E-speak E-xplained

Alan H. Karp
Hewlett-Packard Laboratories
Palo Alto, California

The Internet, World Wide Web, and Web browsers have brought us to the brink of a new way of doing business, and electronic economy, an *e-conomy*, if you will. The picture has not been complete, though, preventing us from moving to the next level. E-speak provides the missing pieces. This report describes the various components of the e-conomy and how e-speak fills the gaps.

1.0 Components of the E-conomy

Buyers and sellers have been part of economic interaction since the earliest days of barter. The problem is how a buyer willing to purchase a product or service finds a seller willing to provide it. Advertising is one approach, but something more is needed for the e-conomy, a *matchmaker.* The matchmaker connects those with an *offer to buy* with those with an *offer to sell*. Unlike static advertising, such as print and broadcast ads, the matchmaker deals with a truly dynamic environment, one in which offers come and go at a furious pace. The matchmaker also recognizes the symmetry between buyers and sellers, something missed in the advertising model. The e-conomy is not just buyers bidding for products or services in an auction; it is also sellers actively seeking buyers in a reverse auction.

The job isn't done once the buyer and seller have been identified; they need to reach an agreement, *i.e., form a contract*. Contract formation can be quite simple in a face to face purchase of tangible goods. The buyer gives the seller cash, and the seller provides the merchandise. Indirect transactions, such as buying from a catalog, are not so simple. What if the buyer doesn't pay? What if the merchandise isn't shipped? Each of these cases involves fraud on the part of one of the parties. Even without fraud, there can be problems. What if the product isn't what the buyer expected? What if the merchandise is damaged or lost in shipping? What if the buyer's payment is not received (cash) or honored (check)?

A number of solutions have been tried. Cash on delivery (COD) puts trust in a third party to deliver the goods and forward payment. Credit card companies provide a similar role without being involved in the delivery of the goods. They guarantee payment to the seller and delivery of the promised product to the buyer. Without these guarantees, mail order sales would be a small fraction of what they are today. The COD service and credit card companies are primitive forms of the last component of the e-conomy, that of *market maker.*

The market maker assists in the formation of contracts between buyers and sellers. There may be many buyers for a given product or service; there may be many sellers competing for a buyer's business. Each wants a contract on the most favorable terms. Forming the contract can be quite complicated, including not just price, but also things like quality, return policy, delivery time, *etc*. The market maker provides the means for negotiating these terms and conditions. In addition, the market maker monitors the transaction, guaranteeing delivery of the goods and payment. These components are shown in Figure 1. Often the roles of matchmaker and market mediator are played by the same party, a *broker*. Sometimes the broker buys from the seller and resells to the buyer, the traditional role of a broker.

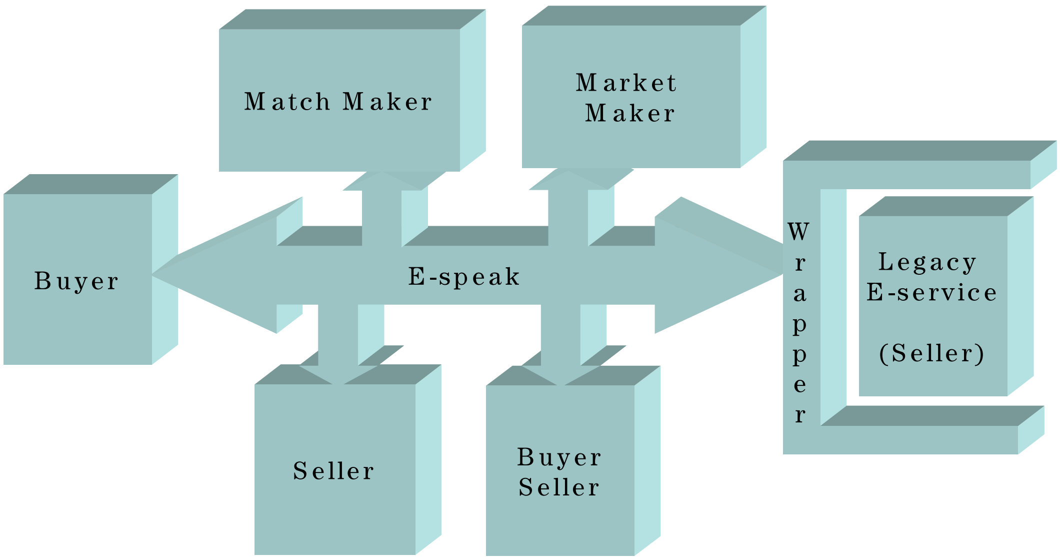

Figure 1. Components of the e-conomy.

The problem today is that these components - buyers, sellers, matchmakers, market makers - have no framework in which to operate. Various projects and products have attempted to address parts of the problem. For example, Microsoft's Biztalk® and the eCo framework from CommerceNet address the problem of describing business logic, but they say nothing about discovery or security. HP's Praesidium® product deals with authentication and authorization, but says nothing about service discovery. Sun's Jini® addresses discovery and service invocation but says nothing about security or billing and manageability.

E-speak assists the creation, composition, mediation, virtualization, management, and accessing of Internet based services. It is an *open services platform* that lowers the barriers to creating new e-services. Each of the emphasized words is significant.

- *Open* - published interfaces, open source
- *Services* - designed for dynamic composition of e-services
- *Platform* - provides naming, discovery, management, and security

2.0 How E-speak Fits In

E-speak has a very modest goal - simply to change the way the world thinks about computing. Today, we think about computing as the hardware we buy and the software we install on it. E-speak is all about thinking of computing as a set of services that we enlist as we need them.

The essential difference is that when thinking about hardware plus software, we're always telling the computer how to do the job. "Go to that server. Use this printer. The paper I want is in the upper tray not the lower one." These are facts I may not know if I've just walked into someone else's office, and I want to print a file. If printing is a service, I say instead "I need a printout of this document, here, in color, with at least 600 dots per inch resolution, by 10 AM." I don't care how the job gets done. Maybe the print shop at the corner did the job and delivered the output.

We can see from this simple example, that e-speak's goal is to make it as simple, transparent, and safe to use an e-service as it is to access a web page using a browser. There is no need to know where the service is, how it is implemented, what kind of machine is used. I'm not putting myself at serious risk of attack or virus infection by using the service. However, adding computation to data delivery dramatically complicates the process of service creation. That's where e-speak comes in.

