Shifting Paradigms with the Application Service Provider Model

Lixin Tao
Concordia University

Built upon developments in selective outsourcing, application hosting, and browser-based computing, the ASP model will shift the purchase of computing power from product to rented-service form.

In the past four decades, several technological breakthroughs have made it feasible to sell computing as a service rather than a product. Supercomputers and clustering technologies have made huge amounts of raw computing power available, while time-sharing operating systems have made computing resources a divisible utility. Personal computers have educated generations of home and office computing users, and they now depend on such devices.

Meanwhile, the Internet has become the world's largest data and computing-service delivery infrastructure, offering a new platform for network-centric computing. The World Wide Web has enabled the widespread growth of electronic commerce, while Web browsers have become universal graphical user interfaces for Internet-based services and thin clients. Component technologies have made it possible to produce huge numbers of reliable distributed software applications that benefit from specialization and greater economies of scale.

Recently, *application service providers* (ASPs) have begun marketing the *ASP model*, which uses the Internet or other wide area networks to provide online application services on a rental basis—commercially delivering computing as a service. Figure 1 shows projections that indicate the ASP market will grow to more than $20 billion in 2003, yielding a compound annual growth in excess of 80 percent.[1] Yet for the ASP model to become the computing industry's mainstream paradigm, ASPs must make significant breakthroughs in networking infrastructure, computing technologies, and rental-based cost models and financial services.

If successful, the ASP model, combined with the effect of distributed component technologies, will foster a new era of competitive network-based computing. Standardizing distributed components will make integration of applications with distributed components a common practice. Standardizing common application data formats or application programming interfaces will further break the ASPs' monopoly, thereby instilling new energy into the competitive network-centric computing platform. A networked economy will be possible that can easily integrate modular services from different providers.

To date, the ASP model has been limited to industry applications, with the research community showing little interest in it. By surveying the relevant technologies that support ASPs, highlighting the technological challenges the ASP model introduces, and studying the impact of the ASP model on the general computing infrastructure, I hope to increase awareness of this important development.

0018-9162/01/$10.00 © 2001 IEEE

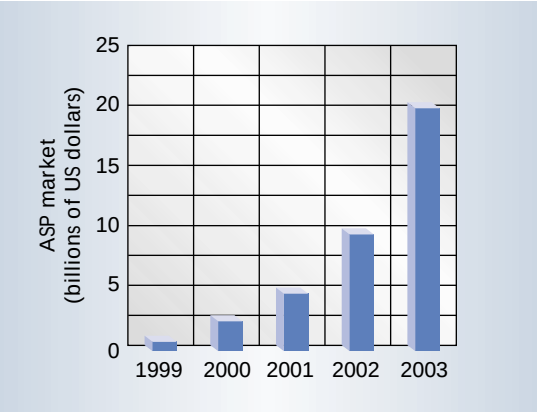

Figure 1. ASP market forecast. If it meets analyst projections that it will reach $20 billion by 2003, the market will enjoy an annual growth rate of more than 80 percent.

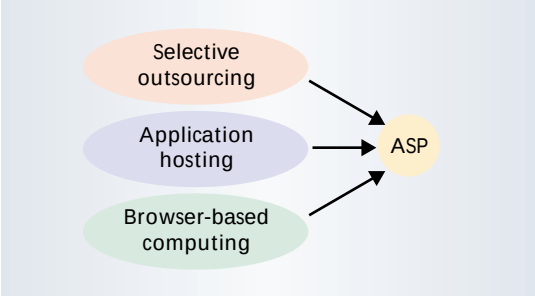

Figure 2. Essential ASP components, which derive from the IT industry's selective outsourcing practices, ISPs' provision of complex application hosting services, and the Web's browser-based interfaces.

ASP TRENDS

Three separate trends drive the emergence of ASPs,[1-3] providing the ASP model's essential components, as Figure 2 shows: selective outsourcing, application hosting, and browser-based computing.

Selective outsourcing

Long a staple of the IT services industry, outsourcing has, in recent years, evolved to provide increasing levels of granularity in the choices the industry offers its customers. Instead of handing over their complete IT infrastructure to an outside provider, organizations have selectively outsourced specific IT functions, ranging from data networking to application management. This development has combined with a trend toward fixed and per-user pricing, often levied in the form of a monthly subscription. ASP propositions emerging from this development strand take several forms, including subscription computing and outsourcing of application, systems-management, infrastructure, and whole-environment tasks.

