E-speak Revisited

Michael B. Spring, Marut Buranarach, Todd Schrubb, and Yanina Zatuchnaya
Department of Information Science and Telecommunications
University of Pittsburgh
spring@imap.pitt.edu
http://www.sis.pitt.edu/~spring

Table of Contents

Introduction

This paper describes design considerations for a distributed computing application, a marketplace, using the e-speak infrastructure developed by Hewlett-Packard. The University of Pittsburgh, along with a dozen other colleges and universities, received a grant of equipment and funds for graduate student support to design applications for this new technology and help students understand how to use the technology.[1] A web site was developed to provides information on how to write e-speak applications. The site also explores the design process and considerations.[2]

The first group of students involved were those in a course on client server system design. This course exposed students to the various paradigms for client/server design. It included discussion of various protocol designs and exposed the students to XDR, RPC, and RMI. A voice annotated slide presentation on e-speak and the project was developed to help orient students. Despite our best efforts, our first experiences with e-speak missed some important points. Based on that experience, a new annotated slide set was developed that captured more of what we had learned. This paper is based on that presentation.[3]

The marketplace design was for an "e'instruction" marketplace in the belief that students would need minimal orientation to the problem area. Students understand instructional processes and can work on the formalization of these processes more easily than they could if they had to come to grips with understanding the application area first. The design shares common elements with the design of a help desk or a knowledge management system.

The terminology, concepts. architecture, API, and utilities of e-speak were new and required time to learn. Hewlett Packard(HP) provided a lot of information and assistance. At the same time, we found that even core HP staff sometimes used one term to mean two different things and different terms to mean the same thing. The problem of terminology and definition is compounded by the fact that e-speak is still evolving. Finally, even if e-speak were stable and unequivocally described, it is difficult sometimes to understand the import of e-speak because it requires a new way of thinking, a new mindset. A recent paper by Alan Karp provides a clear and consistent overview of e-speak.[Karp2001]

What was learned in the process of designing an application? One of the first things that became clear was that e-speak is not a typical 2 or 3 tier client-server system. It has some similarities to RPC and RMI. Probably, if the students had had more experience with CORBA like architectures they would have been more ready for e-speak. The fact is that e-speak is a non-deterministic n-tier model where n gets large very quickly. The number of

[1] We would like to acknowledge the support of HP in the development of this effort. The ideas expressed in the slide set are those of the author and do not imply the endorsement or concurrence of Hewlett-Packard.

[2] The site bazaar.sis.pitt.edu provides a review of our design efforts as well as orientations to e-speak and a series of tutorials A companion site, talad.sis.pitt.edu provides information on distributed computing generally.

[3] This paper is based on a voice annotated slide show presentation providing an orientation to e-speak. The two presentations done on e-speak are available on the web at the following addresses:
http://bazaar.sis.pitt.edu/es_ppt_over/AIntrotoESpeak.htm
http://bazaar.sis.pitt.edu/es_ppt_over/AESpeakRevisited.htm

components and services that are interacting with each other is much higher than in traditional client-server environment. That takes a little bit of time to get used to.

The importance of very precise interface definition became clear. The modules to be developed were defined in terms of methods, parameters, and return values. In retrospect, the specification was rather sloppy. It became clear as the implementation got underway that the number of interfaces would grow dramatically. Failing to define them all absolutely in advance for the programmers assuming they would come to agreement as they moved forward was a mistake. Rather than converging the various components began to diverge in functionality. We discovered toward the end, and probably should have discovered earlier that to define those interfaces both for objects and for methods is incredibly easy using the E-Speak Interface Definition Language (ESIDL). I am sure we were told about ESIDL early on, but we failed to note that it was the RPCgen like utility that facilitates the implementation.

The critical nature of the vocabulary became clear. The e-speak literature talks about the centrality of the distributed extensible vocabulary for e-speak. It clearly is one of the more important and attractive components of e-speak. However, most of the examples we were exposed to involved little more than hand waving when it came to the design of the vocabulary. We learned that vocabulary was not nearly as easy to define as one might think. It is easy to wave your hands at problems of vocabularies early on, but there are a series of very difficult choices that need to be made related to the vocabulary. Early on in our efforts, working with one of our colleagues in business, we had written on the whiteboard the phrase "The vocabulary defines the marketplace". Nothing could be more true. The phrase remains on the whiteboard today and will stay a while longer. If you are going to become involved in e'services, keep in mind that next to the interfaces and the objects passed across them, the vocabulary design is a critical component that will tell you a lot about the rest of the design -- if you do it right.

Lastly, related to the n-tier nature of the model, it was not easy to conceptualize a service. Interfaces to services are not straightforward. As we came to understand the nature of this kind of marketplace, service were split into subservices, recombined into singular services, and otherwise redefined. These new services not only had their own set of interfaces but sometimes required a new vocabulary. It became clear that a taxonomy of services was emerging. What was found will be described later in this paper. Beyond the taxonomy set out in this paper, a broader taxonomy of services will emerge from best practices and academic analysis. As examples of the various types of services emerge, the task of modular design of those services will become much easier. For now, it will be a hard part of the task.

Overview

It is hard to understand the significance of e-speak without a sense of the kinds of problems it might solve. At the same time, not knowing what e-speak can do, it is hard to define a problem for solution. Thus, getting started with e-speak presents a problem. Should we talk about the e-speak technology and what kinds of things it might do, or should we describe a problem that needs a new solution. The dilemma is very closely related to the difficulty Tim Berners Lee describes in talking about the semantic web. [Berners-Lee1999, pp 157-175]

This paper begins with a conceptual overview of e-speak. The reader would be well served to envision the problem as one where there are 1000s of parties distributed across

a network interacting on a distributed loosely coordinated task. In a sense it is similar to web pages and web browsers. Maybe more to the point, consider a web site that has pictures of an office. The cameras snap a picture when someone visits. The pages also activate CGI scripts that look at the contacting browser and get some information. This information is passed to another server that looks up the IP address and resolves it to a domain name. Finally yet another program sounds a chime and displays the information on a screen. This interaction between browsers, web servers, scripts, personal servers and other programs provides a little better sense of what e-speak may lead to. We imagine that ultimately many similar services, having identical interfaces, would be provided in a marketplace where consumers having the appropriate consumer portion of the interface would come to shop. Like the visitors to the web page, the connection would not be constrained by prior agreement and will involve a number of programs working in close coordination. It will be a free and open marketplace for obtaining services in much the same way that the WWW is a free and open marketplace for obtaining pages or nodes of information. As a corollary to this, try to imagine that the service exchanges will tend to be micro transactions and small unit services rather than large long term connections.

The paper is divided into four parts as shown below:

- Understanding e-speak
- Designing a marketplace
- Implementing a marketplace
- Next steps – marketplaces and services revisited

What is e-speak

There are any number of ways to describe e-speak. It would seem to make the most sense to talk about it simply at first and then add information about its functionality. E-speak is first a component transaction monitor. In addition, it is a distributed system, a service providing system, and a system with decentralized controls.

It is easiest to talk about e-speak in the context of a particular engine or core. However, the most important points relate to the functionality of sets of engines and attached services interacting as a whole. The situation is similar trying to describe the World Wide Web(WWW) in the context of a single server with a couple of static pages. Based on experience it is easy to conceptualize the WWW because we have a lot of experience with it. Keep in mind that it took several years after Tim Berners-Lee had built the initial servers, clients, and protocols, before people actually began to use them in ways we consider obvious today. In those early days, before there were examples of what could be done, describing the technological components was not enough to help with the vision[Berners-Lee1999, p.31]. A similar problem exists here.

E-speak as a component transaction monitor

An e-speak engine provides access to a service. As such it is a component transaction monitor. This means that a consumer program is able to connect to the e-speak engine, select a vocabulary, and search using attributes and values of the vocabulary to find a handle to one or more services. The consumer program selects a particular service, and an interface of that service to which it is going to attach. Obviously, the consumer and service will share that interface. Once that connection is made, the consumer can engage

in a dialog with that service monitored by the e-speak engine. When that dialog is completed, a monitored component transaction has occurred between the consumer and service through the e-speak engine. This allows the e-speak engine to enable flexible security mechanisms and permit better management of services, i.e. auditing and accounting. This mediation is a feature that makes e-speak distinct from other distributed computing infrastructure [HPProgCh3, p20]. As Figure 1 shows, e-speak is a component transaction monitor.

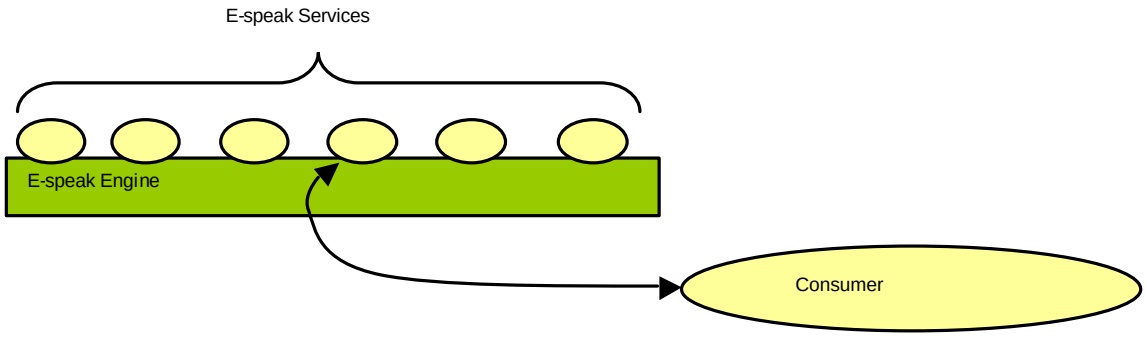


Figure 1: E-speak is a component transaction monitor

E-speak is a distributed service

It may well be that the service is not on the local engine to which the consumer is connected, but on some remote engine. E-speak provides a mechanism that allows the consumer to find a service located on a remote machine. This mechanism permits scalability for service lookups [HPProgCh5, pp 97-103]. From the consumer point of view the service is immediately present. No knowledge about its actual location is needed. Figure 2 shows a directory conforming to the Lightweight Directory Access Protocol(LDAP) [RFC1777]. This LDAP service is the mechanism that e-speak uses to locate services residing on remote machines that have “advertised” their availability. In reality there are a couple different mechanisms by which remote services might be found.