2.1 E-commerce, E-business, E-services

We've all heard these terms. Some pundits say that they're just marketing hype. Sun Microsystems grabbed "e-commerce"; IBM, "e-business"; all that was left for HP was "e-service." This perception is wrong; there are real differences as shown in Figure 2.

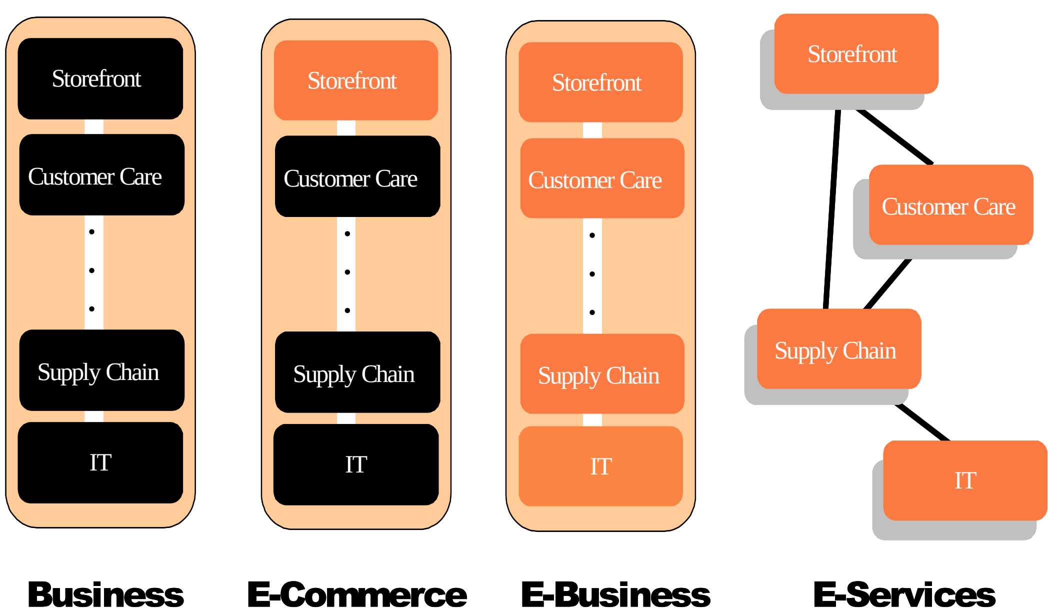

Figure 2. E-commerce, E-Business, and E-service comparison.

Each business today has a certain structure that constrains how data flows. For example, an order passes from the storefront through customer care, to credit management, to inventory, and finally to shipping. E-commerce puts the customer facing part of this process on the web. E-business puts the inward and supplier facing part of the business on the web. These are wonderful things to do; enormous efficiencies are gained. A large part of the productivity growth seen in recent years no doubt comes from this process.

However, neither e-commerce nor e-business changes the fundamental structure of the business to enable it to react to changes in Internet time. That's where e-services come in. With e-services you think of your business as a set of independent services that you enlist as you need them. Now, you don't have to wait for the next corporate reorganization to fix a broken process. Your processes adapt themselves to the need at hand.

There's another advantage to the e-services way of thinking. Let's say that there's one of these services you don't do very well. Out source it! Find someone out there who does the job better than you and is offering it as a service. In other words, out source your core incompetencies. Since your business is already structured to deal with independent services, there will be no disruption.

What if you don't find anyone who does a particular job better than you're doing it? Well, then you've got the best e-service available. Market it! Turn your core competencies into a revenue stream.

2.2 The Open Services Marketplace

Following this line of thought to its logical conclusion leads us to the idea of an open marketplace for such e-services. However, there's a problem. Look at the poor souls in Figure 3. The guy has a great idea for an e-service, but before he can offer it, he has to solve a number of problems, *e.g.*, advertising, billing, security, *etc.* The gal could make more money if she used this service; her customers could have a better experience. However, before using the e-service, she, too, has a number of problems to solve, *e.g.*, finding the service, monitoring its use, security of her system, *etc*.

The problem is that today, they each solve these problems independently. It's a terrible waste of time and money. Money today isn't a big deal, at least here in Silicon Valley, but time to market is critical. However, there's an even more serious problem. Because they've solved these problems independently, they've solved them slightly differently, making it difficult for them to interoperate.

E-speak is a layer of software running on collection of equipment that provides a set of mechanisms for solving this common set of problems. Because the mechanisms are the same for everyone, it's easier for the two parties to interoperate. E-speak also provides a language for specifying a variety of policies. "This service costs $1.00 per use or $50.00 per month." "Joe may use this service, but Sally may not." It is this combination of a platform providing the common set of mechanisms and the language for setting policy that makes e-speak a critical part of the open services marketplace. If this is all you understand about e-speak, you've captured the essence of what it does.

