CHAPTER 2

Peer Architectures

With a clearer grasp of what the p2p concept means, we can now take on the task of classifying peer architectures. This analysis is fairly abstract and non-specific, at least to begin with, because any subsequent discussions of specific technologies and **implementations** must build on a common groundwork.

As a first step, we must define the fundamental terms and the basic p2p models. Including a summary of primary characteristics in this overview proves useful when comparing the functionality of different technologies, and also when we look in Part III at the future development of p2p applications.

We also need to discuss **protocols**, the "glue" that holds networks together. Later chapters dissect particular protocols in considerable detail, so this discussion is not made dependent on any particular implementation. It provides a familiar context in which to place the later specifics. In addition, it can serve as a map to help identify any omissions or simplifications in the protocol design of a particular implementation, which could otherwise be obscured in the mass of technical detail.

Finally, because the focus of the book is mainly from the **end user** perspective, and thus on application implementations where the p2p software is entirely or mostly administered by the user, this chapter is the appropriate place to summarize some of the peer implementations that aren't covered in other chapters. The final section therefore mentions a selection of common peer networking technologies native to different operating systems. These technologies often form the underlying transport layer in host systems that application p2p implementations depend on, yet transcend with their own protocols.

PeerToPeer.book Page 30 Monday, May 20, 2002 12:57 PM

CHAPTER 2 AT A GLANCE

This chapter introduces the architectural models and fundamental terms used in peer-to-peer networking.

From Model to Reality starts with a summary of the conceptual models for p2p, before delving further into detailed terms.

- The *Protocol Types* section analyzes the basics of protocol to introduce some terms and concepts used later to describe particular implementations. *Network Purpose* adds a perspective often overlooked, that of suitability to a particular purpose.

Architectural Models provides an overview of the main types of p2p technologies based on their architecture.

- The models covered are *Atomistic P2P*, *User-Centric P2P* and *Data-Centric P2P*. The *Leveraged P2P* section discusses how distributed implementations might incorporate and blend aspects from these p2p models to improve functionality and performance.

Specific Architectures is mainly a background to Part II and gives an overview of some p2p possibilities not included elsewhere in this book.

- *Native Networking* describes common solutions already built into operating systems, while *Other Application Groups* zips through some p2p and p2p-related technologies not covered elsewhere.

FROM MODEL TO REALITY

First of all, let's summarize into a concise structure the conceptual models of p2p that were introduced in the historical overview in Chapter 1.

Of the many possible ways to structure the information about different conceptual models for information exchange using computers, Table 2.1 takes the perspective of client-server analysis. All the main client-server models are included to give context, while the shaded section in the middle of the table encompasses the established and accepted p2p architectures discussed in this book.

As used in the table, the term "index" means collections of logical links to distributed resources or data, while "directory" refers to collections of logical links to users. Not all applications or networks make this distinction between the two terms and services, but it is a useful one. In either case, this logical addressing is usually independent of the underlying addressing scheme for the network (for instance, the Internet). The latter just functions as a transport layer for the specific p2p protocols.

Although distributed resource sharing, or what might be called the computation-centric model, is often included as a p2p technology, its exchange model actually says nothing about the presence of any p2p architecture. A distributed computation-centric implementation might indeed include p2p characteristics from any of the table's preceding three p2p models, likely data-centric. Quite often, however, node communication is only with the central server that owns the data and distributes the tasks. Next-generation Web or a fully deployed .NET infrastructure, both still in the future, might also include many aspects of p2p but are unlikely as a whole to build on any one p2p architectural model.

Bit 2.1 Sharing distributed resources doesn't necessarily make it p2p.
The defining issues should instead be how these resources or network nodes communicate, and who owns and controls both them and the data.

The Table 2.1 summary also gives a chronological overview of popular trends in communication modes, even though the actual development and use of each listed technology is nowhere as linear and simple as the popularized "paradigm shifts" would suggest. In the earliest **dumb-terminal systems**, for example, the mainframe servers could be interconnected in an atomistic p2p model, refuting the impression that this model came later. Yet it's still true that the PC p2p model did.

TABLE 2.1 *Conceptual models for information exchange*

Conceptual model	Client-side	Server-side
Centralized processing (or "dumb terminal systems")	Display on many (local) dumb clients (terminals)	Data storage, processing, indexing, policies on one server "mainframe"
Client-server (such as corporate NT domain networks)	Processing and display on many smart clients (such as LAN PCs)	Data storage, processing, indexing, policies on one main server
Web server and browser (current paradigm)	Limited processing, display on many clients on WAN or Internet	Data storage and limited processing on many (distributed) servers
Atomistic P2P	**Peer-to-peer models:** The client-server distinction blurs in all these peer models. Data storage, processing, and display on many peers.	No separate servers
User-centric P2P		Directory services on one or few servers
Data-centric P2P		Indexing services on one or few servers
Computation-centric or distributed process	Processing on many distributed clients	Administration, storage, indexing, and display on one or few servers
Next-generation Web (and perhaps .NET)	Data, processing, co-authoring, and display on many clients	Many kinds of distributed services on many servers

The original vision of the World Wide Web by its "creator" Tim Berners-Lee and others was a predominantly data-centric p2p one—a globally hyperlinked content space where no single server had precedence over any other. This vision implied extensive co-authoring and open collaboration by all users. However, much to the disappointment of the first visionaries, the Web instead evolved to have overwhelmingly static and server-centric content, locked to the visitors. There came to be few actual content providers compared to the many users. With users constrained to the role of passive consumers, and little peer communication between them, functionality development of Web browsers focused mainly on snazzy presentation features, not user utility, nor for that matter many of the more basic navigational and collaborative improvements proposed from the very start.

Nevertheless, the importance and use of open p2p models has returned on a new level, user-to-user rather than machine-to-machine, making the developmental chronology implied by the previous table reasonably accurate in that sense.

Protocol Types

Protocol forms the "glue" that holds a network together by defining how nodes communicate with each other to achieve network functionality. We therefore need to examine the terms and concepts that are later used to describe how particular implementations function.

Protocols are specified at many different levels, but we can define some common characteristics concerning how a protocol is implemented. For example, we can consider the kind of *modality focus* of the communication:

- **Message-based** protocol, where the focus is on sending and receiving discrete, packaged and addressed messages. How the messages are carried between two parties is delegated to an autonomous agent, which is like a conversation with messages exchanged by courier, carrier pigeon, or mail.
- **Connection-based** protocol, where the focus is on establishing connections over which messages can be sent. This situation is more like a telephone conversation, where a dial-up connection allows "raw" messages (unpackaged and unaddressed) to be exchanged in real time.

Another, related aspect is the *relative timing* of messages:

- **Asynchronous** conversation, in which one side need not wait for the other side's response before sending another message.
- **Synchronous** conversation, where explicit or implicit dependencies exist between a message and its response. Additionally, there are often internal timing constraints between parts of the same message.

