Lessons from E-speak

Alan H. Karp
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-2004-150
September 7, 2004*

E-mail: alan.karp@hp.com

distributed systems, scalable systems

E-speak was the technology base for HP's E-services initiative, which was announced in 1999. It was designed to be a scalable distributed system, and it met all of its design goals. Although it's no longer around as a supported product, the lessons we learned, both positive and painful, may be helpful to others.

* Internal Accession Date Only
Approved for External Publication

Lessons from E-speak

Alan H. Karp
Hewlett-Packard Laboratories
alan.karp@hp.com

August 31, 2004

Abstract

E-speak was the technology base for HP's E-services initiative, which was announced in 1999. It was designed to be a scalable distributed system, and it met all of its design goals. Although it's no longer around as a supported product, the lessons we learned, both positive and painful, may be helpful to others.

1 Introduction

The basic idea behind e-speak was to improve interoperability in distributed systems that crossed administrative domains by turning everything into a service. Today this approach is called Web Services or the Service Oriented Architecture. For that reason, e-speak has been called "the industry's first web services platform." [5] and "web services before there were web services". The most succinct definition is "E-speak is roughly what you'd get if you crossed CORBA with LDAP and simplified the resulting mess a bit." [6]. Of course, e-speak preceded SOAP, WSDL, and even the widespread acceptance of XML, but all the essential elements of web services, and more, were there.

At one point, over 60 companies were evaluating or using e-speak [5]. Nevertheless, work on e-speak stopped when HP dropped its entire suite of middleware products in 2002. At that time, there were four major users of e-speak, several of whom continued to use their e-speak platforms for a year or more.

We did some things right, and we did some things wrong. After briefly describing what problems e-speak was intended to solve, I'll enumerate some of the lessons we learned. It has been said that a fool learns from his own mistakes, a wise man learns from the mistakes of others. At best we fell into the former category. The goal of this paper is to help you land in the latter one.

2 Architecture

The name e-speak was applied to two, somewhat different architectures. The first was built to be a single system image for the Internet [1]; the second was a B2B platform [2]. Because of their different goals, they used somewhat different mechanisms for authorizing access. However, they were both based on the same set of assumptions.

Large scale: E-speak was designed for a million machines. Hence, it did not have anything centralized and couldn't rely on ever being in a consistent state.

Dynamic: Something is always changing. It is important that we not require developers to deal with such a dynamic environment; it's hard enough to write applications in a static world. The e-speak platform hid many of the changes from the applications.

Heterogeneous: The world is heterogeneous and getting more so, and not just in hardware platform or operating system, but in device capability as well. E-speak's distribution model allowed devices to implement as much or as little of the protocol and environment as appropriate.

Hostile: As we know, there are bad people out there. Some are bad for financial gain; others merely for the challenge of breaking things. Security is critical, but it can't make the system too rigid. E-speak's security mechanisms allowed distrustful parties who implemented completely different security policies to interact while controlling their risk.

Many fiefdoms: There are a variety of organizations that want to use such systems, but getting them to change how they do things is difficult. Getting them all to agree on a single way is nearly impossible, and once you've got agreement, making changes is even worse. E-speak allowed interoperation even across organizations with incompatible policies.

3 Dos and Don'ts

We did some things right; we did some things wrong. This section summarizes some of the lessons from what we did right.

Don't put policy into the architecture. Too many systems build specific policies into the architecture. For example, mandating a specific digital certificate format for carrying information requires everyone to use this format, even if they already use a different format internally. Basing access control on identity requires a unified identity scheme across the entire environment. Most military systems build a particular version of multi-level security into the architecture, which makes it hard for them to interact with those with a different definition or number of security levels. E-speak's flexible mechanisms allowed us to implement a wide variety of policies.

People who never interact should not have to agree. Too many of our distributed systems require global agreement. In the best of these systems, it is only the version of the protocol that everyone must use. Even this level of agreement is too much, since it requires synchronous upgrades, which is clearly untenable in a large scale system. More commonly, numerous policies are hardwired. The most common problems concern naming and ontologies. Too many systems require the entire system use a single name system and a single, global ontology. Since global agreement is needed, updates take too long. People either abandon the system or implement updates in their own communities, fragmenting the system into incompatible parts.

Everything in e-speak was pairwise. We also decided that it was all right if two parties could not communicate. This decision meant that many problems of upgrading components could be left to individual policies instead of being specified in the architecture.

Think about security early and often. Everyone says that you've got to include security from the beginning, but few do, at least in a meaningful way. The first question to be answered is what you mean by security. You must identify what assets you're protecting and the threats you're protecting them from. Only then can you define your security mechanism. Be careful, though, it is easy to to architect policy instead of just mechanisms.

In e-speak everything was a service, so it made sense to control access to the methods provided by the service. Because of the large scale and different administrative domains, basing access control on identity was clearly untenable. Hence, we settled on a capability-like approach [4]. Limiting ourselves to mechanisms proved its value when we found that we could enforce such disparate policies as Unix-style security, multiple security level, and compartments without any change to the mechanisms.

