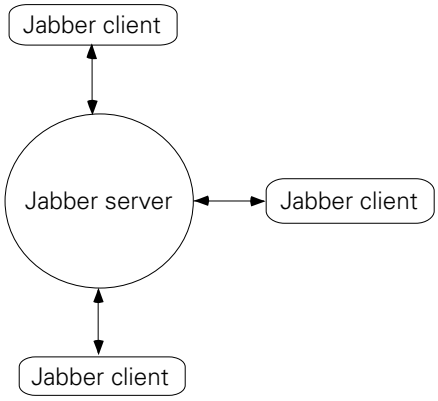

Figure 1.5
Jabber client/server architecture: clients use one connection and only communicate directly with their server.

In Jabber, the client/server model is heavily weighted to favor the creation of simple clients. Most of the processing and IM logic is carried out on the server. Minimal Jabber client responsibilities give Jabber client developers the most flexibility in creating Jabber IM clients that fit the needs of users. Embedded and other limited resource environments can support simple Jabber clients. Heavyweight Jabber clients can concentrate on the user interface and ease of use issues. Finally, the simplicity of creating Jabber clients encourages people to write more clients on different platforms using different programming languages, helping to spread Jabber access far and wide.

Unlike distributed systems, such as those based on peer-to-peer technology,[15] the simple Jabber client/server architecture provides an excellent opportunity to implement centralized control of Jabber domains and provides opportunities for enforcing quality of service guarantees. This is especially important for enterprises that may need to enforce corporate communication policies. In addition, as IM systems are used for mission-critical messaging beyond direct user communication, it will become increasingly important to ensure certain levels of quality of service can be met.

Jabber distributed servers build the Jabber network

Just as the current email system allows separate, distributed email servers to manage email domains, Jabber servers manage a Jabber domain. Like email, Jabber domains are defined by an Internet domain name. So a Jabber server that manages

[15] Peer-to-peer describes a family of technologies that allows network systems to be created without the use of a central server.

the `shigeoka.com` Jabber domain will handle all outgoing and incoming messages for Jabber users in that domain. Jabber addresses, known as Jabber IDs, specify the user's Jabber domain following an '@' character just as email addresses do.

This hub-and-spoke distributed server architecture (figure 1.6) is quite common in messaging systems. Email systems use this architecture. In addition, this is a standard architecture for enterprise messaging servers including the most popular ones that implement the JMS standard.

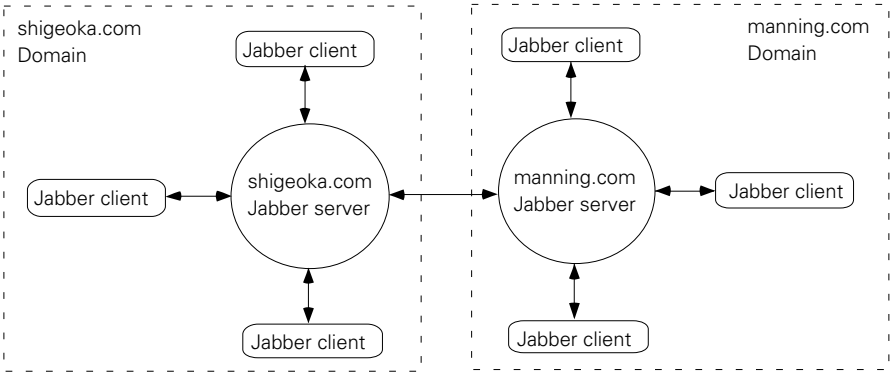

Figure 1.6 Distributed Jabber servers define and control Jabber domains in a hub and spoke architecture. Jabber servers that exchange messages with other Jabber servers create federated domains and expand their Jabber network. Here we see two Jabber servers (shigeoka.com and manning.com) federated to form a Jabber network encompassing both domains.