A third aspect is *state*, as applied to the network used for message transport:

- **Stateless** network, which treats all messages the same with no reference to previous messages. The same message will be processed and interpreted the same way every time it is sent.
- **State-aware** network, which exhibits dependencies. Processing one message can influence how a future message will be handled. Memory of past events is preserved—the same message can give different results at different times.

The current Internet TCP/IP connectivity model is connection-based, asynchronous, and inherently stateless. However, many message-based Internet applications introduce various *ad hoc* mechanisms and protocols to track state.

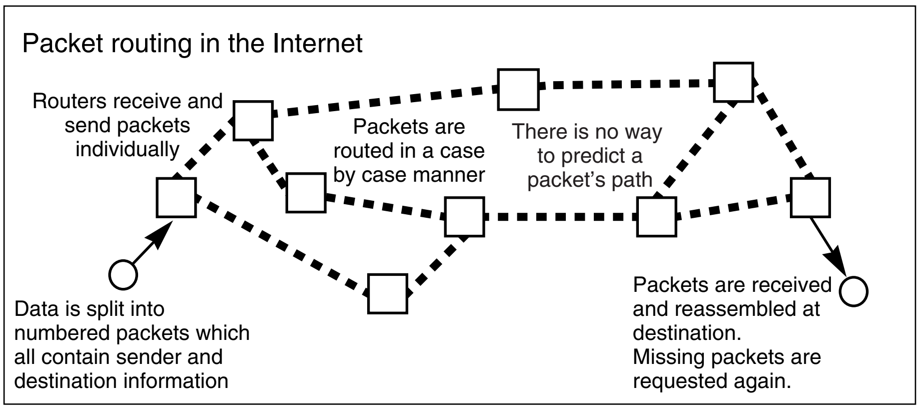
Packet routing in the Internet
Routers receive and send packets individually
Packets are routed in a case by case manner
There is no way to predict a packet's path
Data is split into numbered packets which all contain sender and destination information
Packets are received and reassembled at destination. Missing packets are requested again.

FIGURE 2.1 *Simple illustration of how data transfer between Internet routers is accomplished by asynchronous forwarding of individual packets along whatever path each router at that moment deems suitable*

Constraints of Internet Transport

Internet messages are packaged, addressed, and sent as a number of **packets** determined by underlying transport layers. The packets are routed by independent routers with locally determined priorities and paths. The "sockets" one sometimes sees reference to are logical abstractions of virtual endpoint connections between, for example, server and client, but have no correlation with how messages really travel between them. Figure 2.1 is a simple illustration of that concept.

There is little concern at higher or application levels for how the data transfer is accomplished. The requirement is only that all packets are received within a "reasonable" time, or can be requested again if missing so that the message can ultimately be reconstructed. Hence TCP/IP is characterized as a **reliable transport**, in that it implements handshaking to track and acknowledge packets received.

This pervasive and asynchronous nature of **packet transport** is not a problem unless dealing with various forms of "streaming" content, where packets must be received in a particular order. The severe timing constraints of "real-time" content mean that any missing packets can't easily be requested again. Generally speaking, implementing streaming media over the Internet requires both an acceptance of a buffering delay at the receiver and some tolerance for missing packets. Therefore, streaming connections rely on **UDP** (User Datagram Protocol) as transport, which doesn't try to guarantee packet reception in return for less overhead to manage.

Internet-based p2p applications must usually accept at least the constraint of the asynchronous and stateless nature of the underlying TCP/IP packet routing. They must also build on the current IP addressing model, even if they subsequently construct other directory services with different scope and resolution.

Conversational Modes

The current Internet paradigm, especially the Web, has for some time been predominantly unidirectional in its information flow—from content server to consumer client. This aspect has hampered the development of support at all levels for arbitrary *conversations*.

In the strict sense, most current Internet applications don't easily support real conversations, only requests for predetermined, static content at some more or less permanent address on the network. This is reflected in, for example, Web browser design, which for the sake of efficiency caches content locally but can have problems dealing with sites that generate dynamic content.

That's not to say it's impossible, or even necessarily hard, to support flexible conversations in the existing protocols—p2p applications are a case in point—only that the majority of deployed Internet applications tend to be very rigid and limited in this respect, or to ignore the conversational aspect altogether.

Bit 2.2 P2P innately supports natural, bidirectional conversational modes.
You could use this criterion as a quick litmus test to determine whether a particular technology lived up to its **market meme** of being p2p.

It's mainly for this reason that the deployment of *application talking to application* is still rare and limited in its scope. The vision of autonomous agents deployed to roam the Web, capable of gathering and filtering information according to rules relevant to the interests defined by the user, has long been a compelling one, yet remains largely unfulfilled. To achieve this, not only must bidirectional conversations be a natural mode in the infrastructure protocols, but the information must be accessible in a common structure or metastructure, and the various agents and servers fully interoperable.

Next-generation Web applications do promise a far greater degree of bidirectional conversation support, partly because of proposed extensions to the underlying Web transport protocols, and partly because of a whole range of services geared to distributed authoring and management of content, as exemplified by **DAV** (Distributed Authoring and Versioning). The other part of the equation is that newer content protocols such as eXtensible Markup Language (XML) are designed to meet the linked requirements of common structure, configurable functionality, and ease of extensibility for particular, perhaps unforeseen needs.

Until then, various protocol overlays in the form of existing p2p technologies allow users to retrofit at least some functionality that easily and transparently can support real conversations between nodes in all modes. Some of this is evident in the discussions of architecture models later in this chapter, and in Part II where practical implementations are examined in detail.

Protocols and infrastructure are only one part of the story. The technology must also have an aim, a reason for its design that manifests in the implementation.

Network Purpose

Each implementation of a p2p network has a stated purpose or intent at some level. Usually, this implicit or explicit goal was made early in the design process and so to a great extent determines both just how and for what you can use it.

A given implementation might be admirable in many critical aspects, yet unsuited to the purpose you intend. Many current p2p architectures are relatively specialized. Some focus areas for current p2p implementations include

- Messaging
- File sharing and data sharing
- Content publishing
- Content retrieval (including search and distribution)
- Distributed storage
- Distributed network services
- Distributed processing (and presentation)
- Decentralized collaboration
- Content management/control
- General resource sharing

In addition, the prospective user must evaluate the impact of specific design decisions concerning scalability, **security**, reliability, storage model, and so on.

Actor Conversations

The network purpose also includes the dimension of **actors**, and it is common (and natural) to use the idiom of "conversation" when describing network interactions between peers. Looking at this idiom, we can more clearly see some useful ways of talking about the process of communicating over a p2p network, and some of the essential components of such architecture.

We see three communication situations in p2p network conversations.