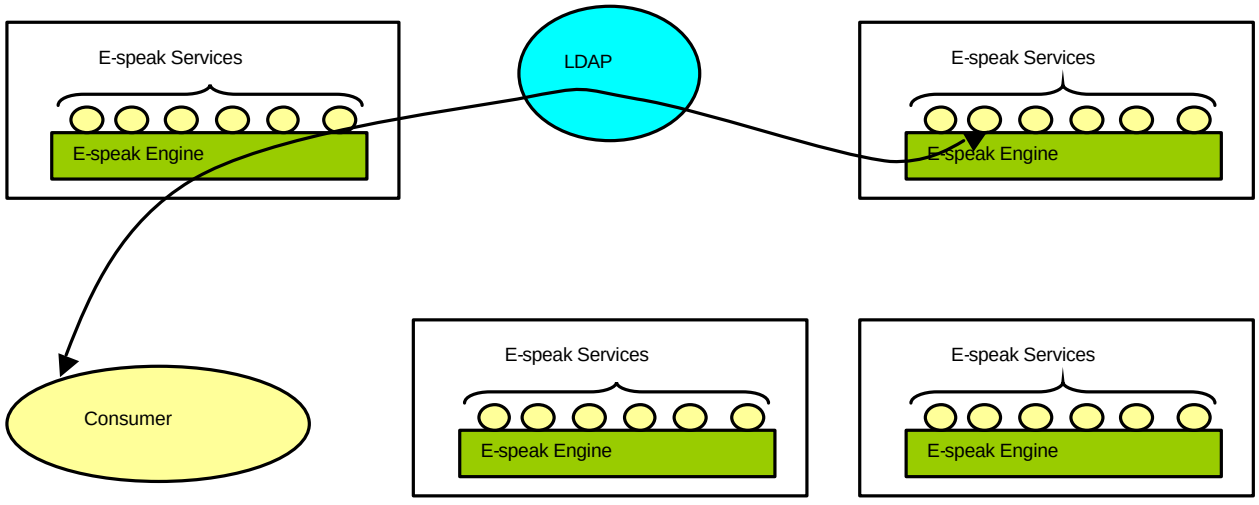


Figure 2: E-speak is a distributed service

The key point is that a consumer can connect to a particualr engine and find services located on other engines. It can then access those services and use them as if they were local services. This makes e-speak is a distributed service.

E-speak is a support structure

Like CORBA, e-speak provides a variety of services that support the development of e'services. There is some debate as to whether any services should be marked as special and differentiated from externally provided services. The goal, as put forward by Karp[Karp2001] is to minimize the set of special services. E-speak provides support for registration, advertising and finding based on vocabularies. It also has an extensive set of utilities that allow it to operate as a certificate authority and subsequently, using these certificates to provide for authentication and authorization. E-speak also provides for secure communications between engines and for access to the engines through servlets or other CGI type technologies operating as attachments to web servers. Each of these is described in a little more detail below.

Advertising and finding based on vocabularies are two sides of the negotiation process. As shown in Figure 3, when an e-speak service registers, it defines itself in terms of values of attributes of a vocabulary. The vocabulary and the attribute value pairs associated with a service may be stored in a local repository on a single engine or they may be advertised to other engines in a community using a Lightweight Directory Access Protocol(LDAP) service. A consumer may go to a local e-speak engine and use the local repository to find a service (the dashed line). If LDAP is used to share knowledge about service availability, the request to the local engine will use LDAP to access a directory server. Using the information obtained, the local engine can connect the consumer to the engine with the desired service and make access to that service available. The consumer, through multiple engines, then connects to and uses the service.

Engines, services, and consumers all have certificates. The service is provided with a certificate that allows it to connect to an engine and register or advertise itself. The consumer is provided with a certificate that allows it to find a service on an engine. A consumer will have certificates signed by the service that allow them to use the service.

Interestingly, one e-speak service might be an authorization service that would provide

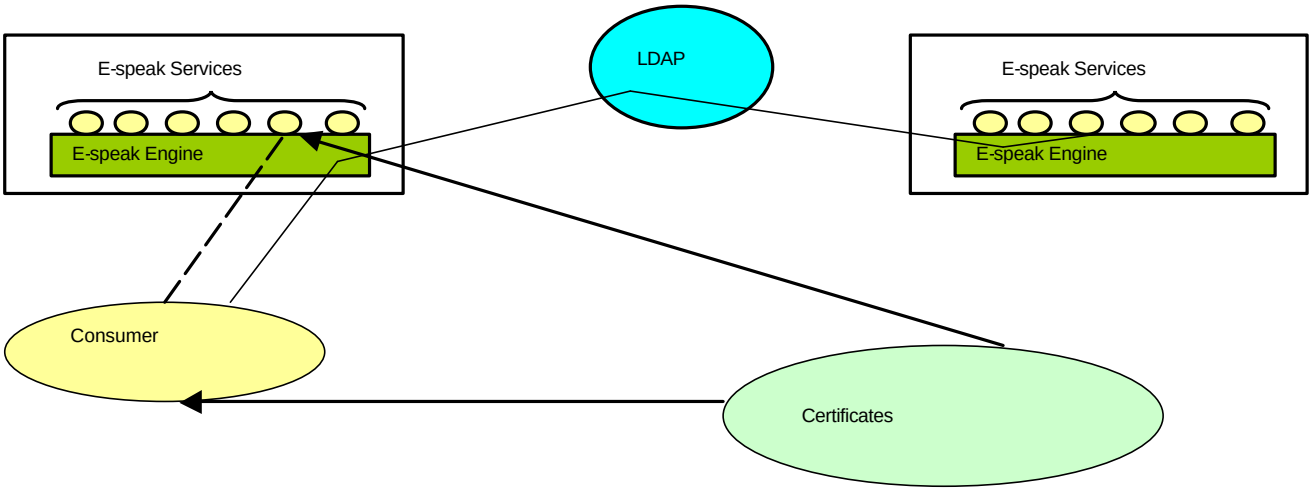


Figure 3: E-speak is a support structure for advertising, finding, and authentication

certificates to consumers. Services could indicate to the authorization service that they will trust individuals authorized by it. Services would then trust certificates from the authorization service as having legitimate access to their particular service. Thus, every service would not need to authorize every individual user. They could simply agree to trust the users certified by some centralized authority. This may help the reader to get a sense of how the n-tiered client server architecture begins to approach a large n. It should be noted that there are ways in which all consumers might be allowed to connect to an engine making it a public engine. Similarly all attached services might provide all consumers authorization to connect to them. So e-speak can be run, as it is at the University of Pittsburgh, as a very open environment. Or it can be run in a very controlled environment where only selected consumer programs can connect to certain engines and then only connect to certain services, and further only to selected interfaces of those services.

If a consumer is connecting to the service on a remote engine (as shown by the dashed line in Figure 4), the exchange between the e-speak engines is secured in an SSL fashion such that communications that cross firewalls are encrypted while traveling through unprotected parts of the internet.

Finally, the fact that requests are encapsulated as XML documents makes it possible to locate and access services using a web browser connected to a web server that has an adapter to the e-speak engine in the form of a servlet[HPProgSec4, pp 195-286]. This so called loosely coupled model for e-speak is still not full defined, but it provides a sense of the kind of connection flexibility provided by e-speak

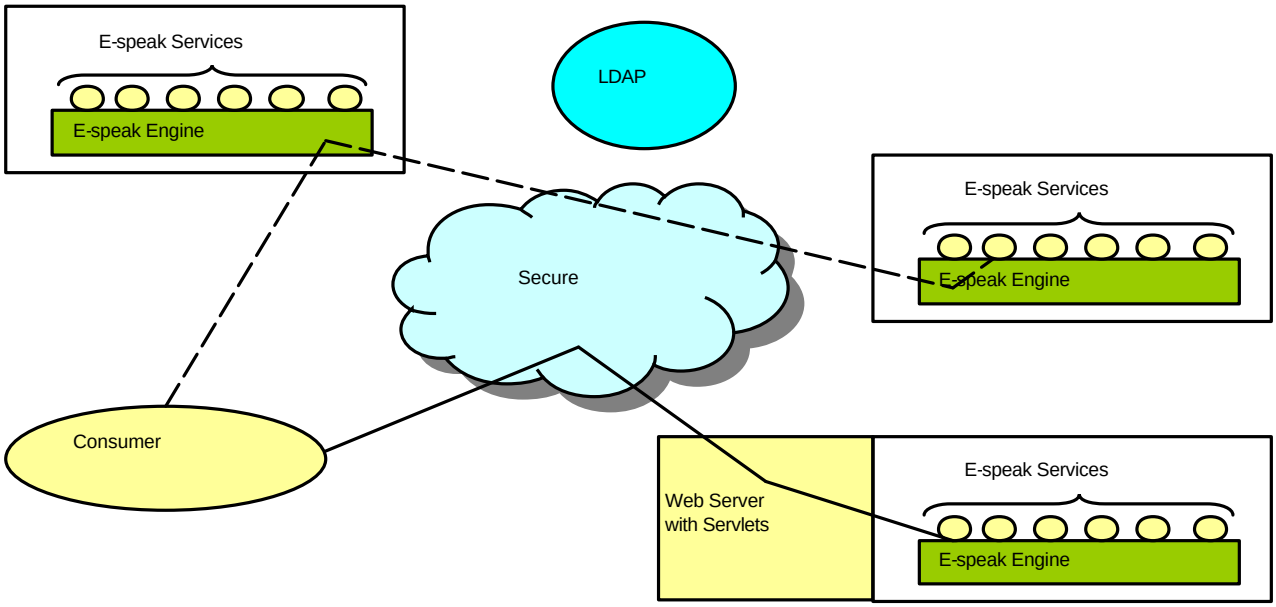


Figure 4: E-speak is a support service for secure transactions and open access

E-speak is an extensible system with decentralized control:

With these fundamentals in mind, it is important to talk about how e-speak provides an extensible system. Given the distributed certificates authority utilities, e-speak engines can be grouped in any number of formal and informal ways, with whatever level of secure

authentication is desired. Further, the vocabularies used for registration, finding, and advertising in an e-speak environment may be defined on an individual, group, large community, national, or international level. There is a standard vocabulary, basically the vocabulary for defining vocabularies. Communities can use few or many vocabularies to advertise their services. As a result, the markets may be simple or complex, small or large. This is one of the most significant benefits of e-speak, but as with any form of decentralization, it represents a potential for fragmentation rather that coalescence.