Creating a distributed Jabber server architecture limits a Jabber server's responsibilities to only handling messaging with its own users and other Jabber servers. A small Jabber server can support a single user and consume minimal resources while large Jabber servers may support hundreds of thousands of users and require large data centers.

Breaking up the server responsibilities so that each server is only in charge of its own Jabber domain helps the Jabber network to grow without requiring massive resources from any particular Jabber server or Jabber domain. As each Jabber domain adds more users, it is in charge of adding its own capacity for handling the increased traffic from those users. You can limit a Jabber server's resource requirements simply by limiting the number of users in its Jabber domain.

This divide-and-conquer strategy helps to level the playing field for Jabber participants. You don't need to be AOL or Microsoft to host your own Jabber system. In addition, it gives each Jabber domain control and autonomy over its own little corner of the Jabber IM network while encouraging interoperability so that your

users can communicate with other Jabber servers. These same advantages have been the primary reasons why the current Internet email system is so pervasive and maintains such a high degree of cohesiveness.

In many ways, Jabber's simple, client/server architecture with distributed servers is old technology. There is nothing really innovative about Jabber's messaging model. In my opinion, this is a major strength rather than a weakness. IM's major innovation is the addition of presence to communication systems. Jabber's innovation is the use of an open XML data format for the data being sent in the communication system. Adding any more innovations at the same time would have probably resulted in a much less elegant and easy to build system.

1.2.3 Jabber's four core concepts

Four central Jabber concepts form the basis for Jabber systems. Before we can look at the Jabber protocols, we must understand these concepts as they exist in Jabber. They're straightforward but it is important for us all to be working with the same set of messaging concepts before setting off into Jabber's details. These four central concepts are:

- Jabber domains
- Users and resources
- Jabber IDs
- Presence

Jabber systems organized into networks and domains

The Jabber universe is broken down into several logical sets and subsets of entities. From the largest to the smallest, we have a Jabber:

- *Network*—All Jabber domains that exchange messages. A network must contain at least one domain.
- *Domain*—A subset of the network containing all entities that handle or belong to a domain. Jabber domains provide local control over parts of the Jabber network while still communicating with users outside of the Jabber domain. A domain is defined by:
  - A valid Internet domain name address.
  - The server that handles connections to that address.
- *Server*—A logical entity that manages a Jabber domain.
- *User*—An entity representing a *logical* message delivery endpoint. Jabber data packets are usually addressed to users, but are always delivered to a resource. Users are managed on the server with user accounts.

- *Resource*—An entity representing a *particular* message delivery endpoint for a user. All Jabber data packets are delivered to resources. Jabber clients play the role of Jabber resources.

A minimal Jabber network is composed of a single Jabber server handling one Jabber domain, and the Jabber clients that use that server, as shown in figure 1.7. Jabber servers can exchange messages using the `jabber:server` protocols. When servers do this, they are federating their Jabber domains to expand the Jabber network.[16]

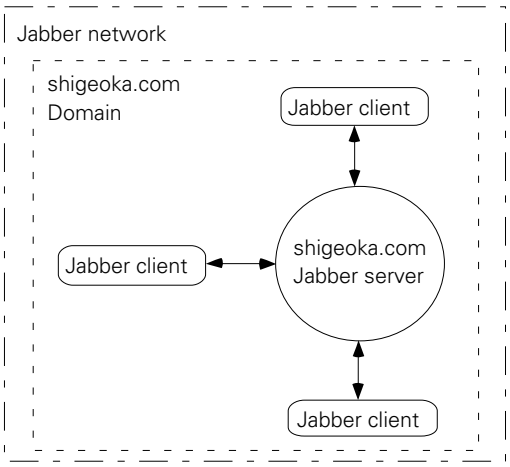

Figure 1.7
A minimal Jabber network with a single domain managed by a Jabber server and zero or more clients.

Server federations are the primary method for Jabber networks to grow. Many Jabber servers are configured to federate with any other Jabber server on Internet. This essentially creates a single Internet Jabber network containing thousands of users.