- Person to person (P-P)
- Person to application (P-A)
- Application to application (A-A)

Peer applications such as messaging are of P-P type, while file-sharing nodes that automatically fulfill typed-in requests by remote users are basically P-A. Both P-A and A-A will likely grow in importance as we get better at designing automata that can interface with people in "normal" conversational contexts, and with each other to manage routine transactions for particular content or services on the network.

We've seen the same trend in telephony P-A, especially recently in the maturing field of advanced voice recognition services, where human operators are rapidly vanishing and becoming only instances of last resort for fulfilling user requests. Web-based customer care centers have also seen rapid deployment as cost-saving ways to allow customers to fulfill their own requests and manage their own accounts. Online Internet booking, shopping, and banking are other P-A technologies that have seen significant investments and deployment, albeit occasionally with mixed results. Online government, so far usually in the somewhat limited sense of requesting information, retrieving forms, and filing tax returns, is yet another P-A area.

Bit 2.3 A-A conversational modes are the next emergent technology.
Except in very limited or rare cases, there are few examples yet of automated agency that can interact with other agents to fulfill more complex user-delegated tasks.

It's been stated, probably correctly, that *the real transformation of the Internet will occur first when A-A conversations become commonplace*. This transformation would be at least as great a shift in communication patterns as when telephony (and especially cellular phones), e-mail, or instant messaging became popular.

Extensive use of A-A would probably imply an equally extensive deployment of P-A, where people communicate with the local client interfaces to the user applications implementing the delegated user authority—see the following section.

Anyone interested in seeing where both P-A and A-A implementations of p2p might lead should look more closely at the open, overtly protocol-based implementations discussed later in Part II, and at the speculations in Part III.

Properties and Components

Looking at conversation properties in general lets us identify some essential p2p properties and components more clearly. Defining these basic terms is important for later discussions.

- **Identity.** This property simply serves to uniquely identify any user, client, service, or resource, and is fundamental to any p2p context. The practical implementation of a network's identity namespace critically determines or limits the scope of application of the p2p network. Another critical identity issue deals with identifying and tracking individual messages.
- **Presence**. As a property, presence defines the arbitrary coming and going of actors in a dynamic conversation. As a component, it represents the mechanism by which users, applications, services, or resources can manage and communicate information about their current state. Note that "presence" can go beyond the simple state model to convey all manner of context-specific information for a particular conversation.
- **Roster.** One most often identifies roster as a list of frequently contacted identities. The corresponding component provides short-list entry points into a chosen peer community but is often underestimated in terms of its potential automation utility to the user—see "agency". In particular, applications and services can make use of a roster to intelligently share resources, filter conversations, and determine appropriate levels of trust automatically, without the user's constant attention and intervention.
- **Agency.** With relationships to both identity and roster, agency defines the ability for an application to act as an autonomous client. This can mean initiating, managing, coordinating, and filtering conversations that the user would be interested in or has set up rules for—e-mail filtering rule sets are but a simple precursor to agency. In some cases, the agent might act with vested formal authority on behalf of the user, probably leveraged with one or another form of digital signature or certificate.

- **Browsing**. As yet comparatively rare in p2p implementations, the ability to browse available peers, services, applications, resources, and relationships is an important but underrated feature that is only marginally supported in the current paradigm of searching the network or central user/resource database. We find its best known implementation in the form of the browser built into Microsoft Network Neighborhood.
- **Architecture**. In this context, architecture mainly denotes how messages are managed and passed between endpoints in a conversation. One dimension for describing this term is to locate the process somewhere on the scale of client-server to atomistic peer. Another is degree of distribution of services and storage. The relevance of any description depends on the p2p context.
- **Protocol**. Current p2p implementations rest on the packet protocol layer of TCP/IP, sometimes UDP/IP, overlaid with more sophisticated application and session protocols to create the virtual network that defines a particular implementation space. Ideally, the implementation should be fairly agnostic about this layering process and be able to transparently translate across different protocols as required.

This list of components is used in the later implementation chapters in Part II to provide a baseline table for a summary comparison between the different implementation architectures.

Although neither exhaustive nor the only way to examine functionality, these items are a useful way to highlight significant differences between implementations. Maintaining a focus on these primary characteristics helps a user evaluate critical features and discount non-essentials in a given implementation's feature list.

The relative importance of each characteristic is largely dependent on intended purpose and scope of the application, but some general conclusions are still possible. One is the overall importance of *identity.* This might seem trivial—surely you need some form of identity to communicate—yet some implementations totally lack any concept of user identity. In such cases, the implementation works because the purpose doesn't require any defined identity. Others allow arbitrary, even multiple, user-selectable names that are unique only within a particular context.

Even with a well-defined identity, the question remains as to what exactly that identity is tied to—message, person, role, digital signature, software, computer component—each has advantages in certain contexts, and clear limitations in others, particularly in the degree of addressability, security, or portability each offers.

Another conclusion is that some characteristics, such as *presence*, are often undervalued in p2p designs. This view applies even to basic online status, let alone to more advanced concepts of presence which might be applicable to a wider context involving autonomous agencies acting on the behalf of a user.

Bit 2.4 Presence is a crucial issue in human-usable p2p applications.
Qualitatively speaking, human users find a well-supported implementation of simple presence as *the single most valuable and useful feature* in a p2p client.

Autonomous *agency* remains very much in the realm of speculation and vision, but is an enticing goal in the face of the ever-mounting floods of unfiltered and unsorted information that confront a human user on the Internet or at work. Such prospects are dealt with in more detail in Part III.

We next examine the primary architecture models of p2p.

ARCHITECTURAL MODELS

The three architectural models for p2p considered here are

- Atomistic
- User-centric
- Data-centric

The last two are similar in that they usually rely on centralized servers to mediate connectivity based on directories of users or resources and data.

Each model is examined to show its main and distinctive characteristics.

ATOMISTIC P2P

In the atomistic model (AP2P), all nodes are equally server and client. There is no central administration or connection arbiter, although such might be implemented as distributed services within the network. For purists, this model is the original and only "true" p2p architecture.

Each node in the atomistic model is autonomous and fully manages its own resources and connectivity. Figure 2.2 shows the schematic connectivity model.