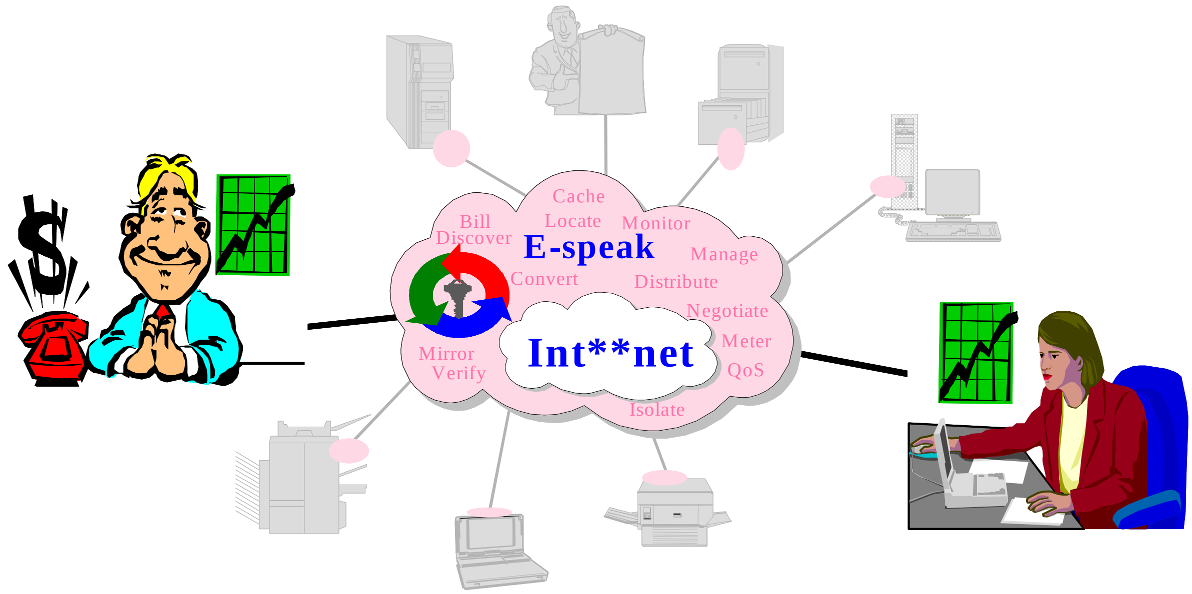

Figure 3. E-speak facilitates the open services marketplace.

2.3 Service Composition

Nobody knows how to make a pencil. One person makes the graphite; someone else, the eraser. A third party makes the little metal band that goes around the top. Very interesting, but what's this got to do with e-speak? Simple. E-speak encourages the same kind of service composition that goes into the making of a pencil.

Say that I want to go into a business with several components, none of which I know how to do. Integrated financial management is one good example. Such a service might include payroll, tax preparation, and cash management. With e-speak, I can easily enter this business as long as each piece is available to me as an e-speak service.

Figure 4 illustrates the process. First, I describe the business logic of my new service. First do the payroll, then calculate the taxes. Finally, in the unlikely event that any cash is left over after taxes, figure out how best to manage it. I'll also describe my policies, such as billing, authentication, and the like. Once that's done, I can advertise the service on the Internet or the Intranet. E-speak doesn't have to be everywhere; you can benefit if you just use it to streamline your intra-enterprise applications.

Once the service has been advertised, a customer can discover it and use it. There's something significant about this process. You'll notice that the service doesn't have to be configured to run specifically in the customer's environment. That's because I've told the system what job I want done, not how to do it. I haven't said "Use this specific computer." Instead, I said "Use a computer with these attributes." As long as the customer environment has a computer with the right

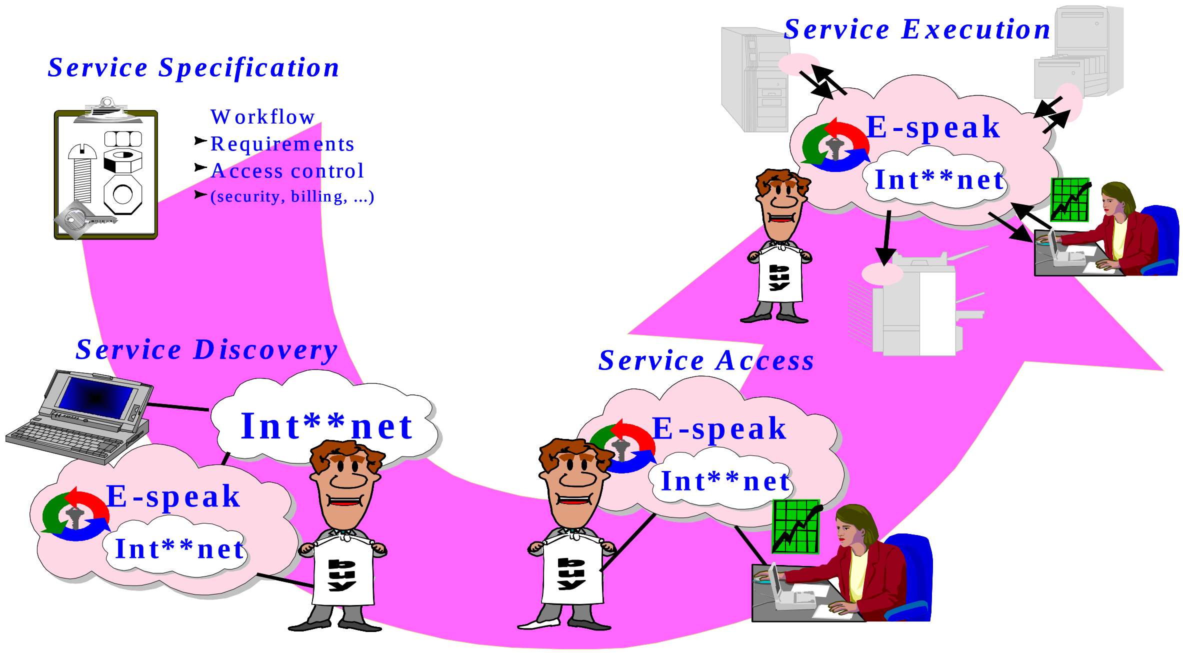

Figure 4. E-speak service composition.

attributes, the job runs correctly. No need for me to come configure the software for each installation; no need for the customer to run Install Shield® and go through six reboots.

What if the customer doesn't have a computer with the right attributes in her environment? That's the real beauty of thinking e-speak. It doesn't matter. As long as someone is willing to provide a computer that can do the job, my service will still run correctly. I didn't have to do anything special to make my application work; the e-speak platform took care of it for me.

3.0 The E-speak Stack

Other products focused on the e-conomy are all point solutions, each addressing a particular aspect of the e-conomy. E-speak, on the other hand, addresses the full set of key problems - naming, discovery, management, security, programming, and interoperability - in a single, coherent architecture. E-speak was not designed to solve a particular problem; it was always intended to be a general platform for doing the kinds of things needed to make the e-conomy flourish.

E-speak is a complete platform as shown in Figure 5. One part implements the necessary mechanisms. Another provides the support to write applications. There is also a means to describe the service's business logic using one of a number of frameworks. Key to e-speak is the service bus that defines a way for services to interoperate. E-speak also provides a set of infrastructure services that provide the basic tools needed to bring an e-service to market.