Application hosting

Internet service providers have always been ASPs to the extent that they provide hosted mail and Web servers. Over time, the ISP industry has divided into entities that provide access and connectivity services and those that offer hosting services. The latter, particularly as they move into sophisticated e-commerce, messaging, and other complex Web hosting services, effectively become ASPs. A new class of application software vendors, which uses the hosting model to provide Internet-based applications and services, has joined them, providing services that fall into several distinct subcategories: Internet Web server hosting, application server hosting, e-business services, and Internet infrastructure services.

Browser-based computing

Web sites, once the home of static content, now host live applications with increasing frequency. To gain the *stickiness* that ensures return visits, information sites have added applications that create dynamic and interactive experiences. Meanwhile, a new generation of software vendors markets their applications as Web-based services, accessed directly over the Internet.

These trends converge in serving either specialized business needs or vertical industry markets, marking the advent of *browser-based computing*, which provides sophisticated online applications alongside relevant content from a Web site, catering to the specific needs of a special-interest group. Significant browser-based computing categories include network-based application vendors, Internet business services, vertical industry Web sites, Internet marketplaces, and enterprise extranets.

BENEFITS

The ASP model's benefits derive from distributing its software applications across multiple servers rather than across multiple clients.[1-3] A combination of a rental commercial model, a component-based application architecture, and a server-based thin-client computing environment provides the greatest benefits.

Software vendor and service provider benefits include the following.

- *No distribution costs.* The ASP model eliminates the need to print manuals, press disks, order thousands of colorful cardboard boxes, manage stock, and operate a returns procedure.
- *No user installation.* Removal of an installation procedure eliminates the expense of providing user support for it.
- *Reduced piracy.* Because the bulk of the software resides on the server, users cannot copy and distribute the full version.
- *Instant upgrades.* Suppliers can implement bug fixes and new features automatically, without waiting for users to discover, download, and install the new code.
- *Consistent user base.* Automatically updated software that resides on the host reduces or completely eliminates the proliferation of different application versions and release levels.
- *Usage monitoring.* Suppliers can monitor usage to gain a greater understanding of user interaction with the product, discovering the most and

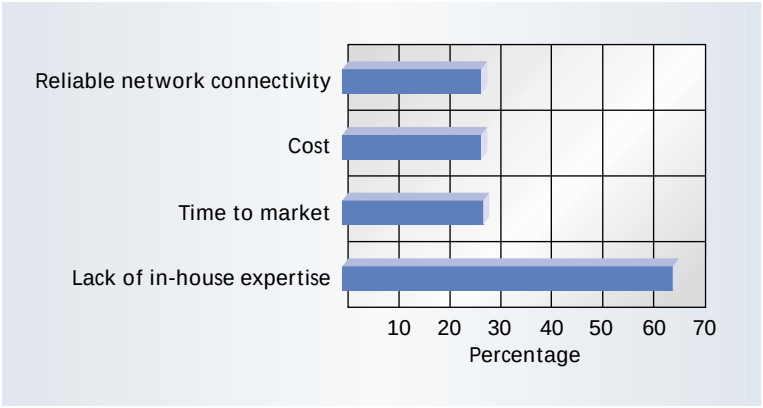

Figure 3. Reasons users adopt ASPs. ASP software's ability to overcome the lack of in-house expertise proved more than twice as important as any other factor.

least popular features, identifying which features appear to cause the most problems, and determining which features must be streamlined to improve productivity.

- *Potentially constant revenue stream*. A steady stream of rental fees releases suppliers from the need to create annual releases simply to generate revenue.

The ASP model's user benefits include the following.