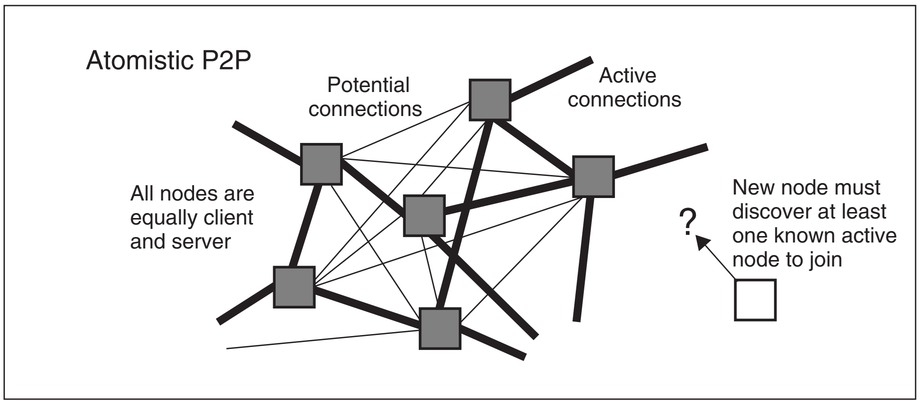
Atomistic P2P
Potential connections
Active connections
All nodes are equally client and server
?
New node must discover at least one known active node to join

FIGURE 2.2 *An atomistic p2p network is constructed from an arbitrary number of nodes. Each typically maintains contact with several others, although virtual direct connectivity can be established between any two.*

The atomistic model contains a fundamental bootstrap problem: *how to join*. The prospective member must somehow first determine which other nodes in the network it can establish connections with. Without a central server, no easy way of determining resource or user availability is apparent in advance.

Two traditional answers are seen to this peer-discovery situation:

- Use a broadcast protocol to send a query to join and await a response.
- Attempt to connect to suitable "known" node addresses that will either accept new nodes or provide further node addresses to try.

When the client has physical connectivity to the p2p network infrastructure, for example in a LAN, it's feasible to use the broadcast method, effectively calling into the dark, "Hello, I'm here. Who's available out there?" The implemented protocol determines how this message is framed to be detected by other client nodes and how these nodes should respond. The query can be a request to join a designated *group*.

Multiplayer games often use this method to establish gaming sessions over a local network. *Ad hoc* sessions are created when several clients detect each other on the basis of broadcast messages plus a matching selection of game, scenario, and other criteria. In these cases, one system becomes the host server for the session.

While broadcast methods can sometimes be used on a larger, general network such as the Internet, the likelihood of successfully reaching active p2p nodes is then much smaller. More practical is to probe known addresses or address blocks for responding clients. The new client can attempt to connect directly to known addresses and through one such node create a response list of other nodes. Alternatively, it can first connect to a published “service provider” node that maintains dynamic lists of active nodes within their horizon. The client downloads this list and proceeds to work through it, attempting to establish a predetermined number of connections.

Bit 2.5 Up-to-date nodelists are a valuable resource in AP2P.
Considerable effort can be invested in atomistic networks to compile and distribute suitable and updated “seed” lists for new nodes, even using external channels.

Once a successful connection is established to at least one node, a client (or other software) can listen to messages passing through the network and build its own list of active nodes for later connection attempts. Furthermore, if the client is configured to accept incoming connections, the user might find that much of the subsequent connectivity is maintained by “incoming” requests from other nodes.

While the formal lack of central administration in AP2P can cause a significant and frustrating threshold to joining the network, it also means that the network is essentially self-maintaining and resilient within the bounds of its capacity. AP2P is therefore the preferred architecture for systems that wish to ensure maximum availability and persistence of distributed data, despite the acknowledged vulnerability in the common practice of distributing nodelists externally.

One or more trusted nodes (or Web sites, or IRC channels) with guaranteed access might in some networks be effectively declared a service provider, mainly to alleviate peer discovery. A fixed address, perhaps of last resort, is then often part of the client distribution and available at first start-up. The fundamental all-nodes-are-equal paradigm is essentially unchanged, except with regard to node discovery.

Lately, some Internet p2p networks are for various reasons moving towards an extension of the service provider concept: a formal two-tier variation of the atomistic model. Particular nodes are then elevated to “super-peer” status because of their proven capacity, connectivity, and reliability. This process is made easier by automatic history and reputation tracking. Reliable or trusted super-nodes can act as list repositories, primary connection nodes, and sometimes search hubs. They provide a sort of backbone network in much the same way that backbone servers came to do for the Internet in general.

Later implementation chapters detail some of the discovery strategies in relation to the specific design goals and security concerns in each case.

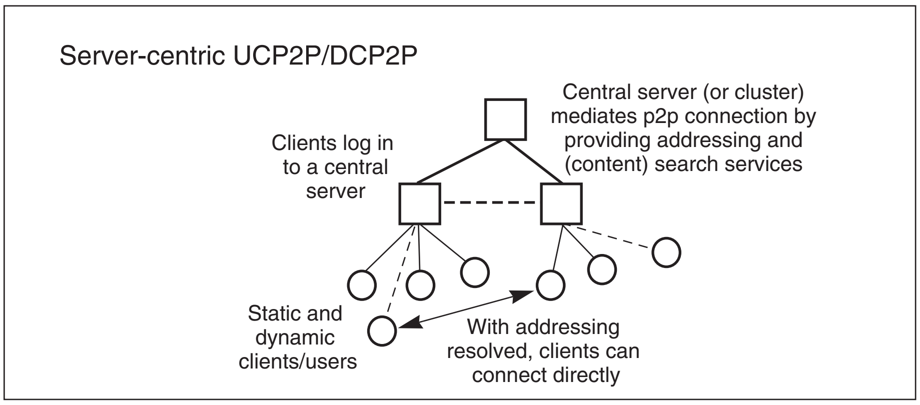
Server-centric UCP2P/DCP2P
Central server (or cluster) mediates p2p connection by providing addressing and (content) search services
Clients log in to a central server
Static and dynamic clients/users
With addressing resolved, clients can connect directly

FIGURE 2.3 *Simple illustration of either user-centric or data-centric p2p. Even though clients usually connect using the traditional infrastructure, the user sees a different, peer-oriented namespace with new services unavailable in the underlying physical network.*

USER-CENTRIC P2P

The user-centric p2p (UCP2P) model adds to the basic atomistic one a form of central server mediation. In its simplest form, the server component adds a *directory* based on (usually permanent) unique user identity to simplify how nodes can find each other. Identities tied to client or other entities are also possible, if sometimes less intuitive. Strictly speaking, one should probably refer to "*directory-centric*" for proper scope, but the term "user-centric" is the one most commonly used.

The directory provides persistent logical links that uniquely (and possibly within a specific context) identify each node, and a translation mechanism to enable direct connection by the user over the underlying network infrastructure. In an Internet context, this translates to the current IP numbers corresponding to the registered and available active nodes. Figure 2.3 shows a simple geometry of this model. In reality, the ongoing direct connections between individual clients would be a much richer web-like structure than this figure suggests.

Clients register with the directory service to announce their availability to the network. Some form of periodic update to this status is tracked so that the server knows when a node is no longer available—either "pings" sent from the client at regular intervals, which is sometimes referred to as its "heartbeat", or responses to service queries. Users might select a specific *transponder* status such as "busy" or "extended away" that will be reported to the server, thereby distinguishing between simple node availability and a more nuanced user-selected availability.