At the bottom of the stack is the transport layer. The transport is not part of the e-speak architecture, only the interfaces to it are. Hence, e-speak today runs on TCP/IP, HTTTP, and shared memory within a Java Virtual Machine. It should be simple to provide for other transports, such as

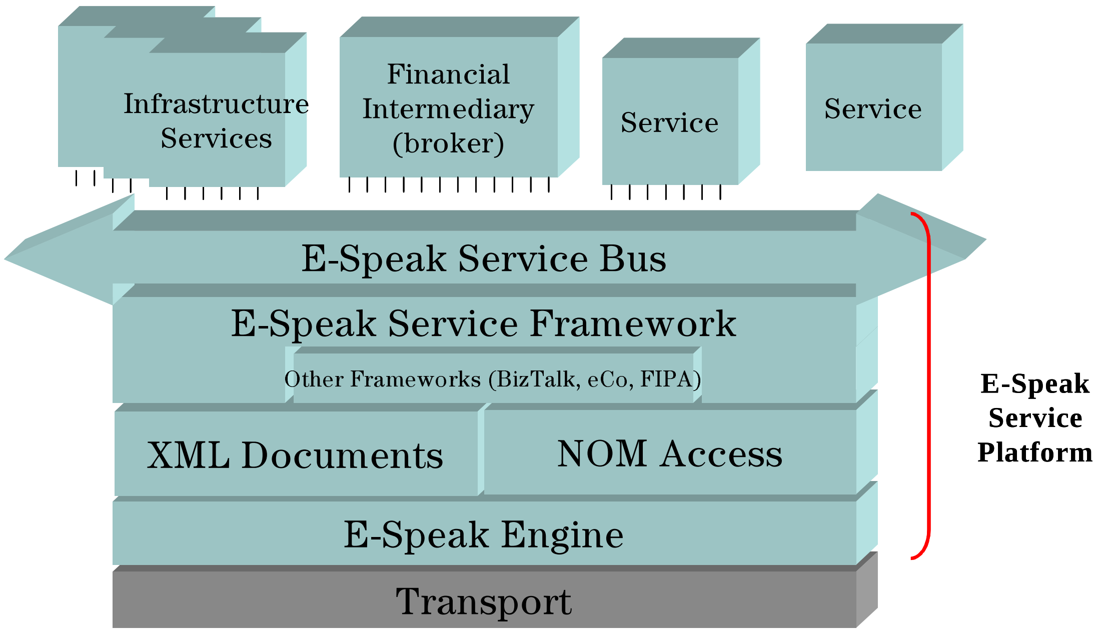

Figure 5. The e-speak stack.

wireless. At the top are the services; they are also not part of the e-speak architecture. However, any service that supports the interfaces defined by the E-speak Service Bus can be used as a building block of a higher level service.

The rest of the stack is defined by e-speak. The engine provides the operating platform on which the other parts ride. It handles the basic abstractions of naming, discovery, management, and security. The engine also mediates all requests, generating the proper management and billing events. Several programming models are built on this base, the main ones being direct messaging, document exchange, and network objects. E-speak does not mandate the way business logic is described. Instead, it provides a framework for using any available system.

The most important part of the stack, the part that makes dynamic composition of services a reality is the service bus. The bus itself doesn't exist as an entity any more than the World Wide Web is a physical entity. Instead, like the Web, the e-speak service bus consists of pieces of services that implement its interfaces. It exists so that any compliant service can be used as a building block of higher level services without a human being needing to read documentation and implement interfaces. It includes specifications for discovery, introspection of both interfaces and protocols, negotiation and contract formation, and service invocation. Hence, the service bus is a key enabler of dynamic service composition.

Today, I can go to your web site, read your user's guide, and start invoking your service. The problem is that this procedure is labor intensive and too slow to react to changes in real time. By supporting the service bus, you are assured that other services can find your service, negotiate and form a contract with it, learn how to use it, and invoke it, all without human intervention.

The e-speak platform and service bus specification reduce the barriers to producing e-services, but there are also infrastructure services that almost any business needs, things like billing, credit management, authentication, and authorization. If every provider of an e-service had to implement these services, the barriers to entering the e-conomy would be quite high. In order to address this problem, the service bus specifies that there will be at least one instance of each of these infrastructure services available. HP is providing reference implementations that will be released to the open source community and is working with partners to provide commercial versions.

The service bus is defined in terms of a set of XML documents that can be communicated over the web on a universally available transport, HTTP. Thus, entering the e-cononmy can be as simple as filling in the blanks in a form.

4.0 History

E-speak was delivered to the world just before the turn of the millennium. It's origins go back almost 60 years, though, to a report by von Neumann describing a *computing utility* and to Vannevar Bush's Memex.

The modern history of e-speak began with Joel Birnbaum's 1982 paper on the "Information Utility". In that paper he used the word "Utility" in the sense of the electric utility; you don't think about it unless it's not there. You never think about electricity until you flip the switch and the lights *don't* come on. In addition, Birnbaum's "utility" was to be pervasive. Every computer, every printer, every automobile, every microwave oven, maybe even every light switch would be part of the utility infrastructure. It was quite an amazing vision given the state of the technology in 1982. The work on e-speak did not start from a formal requirements statement, but this vision had an important impact on its design.

There were other influences. In the mid 1980's, Bill Rozas, a graduate student at MIT at the time, was tired of having to use someone else's names, defaults, environments, even screen colors when using that person's computers. He argued for a *persona* that he could take with him anywhere in the world. In the late 1980's, I came to the rather obvious realization that it's easier to collaborate with someone on the same computer than with someone on another computer. My solution was to put everyone in the world on the same computer. Since one piece of hardware could never handle the job, the question became how to make all the computers in the world look like one computer.

In 1991, concerned about the complexity and expense caused by the rapid advances in technology, Rajiv Gupta started thinking about ways to reduce the total cost of ownership of computers and the associated infrastructure. He wanted to find a way to continue to use older machines even though they might not be able to run the latest applications. Also in the early 1990s, Arindam Banerji was trying to find a way to get a computer to deal with a new kind of thing, like an audio stream, without waiting for the next release of the operating system. His Ph.D. thesis showed how to allow a user to extend the capability of the operating system without compromising its security.

Each of these perspectives influenced the design of e-speak. As you read on, think about how the various features reflect these inputs.