- *Limitless choice*. The Internet gives users access to every rental application available online, creating a model as pervasive and persuasive as the PC. The rental model gives users unbounded potential to access and combine services at will.
- *No installation hassle*. ASP software can be used immediately, bypassing the time-consuming and potentially troublesome client installation. Having the software configured to meet specific requirements or allow for integration with other systems might cause a slight delay, but it provides extra value.
- *No compatibility issues*. Users don't need to worry about whether their system is powerful enough to run online software or if the ASP software will conflict with other applications already installed, because they do not install the ASP products—they simply access them online.
- *No support overhead*. Users don't need to employ expensive administration and support staff to operate complex software installations and the equipment required to run them. The service provider takes care of these issues and builds the cost of doing so into the application's subscription price.
- *Reduced downtime*. Most online service providers do a better job of ensuring 24/7 application availability than customers could because they depend on their products' reliability to stay in business.

Figure 3 explains why current ASP customers are adopting this new computing model.[1]

Although a powerful and enticing business model, ASP software does have its downside. Current disadvantages—which technological solutions should overcome eventually—include difficulty in securing client data and limited performance because of the Internet's limited bandwidth.

ASP CHANNEL STRATIFICATION

ASPs have stratified the ASP channel into several interlocking layers, each with its own areas of core competence.[2,3] Although an end user who purchases an ASP solution will deal with only one provider, in most cases that solution will consist of various components from several different layers. Among those hidden layers might be a company that, while a major contributor to the overall solution, never enjoys a direct ASP relationship with the end user. This stratification results naturally from the ASP model's multitiered computing architecture.

With various elements of the solution performed on separate, specialized servers, an obvious next step is to have separate and specialized providers handle each of those elements. Some ASPs continue to argue in favor of a vertically integrated model in which they own and control every element from top to bottom, while others promote the merits of outsourcing to best-of-breed providers. The former approach can deliver tighter integration and more assured control, while the latter normally offers greater economies of scale.

Every provider, however, outsources at least some element of the solution. Providers who host a server center rarely write their own software. Thus, providers and their customers must weigh the risk of outsourcing against the cost of in-house provision, then strike the balance that best suits their particular requirements.

The ASP channel has four primary subdivisions, as Figure 4 shows. Each of those layers can contain many different components. An ASP solution might consist of contributions from a dozen or more different providers, each responsible for just one component. Alternatively, just one or two providers may provide all the components, with provider's activities straddling several layers.

Network services

The providers of basic communications, server center resources, and value-added Internet Protocol services sit at the network layer. Communications include the physical connections, the routers that handle IP traffic, and the associated performance, reliability, and security applications.

Server center resources typically embrace the provision of collocation space, protected electricity supplies, and physical-security and maintenance services. Value-added IP services include virtual private networking, network caching, streaming media, firewalls, and directory services.

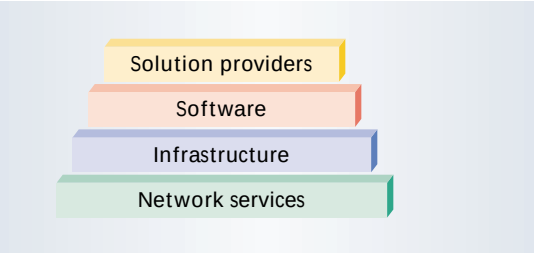

Figure 4. ASP channel stratification. The channel consists of four primary divisions, with a single ASP handling all four only rarely.

Infrastructure

The next layer contains a rapidly emerging space with rich pickings for talented early entrants. Many providers offer individual services such as utility storage and server hosting, or operational resources such as call centers, finance, technical support, and so on. Some ASPs coordinate third-party services along with their own in-house skills and resources, thereby providing a complete infrastructure that lets their clients operate as ASPs.

This ASP infrastructure provider (AIP) role includes the coordination of network and systems management; the supply, operation, and management of systems hardware and software; and the management of ASP subscriber accounts, billing, and customer support. A further important element comprises application management, service-level monitoring, helpdesk infrastructure, and the streamlined messaging of alerts and support information between partners within the ASP channel stack.

Many ASP pioneers have assumed this AIP role, sensing the opportunity to turn their early experience into a marketable commodity they can package and sell to newcomers. They offer the service to independent software vendors and systems integrators who want to bring existing client-server applications to the ASP environment. AIPs also give advice on the fine-tuning and reengineering required to run applications effectively from a shared, Internet-based server center.

Software

Software providers add the vital ingredient that enables the finished application service. The software can be a ready-made, packaged application adapted for ASP delivery or it can be specifically developed for the purpose. Developers can use any of several application server platforms to create ASP offerings, although at present few offer a complete set of services for functions such as service deployment, subscriber management, support, service-level management, and billing. Although independent software vendors do most of the development in house, a growing number of software companies and systems integrators are developing specialized skills in building online application services to order.