A user can scan/search the directory for other users who meet particular criteria, and from the listing they can determine the current connection address. With a known address, the user can then establish a direct client-to-client connection. The target node registered itself as active and known to be online, so it usually responds right away. Depending on the scope of the server mediation, the connection with the directory server might remain, either to exchange supplementary information or to track current p2p transfer status.

User-centric p2p has proven to be the most popular architecture. The most publicized implementation is surely Napster file sharing. This popularity is despite the fact that the UCP2P model has a far greater deployment in the older instant messaging technologies: Miribilis ICQ ("I seek you", now owned by AOL), AOL Instant Messaging (AIM), and MSN Messenger, to name the best known. Napster as a public representative of UCP2P is doubly ironic, because the primary interest of the users in the system is not to find users, but to find particular *content* on some remote computer. One would therefore have expected an explicitly data-centric focus, where the MP3-tracks constitute the permanent index of content—after all, to all practical intents, users are mutually anonymous in the system, often away, and their identities (which are arbitrarily user-chosen at sign-up) need not be tracked at all. On the other hand, UCP2P makes the current transition attempts from a free service to a registered subscriber service much easier from the technical point of view.

One issue of concern with UCP2P networks is this reliance on central directories, which introduces specific vulnerabilities and raises privacy issues. A user directory can track user behavior and generate personal profiles. Such profiles can be worth considerable sums of money to the compiler if sold to advertisers. They also invoke the specter of user monitoring by various overt or covert agencies.

If we take the example of instant messaging, these solutions also illustrate another downside of centralized server solutions: closed, proprietary standards deliberately kept incompatible by the companies that control them. Independent services like Jabber attempt to work around the barriers by providing modular support for the different standards and so allow users access from a single client. It remains to be seen whether this strategy will become more than just experimental.

Ownership and control of directory services is perceived as increasingly important in the business community the larger the aggregate UCP2P user base grows. This conjecture is proven by how Napster clones were bought up by commercial interests even as they were being closed down for alleged music piracy.

All these issues are discussed later in more detail in both the general (Chapter 4) and the application-particular (Chapter 6) contexts.

Data-Centric P2P

Data-centric p2p (DCP2P) is very similar to the user-centric model, and it can thus be illustrated by the same figure used earlier (Figure 2.3). The distinction made in DCP2P is that the central server maintains an index over available resources, not individual users. The index is a collection of logical links that map the resources to registered nodes in the same way that UCP2P maps identified users.

Again, the term should really be "index-centric" or "resource-centric", but prevailing usage prefers "data-centric". However, just as a UCP2P directory can indirectly access content (think Napster), it's possible to indirectly access individuals from a properly structured DCP2P resource index. In this respect at least, UCP2P and DCP2P can be seen as interchangeable and the choice somewhat arbitrary.

When clients register with the DCP2P server, they mainly provide a list of currently available resources on that node. Although this is usually assumed to be data content of one kind or another—that is, files or documents—nothing requires it must be. Resources can include more abstract entities, such as client-managed services, storage, gateway interfaces, and other intangibles.

Users can in DCP2P architecture search for and access data (or content) held on other nodes. With data in primary focus, it's understandable that different forms of content management solutions turn mainly to DCP2P. It is the area of greatest excitement and promise for p2p in business, as opposed to private use.

Access in DCP2P tends to be more governed by rules than in UCP2P, especially in corporate contexts. Not everyone is allowed to access everything, at any time. Exclusion/admittance policy requires more "intelligence" in server index and client management, so this kind of architecture is still very much under development, targeting enterprise solutions. Furthermore, security issues are paramount because of the deep access into registered nodes—both in terms of data and functionality, and the often sensitive nature of the content.

Leveraged P2P

When dealing with the last two entries in Table 2.1, computation-centric and next-generation Web models, I would like to use the term "leveraged p2p" (LP2P), insofar as such implementations are to be considered in purely peer-to-peer contexts. Each can be combined with the other, and also with elements from the previous three p2p architectures. They might also be fully incorporated into these respective p2p contexts to achieve synergies that can dramatically improve network performance. However, it didn't feel appropriate to include deeper discussions of these models in this book.

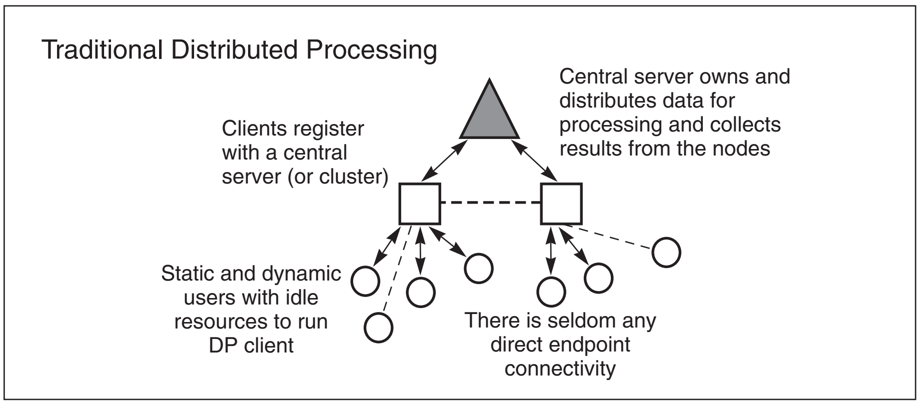
Traditional Distributed Processing
Central server owns and distributes data for processing and collects results from the nodes
Clients register with a central server (or cluster)
Static and dynamic users with idle resources to run DP client
There is seldom any direct endpoint connectivity

FIGURE 2.4 *Simple illustration of traditional distributed processing. Typically little or no communication occurs between nodes. Data is owned by the central server(s), and tasks are sent out to nodes that sent back results.*

Take for example distributed processing. Traditional DP implementations tend to be highly server-centric, as shown in Figure 2.4, because the primary interest has been for a single owner of large amounts of raw data to process it using distributed and otherwise idle resources. Owners of idle capacity, say a PC on a network, install client software and register with the server. They then let the client autonomously request and process data for the server owner, but generally have no insight or control of the process, or the data. The best known example is the SETI@home Project, which tries to analyze the vast reams of data available from radio telescopes in order to detect evidence of intelligent life in the universe.

The thorny issue of data ownership and potential rights to processed results in a public DP setting arose with a comparable distributed effort to identify genes in human DNA. Ostentatiously hosted by a university for cancer research, any results gleaned turned out to be patentable and exclusively owned by a private company. When this fact was made public, many users left that DP network in disgust.

Using DP in a proper p2p context is comparatively rare as yet. In fact, the usual definition of "large processing tasks segmented into component blocks, delivered to a number of machines where these blocks are processed, and reassembled by a central machine" is pretty clear about the strict hierarchy model.