We started the work that led to e-speak toward the end of 1995. By March of 1996, Bill Rozas and I had developed the beginnings of an architecture and a first prototype. Rajiv Gupta, at that time manager of HP Lab's Future Systems Department, would comment on what an interesting service we had just shown him. He often made suggestions, both on the architecture and the demonstrated uses of it. When we had implemented these ideas, he'd comment on how useful or profitable such a service would be. Over time, perhaps the six months from March to September of 1996, he formulated the idea of a world of e-services. Not just isolated e-services, but a world in which everything became an e-service. It was an interesting process to watch.

5.0 Why We Weren't There

Until e-speak, none of the perspectives presented above had been met. Why? One reason might be bad assumptions.

In the early days of computing, the 1950's and into the early 1960's, a computer was a big machine kept in a locked room attended to by specially trained people. Most users had limited interaction with the machine. Normally, you'd hand a deck of cards through a small window, and a couple of hours later you retrieved some paper that showed you'd forgotten a parenthesis in your Fortran program!

As time progressed, computing moved away from the locked room in the form of terminals, first teletypes, then other typewriter based devices, and finally cathode ray tubes. Programmers were freed from physically visiting the computer or waiting for a courier to do so. Unfortunately, the mainframe computers were expensive, and many felt that these precious resources were wasted dealing with all those keystrokes.

Once inexpensive ($100,000) computers could be built, there was a better solution in the form of client-server computing. It is based on the idea of a small computer close to the user that handles the mundane tasks of input and output while the *real* work is done by the mainframe.

The problem was that we carried our assumptions forward as we rolled out the infrastructure. One such bad assumption was that all users were dealing with a shared pool of resources. After all, that's really what a mainframe computer is, a shared pool of resources. When problems caused by these bad assumptions became apparent, people attempted to find point solutions to address a particular symptom. Network File System® from Sun Microsystems is one such point solution.

As time went on, we evolved from the controlled, proprietary client-server world to the wide open Internet. Unfortunately, we carried our assumptions forward with us again, and again we are attempting to rely on point solutions to address the problems. One such problem is caused by carrying forward from the client-server world the assumption that, even though the client machines are widely distributed, they are all under the physical control of the owner of the server. That's not true on the Internet, and it's causing problems. Once again, people are proposing point solutions to fix specific aspects of the problem. Hence, the interest in IPSec, SSL, PKI, and a whole host of other acronyms.

6.0 Assumption and Implications

When we started the research on e-speak, we realized that we had a unique opportunity, a chance to design a distributed system from the ground up, knowing that the Internet was there as a commercial platform. Had we started a year earlier, we might not have realized that business could be conducted on the Internet. After all, through most of its life, the Internet was a means to deliver data, not services. None of the earlier designers could have known what we knew; those that followed would be working against long established patterns.

How to start? We decided to ask what assumptions we should make. From them we could construct our design principles from which the architecture would follow. OK, it's never that clean. In fact, it was an iterative process. As the early work proceeded we would re-examine assumptions or introduce new ones. Fortunately, the process quickly converged to the following assumptions.

6.1 Large Scale

We assumed that we would have to deal with 1,000,000 machines. We now know how ridiculous a number it is, ridiculously small, that is. However, thinking about a million machines focuses your attention. We realized that we could not have a centralized anything, not a database, not a control point, not an authenticator. Any such a thing would eventually become a bottleneck.

A very large number of machines also means that you have to give up the idea of keeping data consistent. If you try, you'll spend all your time waiting for changes to propagate through the systems, and you won't get any useful work done. In e-speak, any piece of data you get, even from the local machine, may well be out of date. Most of the time what you get is good enough. If you need current data, go get it. It turns out that if you assume your data is consistent, and it's not, you've got big trouble. However, if you assume your data is inconsistent, it's much easier to deal with out of date information.

6.2 Dynamic

With a million machines, someone is always tripping over a power cord. New services are appearing; old ones are going away. E-speak is built to work in a dynamic environment like the Internet. Every part of the system, from naming to invocation and replies, has special properties for dealing with this world.

Services need to be written differently. In a static world with predetermined configurations, a program can be written assuming that what it needs can be found, and once found, used for as long as necessary. This approach won't work on the Internet. E-speak encourages programming for a dynamic world. If a service isn't found, try again later or compose what you need out of other services. If a service you've been using disappears, find a provider of an equivalent service.

6.3 Heterogeneous

The world is heterogeneous and getting more so, not less as Microsoft and Intel would have you believe. For example, a Saab has 50MB of software, a network, and 12 operating systems. Your electric shaver has 8KB of software and an operating system. You don't even want to think about what goes on inside a rice cooker! The heterogeneity is not just in hardware platform or operating system, though. It is also in functionality. E-speak was designed to work on devices ranging from supercomputers to pagers or even light switches.

6.4 Hostile

The Internet environment is hostile. There are bad people out there; there are careless people out there. Security is difficult to do, so difficult that system designers often leave it for last. After all, if the thing isn't going to work at all, there's no reason to do all the hard work of adding security mechanisms. Unfortunately, if you try to add security later, it is very difficult to block all the ways around the mechanisms.

E-speak started with a way to control access to resources. In fact, the first two machines we connected each ran a script called "malicious", attempting to get the other machine to do something it shouldn't. There are data structures in the code today that appeared in the first prototype. Security is built into the very fiber and being of e-speak.

6.5 Many Domains of Control

There are different fiefdoms. The fact that your security is based on classifications, such as secret and top secret, and mine is based on compartmentalization should not prevent us from cooperating. You should be able to manage your system any way you like; I should be able to do the same with mine.

E-speak addresses this problem by never looking inside another machine. If you ask me in the right language, and permission is granted, I'll do it. I don't care what operating system you're using, or whether you're even running e-speak. You might be a squirrel with an abacus for all that I know. All I care about is that you've asked me to do something that I've agreed to do on your behalf.

7.0 Design Principles

These assumptions led us to a number of design principles. First of all, the three of us weren't going to be able to write a million lines of code to do what we wanted, so we needed to keep things simple. On the other hand, we couldn't forget the problem we wanted to solve. Our solution was to develop a small set of abstractions and use them over and over again. Remarkably, the abstractions that we developed early in the project survived the test of time.