Think about naming early and often. Designers of distributed systems invariably assume that the name space can be partitioned. However, in a dynamic environment with hostile participants, this assumption is unwarranted. The problem with names is that they reside in many places – programs, data files, even people's heads. The goal is to build a name system in which applications don't break when someone renames something.

It has been shown [10] that no single naming system can be human meaningful, securely collision free, and globally context free. A human meaningful name has meaning in some particular context. URLs have this property to some

extent, but they lack other desirable properties. Securely collision free means that names can't be spoofed. Clearly, return addresses on email do not have this property. The usual approach is to use a private key in a private/public key system to construct the name. Globally context free means that the name doesn't depend on the location of the namer or the named. Sufficiently large random numbers have this property.

It is also important that names in a large scale distributed system have both spatial and temporal integrity. Spatial integrity means that the name used for something doesn't change when the namer or the named changes locations, as it does today when moving from inside to outside a corporate firewall. Temporal integrity means that the name shouldn't change because the passage of time caused some external factor to change, as it does today when companies merge or when a private key used to construct a name must be changed.

E-speak was based on path based names. Each client, sort of like a process, had a private name space. Pairwise translation was used to move the request between the namer and the named. The advantage was that any pair could change the name used without affecting anyone else along the path. The disadvantage was that names had no meaning out of band. Also, if an intermediary was unavailable, the service was unreachable. However, paths could be shortened by introduction.

Avoid special cases. Special cases are an architectural nightmare, a development nightmare, a maintenance nightmare. We went to considerable effort to avoid special cases in e-speak, and it paid benefits. The service engine, analogous to an operating system kernel, was relatively small and only had about a dozen distinct resource types to deal with.

Sticking to this policy also had an unexpected benefit. E-speak provided for service discovery with constraint based search using vocabularies, an ontology representation based on attribute value pairs. Since everything in e-speak was a service, so were vocabularies. That meant a vocabulary could be advertised would any other service. A search might turn up services and new vocabularies that extended the descriptions used to find them. The result was that e-speak provided a dynamically extensible ontology framework.

Plan for delegation/Plan for revocation. Most of the systems we use today depend on identification or authentication to determine access. There are many problems with such an approach, such as confused deputy attacks [3], but the biggest problem is the inability to designate a delegate. The unfortunate result is that people tend to share their identities.

Delegation also simplifies management when access crosses administrative domains. In today's world, I get a list of employees (or roles, it doesn't much matter) in your company who are authorized to use my service. When someone in your company changes jobs or a role changes responsibility, you tell me, and I update my list. The problem is that we each have thousands of partners and spend all our time updating our respective lists. With easy delegation, I give your company a delegatable right to use my service. How you manage it is up to you. If the right is easily revoked, you can delegate it to the appropriate subset of your employees.

Support Voluntary Oblivious Compliance. There will be people who want to break the rules. Not just strangers, but people in your organization, too. Unfortunately, there's nothing you can do to prevent misuse of a legitimate authority. Don't even try. In other words, DPWYCP (Don't Prohibit What You Can't Prevent). However, those who want to follow the rules need some help. The rules are complex, and they frequently change. If your system requires that everyone know these rules in order for them to be enforced, they won't be.

E-speak supported what we now call "Voluntary Oblivious Compliance" (VOC). Let's say you ask me for access to a service. Should I give it to you? I could certainly expend some effort to find out, but there's a race condition. Your access might be revoked just after I ask. E-speak took a different approach. I'd just send you a reference to the service, and the platform would prevent you from using it if you shouldn't have gotten it. The specific mechanism, nega-

tive permissions from split capabilities [4], isn't as important as the ability to support the concept.

Design for Consistency Under Merge. People will build private copies of your distributed system. At some later date they will want to merge these copies. If you're not careful, one side or the other will have to go through painful modifications in order to eliminate conflicts between the systems.

You can't rely on a partitionable name space to help you, either. A major supercomputer center spent several weeks trying to merge two large clusters until they discovered that two ethernet cards had the same MAC address. In fact, AOL at one time assigned the same MAC address to every dial-in user. Several attempts at merging private UDDI repositories failed due to conflicting GUIDs, even though the GUID algorithm supposedly generates unique strings. E-speak was consistent under merge because of the name system and the distribution model.

Don't authenticate when you want to authorize. Too many times our systems ask "Who are you?" when they want to know if your request should be honored. Most times knowing who you are doesn't carry enough information to make an informed decision. If my access is granted because I work for a business partner, you don't want to know who I am. You only want know that business partner has authorized me to make the request.

Relying on identity also makes delegation difficult. The unfortunate result is that people share their passwords, which loses a valuable use for identity, audit trails. A final problem is that it isn't a person making the request; it's software. There is no assurance that the software is acting in the user's best interest. A virus certainly doesn't. If access is controlled by identity, then the software necessarily runs with the user's privileges. E-speak properly separated identification, authentication, authorization, and access control.