One potential network-specific application of DP based on p2p is distributed indexing, where nodes in a DCP2P network assume responsibility for a portion of the total indexing task and work together in fulfilling search requests. Various distributed search schemes are possible, index-based or instance-compiled on varying scope.

A combination of DP and the new Web, sometimes called "Web Mk2", offers other prospects. In the p2p context, one can envision autonomous, roaming Web agents opportunistically using p2p nodes as temporary hosts as they go about their assigned information-gathering tasks—a kind of benign Internet "worm". These and other visions of future potential are dealt with in Part III.

SPECIFIC ARCHITECTURES

This section describes how some common p2p-capable implementations, especially those built into operating systems, have applied one or another of the general architectural models. We examine how closely each implementation adheres to a given model, and of particular interest is to note how each has tried to solve issues relating to user convenience, reliability, scale, and so on.

These examples are intended as illustrative of concept, not as exhaustive analysis, and they mainly provide a general backdrop to the detailed analysis of more recent technologies in Part II. Although capable of creating legitimate and often perfectly adequate p2p solutions in the LAN context, they aren't in themselves necessarily practical solutions for deploying peer networks today. Modern p2p solutions are based on **open source** applications that create virtual networks and can run on any network-aware system, which gives numerous advantages compared to proprietary solutions that depend on particular operating systems.

Before looking at the p2p solutions that can build on any operating system, the following section takes up native networking abilities in the main operating systems encountered by individual users today. I discuss "OS-bundled" networking in this limited way, because it falls somewhat outside the intended scope of this book.

NATIVE NETWORKING

By "native networking" we understand the inherent ability to connect to a network (of peers) using only those components already present in the respective operating system, possibly with further installation of some non-default ones.

Such networking ability enables "instant" peer networking on the machine level, at least for messaging and resource sharing across the network. In today's systems, this native capability almost always means Internet connectivity, in addition to transport support for local networks. Native networking is distinguished from the application-level p2p networking that is the main focus of this book, and it is often neglected in discussions of p2p technologies.

Application-level solutions communicate on top of the established machine-level networking, but can be independent of the latter's addressing and peer or non-peer ability, and are therefore seen as complementary enhancements.

Unix to Unix

As mentioned in Chapter 1, the first computer networking was between mainframes. It quickly evolved to communication between Unix machines, which early had a basic peer protocol called UUCP (Unix to Unix Copy Protocol).

Because UUCP is a standard copy process between all Unix machines that can be applied to any content, it was also used to transport messages. Many newsgroup servers still rely on UUCP to transport messages to and from other systems. Although ancient and not especially efficient, its main merit is that it's always available, whatever the Unix system. Early chat, e-mail and the newsgroups (on **Usenet**) were built on top of this protocol. Unix defined the interoperable standards for all e-mail support, mailbox format, and applications—and were inherited by Linux.

Reference to Unix characteristics common to a variety of implementations of either Unix or derivatives like Linux is commonly denoted by writing "*nix".

Peer Networking in MS Windows

One of the better design decisions in 32-bit MS Windows was the integration of generic networking components. Once the hardware, protocol, and workgroup configurations are properly set up, basic peer networking is essentially plug-and-play, whatever the mix of Windows platforms—95, 98, ME, NT, 2000, or XP.

Network components however are not default in Windows (prior to XP) but are installed and configured whenever hardware or software that requires them are added to the system. Installing for example a network interface card or a modem not only requests the appropriate device driver, but also sets up the corresponding client and protocol layers to handle network abstraction. Figure 2.5 illustrates the model used, with reference to the OSI protocol layers.

The supported network protocols are not just Microsoft's enhanced version of network BIOS, NetBEUI, for Microsoft Networking, but also for Novell NetWare (IPX/ISX) and Internet-compatible networking (TCP/IP). Others can be added. Network support is largely transparent to the user.

The default configuration prior to Windows XP relies on the proprietary NetBEUI protocol, although it is relatively painless to reconfigure it to TCP/IP from the Network Properties dialogs—XP defaults to TCP/IP. The advantage of NetBEUI

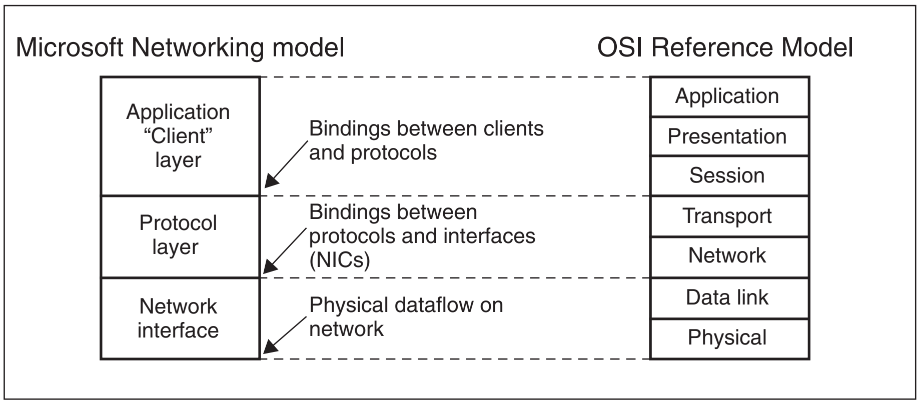
Microsoft Networking model
OSI Reference Model
Application "Client" layer
Protocol layer
Network interface
Bindings between clients and protocols
Bindings between protocols and interfaces (NICs)
Physical dataflow on network
Application
Presentation
Session
Transport
Network
Data link
Physical

FIGURE 2.5 *Microsoft Networking models the network as three layers and two bindings, here compared to the OSI model. The user must configure one or more appropriate binding paths Client-Protocol-NIC for each application and network used.*

in the small local network is that it doesn't require any setup apart from uniquely naming each machine and assigning it to a common workgroup. Once connected, machines will "see" each other using the integrated network browser. The integrated network browser makes access of remote files and resources transparent, and to the user as easy as accessing local files and resources.

Bit 2.6 Workgroup networking in Windows is an atomistic p2p architecture.
The primary focus of Windows native networking is resource and file sharing, with shares managed by the individual machine's administrator/user.

As each machine's resources (such as hard disk partitions or printers) are locally declared shared, they can be accessed by other machines in the workgroup. There is no built-in central administration. Resources and access are controlled locally for each machine by the respective user (in Windows NT, by the local Administrator account). In the corporate or home LAN case, all the machines tend to be physically administered from external notes and lists managed by one person.

It's straightforward to use machines running some flavor of 32-bit Windows as p2p nodes in a NetBEUI-type LAN with up to perhaps 10 or 15 PC workstations. Beyond that scale, the complexities of consistently administering names, shares, and permissions easily get unmanageable.