E-speak was designed to deal with technological advances. In order to deal with a dynamic environment like the Internet, e-speak has to be seamless, flexible, and to allow for dynamic evolution, able to deal with current and future requirements. That word, *future*, is very important. There is no need to wait for the next release of the software, no need to shut down the system and restart, even for a kind of thing invented after the system started running.

E-speak is designed to be scalable, manageable, and secure. Designed to operate on the Internet or Intranet, e-speak avoids bottlenecks such as centralized databases and a reliance on data consistency. Its mechanisms provide the hooks necessary to manage and secure the system without forcing everything to be done in a specific way.

E-speak has no special cases. Any special case needs very strong justification, and none passed the test. Each is an architectural nightmare, a development nightmare, a maintenance nightmare. Every time a special case was proposed we would lock ourselves in a conference room until we figured out how to do without it. Doing the work in a research environment gave us the luxury of taking this time, but its benefits contributed to the speed with which the product was produced.

E-speak has a completely uniform access model. E-speak makes no distinction between a service, such as a billing service, and a resource, such as a transaction to that billing service. Even though people normally think of a service as being active and a resource passive, e-speak's mechanisms are the same for both.

E-speak deals only with service control information, not service semantics. If you say "open a file", and permission is granted, your file gets opened. If you say, "open a butterfly", and permission is granted, the butterfly opens its wings. The important thing is that e-speak does the exact same thing in each case. E-speak makes no attempt to understand the semantics of your service. That's why it can deal with a new kind of thing; e-speak doesn't even try to understand an old kind of thing.

E-speak mediates and virtualizes every request. In a dynamic world, things change rapidly. Getting policy information and using it for any length of time can be dangerous. Furthermore, services can come and go at any time. E-speak needs to be able to deal with such occurrences without breaking the application. Mediation lets e-speak keep management informed, and virtualization let's e-speak hide the changes from the application.

E-speak is built on industry standards wherever possible. There is no need to reinvent what the industry has adopted. E-speak leverages transports such as TCP/IP and HTTP, business logic such as XML, and standard ways of encrypting and authenticating.

8.0 E-speak in Perspective

Think about your computer. There's a lot of stuff inside of it, most of which you don't want to know about. One thing that's not inside your computer is anything called a file. There's a disk with tracks and blocks and bytes, but nothing called a file. Yet, you can talk to the operating system about a file. The operating system uses some of that stuff inside your computer to present the *abstraction* of a file.

There is nothing on the Internet called a file. There's a link which refers to a URL. A URL has some components - a protocol, a location, maybe a port number, and a path. Eventually, this all gets converted to a set of bytes in a block in a track on a disk in a computer, but there's no such thing as a file on the Internet. As shown in Figure 6, E-speak lets you present the *abstraction* of a file on the Internet, just the way the operating system on your computer presents the abstraction of a file.

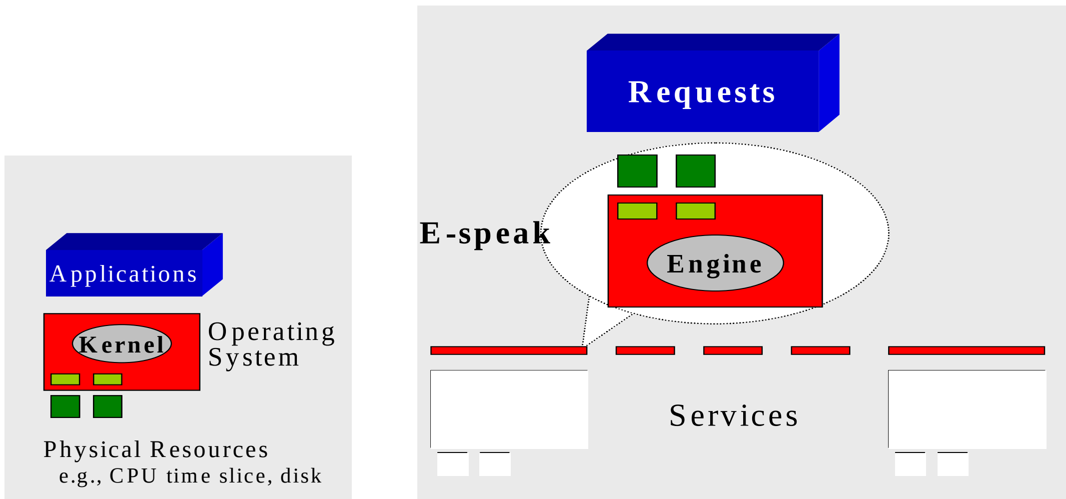

Figure 6. E-speak on the Internet compared to an operating system on a computer.

Think about it. What does your operating system provide? In one view, an operating system does just three things, namely it

1. presents a programming model based on a resource abstraction,
2. provides means to control access to those resources, and
3. talks to the hardware.

E-speak does the first two. In that sense, e-speak is sort of like an operating system for the Internet. It's not really an operating system, but you can think of it as one. If you do, your job of producing an e-service is much easier. Program to the e-speak abstractions as shown in Figure 7, just the way you program to your operating system abstractions today. Someone, perhaps HP, perhaps partners, perhaps competitors, provides the basic set of infrastructure services needed to run an e-service. The e-speak stack takes care of hiding the complexity of the Internet, allowing you to get to market faster with less effort.

9.0 Technology

E-speak is a single, coherent architecture that deals with the four key problems of distributed computing, namely, naming, discovery, security, and management. Here I'll briefly describe how e-speak addresses each of these problems.