One important way in which e-speak is extensible is in the dynamic composition of services. It is easy to imagine basic services that people wish to access. This is the simplest form of e-speak. It is also easy to imagine that someone might write a service that would coordinate the access to two or more basic services. It is also not hard to imagine that someone else might write a service that would make the job done by several basic services easier, even though it would never be used directly by a consumer. These services could be provided by the market maker or by third parties.

The dashed line on Figure 5 shows a consumer connecting to the e-speak engine, querying the repository, which may be connected to a LDAP server working as a joint repository for multiple engines, and finding a particular service. The consumer is given a handle to a service and accesses the service through the e-speak engine. The solid line on Figure 5 shows a consumer connecting through some sort of front-end service to a basic service. This might be an enhanced rating service. The consumer finds the particular service they want through the front-end service. The consumer then interacts with the basic service, which may in turn use a back-end service. This back-end service might be some kind of document delivery or translation service. The service returns the object to the basic service. In this case, the consumer does not even know that the back-end service existed. It need only be known to the basic service. And the basic service might respond directly back to the client, or as the case in the example that we show here, it is responding back to the client through the front-end service. So vocabularies, and services, infrastructure, and marketplace that are supported are all extensions of the e-speak, are all ways in which it may be extended. In the implementation section, a more precise definition of these service categories will be provided along with some thoughts about their design and implementation.

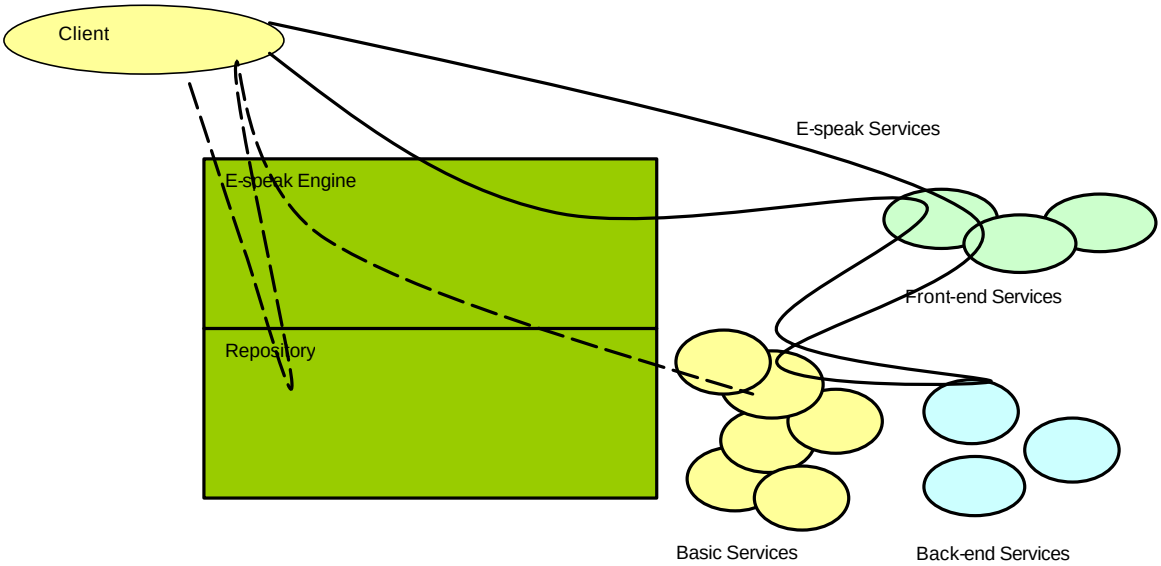


Figure 5: E-speak is decentralized and extensible

Terminology

The terminology used to describe e-speak can be confusing to someone trying to get a hold of all these new concepts at once. The following terms are important to understand and distinguish.

E-speak engine The e-speak engine is the basic unit of e-speak platform. The engine is the component transaction monitor. It is roughly parallel to an ORB in CORBA or to the web server in the WWW.

Repository The repository is attached to the engine locally or through a directory server and holds the vocabularies, attributes, and values used for both registration and for finding.

Service Bus The Service Bus is the distributed collection of engines, repositories, and services. The engines may be located behind firewalls with the ability to securely connect to other engines and communicate. Given the decentralized nature of e-speak, those collaborations are defined on an engine-by-engine basis. The service bus is a virtual bus; that is to say that it's a bus across which consumers connect to providers. There is no requirement that those be on the same physical e-speak engines. There may be multiple separate e-speak engines that provide, from the consumer's point of view, a singular service bus for identifying services.

SFS The Service Framework Specification(SFS) specifies the nature of conversations in which consumers and the producers will be engaged. Keep in mind that the line on the various figures between consumers and services may well represent an extended dialog, or multiple interchanges between a consumer and a provider. The structure of these dialogs is defined by the SFS of e-speak or some other dialog convention such as UDDI.

Registration Registration is a core service of the engine. It allows a service to make itself known so that it can be found.

Advertising Advertising is a core service related to registration. When registering, a service may choose to advertise itself, which uses LDAP to make itself known to other engines. There are various forms of advertising.

Finding Finding is a core service used by a consumer who would like to find a registered service. Services are found in terms of attribute-value pairs in the vocabulary and in terms of the nature of the interface that is supported by that service. Like registration, there are variations on finding. From the e-speak perspective, a group is formed when a set of engines use a shared LDAP environment to allow each other to know about services. In this case, the user does not necessarily know where the service actually resides. It is possible for a user to identify a series of e-speak engines or groups which are to be checked for given service. In e-speak terminology, such a collection of engines, defined by the consumer, is referred to as a community.

Standards

E-speak's communication protocol (ESIP) interoperates with the traditional transmission protocols, i.e. TCP, IP, or HTTP and wireless communication protocols, i.e. WAP or IrDA [HPArchCh1, p13]. The messages exchanged via these protocols may be market defined or they may conform to some existing message format such as EDI or BizTalk. These messages may be enclosed within envelopes as simple as mail or more appropriately something like SOAP. The structure of the dialog, as we have already suggested would be defined by such protocols as UDDI or SFS.

The security model of E-speak bases on the Simple Public Key Infrastructure (SPKI) and Session Layer Security (SLS) Protocol. SPKI provides authorization mechanism using certificates. SPKI certificate [RFC 2692, RFC 2693] can also interoperate with the ISO/IEC X.509v3 certificate [RFC 2459]. The SLS protocol provides secure communication between clients, e-speak cores and resource handlers [HPArchCh5, pp121-139]. The SLS protocol is based on Transport Layer Security Protocol (TLS) [RFC 2246].

The role of XML is multiple. Obviously, it plays a role in things like document translation and document transformation, but it also plays the role of being the basic syntax in the exchange of data structures that are to be moved back and forth, consistent with the growing use of XML as a Data Definition Language (DDL).

E-speak Envisioned

The preceding section offers a technically correct introduction to the e-speak technology. There is much more behind each of the points made and the interested reader is referred to the architectural specification of e-speak and to the JESI API for more details. While the description provided is reasonable, it is incomplete. The limits of the language and the graphics we have used may have simplified the picture too much. Keeping the following eight points in mind will help to expand the vision.

First, an accurate vision of e-speak is one where there are millions of machines engaged in thousands of marketplaces: Thousands of machines will be providing services in individual marketplaces consisting of hundreds or thousands of servers at varying levels of coordination. Tens of thousands of users connecting to the services will engage in millions of transactions. If you can adopt this mindset, it is a little bit easier to see that whether decentralized services come or go, given that there are hundreds or thousands of similar services will insure continued access. Thus, the system is meant to help negotiate relationships between tens of thousands of users of services and thousands of providers of services, where many of those services are similar. This is different conceptually from a mindset that views Amazon.com as a single critical service provider. Rather, imagine that bookstores on the service bus are like gas stations on your way to work, except that the ones on the service bus are all equally convenient to access. If one or another closes, there are always more to access.

Second, if this is to come about, it is essential that someone structure the marketplace in which service consumers and services providers can meet. If the marketplace is poorly constructed consumers and providers will find rendezvous difficult and will shun the marketplace. If that marketplace is well constructed and easy to use, a lot of people will come and use it.

Third, it is important to understand that there will be competing service providers, not a single monolithic provider. Move aside Amazon.com; enter lots of smaller specialized e-tailers who can now efficiently compete because they can make use of marketplace and infrastructure services as needed. Although there are situations in which we can imagine e-speak working well with only a single service provider, the technology is not very compelling when there are not competing service providers.

Fourth, the offerings will be dynamic. Given a well defined interface across which a transaction occurs, the processes that are engaged behind that interface, can be changing on a very dynamic basis, that new players can be coming in offering better services because of behind the scene processes.