Microsoft later implemented a scalable server-centric model for Windows Networking based on NT Server, where resource and access control is handled by a designated Primary Domain Controller (PDC) in the LAN. Users must then first log in to the PDC using their local client before gaining access to the network. Using Microsoft domains adds considerable complexity, but also the kind of power and centralized control that larger corporate networks usually need. This kind of domain-centric network scales tolerably well to thousands of nodes.

The default NetBEUI protocol is furthermore constrained to a small physical LAN, because it only handles network data frames with explicit hardware addressing. The price NetBEUI pays for its simplicity is the inability to cross network boundaries. To route across virtual networks, you need support for software addressing, such as in TCP/IP's packet addressing. Hence, for maximum flexibility, Windows systems should be configured for TCP/IP. Given the continued Internet focus of Microsoft, TCP/IP might become the only networking option in future versions of Windows.

With TCP/IP as the protocol, it's possible and often desirable to install support for point-to-point tunneling protocol (PPTP), which is an **encryption**-protected virtual private network (VPN) connection between NT servers, or between a Windows client and a server. This protocol is primarily intended to provide secure access to corporate networks from external, dial-up users. It however could also be used to construct a virtual, distributed, and private p2p LAN of up to 256 connections per node.

Home LANs have become more common in later years: several machines in a p2p network sharing an Internet connection, possibly through a cable router. A better understanding of p2p principles even in Windows can greatly enhance the utility of such home clusters by allowing different approaches to how data and resources are deployed, and perhaps shared from outside the home as well.

For the purposes of this book, it's assumed that most readers have Internet connectivity with either some flavor of Windows or Linux, or a Mac, on which system they intend to install application-level p2p solutions.

Peer Networking in Apple Macintosh

Apple's Macintosh early included peer network capability in their operating system. Native support for ubiquitous 10/100 Base-T Ethernet makes physical network connection easy.

The Mac supports either proprietary AppleTalk or open TCP/IP protocols, and can natively build peer networks. Stringing together some Macs with AppleTalk is the easiest route, but easy comes at the price of the power and sophistication that more complex protocols give. AppleTalk is in addition known as a "chatty" broadcast protocol that doesn't scale very well to larger networks.

The early dominance of Mac systems in corporate and educational environments has waned over the years, although they are still fairly common in the latter. The proprietary architecture, solutions, and protocols have always been an impediment to broad interoperability with other platforms, networked or not.

A number of solutions exist to interconnect Macs and PCs to the respective proprietary protocol networks, but TCP/IP is usually the protocol of choice. As with Windows, Mac p2p applications install on top of the current transport. The newer OS X is a Unix derivative, so it supports many *nix tools and applications—with broader support for p2p than older Macs.

OS/2 Peer Networks

IBM's OS/2 is no longer a current operating system for the average user, although not so many years ago, OS/2 Warp was billed as the next dominant desktop OS. It might have taken a significant share of the market too, if IBM hadn't so abruptly dropped support for it and instead begun to bundle Windows.

OS/2 is worth mentioning because it lives on in some corporate networks and among a core group of enthusiasts. Even today, some still maintain its suitability for office use, with words much like this:

> *If you want a consistent, friendly interface that has the power to run the office, run your old DOS/Windows programs, and connect to the outside world (all simultaneously), then OS/2 Warp Connect is worth a look.*

The last OS/2 Warp versions, Connect and v4, were true 32-bit, multitasking and network-aware operating systems roughly comparable to Windows NT or Linux. Warp consumes less resources than NT, more like Windows 95, and can run most Windows programs intended for the early 32-bit extensions. For our purposes, the question is how well OS/2 supports peer networking.

Network software setup for OS2/Warp is similar to the Mac in its ease of use. Finding a working NIC driver can sometimes be problematic, given the lack of vendor OS/2 support for newer hardware, but the rest is an easy walk. Other platforms may match up to Warp in any one area, yet IBM covered its bases much better overall.

IBM's proprietary networking protocol, OS/2 Peer, is limited to sharing resources among machines running Warp Connect. However, TCP/IP is also supported, albeit an older version that's less easy to configure. Protocols are session

TABLE 2.2 *Main OS/2 Peer Networking components*

Component	**Function**
Sharing and Connecting	A program that enables you to connect to the resources of other users, and declare which of your resources are available to other users, and to what degree.
Network Messaging	OS/2 Peer's internal e-mail system.
Peer Workstation Logon / Logoff	Logon and logoff service for network access.

specific, so you can log on to OS/2 Peer, IBM LAN Server, Novell NetWare, Microsoft LAN Manager, Windows for Workgroups, and TCP/IP networks simultaneously.

The OS/2 desktop has three network-related folders: OS/2 Peer, Network, and UPM Services (the latter for user and password maintenance). The OS/2 Peer folder contains all of the good stuff, the most important being Sharing and Connecting, Network Messaging, Clipboard and DDE, Information, and Peer Workstation Logon/Logoff—as explained in Table 2.2.

Linux Networking

A Linux installation is inherently a full-featured server in the Internet networking model, natively supports TCP/IP, and also includes all the associated client software. Linux is based on Unix, which to all intents and purposes *is* the Internet.

Linux exists in a variety of branches and distributions, all similar and generally interoperable, but with different configurations and purposes. Full-scale Linux installations are admittedly not easy to master, but they do have all the power and options for networking you could possibly want. A network of machines running Linux can therefore easily function as both client-server and p2p node using the full array of software developed for the Internet.

This kind of system, partly due to the more "experimentally involved" attitude of the typical Linux user, readily participates in many p2p contexts, locally and over the Internet. One such common context is to return e-mail to the p2p model, because Linux machines can, and often do, each run the server software for sending and receiving e-mail in a variety of protocols, messaging, and sharing files or other content. That way Linux users easily turn their machines into p2p endpoints for a broad range of services. Similar functionality is available in Windows and other systems by adding comparable third-party software, but Linux support is native.

QNX Networking

QNX is a mature distributed *nix-like operating system, generally found in but by no means restricted to embedded real-time systems. For several decades, QNX development has sort of paralleled Linux, and can on a PC generally emulate or run much Linux software.

QNX has native networking at several levels, including support for distributed processes and modules, and it of course supports the ubiquitous TCP/IP. It's however rare to find QNX on a desktop PC outside of special developer contexts.

OTHER APPLICATION GROUPS

This chapter ends with a brief tour of application-level solutions that might or might not be strictly p2p, but are related in concept at some level.

Peer Servers

As a kind of catch-all, the term "peer servers" can be used to designate various forms of Internet or LAN servers that maintain p2p connectivity with each other, while serving a host of clients in a traditional client-server role.

Most of the discussions about p2p networking are equally applicable to the server-to-server p2p role, even though little is said about this role in this book. This is in part because the main focus is on the end-user perspective, and not so much on software such as traditional servers that are not administered by the user.