It is also possible to bridge Jabber systems by using the client protocols to exchange messages between domains as shown in figure 1.8. I'll refer to this process as *bridging* to differentiate it from the privileged server-side access that server federation requires.

[16] Jabber servers find each other by opening connections to the domain name address of other servers (see chapter 9 for more details). Jabber server administrators can play some "DNS tricks" such as allowing multiple physical machines to act as the same logical Jabber server. This trick, commonly referred to as *round-robin DNS*, creates *server farms* in advanced Jabber installations. Jabber servers are also free to develop their own proprietary server-to-server connections in addition to (or as a replacement for) support for standard Jabber server-to-server connections if needs demand.

In jabberd, the Jabber server reference implementation written in C, a hybrid bridging solution is provided by server components known as transports. They have privileged Jabber server access but typically use client protocols to access foreign systems.

For example, one of the most popular and most controversial is an AIM transport. The AIM transport lets Jabber users transparently send messages to AIM users and track AIM users presence, an important capability when most IM users are currently using AIM. However you should be careful about creating transports or bridges as it may violate the usage policies for foreign services. AOL in particular has been resistant to AIM transports.

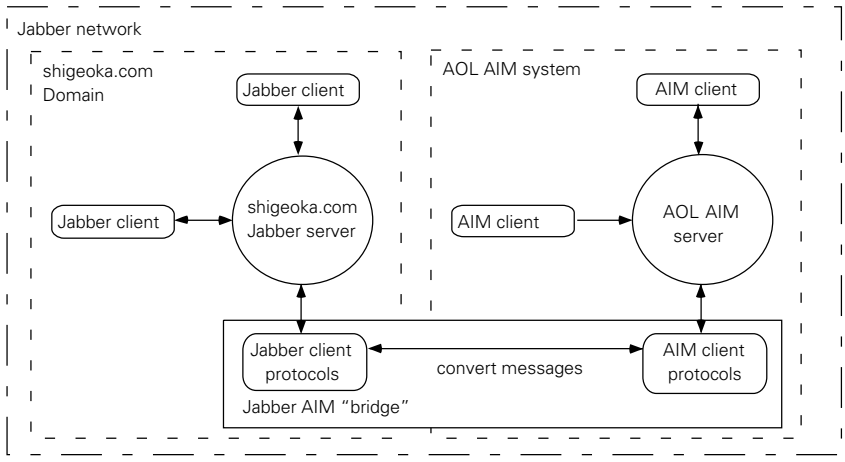

Figure 1.8 Jabber clients or server components known as transports can bridge IM systems by serving as bridges. Here a Jabber AIM bridge acts on behalf of Jabber users to deliver AOL AIM messages.

For example, imagine you have a small business with an intranet local-area network (LAN) that is isolated from the Internet. If you set up a Jabber server on that network and use Jabber clients on your workstations, you have created a companywide Jabber network. You can also create isolated Jabber networks on the Internet or other large networks by preventing your Jabber server from connecting to other Jabber servers. Your clients will only connect with your server, and your server can't connect with any other Jabber servers, creating a separate Jabber network.

Jabber entities: users and resources

Each Jabber domain hosts zero or more Jabber users. A Jabber user is a logical messaging endpoint usually representing a person or user account. However, Jabber users can be anything to which you would want to send a Jabber message. Users can include automated services and gateways to other messaging systems. A Jabber user is addressed by their username. Jabber usernames follow the email guidelines for email usernames.[17]

In the common case where a Jabber user represents a person, it is possible for a single user to be simultaneously using separate clients to access their Jabber server. For example, a user may log in to the server using their PC at work to check messages. When they're away from their desk, they may use their mobile phone to check messages while their PC remains logged in.

This situation is not handled by explicitly by email. Instead, email clients must decide how to trick the server into supporting their simultaneous access. This is usually accomplished by leaving messages on the server so each client receives a copy. It is not an entirely satisfying or successful strategy as anyone that uses multiple computers to access a single email account can verify.