Fifth, it is important to understand that every e-speak marketplace can operate under a distributed certificate authority. This is in contrast to the current model, which has a limited set of recognized certificate authorities such as VeriSign. Distributed certificate authoritiess, within marketplaces will become commonplace. E-speak allows communities to be developed, and if they decide that there is a need for authentication, that e-speak community can become its own certificate generating authority. With time, this small community may distribute certificates that will be trusted by another community. In this way, e-speak will contribute to the "web of trust" conceptualization. The central notion here is that we now have the capability of securing an environment in which local communities have the capability to authenticate their own members and further to act as trusted partners within a "web of trust."

Sixth, envision many intermediaries in the process. Where one of the visions of the WWW has been disintermediation, as Brown and Duguid have suggested, there is likely to be a reintermediation on the web.[Brown2000] E-speak is one mechanism that might allow for intermediaries of this new form. The market place maker is one form of the intermediary. But others can define intermediary services between consumers and providers. So, there are multiple layers that might be imagined. We have already suggested the notion of rating services as intermediaries.

Seventh, think about smart devices as consumers. As hand-held and embedded devices populate the web, they can connect to the marketplace and access services that enhance their capabilities.

Eight, and finally, it is important to imagine an environment populated with knowledgeable consumers. The process of working in the World Wide Web today has been a process of asking for information, simply asking for a document. And much of what is going on in this environment has been a matter of simply browsing pages. The e-speak environment imagines that consumers are somewhat more aware of what they are looking for and of their ability to conduct transactions. A smart consumer wants a particular service, knows how this service is defined, knows the transaction will be secure, and is willing to go out into environment and find and use it. This is a little bit different from a casual browsing that we image in many of the World Wide Web situations.

The System in Overview

Figure 6 attempts to show e-speak in its totality. Beginning at the far left, a consumer exists behind a firewall. This consumer may be an individual looking for a service via an interactive program or a program triggered by an event in an Enterprise Resource Planning System that is looking for a product. The consumer makes a connection to the local e-speak engine. That e-speak engine, like the consumer program, resides on a

machine behind a firewall – indicated by the heavy dotted line. If they are looking for a service, which does not happen to be on that engine, the engine seeks information provided by an LDAP server. The request is transferred securely as an XML document through the Internet and across the other firewall to another e-speak engine. In the center, in addition to the centralized directory, there may be some form of open marketplace. The light-green circle through which the solid line is going suggests that the consumer is making use of some kind of marketplace service to access the basic service. It may indeed be in a process of actually connecting to a remote service based on the LDAP

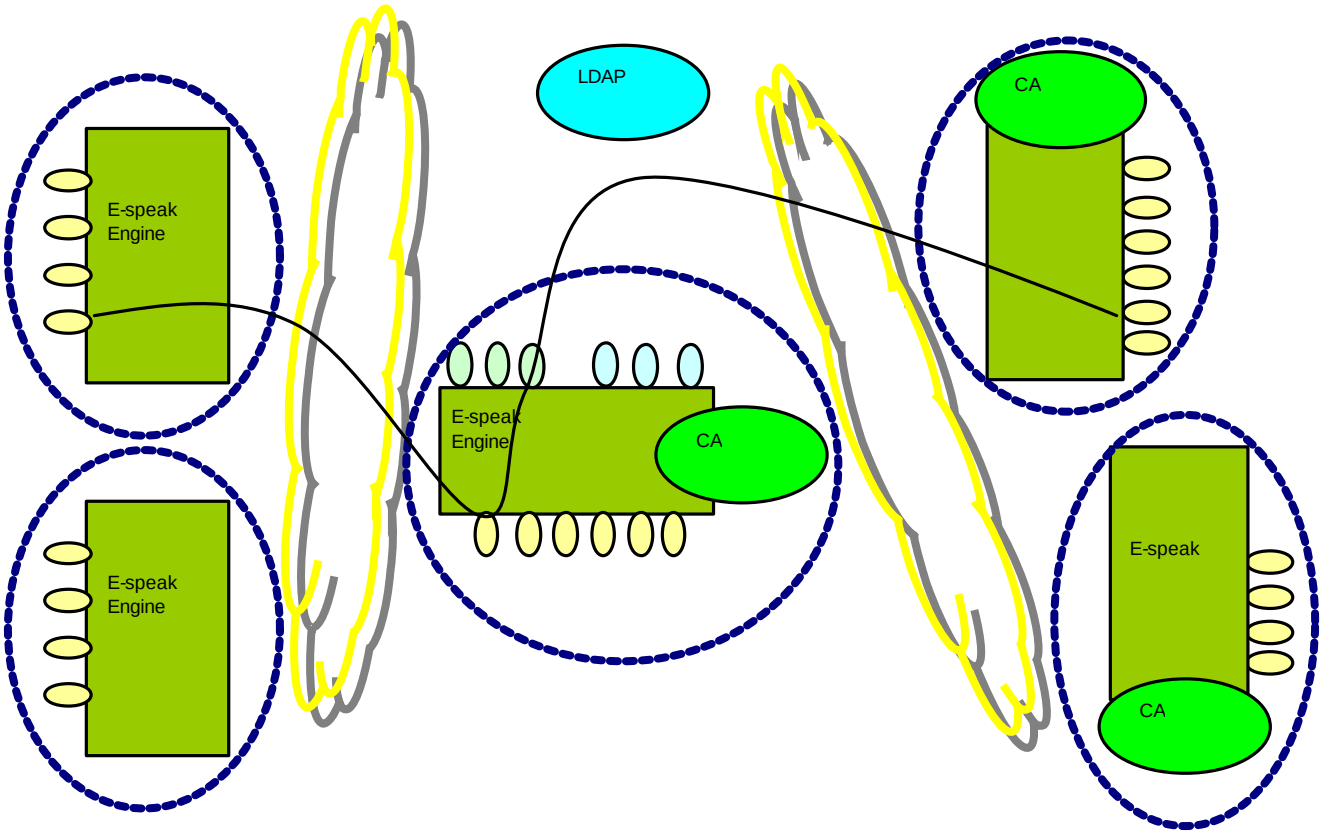


Figure 6: The Big Picture

information. Via the marketplace service, the consumer again securely traverses the firewall and transfers information across the Internet in a secure fashion to the e-speak engine that has the actual service that user wants to engage. Notice that centralized e-speak engine, as well as remote e-speak engine on the far right, have certificate authorities (CA) attached. In order to register, find, and engage services, both the provider and the consumer will use certificates and webs of trust to establish their right to undertake actions.

Finally, It is important to understand that once that secure connection is made through the firewalls to the remote engine, the single line may represent a whole series of interactions between the consumer and the provider. This set of interactions represents a dialog or conversation that will be defined in some framework. Note that there are many e-speak engines, some of which are not engaged in this particular transaction, that may have consumers attached to them and that may go through some centralized kind of marketplace. The marketplace may provide any number of different kinds of services. It is possible to imagine basic services that consumers are attaching to on the engine on far left and it is possible that services that are advertised might come from multiple engines in

a community; some of which are selected for a given transaction or conversation and some of which are not. This is the broad vision of e-speak.

The Design of a Marketplace

Setting up the Project

The goal of this effort was to build a marketplace and core services. The project constituted the final assignment in a client server course. Because it would take place as a group project, and because a high degree of coordination was required, it was decided to structure it along the lines of a software engineering project. The instructor would serve as the architect, the graduate students supported by HP as designers, and the students in the course as implementers in groups. The architect spent time prior to the term building a vision for the project including both a functional and structural model. A number of potential projects were considered and the capabilities and requirements of e-speak system were explored. As the fall term began, the graduate students who were to design the system began to define particular interfaces in terms of the vision set out by the architect. This was more difficult than we imagined for two reasons.

- It wasn't clear that the architect's vision was correct. In essence the vision was one where the vast majority of services were representations of individual students or documents. Thus, a student could post a document that would reside as methods for delivering the document, translating it, billing for it, streaming it, etc. Similarly, students would connect to the service bus as available and share their expertise interactively. Assuming that 100's of students were involved, there would always be some subset of the possible services attached to be found.
- It took a long time to communicate this vision to the graduate students who had to do the design. It is not clear that the graduate students internalized the vision. For one thing, they were focused on very immediate problems of getting code to function. The system as it finally emerged had many more "brokers" than the architect imagined. In essence what the instructor had imagined as producer services accessed by consumer services became consumers of a broker service. The broker service was then also the provider of service to the consumer. In essence, the vision of a very decentralized system of service providers and consumers was transformed step by step into a more conventional monolithic database application. The DBMS and the repository began to provide duplicate functions.

The vision also had to be communicated to the graduate students who would actually implement the code. Building a vision, communicating it, and implementing it, present real challenges in terms of communication, particularly in an environment where the vision is based on a new mindset. The reader is invited to assess our introduction to e-speak (see reference in the introduction). While the feedback on this presentation has been positive, we conclude that it was lacking in communicating the mindset needed for e-speak and the essential vision of the marketplace.

The designers and coders tended to build the kinds of systems they built in the past. The vision of the architect, which we believe is consistent with the new capabilities provided by the e-speak, was not adequately communicated to the participants.

Despite the problems, the plan to have architects, designers, and implementers appears sound. A single individual serving as an architect can bring clarity to the project and

resolve questions about the modification of the design. The designers should focus on the interfaces, objects, and persistent stores. As will be described below, the use of javadoc and the ESIDL compiler greatly simplify this task. The high level vision for the project set out here is tempered by experience over the first iteration of the effort. The insight gained by hindsight is significant. While it was clear that the task was to build a marketplace, it was not as clear that marketplace development was more involved than simply defining services and vocabularies.