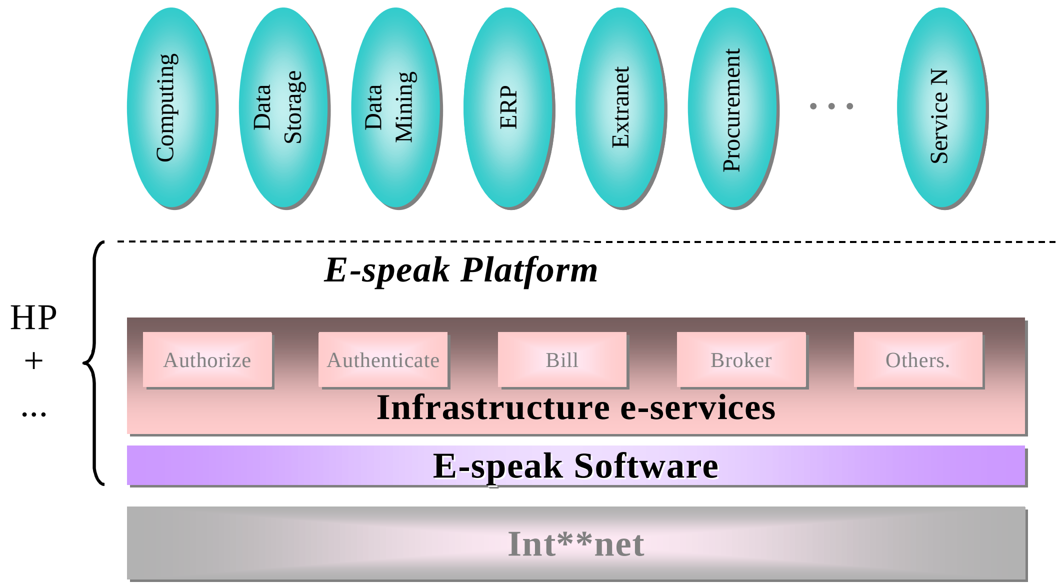

Figure 7. Viewing e-speak as sort of an operating system for the Internet.

9.1 Naming

Most distributed systems give the problem of naming short shrift. Most simply use URLs or something similar. As simple as a Universal Resource Locator sounds, it is not sufficient in a dynamic environment like the Internet. First of all, you have to know the name to use the service. Secondly, the dynamic nature of the Internet means that the service you're requesting may have been renamed, may not be available, or even still in business.

E-speak addresses these problems by mapping each name to an algorithm for finding the kind of service you need. An e-speak name can be bound to a single service, a list of services, a search request for a service, or any combination. Thus, if the service you used yesterday isn't available today, your name binding can be used by the system to find an equivalent provider, all without your application crashing or needing special code to deal with the situation. This example is one way that e-speak lowers the barriers to service creation.

9.2 Discovery

If names aren't used to find services, what is? E-speak encourages you to think of what job you want done instead of how to do the job. Naming a specific service falls into the category of telling how to do the job; finding a suitable provider is telling the system what job you want done. E-speak provides some very powerful mechanisms for discovering service providers.

We all know what's wrong with many search engines; you either get no hits or 40,000. Why? Because there's no context for the search. You're looking for GATES, but are they gates for your garden, gates in a circuit, or are you a lawyer for the US Justice Department? Without a context the search engine must return all matches.

The normal way to provide this context is through a vocabulary. Many systems provide such frameworks, but if it's not complete enough for you to describe your service, you're out of luck until you can get your changes accepted. E-speak allows anyone to define a vocabulary at any time. This vocabulary is an e-speak resource like any other and can be discovered if it is advertised in another vocabulary. (We provide a base vocabulary that everyone understands to get things started.)

E-speak also provides a means to discover services on other machines, an *advertising service*. The advertising service is also used to form *communities*, groups of cooperating e-speak systems. In general, you can find any service offered by any e-speak machine in one of the communities you belong to.

9.3 Security

One reason we have so many problems with security on our systems is that the usage model is wrong; these systems were designed for an environment that didn't include the Internet. Not so for e-speak. We built e-speak for a world where each person wants to share part of a machine with one person and a different part with someone else. We also designed e-speak with the knowledge that bad or careless people would have access to systems belonging to others. Unlike many other systems, security is built into e-speak from the very beginning.

E-speak access control is based on two important concepts, fine granularity and dynamic roles. Fine granularity means that every e-speak service, every e-speak resource can be controlled independently of any others. In fact, individual operations on individual resources can be separately controlled. Dynamic roles recognize the fact that who I am is often less important than what job I'm doing. Sometimes I'm the engineer; sometimes I'm the Daddy. If I change jobs, my access rights as an engineer should go to my replacement. The right to discipline my child should not.

An important part of e-speak's security infrastructure is the ability to have e-speak messages cross corporate firewalls in a secure way. This feature frees the application designer from needing to worry about such issues. Because e-speak separates this process into separate parts, the application part and the security policy part, services are easier to write, and are more likely to work in a changing environment.

9.4 Manageability

E-speak mediates every request. There is a cost in latency, but a tremendous gain in manageability. Since the e-speak engine sees each request, we can rely on it to generate the appropriate management events. There is no need to build this piece into every application, verify that the implementation is correct, and make sure that everything gets updated for every new version.

E-speak provides the tools necessary to convert these low-level events into those appropriate for high-level management systems such as HP's OpenView®. Equally important is the fact that e-speak events are carried as e-speak messages. Since e-speak messages can cross firewalls, so can management events. This feature opens up the possibility of outsourcing system management. It is even possible to have different contractors manage different aspects of a single system. E-speak security ensures that no information leaks to unauthorized parties and that control of each part of the system is only possible by those given the relevant access rights.

10.0 Summary

E-speak is an environment that provides a number of advantages for service providers and their users. Figure 8 shows how the pieces fit together. The central part is the e-speak engine and the intermachine communication. HP is currently supporting the three platforms shown and provides low level libraries for Java, C, and Python programmers. There is also a high level library for Java programmers and a set of XML schemas that each provide abstractions suitable for creating e-services. The e-speak service bus and infrastructure services are key to making it easier to bring the next layer of applications and services to market.