Jabber's designers recognized this shortcoming and provided explicit support for multiple client access. To do this, they introduced the concept of *Jabber resources*. A Jabber resource represents particular messaging endpoint for each Jabber user as shown in figure 1.9.

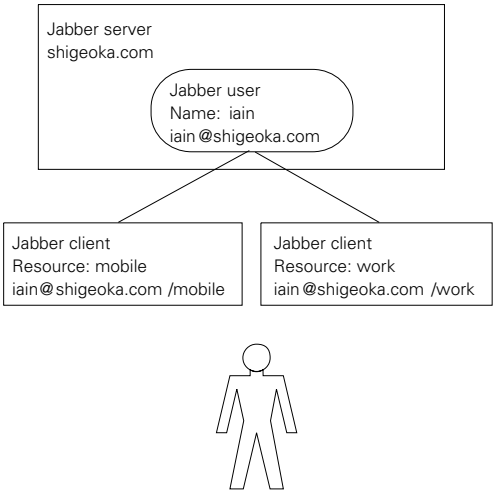

Figure 1.9
A user has two clients logged into the "iain" user account on the Jabber server shigeoka.com. They are represented in the Jabber network by two resources, creating three distinct messaging endpoints for the same user: the generic iain@shigeoka.com and the client endpoints iain@shigeoka.com/mobile, and iain@shigeoka.com/work.

[17] In general, usernames are alphanumeric names that can contain a few special characters like dots, underscores, and dashes, but must avoid other characters like quotes and the @ symbol.

In most cases, you send packets to users. Packets are always received at resources. The Jabber server takes care of properly routing packets sent to a user, to the best resource available for that user.

For example, I want to send a packet to the user "iain" in the Jabber domain shigeoka.com. I don't care how the packet gets to "iain." The server receives the packet, sees that it is addressed to user "iain" and checks to see what clients, if any, are connected. If none are, the packet is stored for later delivery. For this example, imagine that I have two clients connected to the server and logged into my Jabber user account. The clients use different resource names: mobile and work. The server detects this, determines that "mobile" is my preferred resource if available, and sends the packet to that client.

I can override the server's routing by addressing the packet to user "iain" at resource "work" using the address iain@shigeoka.com/work. This allows me to chat with user "iain" at the "work" resource even though the packet should go to the "mobile" resource by default. If the "work" resource becomes unavailable the server will automatically route the packet as if it were addressed to the user. In most cases, clients should accept the default packet routing provided by the server rather than specifying resource addresses.[18]

JABBER IMPLICIT ADDRESSING

The Jabber protocols make certain assumptions about addresses depending on the protocol and the context of a Jabber packet exchange. The server will often override any address you use in packets and replace it with these implicit addresses.

For example, when your client connects with a server, it establishes a Jabber session. This session sets up certain implicit addresses. First, the user's account becomes the default address for all packets sent from the client. If you don't specify a packet recipient, the server will assume that the implicit recipient is the user's account. Second, when a client authenticates with the server as a user and sets its resource, that user and resource are used as the implicit sender address of all packets originating from that connection.

Most Jabber servers will automatically set the sender address of packets to the session's sender address overriding anything the client may have set. This helps the server to prevent clients from sending messages with bad or purposefully wrong sender addresses. If you aren't aware of these implicit addresses you can run into unexpected behavior or errors.

[18] Obviously, there will be many cases where default packet routing is *not* the behavior you want. For example, if you are chatting with someone, you will want to send all messages in the conversation to the same resource regardless of their normal server packet delivery settings.

Jabber addressing using Jabber identifiers

As the earlier sections have shown, Jabber addressing involves properly specifying the Jabber domain, and optionally a username and resource. The Jabber protocols use a standard Jabber identifier, often referred to as a Jabber ID or JID, to format this information into a single, easy-to-use address: user@domain/resource.