The initial vision for the system was one with many micro transactions. They are micro transactions in two ways. They are transactions between individual consumer and provider – not between collectives. Second, the services are as small as possible. Rather than extended interfaces, to huge monolithic services, we wanted small focused services that could be combined in different ways to provide more significant services. Even in the third iteration of the design, we are struggling to make the services less monolithic. At every turn, the designers find another reason to create brokers rather than individual services. The logic is compelling, but the result is that the design gravitates to big monolithic brokers rather than sets of services that compete.

The marketplace envisioned was one made up of programs representing a set of talented graduate students. Surrounding the traditional classroom experience these students constitute an expertise marketplace. Students help each other on a regular basis to do things, so the expertise of the individual students creates a new kind of the market place for the exchange of that expertise. Students create programs, they create notes, they create papers. These are all artifacts that may have value to other students.

There are additional services that we can envision for this marketplace, but basically, it is a marketplace of experts and consumers of that expertise. From a design perspective, students and their expertise are no different than vendors in a broader marketplace. The various artifacts that students produce, answers to questions and notes they make, are like products.

Given this vision of the marketplace, the next step is to provide the support services. The support services might involve transaction monitoring, with the marketplace monitoring the cost of a question and answer session and inputting that information into a billing service. The assessment could be based on the amount of time it took, or the number of words exchanged, etc. The marketplace also needs to provide a billing service and along these lines, might provide for anonymity through escrow services.

Functional Architecture

Figure 7 shows a functional overview of the system. The system surrounds and supplements a traditional classroom environment. It consists of a collection of services grouped roughly into four classes – Q&A, Notes, Whiteboard, and Auction.

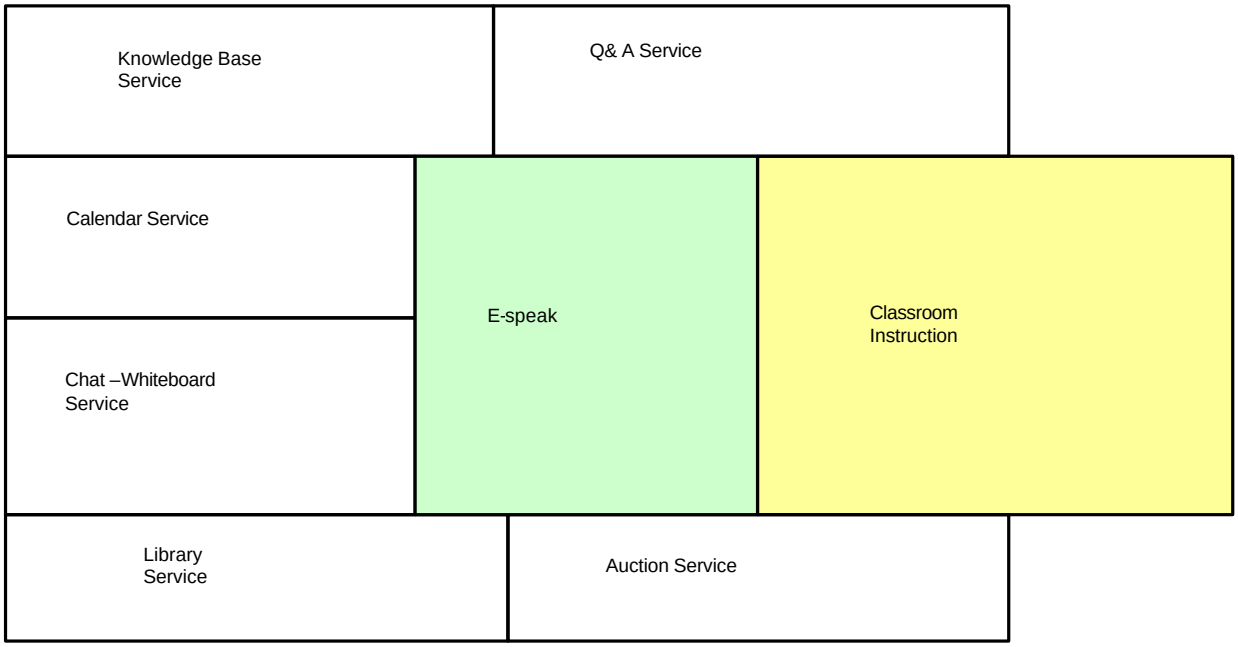


Figure 7: Functional Architecture for an E'instruction Marketplace

The first service, the flagship service of the marketplace, is the Q&A service. It is based on the simple notion that students do ask each other questions all the time. The goal was to make what happens in same time/same place situations work across space, and potentially across time. A student would attach to the e-speak bus and advertise themselves as an expert able to answer selected questions. When they answer a question, some form of exchange will take place representing a charge. Related to the service is another for storing the transaction – i.e. questions and answers might be stored for the future use. Thus there might be knowledge base service built on the Q&A service that would allow a question to be answered from the store if an expert were not available on line.

Down at the bottom left-hand corner is Library service. This represents a collection of individual Note services. We imagined that some forms of presentation might be so compelling as to cause students to buy them. The instructor, by way of experimentation took a half dozen powerpoint presentations and voice annotated them. He then invited students to look at them and assess how much they were worth. As we explored this notion, we found all sorts of ideas. Some students make batter class notes than others -- these might be exchanged. Some students take class notes in their native language – these might be of value to some. As we explored this notion it began to expand – format translations on the fly, streaming presentation of notes, etc.

Students might want a Chat-Whiteboard service focused on their academic area or area of personal interest. A Calendar service provided an opportunity to begin to explore how general failures of group calendaring might be overcome through a brokering system.

Finally, the vision imagined a local Auction service where the graduate population, which involves many international students, might use the service to buy apartment furniture

when they first arrive and to sell it when they leave. Such a limited auction might allow students to exchange items at low cost.

Structural Architecture

Given this basic functionality, the structural architecture defines the arrangement of infrastructure and marketplace services needed to support them. The design process failed to structure the dependencies and the information needed so that essential services were defined first allowing the system to be constructed efficiently. Despite the failure to define construction dependencies, the structural vision was correct.

As shown in Figure 8, consumers (students) would attach to basic services like Q&A or Notes through the core. Notes services and Q&A services registered in terms of the area of expertise, the nature of the service (interactive, mail, etc.) The student connects to the core. The core, in one case, connects to the given Q&A service. A student at the lower left is shown connecting to one of the Q&A services. The Q&A Service then connects to another individual who provides the answer. Note that the Q&A service makes a connection to the banking service to record the transaction.

Another student (brown) makes a connection through the core to one of the Notes services (tan). The Note service makes use of a Delivery Service to provide the document to the consumer. In this case, the Delivery service is an infrastructure service for the Note service. Again, like the Q&A service, the Note service makes use of the Banking service to record the transaction.

It should be noted that the full extent of the interactions is not shown here. In implementation, the consumers would also use the banking service to authorize transfers to the Notes service and the Q&A Service. Indeed, as the system now stands, there are more than a dozen services to assist, broker, support and transform basic services.

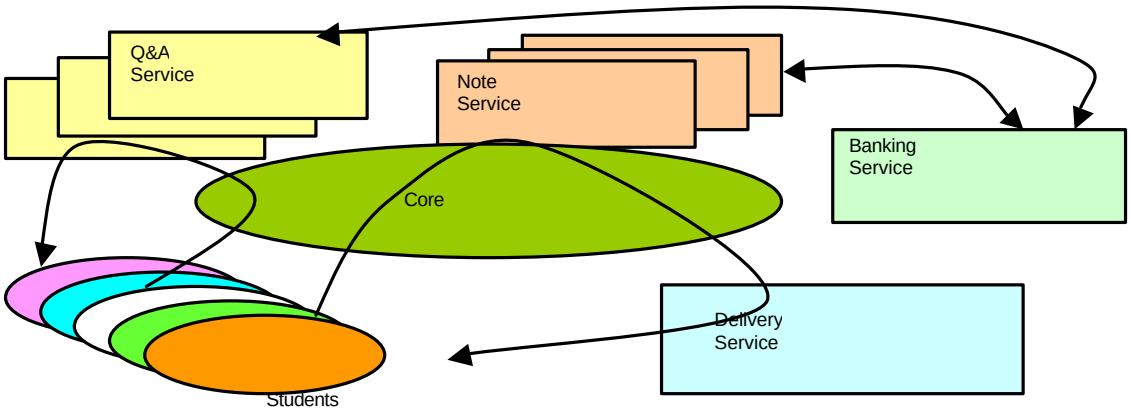


Figure 8: Structural Architecture for an E'instruction Marketplace

Project Terminology

It is difficult do talk about those things for which you do not have language. This is surely the case in the development of a marketplace. Beyond the terminology provided by HP, it was useful to define some additional terms for the project.

Marketplace — The marketplace is the collection of basic services and the services they make use of. The marketplace is best associated conceptually with the service bus – i.e. it may span multiple physical machines.

Market Services — The marketplace services include those provided by the market maker as well as those provided by third parties. Marketplace services are required to make basic services work. Without them, the marketplace cannot function. While marketplace services will normally be essential services, they may in some cases be optional. The banking or credit exchange system for the use of services is a good example of a marketplace service.

Infrastructure — Infrastructure services may be provided by the marketplace or third parties. They support basic services but are not required by them. The distinction between marketplace and infrastructure services is twofold. First, marketplace services are owned by the market maker. Second, infrastructure services can not be essential to the operation of the marketplace. Infrastructure services are the services that enhance the marketplace but are not required to make basic services work. At this point in time, we have distinguished two forms of infrastructure services – front-end and back-end. Back end services are those used by the basic services and as such are generally invisible to consumer programs. Front-end services are those that are accessible to the consumer. They may broker or dynamically compose basic services. From a consumer perspective, a front end infrastructure service may appear as a basic service.

Basic Services — A basic service is one that is responsible for producing a product or a service to be acquired by a consumer.