Solution providers

Fulfilling the final step in the chain, solution providers are the true ASPs. They package the software and infrastructure ingredients with business and professional services to create a complete service product.

SUPPORTING TECHNOLOGIES

Applications running on ASP servers need special properties, such as separation of business logic from presentation with Internet protocols, reentrant code, scalability, efficient storage and retrieval of session data, and lifetime management.

Most existing client-server applications are unsuitable for ASP hosting. For those developers who value time-to-market more than quality of service, however, some technologies—such as Microsoft Windows 2000 Terminal Services—support fast-track adaptation of existing client-server applications for ASP hosting by logically extending the connection between ASP servers and client PCs' I/O devices. WTS lets standard Windows-based client-server applications run on the server instead of on a client PC. Clients running Windows terminal software can then use the Microsoft Remote Display Protocol to access the sessions.

Citrix, which developed the core technology underlying WTS, has its own independent computing architecture for delivering sessions to clients. It supports non-Windows clients on platforms such as Java and Unix, as well as the Windows clients that Microsoft RDP supports. The company also offers a technology called Application Launching and Embedding, which allows access to Windows applications on the server from any browser, without special Citrix or Microsoft client software.

To fully benefit from the ASP infrastructure, applications must be tailored specifically to meet the ASP's intended use. The complexity and cost of such applications mandate adoption of the component approach. Even though different ASP applications can provide different services, they share many functions, such as user subscription and management, billing and payment processing, service quality control, data storage and management, and authentication and verification. The success of a small- or medium-sized enterprise depends on the following rule of thumb: Spend 90 percent of your investment optimizing 5 to 10 percent of the components in your expertise domain, and adopt commercial-off-the-shelf components for the rest of your applications. It may be better to let other specialized ASPs host advanced external components via service integration.

External component quality presents a common concern when using the component approach because the external source code is usually hidden from in-house system integrators. I believe the component approach's advantages outweigh these concerns for the following reasons:

- in-house code is usually inferior to components implemented by specialists,
- market competition provides the major driving

Corba components support all the essential online and offline services a component system requires—and much more.

force for those who create components based on publicly accepted standard APIs to perfect their products, and
- public adoption of a commercial component will lead to early discovery of its bugs and deficiencies.

Distributed-components reference model

A component is a binary module of code—usually implemented as an object with some extra properties—that supports system integration. A distributed component further supports interoperation and collaboration of components running on different processors across a network. A full-fledged distributed component usually has the following properties.

- *Universal reference.* Each instance of the component must have a reference or ID that is unique across the world, so that other component instances can address it and call its methods through the Internet. Such references should be valid across various computing platforms, component implementation languages, network protocols, and geographical distances.
- *Network interoperability.* Any two component instances on the Internet should be able to interact with each other without regard to computing platforms, implementation languages, network protocols, and geographical distances.
- *Introspection.* Without a component's source code, the computing environment or other component instances should be able to dynamically find the component's API, including the types and signatures of public attributes and methods. This capability enables dynamic interaction between two component instances unknown to each other.
- *Customizability.* A component's attributes and behavior should be customizable offline, usually with an integrated-development-environment tool that leaves the instantiating component's source or binary code untouched.
- *Toolability.* Customization, as well as component integration, should be carried out in a visual tool environment, enabling system integration and management without coding.

Because the component instances will interact and collaborate across networks, a component-support system must provide various services, either online or offline. Typical services in this category include the following.

- *Naming.* The system associates user-friendly names with references to component instances, using globally unique names.
- *Trading.* Like a yellow pages directory, the trading service enables publishing new components on the Internet, listed according to service category.
- *Life cycle.* This service handles the instantiation, migration, copying, and destruction of component instances and provides key support for the visual dynamic integration of applications and services.
- *Persistence.* This service lets the system automatically activate component instances upon client invocation. It will automatically save the component instance's state during a server crash, restore the saved state upon re-instantiation, and support the illusion that component instances and the references to them are persistent.
- *Event.* The key to supporting event-driven execution among components, this service is itself usually a component. Event source and sink components can register with an event service component. A sink component can register itself as either a *push* or *pull* client for a particular category. Upon notification from the event source, the component will broadcast the event to all its registered push clients for a particular category. The pull clients can check the events with the event service component in their own time. The event service also provides a convenient tool for supporting the push and pull of information.
- *Transaction.* This service makes a sequence of interactions among components atomic: either all succeed, or none will commit any state change for the relevant components. Because such transactions are not specified in source code, the transaction service can support either dynamic or integration-tool-based transaction specifications or transactions among components the system implements in different languages.