4 Why E-speak Died

HP abandoned E-speak in 2001 for a number of reasons. Had we been able to build a larger user base in the time we had, e-speak might still be in widespread use. Unfortunately, we made some mistakes that slowed e-speak's spread.

We didn't devote enough resources to the Open Source effort. We knew that keeping e-speak proprietary to HP would doom it, so we released it under the GNU Public Licenses. Unfortunately, we didn't realize until too late that this was not enough. We needed to devote substantial resources to building a community of developers. Without that push on our part, we never developed a critical mass that could convince potential customers that e-speak wasn't just an HP product.

We made people change their mental models too much. When we explained all of what e-speak could do, customers told us it was too much to grasp. When we explained only the part relevant to their particular environment, they told us it was just CORBA or DCE or ... (fill in your favorite environment).

Once we got past that barrier and started showing people how to use it, we ran into another problem; developers had to change their way of thinking. It isn't difficult for people to think of a print service as a service, but we also wanted them to think of the file being printed as a service. Doing so had a lot of advantages, such as not accidentally printing a confidential document on the printer in the lobby. Once users adopted this way of thinking, they found their problems were easier to solve. We just didn't have a good way to get them across that barrier.

We focused on the technology, not the business problem. We're technologists, and we developed some pretty neat technology. Unfortunately, we turned off some customers who wanted us to help them solve their problems, not demonstrate our cool stuff. The customers we ended up with were the ones who couldn't see any other way to build their business, so they listened to our techno-babble. We might have built a big enough customer base to avoid being shut down had we focused more on the cus-

tomer's problems.

We didn't give adequate attention to the development environment. It's one thing to have a good solution to people's problems, but it's got to be something they want to use. If the only debugging tool is "System.println", you're not going to attract many developers. It's a credit to e-speak's potential that we had as many as we did. We would have been better off devoting a sizable part of our budget to building a good development environment.

We worked in the wrong industry. E-speak was software. Even worse, it was middleware, which is software that's supposed to be invisible. HP at that time was largely a hardware company. In fact, we started in the business unit that sold HP-UX servers. Everything about software is different from hardware. It's made differently; it's sold differently; it's procurement cycle is different; it's support structure is different. All these differences made it hard both for HP management to know how to deal with it and for customers to deal with HP in this new way. HP could certainly make a wonderful refrigerator, but it would be hard to break into the market because refrigerators are so far from customer's expectations of HP. In 2000, middleware was farther from HP's core business than are refrigerators today.

E-speak is no longer a supported product, but aspects of it still live on. NTT still has a web site describing the services platform it built with e-speak [9]. More significantly, various aspects of e-speak have influenced the development of the E language for secure distributed computing [8]. The e-speak vocabulary system [7] has taken on a life of its own because of the way it solves the problems people try to address with global ontologies. Finally, some groups trying to build scalable systems with the web services standards and finding them inadequate are taking a look at e-speak. Maybe our biggest mistake was being too early.

References

[1] Hewlett-Packard Company. *E-speak Architecture Specification*, September 1999. http://www.hpl.hp.com/-personal/Alan_Karp/espeak/version2.2/-Architecture_2.2.pdf.

[2] Hewlett-Packard Company. *E-speak Architectural Specification, Release A.0*, January 2001. http://www.hpl.hp.com/-personal/Alan_Karp/espeak/version3.14/-Architecture_3.14.pdf.

[3] Norm Hardy. The confused deputy. *Operating Systems Reviews*, 22(4), 1988. http://www.cap-lore.com/CapTheory/-ConfusedDeputy.html.

[4] Alan H. Karp, Guillermo Rosas, Arindam Banerji, and Rajiv Gupta. Using split capabilities for access control. *IEEE Software*, 20(1):42–49, January 2003. http://www.hpl.hp.com/techreports/-2001/HPL-2001-164R1.html.

[5] Jim Kerstetter and Peter Burrows. Hp's e-speak: Good products, botched marketing. *Businessweek Online*, July 3 2000. http://www.businessweek.com/-2000/00_27/b3688173.htm.

[6] Eric Kidd. Customdns. http://customdns.-sourceforge.net/internals.php.

[7] Wooyoung Kim and Alan H. Karp. Customizable description and dynamic discover for web services. *ACM Conference on Electronic Commerce (ACM EC'04)*, 2004. http://www.hpl.hp.com/-techreports/2004/HPL-2004-45.html.

[8] Mark Miller. Open source distributed capabilities. http://erights.org.

[9] NTT. Teatray. http://www.nttcom.co.jp/-teatray/english/base/.

[10] Bryce Wilcox-O'Hearn. Names: Decentralized, secure, human-meaningful: Choose two. http://zooko.com/distnames.html, September 2003.