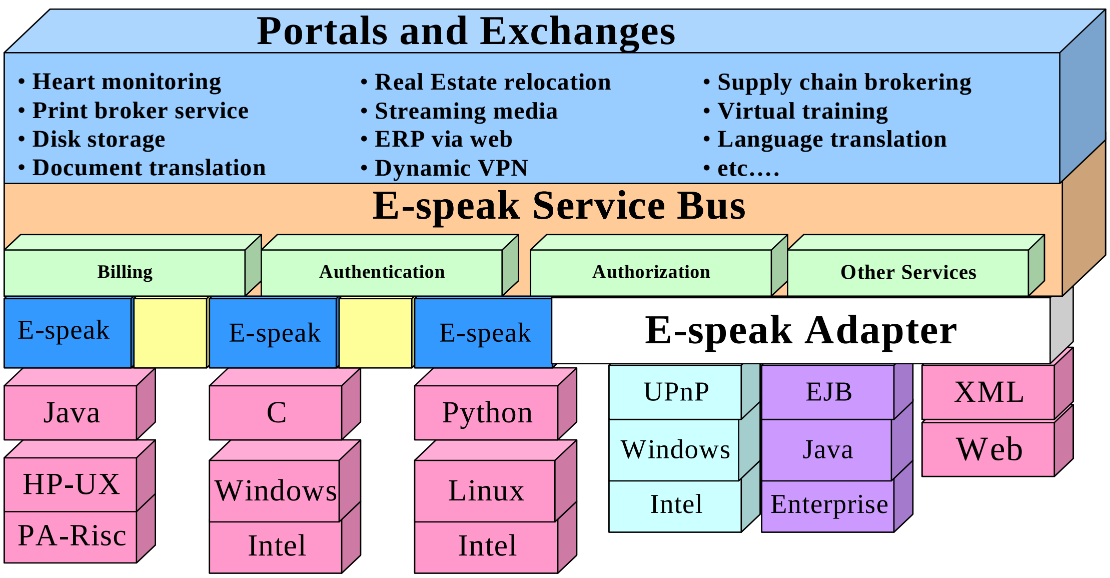

Figure 8. E-speak as a service development platform.

The e-speak adapter was originally developed so that e-speak services could be provided and used solely through the exchange of XML documents. In fact, by translating HTML into XML, we make it possible to provide services to clients who have nothing but a standard browser, one without so much as an e-speak plug-in.

Once we have the idea of an adapter, it’s not hard to envision including complementary solutions into the e-speak world. Prototype code has been written that allows any Jini enabled device or Enterprise Java Bean to be an e-speak service without any modification. The effect is that the Jini device or bean can be discovered, managed, and secured using the full power of e-speak. They also benefit from the language independence of e-speak and its ability to work on the Internet, including across firewalls.

In summary, what is e-speak?

> E-speak is an operating environment for the Internet that reduces the barriers to creating e-services.

How much does it reduce the barriers? Well, our first customer built the first version of its e-service in 3 months in mid 1999, in spite of our changing interfaces between the Beta 1.0 and Beta 2.0 releases. Our next customer took only 2 weeks in early 2000. Even more impressive was the company that called to say that they downloaded the open source material, studied it for a couple of weeks, and produced a working version of their service in only 2 days.

That's not good enough. The stated goal of e-speak is to reduce the time from idea to market to 2 *hours*. How is that possible? A partner is working on a tool that will allow a business planner to describe the business's logic in a UML (Unified Modeling Language) diagram, something being taught in the business schools today. If every node in that diagram is an e-speak service, the job is done. Talk about reducing the barriers to producing e-services!

11.0 Contributors

E-speak is the work of many hands. This section acknowledges the contributors who have had a significant impact. Since no work succeeds solely on its technical merit, I'll point out the significant management contributions as well.

11.1 Technical

Work on e-speak started at HP Labs in late 1995. Bill Rozas and I put together the first prototype. I coded the earliest version of the e-speak engine in Perl, and Bill produced an e-speak aware file system that included mirroring and code to allow us to move desktops from one machine to another. The latter did not make it into the product, but was instrumental in helping people understand our vision.

Arindam Banerji joined the project in mid 1996, just at the point where we were ready to rearchitect the system. The three of us disappeared into a conference room for a couple of months and produced the architecture that survived largely unchanged through the Beta 2.2 release. Throughout this time, Rajiv Gupta not only managed our department but also made substantive technical contributions. The e-speak e-services vision is largely his.

About the time we started implementing the new architecture, we added Chia Chao and Sandeep Kumar to the team. Chia and Sandeep largely developed the means of matching lookup requests with resource descriptions. Sandeep also produced an in-memory repository, one that survives today. Sven Graupner joined next, first as a student, then as a regular employee. He was the first one to use the new architecture to get two machines talking to each other, in spite of a rather sketchy inter-machine specification.

After HP decided to productize e-speak, several changes were made. Sandeep managed to get a persistent repository working with an Oracle database, while Chia took over improving the engine. Tom Rokicki joined the team and completely reworked the messaging layer. Mike Saboff did the same for the name resolution code. Tom also helped design and implemented the quota system. Wooyoung Kim took over developing the advertising service and recognized that it could be used to form communities. David Stephenson architected and implemented much of the e-speak management system. Mike Wray and Nigel Edwards produced the architecture for providing end-to-end security that first appeared in the Beta 3.0 release, with Mike doing much of the implementation. Naiem Dahti put together the picture of the e-conomy in Section 1.0 and introduced the term “e-conomy” to me.

11.2 Management

It is likely that e-speak would have amounted to little more than some conference talks or papers in dusty journals had it not been for Joel Birnbaum, the Director of HP Labs at that time. His support led to the beginning of the research, “How’d you like to work on this for 6 months?”, to bringing executives from HP and a number of companies to see the demonstration we put together, to helping us get funding.

Rajiv Gupta is the kind of manager who will tell you, “That’s the craziest idea I ever heard; let’s try it.”, an important attitude in the early days of the research. Once a decision was made to go to market, he put together an outstanding team, starting with Greg Kleiman as marketing manager and the first person Rajiv brought into the newly formed Open Services (now E-speak) Operation. Rajiv has managed a group that has grown from 6 people in late 1998 to over 200 by mid-2000. All the while, he has continued to push on the people he reports to, often in an environment where software was treated as only an adjunct to selling hardware. It is largely through Rajiv’s efforts that e-speak was released as open source, and he has continually fought to ensure that E-speak Operation is measured on the pervasiveness of e-speak, not revenue.

Arindam Banerji, a relatively new Ph.D. with no previous management experience has led the engine development team. Starting from the unenviable position of having researchers as developers, he put together all the pieces needed for successful product development. In spite of the aggressive, some would say unrealistic targets set for him, his team has yet to miss a deadline. Going from a prototype that almost worked in January of 1999 to the Beta 2.2 release in December, 1999, is a testament to AB’s ability to organize, motivate, cajole, threaten, do whatever it takes to make the targets. He managed to do this in an environment where start-ups were always recruiting his best people. It takes organizational skills beyond the ordinary to absorb the loss of key people without slipping the schedule. The fact that he has done all this with no previous experience managing a project of this scale makes is accomplishments all the more remarkable.