COMPONENT TECHNOLOGIES

The following three major distributed-component models represent—according to my component-reference model—the latest industry technologies that support ASP server applications.

Corba

Common Object Request Broker Architecture[4] components support all my required component properties except customizability and toolability. Currently, Netscape Web browsers have a built-in object request broker to support Corba-based applications embedded in Web contents. Corba components support all the essential online and offline services a component system requires—and much more.

Corba can easily wrap legacy code in wrapper components to provide a fast-track approach to adapting legacy code to the ASP model. To achieve its ultimate

goal of system integration, Corba uses IDL to standardize the specification of vertical and horizontal common facilities.

Enterprise JavaBeans

EJB[5] by itself can support only the integration of Java components. But because EJB and Corba complement each other so well, Corba has become the favored implementation technique of EJB Remote Method Invocations, while EJB provides Corba with a friendlier user interface. EJB augments Corba with declarative transactions, a server-side component framework, and tool-oriented deployment and security descriptors. Corba augments EJB with a distributed object framework, multilingual client support, and Internet-Inter-ORB Protocol interoperability.

To integrate an existing Corba component into an EJB framework, we can use the IDL-generated JavaBean proxy to represent the original Corba component, easily taking advantage of both these component models.

Microsoft's Web solution platform

Microsoft's Web Solution Platform, formerly known as DNA,[6] provides a framework for fitting Windows and the PC into the 3-tier application concept. The platform represents Microsoft's vision of networked computing: an application architecture that fully embraces and integrates the Internet, client-server, and PC computing models to support the development of scalable, multitier business applications that can be delivered over any network.

At the Web Solution Platform's core is COM+, whose strengths include its position as a mature, core supporting technology for Windows applications, close integration with those applications, and a user-friendly development environment. Native to Windows, COM+ also has some disadvantages, including its limited services for distributed computing and limited scalability relative to Corba. It also lacks the serious competition that could drive Microsoft to perfect this technology.

Microsoft has made great efforts to port COM to other platforms. But COM's supporting environment, MTS, suffers from such extreme platform dependency that it was too great an obstacle for this effort. In recent years, Microsoft tried to use interoperability to compensate for COM's platform dependency. With DCOM–Corba bridges, COM component instances running on Windows can interact with Corba or EJB component instances running on other platforms. Microsoft does support Java, but only as a language, not as a platform.

ASP CHALLENGES

To truly benefit from the ASP computing model, the industry must overcome several technological challenges.

ASP server scalability

Application hosting servers must support tens of thousands of concurrent service sessions with high availability and short response times. Unlike Web servers used mainly to support stateless HTTP connections that request Web contents, ASP application servers must support connection sessions during which they must keep some session data on the servers. Such sessions may last hours or days, and servers cannot predict the connection patterns.

Application hosting servers must support tens of thousands of concurrent service sessions with high availability and short response times.

For Web servers, current techniques for improving server performance include RAID disk arrays, server farms based on dozens of processors interconnected by buses or shared memories, and extensive caching. For example, the Yahoo Web site uses an array of around 50 high-performance server processors. Nonpreemptive scheduling algorithms balance the workload among the processors. Some ASPs also support external caching as a generic approach to boosting Web server performance. Today's Web server market, based mainly on proprietary ad hoc techniques, cannot support the level of service quality most ASPs need.

Given that it must support session data, caching is less effective for ASP servers. Dozens of processors may be insufficient to support a full-fledged ASP server's data processing power. Further, current bus or shared-memory architectures introduce server performance bottlenecks.

Preemptive process or object scheduling techniques[7] can help to achieve the scalability that ASP servers require. A server consists of one or more master processors and a cluster of client processors. The master processors maintain the client processors' dynamic load information. The client service request first reaches a master processor, which becomes the gateway for further communication between this client and the master's client processors.