Consumer — A consumer program is the mechanism through which a basic service is obtained by the consumer. It may be directly accessed by a human in which case the consumer program consists of a graphical user interface for interaction with a person, some kind of business logic to manage the conversation, and a series of network interfaces to connect to the e-speak service bus, and find and engage services. We believe that in most e-speak applications this consumer program will attach to an internal system within an organization, such as an Enterprise Resource Planning System. The interaction will be triggered by some event – either internal to the organizational system or on the service bus.

Producer — Producer programs are in essence programs that produce basic services. In our project, in order to generate a critical mass of basic services, we imagined services that differed only on vocabularies. In recognition of this, we defined a “producer program” as a program that could generate hundreds of different basic services where the difference in those basic services was the values of the attributes in the vocabulary

used in registration. There is no reason why producer programs might not produce basic services that differ in functionality but still use some defined interface. With this in mind, we initially defined producer programs as programs that would query a user and based on the inputs generate code that could be compiled and registered as a service. In the final analysis, we simply used a single piece of code that used different values for the vocabulary attributes to register a service. Thus, many basic services were actually producer programs that were differentiated by the files they read at start-up time.

Clarifying Assumptions

It may still not be clear how things actually work. How is the software distributed? Where do the services actually exist? How are the interactions fostered. The reality is that it will operate in many different ways. Exactly how is something that will only be known with time. However, for purposes of orienting people to the system, we will make a series of assumptions that we hope will clarify at least one way in which the system might work. First, there will be sets of services. When a consumer attaches to an engine, and thus the service bus, they will find a whole set of services by querying the repository using a vocabulary. The services will have identical interfaces because they will all have been produced by the same producer program. (There is no reason why various services could not be hand tooled to comply with a defined interface.) In this case, having a single program produce all the basic services guarantees interoperability.

A consumer program is capable of attaching to any one of the basic services – sets of which differ only in terms of their attribute – value pairs. But how does the consumer get the software? We assume that they connects to a web site associated with an e-speak engine and download a signed jar that contains the application. Thus, the consumer side is a thin client. When a user wants to access services, they download a program that has the right interface and access the service bus through a local engine to access the services. We are currently toying with services (marketplace or infrastructure) that would allow the consumer to dynamically update itself as new interfaces are defined.

The vocabulary provides the basis on which the marketplace exists. The vocabularies are known either through web access, for semantics and explanation, or through consumer software that has been downloaded.

One way to think about this environment is as a rich RPC/RMI like environment that uses http for support and software distribution. Once the services are found and the software distributed, the browser and web server move to the background and the more specific software takes over the dialog.

Simplifying Assumptions and Mechanics

A series of simplifying assumptions and standard operating procedures were also adopted. One of the great capabilities of e-speak is that it can deal with Internet appliances and Wireless Access Protocols. We chose in this first iteration not to include any of these. Plans were made, for example via the translation service, to allow them to be developed in the future as they are important long-term applications of this technology.

We decided not to use the web interface for finding. This mechanism is well defined in e-speak, and it will be is a very convenient function in the long run, but we felt that at this

point it was important to simplify the task. Given the plan to use a tightly coupled e-speak application to invoke the services, we decided to run the whole process in a tightly coupled mode rather than complicating the situation by having to develop some steps in a loosely coupled mode and others in a tightly coupled mode.[4]

As indicated, to overcome the problem of producing enough services that have the same interface we chose to engage the notion of producer programs that will be used by individuals to produce services. This greatly simplifies the process of producing the service, taking away much if not all of the coding. The individual who wishes to mount a service simply defines the service in terms of the vocabulary. As indicated consumers will be thin clients that a given user can simply download from a web site.

Implementation of a Marketplace

To implement the marketplace, we needed to specify the vocabulary, define the interfaces, and build the components. Defining the vocabulary, the attributes, and the range of the values for those attributes was assumed to be an easy task until we started the process. Defining the interfaces also seemed easy – simply define the input parameters and the return values for each of the methods defined by the service. With these defined, the components could be coded and the system would be up and running. While we imagined these steps as an easy sequence, implementation required a system of rapid prototypes that allowed iteration through the three steps several times in order to work out the kinks. The vagaries of each of these interwoven activities are described in the sections that follow.

Vocabulary Design

Vocabulary design involves the resolution of a number of issues. Existing examples of vocabularies tend to be simplistic – e.g. finding a cab in New York City or accessing a printer. There are several issues that need to be answered in defining a vocabulary. The ones that seem most important at this point in are the following.

A market place is made up of a number of classes of services. It is possible that there will be an overlap in the vocabularies for these services. Should each service class have its own vocabulary, or should a combined vocabulary be developed for the services? If a shared vocabulary is used, how should attributes not relevant to a given service be handled?

The size of the vocabulary needs to be determined. This issue involves two questions. How large should the attribute set be and how large should the value set for an attribute be? What should the attribute to value ratio be? Should there be many attributes with few values, or few attributes with many values?

All of these questions have impact both on service design and on the interaction with the consumer. Will users want to obtain unique services based on a search or will they want to obtain a set of possible services from which they can select the best. Will users be

[4] In e-speak terminology, access to the engine using programs written in conformance to the Java E-speak Interface (JESI) API is tightly coupled. Access using a web server and XML exchanges is loosely coupled. As of this writing, only selected aspects of interactions are provided for under the loosely coupled model.

allowed to search on partial subsets of the attributes or will they have to specify all of the attributes?

Lastly, and probably the most important issue has to do with the nature of the values used. This begins to focus on the semantics of the vocabulary. At the simplistic level, consider the issue of single or multiply valued attributes. In the example we propose, area of expertise is a potential attribute. The values of the attribute might be the areas in which a person is expert. Should we have one value for each area and have the attribute define the extent of expertise, or should be have expertise as the attribute and allow the area to be defined as a value? Should the values of a multi-valued attribute such as expertise be predefined or should providers and consumers be allowed to use free form strings? How easily extensible is each approach for the provider and the consumer. What are the impacts on usage?

It is possible in e-speak to change the values of attributes. This begs the question of whether the values should be static or dynamic. For example, it is possible that a user would want to use a service that has been used by lots of consumers. A service could update a value related to its usage each time it is used. Should the data in the values be restricted to data used to find a service? Or can they be used to tell us something more about a service? So for example, a cumulative rating of a given service could be included as an attribute-value pair. This involves each service regularly updating a value associated with an attributes of the service. In the last analysis dynamic data values were deemed inappropriate for the uses we imagined. Separate services should be defined for them. Indeed, a trusted independent service to provide ratings would probably be the only way that consumers would trust the ratings.

Interfaces and Objects

The single biggest mistake made in the first effort to define a marketplace was the failure to use ESIDL (E-speak Interface Definition Language) to define the interfaces. Initially, methods, parameters and outputs were specified using a modeling language. While this is essentially the same process, it does not allow use of the ESIDL compiler to simply and quickly generate the framework code. It was only after we went well through the initial design when we realized that the compiler (the Interface Definition Language compiler for e-speak) did a marvelous job of defining serialization of the objects across the interfaces, defining the methods that would be used on the interfaces, and of providing easy access to javadoc to provide documentation.[5]

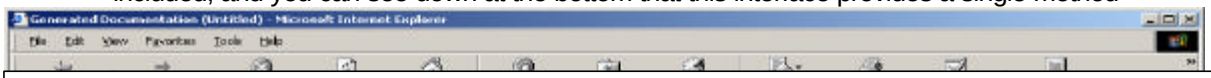
Despite the benefits of using ESIDL, it is still going to be true that there is an iterative process in design of the interfaces. There are two forces that pull at the definition of these interfaces. The first is the optimization of DBMS tables that support the services operating in the marketplace. If there are DBMS needs in the marketplace and not just at the consumer and provider ends, the functionality of the DBMS and its structure will impact the nature of the interfaces. As we went back and forth, we found that there was an intimate relationship between interfaces and DBMS tables. We also found that as the interfaces were defined, we noted places where services we had seen as separate had to be combined and other services needed for one reason or another to be split apart.

[5] Given our experience with RPCGEN and rmic, it is not at all clear how we missed this step and it is rather embarrassing to admit it here, but if it saves others from making the same mistake the embarrassment may be worth it.

The E-Speak Interface Definition Language provides the kind of specification required to allow teams to write basic services or consumer programs that work together. In addition, ESIDL, because it is simply defining a java class or java interface, allows the user to make use of javadoc, which provides documentation of an interface and its functionality.

E-speak uses its own form of serialization for objects that are exchanged across the various interfaces. If you have a series of object classes in a directory along with the interface class that makes use of those objects, the ESIDL compiler not only compiles the main class also automatically processes all of the classes that have been defined as objects to be passed across those interfaces. For this reason the ESIDL compiler is a very powerful utility, especially when used in connection with javadoc.

The text box below shows a simple ESIDL file. It defines the imports and a public interface to the banking service. You can see some of the javadoc comments that have been included, and you can see down at the bottom that this interface provides a single method



```
/**
* BankingServiceIntf is one of Banking Service's interfaces.  Acting as a back office,
*  this interface provides methods for Service Brokers such
*  as QAService, NoteService and RatingService.  @author Ganokporn Rungsritampanya
@version %I%, %G% @since JDK1.2.2  */
import net.espeak.jesi.ESService;
import net.espeak.infra.cci.exception.ESInvocationException;
import net.espeak.infra.cci.exception.ESServiceException;
import java.sql.*;
import java.util.*;
public interface BankingServiceIntf extends ESService {
   /**
    * Transfers money from  one account to another.
    * @param  fromAccId          The account from which the money is transferred.
    * @param  toAccId            The account to which the money is transferred.
    * @param  transAmount        The amount to be transfered.
    * @param  serviceName        The broker, QAService or RatingService, who call the method.
    * @return                    <code>refNo</code> if the transfer is done successfully;
    * <code>an error code</code> otherwise. Error code example:<ul>
    * <li>-101 Receiver account not found,
    * <li>-102 Sender account not found,
    * <li>-201 Receiver account closed,
    * <li>-202 Sender account closed
    *  <li>-300 Not enough balance. </ul>
    * @exception          <code>ESInvocationException</code> If an invocation exception occurred.
    * @exception          <code>BankException</code> If a bank exception occurred.     */
   public int fundTransfer(String fromAccId, String toAccId, float transAmount, String serviceName)
     throws ESInvocationException, BankException;}
```

Figure 9: ESIDL File

called fundTransfer, with the defintion of the exceptions it throws.

