Lessons from E-speak

Alan H. Karp
Hewlett-Packard Laboratories
alan.karp@hp.com, http://hpl.hp.com/personal/Alan_Karp

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

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.

Nearly 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 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 levels, 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 that no single naming system can be human meaningful, securely collision free, and globally context free [11]. 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 as 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, negative 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 Questionable Decisions

Some decisions we made weren't obviously right or wrong. We had good reasons for doing what we did, but perhaps those reasons didn't justify the decisions.

We listened to the experts even when we knew their conclusion was based on a misconception. Shortly before the release of the Beta version, we commissioned a well-known security firm to do a review of the architecture. The rules they insisted on were that we give them any documentation we had, and they would prepare a report. No interaction with them was allowed during the review process, and their conclusion was final. No response from us would affect their report.

They stated that the security model was flawed because it was based on "name hiding". We knew this statement was due to a misconception on their part, but there was no procedure for correcting it. Nevertheless, this review was an important factor in the decision to change the basic access control mechanism.

It was too late to change the architecture for the Beta release, but the decision was made to change the architecture for Release 1.0. I argued for keeping split capabilities in spite of the flawed review. The main opposing argument was that we could hardly go to market with a version that a well-known security company said wasn't secure. I felt that we could if we showed the flaw in their understanding of the system. Needless to say, I lost that argument. Split capabilities were replaced with Simple Public Key Infrastructure (SPKI) attribute certificates.

The change to SPKI certificates had a number of effects. First of all, the architecture was better suited to a B2B environment. It made certain end-to-end guarantees easier to enforce and, equally importantly, easier to explain. SPKI, because of its PKI heritage, was more familiar to our customer base, even though our use of the certificates was quite different from their conventional use. Increasing marketability in B2B was an attractive proposition considering the projections of a market size of $1,600B by 2004 [10].

The downside was that we no longer had a general purpose, low latency base. When the B2B boom failed to materialize, we couldn't easily build other kinds of systems. The very features that made the e-speak product attractive to our B2B customers, particularly the heavy reliance on expensive cryptographic protocols, made it hard to go after other businesses. For example, e-speak could have made a good platform for building a distributed collaboration system, but the SPKI operations made the latencies too large for interactive use.

We sacrificed an important architectural principle. We wanted the core (analogous to an OS kernel) to handle one request in its entirety before starting work the next one. Doing so would avoid any possible race conditions since message handling would be logically atomic. Of course, that meant that our throughput was controlled by the slowest commands. Discovering locally available resources used an SQL-like repository lookup, which was very slow when the repository was large.

We tried to find a way to make the repository a client of the core, but we didn't like the idea that this client would be able to use all system resources. Subverting this client would give an attacker full control over the system. We ultimately decided to add additional threads to the core solely to handle lookup requests. The result was a considerable increase in the core's complexity. We might have done better by taking our chances with an external lookup service.

We invented something new before thinking hard about possible optimizations. The original prototype for e-speak used capability lists (c-lists) for access control. Each client had a list of resources it could name. No attack was possible against a resource not in this list. Each entry in the c-list included the specific permission, such as read or write, it granted. Each separate permission required a separate c-list entry. The problem was that each entry had a considerable amount of metadata, typically 100-1,000 bytes. It was this extra metadata that concerned us.

Since most of the metadata was the same on all the entries for a given resource, we thought we could come up with an optimization. However, we decided to implement split capabilities [4] instead. Split capabilities solved the metadata problem and made configuring a variety of security policies quite straightforward. However, by separating designation from authorization we opened up the system to some attacks. More importantly, adding something completely new made it even harder for people to understand the system.

We didn't control the urge to add new features. Part of the reason e-speak was hard to understand was that we had too many features. They were all useful in the sense that every feature was used in least one deployment, but it took a trained eye to sort through the features to find the relevant ones for a particular service. Paraphrasing Einstein, "E-speak needed to be as simple as possible but no simpler". It wasn't. Even if the architecture included all the features, we would have done better had we turned on only a basic set for the earlier releases. Once developers got used to thinking the e-speak way, we could have made the additional features available in steps. The hard part, of course, is deciding which features are really needed.