The master processor starts a server process on a client processor based on the processor's current load distribution, the level of service quality the client requires, and the server's scheduling policies. Because clients use the ASP servers in unpredictable patterns, a lightly loaded client processor can become heavily loaded. With preemptive scheduling, a process can migrate from one processor to another to rebalance the workload. Major challenges include how to minimize process-migration overhead, share and maintain session data, and reroute the communications between the gateway and the client processors while leaving a minimal footprint in the client processors.

To fully benefit from their new operations model, ASPs require smooth, secure, and efficient online billing and payment mechanisms that support very large and very small transactions.

Internet infrastructure

Distributed components and services mainly use URL addresses to identify each other. Currently, the Internet uses the 32-bit Internet protocol addresses that IPv4 specifies, but projections show we will run out of IPv4 address space by 2005. IPv6, which must be in place by then, uses 128 bits to represent a URL address and thus allows ample expansion space for future component-based ASP services. Windows 2000 already supports IPv6.

With the ever-increasing communications volume on the Internet, ASP services' response time depends on the speed at which the information flows between clients' Web browsers and ASP server centers. Today, packet switching provides the main Internet support for data communications. With an achievable bandwidth of 1 terabyte per second per optical fiber, the Internet's communications delay will be dominated mainly by the number of hops from source to destination, not by an individual carrier's bandwidth. When a packet arrives, the router must buffer it, extract its header for the destination address, and use a routing table or algorithm to determine its outgoing channel. The router will cut a typical message into many small, fixed-size packets, incurring significant overhead in routers along the way to the destination.

One solution to router processing delay adapts the wormhole routing[8] of parallel computing to the Internet. This approach cuts a message into packets, then further cuts each packet into smaller units named *flits*—flow control digits—the bits of which can traverse the communications carrier in parallel. For each packet, the first flit contains the destination address and the last flit signifies the packet's end.

When the first flit arrives at a router, the router hardware sets up a passage based on the destination address and routing algorithm. The following flits then just bypass the router without being buffered or processed. The last flit's arrival breaks the hardware passage between the incoming and outgoing channels and recycles the router resources.

With this approach, if the network is not congested, and the number of a packet's flits is much larger than the number of hops the packet needs to traverse, the packet's delay will be roughly proportional to the packet size, not the hop number. Cisco has implemented a variant of this approach in some of its switches.[9] Based on today's Internet infrastructure, a packet can move around the world in 14 hops or less.

Micropayments

To fully benefit from their new operations model, ASPs require smooth, secure, and efficient online billing and payment mechanisms. These mechanisms should support both very large and very small transactions, the latter possibly involving only a few dollars. The major challenge is to design mechanisms that support the collection of very small sums, or micropayments, so that clients can pay as they go without being deterred by complicated payment overheads. Current approaches under investigation include smart cards and electronic, virtual, or digital money.[10]

Micropayment becomes much more complicated when used with an ASP service implemented through integration of distributed components from several ASP providers. Client sites may download such components only during use to ensure better performance or security. In this situation, the system needs to distribute client payments transparently to multiple involved service providers based on accurate usage statistics and service contracts. Handling the situation will be easier if all commercial distributed components support standard APIs and microaccounting mechanisms that record and maintain usage statistics.

Security

ASP security addresses both client data and server availability. Today's virtual private network technology can make any Internet connection highly secure against outside interference. Similarly, encryption can easily make either dial-up or leased-line direct access secure. At the server centers, firewall technology further guards against unauthorized access.

Establishing internal staff procedures sufficiently robust to protect against security breaches presents a greater challenge. The vast majority of security lapses involving information technology today stem from careless or malicious acts by employees. ASPs must establish stringent procedures to avoid compromising the integrity of customer data while it is under their care.

For extremely sensitive data, ASPs must let clients download the necessary subset of components to process data on the clients' desktops. Further, ASP server centers must guard against malicious attacks that monopolize communications and server resources. Currently, most Web servers—including Microsoft Internet Information Server and Apache—will crash, taking down their underlying operating systems as well, if their concurrent client connections exceed the servers' capacity to scale.