For example, if I have a user "iain" on Jabber domain shigeoka.com with the resource "work" then a full URL would be: iain@shigeoka.com/work.

Both the user and resource components of the Jabber ID are optional. The most common form is to simply omit the resource: iain@shigeoka.com. This form is easy to remember and resembles the ubiquitous email address. In fact, I have a feeling that many Jabber systems will simply reuse email accounts so that users can have the same Jabber ID and email address.

There is an assumed anonymous Jabber user associated with the server address. You can address messages to a Jabber server by simply specifying the server name (the empty user part of the Jabber ID implies delivery to this anonymous server user): shigeoka.com.

The most common usage of server addresses is to send packets to Jabber servers outside of your own Jabber domain. To deliver packets from a client to its own server, it's more efficient to not specify the recipient address. The server knows that any unaddressed packets have the user's account as their implicit address. The server will handle these packets on behalf of the user, routing or processing them according to the Jabber protocols.

It is possible to send packets to a resource at the server: shigeoka.com/admin. This form is fairly rare, though, as most server messages are sent from clients to servers without regard for resources.

The compact and familiar format of the Jabber ID makes it easy to remember and use Jabber addresses. The only real danger in its format is the possibility of confusion between Jabber IDs and email addresses. Jabber domain administrators can reduce this confusion by simply using the same address for both so that Jabber messages or emails sent to a particular address will both go to the same user. It is also possible to bridge the two messaging systems so that users can use a single Jabber client to handle both Jabber messages and email, thus further reducing the chance for problems.

The final core Jabber concept is the IM concept of presence.

Jabber awareness using presence

IM systems frequently rely on the "instant" delivery of messages and real-time interaction between clients. These real-time interactions can be simple text chats,

or they can be complex collaborative applications such as groupware and massively multiplayer, online games. Both features require the server and clients to be able to determine who is currently available to receive messages.

Jabber provides standard IM support of presence to indicate each user's online status. In most cases, the simple "available/unavailable" status is enough. However, Jabber allows users to customize their presence status to indicate any presence status such as "away to lunch" or "gone fishing." These custom presence states aren't as useful for automated tools, but they help users to interact in rich and flexible ways.

Jabber also allows you to create rosters, often referred to as buddy lists. This feature lets you maintain a list of other users and their current presence status. Jabber rosters are stored and maintained on the server so your rosters will always be available when you log into the Jabber system.

Finally, the Jabber presence protocols allow you to approve or disapprove presence subscription requests from other users. This feature allows you to protect your privacy and determine who has permission to see your presence status. You can also revoke previously approved presence permissions if you change your mind.

Jabber's presence system is flexible enough to apply to a variety of applications outside of simple user presence management. For example, imagine using a motion detector from a home automation kit to send Jabber updates indicating the "presence" of a car in a particular parking space. You can then write a simple Jabber client to watch for the "car in space" presence update and send you a message if your Jabber presence is set to "chat." That way, if you're Jabbering with your friends on the job and your boss arrives you'll be warned to get back to work, but it won't bother you if you're really working.

Now that we have a basic understanding of the core Jabber concepts, let's discuss the benefits and drawbacks of using Jabber for our Java IM system.

1.3 Benefits of the Jabber protocols

The Jabber protocols offer a wide variety of benefits to developers. One of the most important is its open nature where sharing, experimentation, and cooperation are always encouraged. This has led to the rapid growth of Jabber's user and developer communities. Developers have an amazing level of access to both the major creators of Jabber software, as well the option to influence the creation and evolution of Jabber standards.

From a technical standpoint, Jabber's simple XML packet format provides a nice compromise between a conversational, human readable format, and something that machines can easily interpret. It is easy to hack and explore the Jabber protocols by reading and typing in raw XML using simple tools like telnet. If Jab-

ber had used binary data formats, special tools would be necessary to both read and send valid messages.