5 Don'ts

Flaws in an architecture are usually subtle and depend on specific details. After all, if they were obvious, they wouldn't have made it into the release. Understanding these flaws, then, necessarily requires a reasonably deep understanding of the architecture. Space does not permit a sufficiently detailed description of e-speak here. Still, it's important to document what was wrong. See the relevant architecture documents [1, 2] if you want to understand this section.

Don't let the implementation drive the architecture. Our first implementation was dreadfully slow. Some investigation showed that each name resolution required an average of 500 hash table lookups. Some people argued that we should fix the problem by abandoning our naming scheme. Here we held firm. After some tuning, we reduced the name resolution to an average of two hash table lookups.

We weren't so fortunate in two other cases. The initial architecture called for the explicit rights being requested, *e.g.*, read or write, to be associated with the corresponding resource. Somewhere along the line, this feature got *optimized* so that there was only one place to specify the rights for all the resources named in a request. This change made the system susceptible to a confused deputy attack [3].

The second failing was the handling of incoming messages. The original architecture specification was unclear on whether all messages went to one inbox or there was a separate inbox for each message. The latter is more secure since there's less chance of mixing the authorities from different senders. However, the former had an even worse problem. Since there was no way for the system to know when the client no longer needed an entry, they just kept piling up until the JVM ran out of memory. This choice also meant that the sender could not choose the name the receiver would use for the resource, a definite change in the architecture.

Don't make the easy way the insecure way. We wanted to be able to delegate revocable authorities, so we introduced the idea of key rings. Each user also had a mandatory key ring, which was needed to support negative permissions [4]. Unfortunately, we let users add keys to their mandatory key ring. The result was that users tended to put all their keys on this key ring, making them susceptible to confused deputy attacks.

Even had we not given users access to their mandatory key rings, having such a thing in the system encouraged grouping large collections of authorities. Later in the process we realized that we could clone keys to make revocable authorities, obviating the original reason for key rings.

Don't ignore end-to-end issues. The Beta release was path based; all requests passed through a series of intermediaries unless the end points were introduced explicitly. While the message payloads could be encrypted, the headers were necessarily visible to the relaying machines. The original architecture did not attempt to protect the headers from prying or tampering. By the time the problem was identified, signing the headers to prevent tampering was a problem because the naming system required a change on each hop. A proposal to implement tunneling (e-speak in e-speak) was never implemented because the decision was made to change the architecture.

Don't forget about performance. The Beta release was designed to be sort of like an operating system, so we worried quite a bit about performance. Any interaction between machines that exceeded a few tens of ms was a target for optimization. The e-speak product was built for a different environment, B2B systems. A B2B interaction often takes minutes or even days to complete. That being the case, we didn't worry too much about performance. The unfortunate consequence was that setting up a connection took several seconds of CPU time, and each interaction involved a reasonably expensive cryptographic calculation. Although the latency of these steps wasn't an issue, the CPU load was. As a consequence, our customers needed more machines than they would have liked to support their business partners. Planned optimizations never got into the system.

6 Why E-speak Died

HP abandoned E-speak in 2002 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 use. Unfortunately, we made some mistakes that slowed e-speak's spread.

We didn't do a good job explaining the system. When we described 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). We never did find the middle ground.

Although e-speak has fewer than 20 components that need to be understood, the system was quite flexible because of the way they could be combined. For example, events are based on e-speak vocabularies and split capabilities. That meant that events could have a variety of personalities depending on how these pieces were used. We kept trying to show users how flexible the system was, which only served to confuse them. We would have been better had we settled on a small number of use patterns to present.

We made people change their mental models too much. We thought we were starting a revolution, but businesses want evolution. Kannan Govindarajan, one of the e-speak architects, has said that we needed to "evolve users, not revolve them". Indeed, some of them told us that we made their heads spin.

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 customer'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 bigger part of our budget to building a good development environment and a smaller part to adding more features.

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 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.

7 Conclusions

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.

8 Acknowledgements

Guillermo Rozas, Arindam Banerji, Rajiv Gupta, and I are mainly responsible for any failings in the original architecture. Nigel Edwards and Michael Wray made important contributions to the product version.

9 Availability

E-speak was released under GNU Public Licenses and is available for download from the author's web site.

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.source-forge.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] Rob Rosenthal. The Internet commerce market model: B2B versus B2C around the world. Technical Report 22745, IDC, July 2000.

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