The definition is run through the ESIDL compiler to produce the Java classes that make up the framework for the service. The same file, run through javadoc, provides the kind of documentation shown in Figures 10 and 11. It defines the parameters, return value, and the kinds of exceptions that are thrown. This definition, supplemented for developers with business processing logic, coding suggestions, and information about DBMS tables and inputs, provides programmers with a very clear definition of what is required.

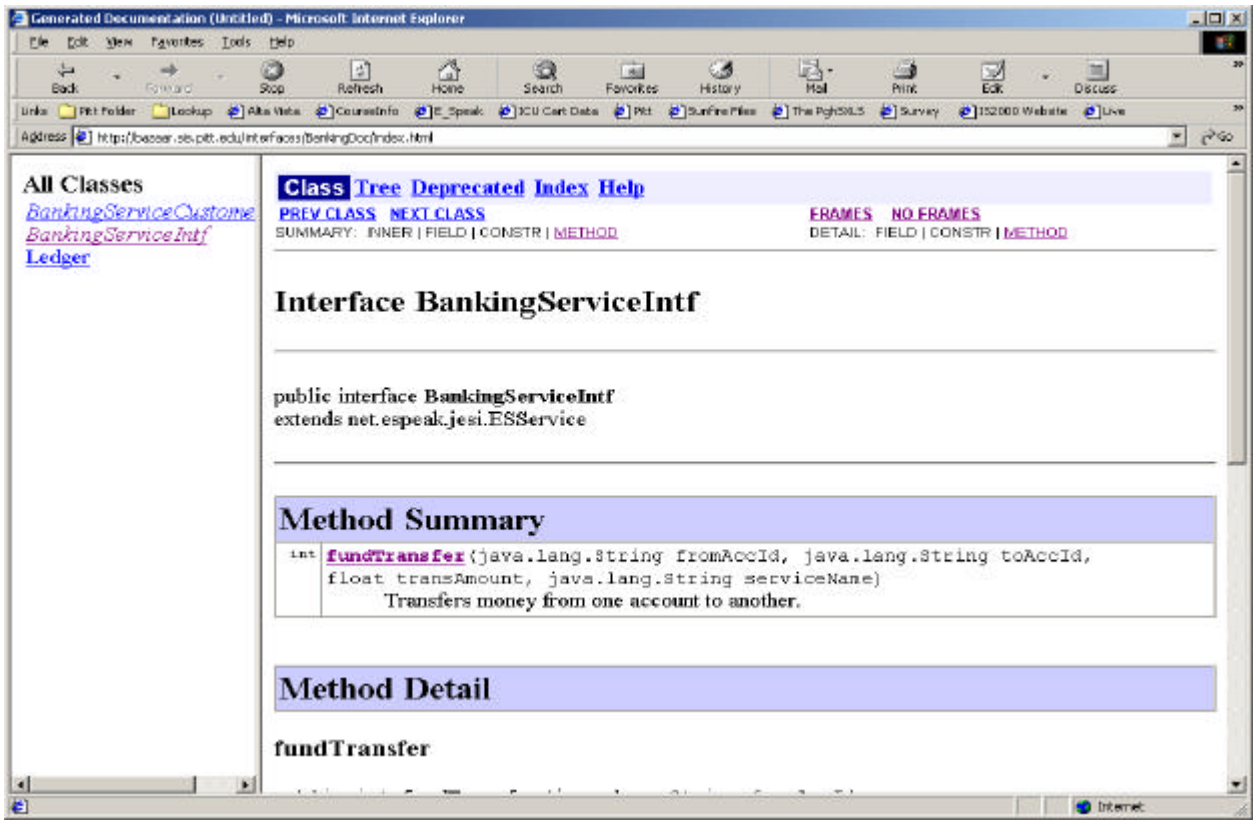


Figure 10: Javadoc output

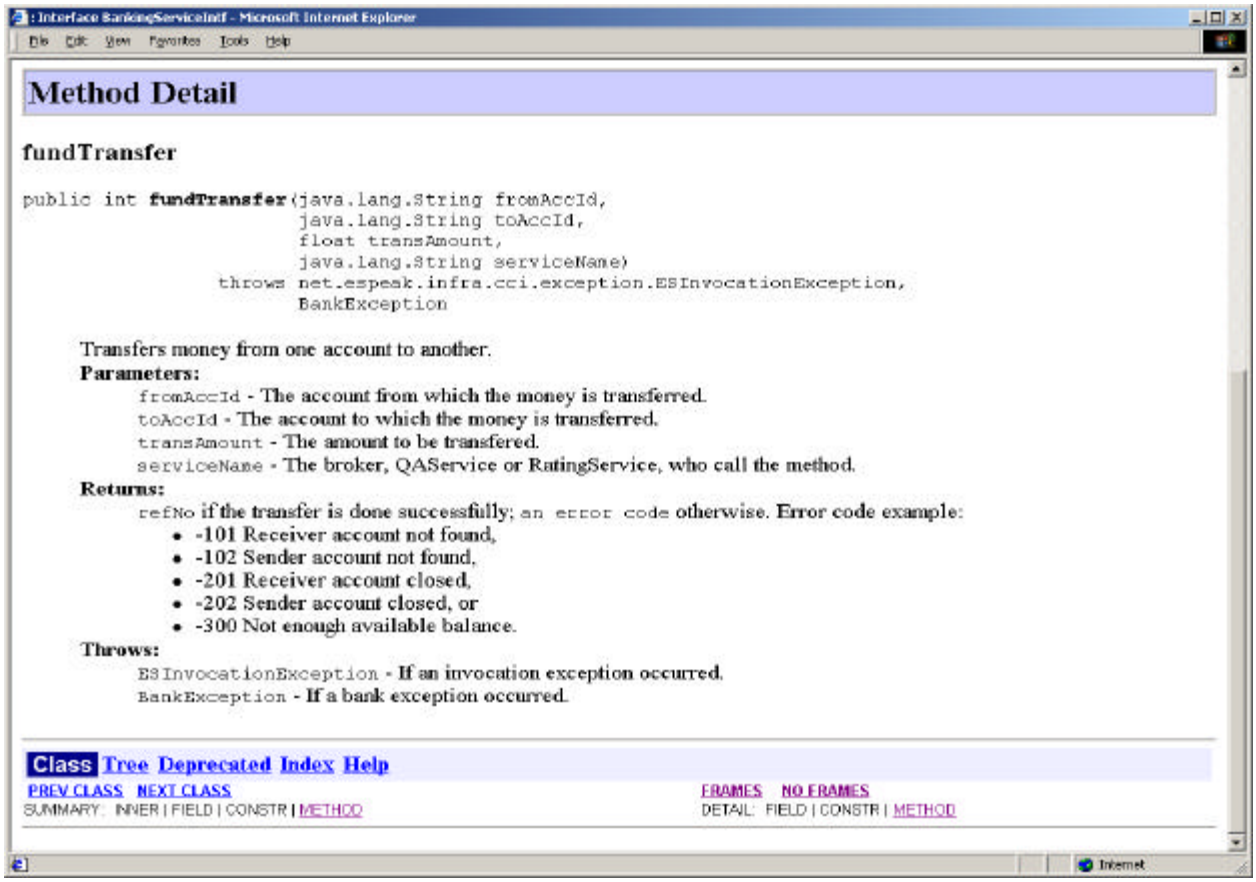


Figure 11: Javadoc output

Components

It is clear now – to the point of being painfully obvious, that not all services are created equal. There are services that support the basic services. They may be provided by the market maker or by third party vendors who are attempting to enhance the basic services – either on the back-end or the front-end. (One infrastructure service might help a basic service by doing something at the request of the basic service – back-end; another service might allow the consumer to coalesce several basic services or select one from a set of basic services – front-end.) And finally, there are marketplace services. These are services that cannot reasonably be eliminated without destroying the marketplace. The bank is an example of a marketplace service.

It also became clear that infrastructure services, such as translation and rating, and marketplace services, such as authentication, authorization, and banking need to be carefully defined and finalized before defining basic services. There is tremendous utility in classifying your services and working on some before others. While a taxonomy of services will be expanded and refined over the coming years, for now, the following taxonomy seems to offer some guidance for ordering development of components. It also specifies the order in which the services should be developed:

1. Marketplace services
2. Backend Infrastructure Services
3. Basic Services
4. Front-end Infrastructure Services
5. Consumers

Until the marketplace and backend infrastructure services are stable, it is of little use to begin development of the basic services. The one exception to this is services that may be stubbed because they provide a completely isolated service that is only used at one point. A login service might be an example of this. On the other hand, marketplace services such as a banking or escrow service, which have intimate exchanges among consumers and providers, need to be developed early on.

The prototype basic services that are produced by producer programs need to be produced in advance of the backend services they will use.

As design proceeded, it was necessary for a number of reasons to reconceptualize basic services as consumers of brokering services. Thus, it may be that the definitions proposed need to be further refined to define a basic service as one that is used by two kinds of consumers. Consider for example a note broker service that accepts a note and charges the note maker a storage and advertising fee. The same broker has another interface which allows a note buyer to access and acquire the stored note. Despite efforts to avoid this kind of monolithic service, there are times when such a consolidation just can't be avoided. We found a similar situation related to the Q&A services. All of the experts use a single routing service to manage email exchanges. Thus, basic services could be micro transaction oriented either for themselves or they could manage micro transactions for a number of different providers outside the e-speak engine environment.