Nevertheless, it's also true that the node application in p2p technologies is quite clearly a "peer server" because all the nodes participate in this role. The server role becomes more explicit in cases like Mojo Nation and Freenet (see Chapters 8 and 9, respectively), where the node software does have a clear client-server role towards separate client software (at the user endpoint) running on the same machine.

Internet Relay Chat gets a brief mention here only because its relay servers function p2p with each other. The IRC client-to-client chat transactions almost exclusively go through the servers, so these relationships are not p2p. Nevertheless, IRC one-to-one chat and many-to-many (or chatroom) discussions can be a method to discover potential peers for other direct p2p connectivity. IRC support is therefore a common extra component in many atomistic p2p technologies. A multi-transport chat client such as Jabber (see Chapter 6) is especially useful in such contexts because it supports several other p2p messaging and file transfer protocols in addition to IRC chat and its own open client and services protocol.

HailStorm

At the time of writing, Microsoft's new and controversial p2p entry, dubbed HailStorm, is barely past prototype status. Although details will surely change over time, Hailstorm and its associated services appear clear enough in principle that mention of this implementation should be made.

Launched as the first real .NET (pronounced and sometimes written as "dot-NET") initiative in March 2001, it is described as a set of user-centric *Web services* that will "turn the Web inside out". The concept assumes some aspects of the traditional client-server relationship. HailStorm defines a basic network framework around which third-party developers are invited to write applications that rely on user identification. The approach has been described by Microsoft in this way:

> *Instead of having an application be your gateway to the data, in HailStorm, the user is the gateway to the data.*

Unusually for Microsoft, the framework rests on a set of open standards, XML and Simple Object Access Protocol (SOAP), rather than proprietary protocols. It remains to be seen, however, whether these assimilated open standards will in future be extended in proprietary ways. HailStorm's security protocol, based on Kerberos, has already been extended by Microsoft, with unclear consequences for continuing its open and interoperable characteristics.

While officially described as an open p2p system, closer inspection shows that HailStorm sits in an uneasy balance between the centralized closed server and the open p2p models. Depending on how the final implementation designs play, Hailstorm could turn out to be the largest client-server architecture ever devised, with rather minimal peer focus overall.

The core concept depends on a user-centric, or strictly speaking an *authentication-centric* server model. This has the audacious goal of centrally validating any and all Internet user identities in the world! It would (by way of the Passport service) mediate and authorize valid user access not only to all distributed Web services, but also to *locally installed software*. The thought is that software registration management will be yet another service sold by Microsoft.

It should be noted that the concept of "personal identity" that HailStorm deals with is not just a simple *who am I*, but is at minimum a three-tier structure that uniquely specifies the *individual*, the *application* that the individual is running, and the *location* where that software is running. This information is encrypted into Kerberos application requests sent to a Passport server for **authentication** checks.

Passport defines identity, security and description models common to all services. As currently deployed, Passport identity is keyed to the user's e-mail address, whether existing or created on the Passport server just for authentication purposes. The official motivation for this massive central control is that all the proposed .NET services, especially commercial and banking, are defined as tied to a unique user identity that has to be administered globally. Microsoft is promoting Passport as a one-stop service for identifying people at online outlets.

An example of large-scale, public use of the Passport service for user authentication is the multi-user game *Asheron's Call*, a Microsoft Zone gaming site that in December 2001 began using the new identity verification system. Users that log in are shunted to a Passport server to verify their identity before being allowed into the game. It's not clear whether continued participation depends on Passport tracking user presence, but that feature is mentioned in other Passport contexts. Windows Messenger (WM) relies on presence tracking with Passport authentication.

Central Authentication

Crucial to the Passport concept, as its name implies, is that the distributed services and software honor the identification protocol. Significant is the list of standard functions, such as myAddress (electronic and geographic address), myProfile (personal information), myBuddies (contacts roster), myInbox (e-mail), myCalendar (agenda), and myWallet (e-cash), to name the first offering.

Needless to say, not everyone is comfortable with the idea of one company (with a less-than-reassuring track record for online reliability and interoperability) totally in control of individual and corporate public identity at this global level. One worry is that all identity credential transactions, and hence by extension most commercial transactions, would require participation of central Passport server(s).

Early criticism of the system can be summed up in the sentence:

> *HailStorm is the business idea of getting you to give up your identity to Microsoft, who will then rent it back to you for a small monthly fee.*

Undeniably, there is at least one obvious ulterior motive for implementation of Passport: central identity validation for the new pay-by-use, personal-rental model for software-as-service that the company is adopting to replace the previous user licensing model of software-as-product. Deploying the infrastructure for a single global identity for each individual makes it much easier to manage registration and payment.

Significantly, the HailStorm infrastructure moves the revenue model from selling or licensing proprietary products on a proprietary platform, to pay-per-use fees culled from anyone running anything on any Internet-connected platform. HailStorm is a very "egalitarian" commercial venture in this respect—it asks both developers and users to pay for access, though the nature and size of these fees are far from worked out. Assume some form of periodic subscription or pay-by-use.

Privacy groups and others have meanwhile complained the service lacks adequate safety measures for securing sensitive consumer information, charges that Microsoft denied, despite the discovery in October 2001 of a security flaw in the Wallet part of the Passport system that could have exposed confidential user financial data to intruders. Glitches in the transition of the Gaming Zone site in December 2001 reawakened public skepticism.

Nevertheless, momentum is growing for Passport as more companies sign on and switch to a Passport-mediated log-in, use the MS bCentral portal service, or build new applications that lean on Passport services. Needless to say, Microsoft's applications for Internet communication, for example, Exchange Server or the IM client Windows Messaging, all tie into the Passport authentication scheme—sometimes as an option, sometimes as a necessary component.

Open Access

The distributed aspect of HailStorm is described as "open access", meaning that in principle any minimally connected device that is compliant with the XML Web SOAP framework can access applicable HailStorm services. No Microsoft runtime or tool is required to call them.

This concept seems clear enough at the client level, but less so at the server level. There, Microsoft has only vaguely stated that servers running on non-Microsoft operating systems like Linux or Solaris will be able to "participate" in HailStorm; the degree of actual integration has not been specified any further. In September 2001, Microsoft opened the gates by announcing that .NET will allow third-party identity providers to compete with Passport. This move is promising, albeit surprising at this early stage of deployment because the detailed strategy of the company can only be the subject of speculation. It however is strengthening the utility of HailStorm (which, by the way, has been renamed ".NET MyServices").

A good place to start if looking for more overview material on the many aspects of .NET and HailStorm is a Belgian site at I.T. Works (www.itworks.be/webservices/info.html). Another, more technical resource is DevX (www.devx.com).

In the next chapter, we leave the overview for more practical matters.