Jabber's XML design also permits the routing of any information that can be expressed as XML.[19] With the growing number of XML technologies being brought to fore, especially in advanced business systems, this places Jabber in an interesting position to become a core part of future XML messaging systems that goes beyond simple IM.

Jabber's XML protocols have also been designed to transparently accommodate extensions. Developers can use these extensions to support new applications on top of Jabber such as games and collaborative groupware. Work is already under way in the Jabber community to bridge Jabber systems to other communication systems like pagers and Short Message Service (SMS).

Finally, as we'll see in this book, the Jabber protocols are simple. A small team of developers using a modern language like Java can create a Jabber system in a very short time. This simplicity lets you concentrate on features beyond IM. Whether your goals are to create applications on top of Jabber, integrate Jabber into your current software, or create massively scalable Jabber software, the Jabber protocols won't get in your way.

1.4 Drawbacks of the Jabber protocols

Unfortunately, Jabber is not trouble-free. There are problems with the Jabber standards that you must be willing to accommodate in order to use the technology. The most glaring problem is the relative immaturity of the Jabber standards. In many cases, official documentation for Jabber standards and protocols are incomplete or outdated and the only definitive answer to protocol questions is to check the behavior of the Jabber open source reference server.

The Jabber community is working on addressing this shortcoming. The Jabber Software Foundation (foundation.jabber.org) has recently been formed to help manage the Jabber standards process. It is hoped that before the end of 2002, new Jabber standards under the guidance of the Jabber Software Foundation will provide definitive documentation for Jabber standards.

The Jabber protocols suffer from inefficiencies directly related to its conversational, XML-based nature. Binary data formats can greatly reduce the bandwidth

[19] In particular, the Jabber-As-Middleware and jabber-rpc working groups in the Jabber community (www.jabber.org) are looking into these very issues. The former is concentrating on what is needed to make Jabber enterprise ready, while the latter is specifically looking at transport bindings for XML-RPC over Jabber.

required by a system, as well as provide other features such as error detection and correction. However, the Jabber designers decided that gaining the XML-related benefits mentioned earlier outweigh the resulting inefficiencies. I agree that this was a good decision. However, if this overhead is unacceptable to your application, you may need to search for an alternative.

Another drawback with the Jabber system is its underdog status in a fragmented IM market. The current market leader by a long shot is AOL that controls both AIM and ICQ. Other large IM providers like Microsoft and Yahoo! also have much large user populations than Jabber. These IM leaders have also been resistant to interoperability and open IM standards that could jeopardize their control over their captive IM user communities.

Jabber has been fighting this problem by both pushing standards efforts and reverse-engineering the proprietary IM protocols to create transports[20] to bridge Jabber networks to these networks. Unfortunately, these standards efforts have not been very productive to date.

Jabber's strategy of creating unauthorized transports to bridge Jabber networks to other IM networks may also meet with problems in the future. Until recently Jabber has been a relatively small effort. However as the technology moves forward and larger organizations begin to use it, there may be legal and business problems with bridging to these networks without permission from their operators. Companies interested in using transports should thoroughly investigate the legal aspects of transports before adding them to their Jabber servers.

Finally, the Jabber protocols lack standard enterprise features like transactions and quality of service support. It will be impossible to build mission-critical applications on top of Jabber without adding these features. Fortunately, there is a working group in the Jabber community called Jabber-As-Middleware (JAM) looking into these issues. If you're interested in using Jabber for mission-critical applications, I highly encourage you to join the JAM working group, hosted by the Jabber Software Foundation, and help define these standards.

1.5 Conclusion

Despite some drawbacks, Jabber is one of the best ways for Java developers to build IM systems. Creating IM systems in Java is important as IM is poised to be the next great Internet communication revolution for both consumers and enterprise users. In the next chapter, we'll take a look at Jabber from a technical standpoint and examine how the Jabber protocols work.

[20] We'll discuss transports in more detail in chapter 9.