Dynamic configuration

In the ASP environment, multiple clients with different configuration requirements can use the same application and run it concurrently. Therefore, an ASP application must support separate configurations for each independent user group. This capability contrasts starkly with current practice, where the norm is to enforce a common standard throughout the enterprise to ease systems management complexity. ASPs do not have the option of mandating consistency across the

user population, but instead must embrace and manage complexity. Neither conventional client-server nor next-generation e-commerce application architectures currently provide satisfactory answers to this challenge.

ASP service integration

For several reasons, an enterprise may not want to entrust all its computing needs to a single service provider. First, it may have a significant amount of proprietary or legacy systems and applications to run. Second, it may have sensitive data to protect. Third, it may want to benefit from competition among its service providers to offer better service quality and lower costs.

On the other hand, few service providers can afford to be fully self-contained. For example, many ASPs will opt to let professional financial institutions run their credit card payment services, thereby reducing operating costs and increasing their clients' confidence level. This option implies a strong need for easy integration of existing ASP services to create new services. Thus, ASP applications should support integrating services from different ASP providers and the integration of ASP services with client applications. The industry can achieve these goals by standardizing either the common applications' data formats or their APIs.

ASP IN PERSPECTIVE

The ASP model can easily lead to a service monopoly. The current ASP market consists mainly of service providers for existing stand-alone or client-server applications. Applications in the same category usually have similar functionalities, but different data formats or user interfaces. While possible in some simpler cases, changing the user data format poses problems, especially when a proprietary data format is involved. Clients of such service providers will soon find it almost impossible to switch to other providers that offer similar services with a different application. Although many ASPs cite the "high cost of switching" as a major factor in retaining customers,[1] such switching costs will keep many new clients from adopting ASPs, thereby impeding competitive innovation across the computing industry.

Promoting competition in the ASP market and expanding the market with noncommitted free trials requires accelerated public standardization of services on two levels. At the lower level, we must standardize the data formats of major applications. At the higher level, we must standardize major application user interfaces. Customers can easily adopt services based on new applications that comply with such standards. Innovative companies can then improve the performance of existing applications and use higher-quality service to seduce customers away from less-competitive service providers.

If it can overcome the challenges facing it, the ASP model will foster a new generation of distributed, component-based computing services characterized by finer computing granularity, global cooperation and specialization, multimedia data handling, binary integration, mobile computing, and pervasiveness across a full spectrum of electrical and electronic devices. Applying generic network-centric computing will lead to a networked economy characterized by service integration. ✸

References

1. ASP Island, "ASPs: The Net's Next Killer App," Jan. 2000; http://www.aspisland.com/trends/jcbradford/ (current Sept. 2001).
2. P. Wainewright, "Anatomy of an ASP: Computing's New Genus," *ASP News Rev.*, Jan. 2000; http://www.asp-news.com/premium/article/0,,4221_375621,00.html (current Sept. 2001).
3. P. Wainewright, "Packaged Software Rental: The Net's Killer App," *ASP News Rev.*, Jan. 2000; http://allnetresearch.internet.com/item/1,3008,45601,00.html (current Sept. 2001).
4. T. Mowbray and R. Zahavi, *The Essential Corba: Systems Integration Using Distributed Objects*, John Wiley & Sons, New York, 1995.
5. J. Gosling et al., *Java Programming Language*, 2nd ed., Addison Wesley Longman, Reading, Mass., 1998.
6. D.S. Platt, *Understanding COM+*, Microsoft Press, Redmond, Wash., 2000.
7. A. Goscinski, *Distributed Operating Systems: The Logical Design*, Addison-Wesley, Reading, Mass., 1991.
8. L.M. Ni and P.K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *Computer*, Feb. 1993, pp. 62-76.
9. Cisco Product Documentation, "Bridging and Switching Basics," June 1999; http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/bridging.htm (current Sept. 2001).
10. M.H. Sherif, *Protocols for Secure Electronic Commerce*, CRC Press, Boca Raton, Fla., 2000.

Lixin Tao is an associate professor of computer science at Concordia University, Canada. His research interests include distributed component technologies, Internet computing, parallel computing, and operations research. He received a PhD in computer science from the University of Pennsylvania. He is a member of the ACM and the IEEE. Contact him at taol@acm.org.