Producer Components

A basic service is a piece of software that provides some service. Basic services quickly become very complex in terms of the multiple interfaces required. Not only is there an interface to the consumer but to all of the other backend and marketplace services needed to provide the service. Given this complexity, it is useful to try to reduce the load on those

who have to produce services by reducing the amount of code they need to write. Ideally they only need to write the code that specifically relates to their core competency. To this end, we looked to a generalized piece of software; a producer program, that attaches a specific remote service when it is run and that is differentiated by the vocabulary input form a file or interactively at the time it is run.

A producer component could also be a piece of software that writes another piece of software that then could be compiled as an differentiated service when it is run. This would allow the individual writing the code to modify specific methods or classes as appropriate to make the resulting code more consistent with needs.

Consumer components

The consumer program, which might be downloaded from a web site as a signed jar makes a connection to the core, has a mechanism for user input, and for selection to allow finding a service and selecting it, and then for executing that service. Consumer programs should be written to conform to the published interface, they should be available for download from the web. In the final design, all of the consumer interfaces for the marketplace were integrated into a single package. The user downloads the framework from the web which in turn downloads and activates the specific service interfaces that they wish to use. These interfaces can then be updated dynamically by the framework.

Next Steps

Having defined the classes of services and rigorously defined the interfaces, we are now at the point of opening a marketplace. Despite significant efforts the marketplace that has emerged seems rather anemic. A sense of the scope of the effort to develop even a toy marketplace has begun to take shape. As best practices for marketplace development emerge, it is likely that ideas and maybe whole infrastructures will simply be copied to create new marketplaces more quickly. It is useful to reflect on what we think needs to be done to complete even this toy marketplace. We do that by taking a closer look at how marketplace services and the two primary functional services in our marketplace might evolve.

Q&A Service

As originally envisioned, Q&A services registered as an expertise and held a live connection. A consumer connected to the engine, searched for expertise. The core responds with zero or more experts that have been found. If one or more is found, the user selects an expert and dispatches a question. The expert answers the question and returns it to consumer. There is then some accounting for the transaction.

The Q&A service, as it is currently defined, has multiple sub-interfaces that create the potential for a marketplace that might be used by any class of professionals to provide answers to questions posted by clients. Patients could connect to Q&A service supported by a group of physicians, and with appropriate anonymity, or authentication, get answers to questions that they might have. As currently conceived, the Q&A interfaces are the following:

- QAArchiveManager

- QAConfigurationManager
- QAQueryAgentConfigManager
- QAQueryManager
- QAResponseManager

Q&A extensions

Beyond these basic functions, imagine a variety of add on services. An "operations" service might allow experts to register their services providing multiple contact options: interactive services; emergency paging services, mail contact, etc.. It might be that basic service would say there are people who would be able to answer your questions, but they normally answer them between 8 and 10 at night. And that information might be generated based on collected data about when the services are open, or on advertised data about when the services are open.

There is a whole opportunity for selection and bidding here to allow a consumer to ask experts to bid on the question. So, rather than having a user choose the cheapest or the most expensive service based on static data, services could allow the consumer to provide a question on which experts would do a reverse auction.

Billing as imagined is currently based on a fixed price. For a given question, there would be a charge, and that charge would be so much money. But this billing system could be extended and enhanced in a variety of ways such that it was done based on the amount of time, or the number of words that were used.

Additional services could address the process of submitting a question to multiple experts, handling "concurrency and locking" of experts when more than one is invited to respond. Who would pay what, particularly if people are engaged in partial answers, would also have to be addressed.

The current version has an e-mail option. Should the system develop a reminder service that tells experts they have been asked a questions and promised to answer it in 24 hours and 24 hours are now have passed. This capability would build on the store and forward capability already built into the email manager.

Notes Service

The original conception was that a document would be registered. The consumer would then search for a document. If the document that met their requirements was found, that document would be delivered, and there would be some accounting for the transaction.

Notes service extensions

The Notes Service might be enhanced is by providing a monitoring service. This would be an infrastructure service that might provide any number of different kinds of functions. For example, it could take a document from the Notes Service and stream it to the users billing them only by the minute connected. It could also control the users' ability to capture documents, such that they might be able to view only, view and annotate, or view, annotate, and edit.

Documents exist in a particular format. Another service could transform them as needed. A transformation service might be used to convert documents or dialogs from one form to another. It might select a document out of the library, and convert it to meet the bandwidth need of a connection. It might provide services that would convert documents in one form to some other form. So, it might convert a power-point presentation to PDF.

A rating service might create additional value for service consumers. This service would provide collaborative assessment of notes in either an active or passive way.

Marketplace Components

While the certificate authority capability of e-speak will be used for authentication, as has been suggested in examples provided by HP, we believe that we should develop a marketplace service that serves as the centralized certificate authority for the marketplace. We developed a rudimentary accounting service to allow for payment for services. This accounting service might be extended in a variety of ways that we are beginning to explore; such as escrow services that provide e-money like capabilities. In particular, we are toying with a scheme that allows for anonymous and secure funds transfer based on tokens providing a guarantee of anonymity in the marketplace.

Conclusions

The design of an e-marketplace has a better understanding of e-speak? Probably the most important thing is that this requires a whole new mindset. In a design effort, great care has to be taken to communicate and educate those involved about what this new technology is and can do. It is a vision of millions of consumers and thousands of providers, even if that is not how it starts out on day one. It is about the creation of a marketplace and the infrastructure needed to support that marketplace. Because there will be hundreds of competing service providers, the existence of one or two particular services, which is how the marketplace will be initialized, is generally unimportant. This mindset is very difficult to adopt. The problem of mindset is compounded by the fact that the service marketplace involves a number of different categories of services that are interacting with basic services using infrastructure services and marketplace services in order to respond to a user. The decomposition and dynamic composition of services is something that requires a new way of thinking.

The project emphasized the importance of three steps in the design process. The first is that the architect's vision of this new way of doing business has to be very carefully communicated to the designers and implementers. There are all sorts of reasons why people want to have a more traditional notions of how components should interact. The e-speak approach is a little bit of a stretch for most coders and most designers, and so a lot of time has to be spent communicating this vision.

Secondly, there is an intimate relationship between the interfaces and database management systems that control persistent data in the marketplace. Further the E-speak Interface Definition Language is a very important tool for defining the methods and the objects to be exchanged by those methods. The ESIDL compiler eases the process of coordinating the interrelationships between various ESIDL classes and does a solid job of producing all of the appropriate structures required based on those definitions. In addition, extensive use of javadoc comments within the ESIDL definitions is a very powerful tool for helping coders implement that code.

Finally, the difficulty of defining vocabularies was underestimated. It seemed easy early on to define those vocabularies. In some cases, it will be easy. But there are a number of questions that need to be asked about the breadth, depth, and semantics of vocabularies.

Reference

[Berners-Lee1999] Berners-Lee, T. Weaving the Web. San Francisco: Harper Collins, 1999.

[Brown2000] Brown, J.S. and Duguid, P, The Social Life of Information. Boston: Harvard Business School Press, 2000.

[HPProgCh3] Hewlett-Packard, "J-ESI Basics," *E-speak Programmer's Guide: Documentation Release A.03.11.00 January 2001* 2001, pp. 19-30. < http://e-speak.hp.com/media/a0/programmers_guidea0.pdf>

[RFC2459] Housley, R., Ford, W., Polk, W., and Solo, D. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. RFC 2459 . 1999. IETF Network Working Group. <http://www.ietf.org/rfc/rfc2459.txt>

[HPArchCh1] Hewlett-Packard, "Architecture Overview: Standards," *E-speak Architectural Specification: Documentation Release A.03.11.00 January 2001* 2001, pp. 13. <http://e-speak.hp.com/media/a0/architecturea0.pdf>

[RFC2246] Dierks, T. and Allen, C. The TLS Protocol Version 1.0. RFC 2246 . 1999. IETF Network Working Group. <http://www.ietf.org/rfc/rfc2246.txt>

[RFC2692] Ellison, C. SPKI Requirements. RFC 2692 . 1999. IETF Network Working Group. <http://www.ietf.org/rfc/rfc2692.txt>

[RFC2693] Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., and Ylonen, T. SPKI Certificate Theory. RFC 2693 . 1999. IETF Network Working Group. <http://www.ietf.org/rfc/rfc2693.txt>

[HPArchCh5] Hewlett-Packard, "Communication: Session Layer Security (SLS) Protocol," *E-speak Architectural Specification: Documentation Release A.03.11.00 January 2001* 2001, pp. 121-139. <http://e-speak.hp.com/media/a0/architecturea0.pdf>

[HPProgCh5] Hewlett-Packard, "Extended Services: Communities," *E-speak Programmer's Guide: Documentation Release A.03.11.00 January 2001* 2001, pp. 97-103. < http://e-speak.hp.com/media/a0/programmers_guidea0.pdf>

[HPProgSec4] Hewlett-Packard, "Gateway Usage Model," *E-speak Programmer's Guide: Documentation Release A.03.11.00 January 2001* 2001, pp. 195-286. < http://e-speak.hp.com/media/a0/programmers_guidea0.pdf>

[RFC1777] Yeong, W. and Kille, S. Lightweight Directory Access Protocol. RFC 1777 . 1995. http://www.ietf.org/rfc/rfc1777.txt

[Karp2001] Karp, A. H. E-speak E-xplained. Draft paper, Hewlett-Packard Laboratories, 2001. http://www.hpl.hp.com/techreports/2000/HPL-2000-101.pdf