EtchProposal

Abstract

Etch is a cross-platform, language- and transport-independent framework for building and consuming network services.

Proposal

Etch is a cross-platform, language- and transport-independent framework for building and consuming network services. The Etch toolset includes a network service description language, a compiler, and binding libraries for a variety of programming languages. Etch is also transport-independent, allowing for a variety of different transports to used based on need and circumstance. The goal of Etch is to make it simple to define small, focused services that can be easily accessed, combined, and deployed in a similar manner. Ultimately with Etch, service development and consumption becomes no more difficult than library development and consumption.

Background

Etch was started because we wanted to have a way to write a concise, formal description of the message exchange between a client and a server, with that message exchange supporting a hefty set of requirements. The messaging technology should support one-way and two-way, real-time communication. It should have high performance and scalability. I should support clients and servers written in different languages. It should also support clients/servers running in a wide range of contexts (such as thin web client, embedded device, PC application, or server). It must support anyone adding new language bindings and new transports. It should also be fast and small, while still being flexible enough to satisfy requirements. Finally, it must be easy to use for developers both implementing and/or consuming the service.

Rationale

Existing systems were either too slow, hard to use, bloated and/or proprietary. In any case, none fit our matrix of requirements perfectly.

Developers of applications that must leverage the capabilities of network-hosted services have a daunting challenge. They must cobble together a heterogeneous collection of services that expose different APIs with different communications technologies just to integrate with the services, essentially spending a great deal of energy and effort on just the basics of inter-service communication rather than core business logic.

So the desired state then is when developing applications that leverage the capabilities of dispersed and heterogeneous network services, APIs must be simple, cohesive, and coherent across network services. APIs should be easy to consume by developers regardless of the implementation technology of the service used or the domain a service is being built within- from client-side web applications to complex real-time server systems. Put simply, developers ideally should feel that they are developing to a platform.

API development is a much better understood and simpler subject if you are building those APIs to be run *locally* on a single machine or service. Microsoft and Linux centric API developers have the luxury of the massive assumption that a standard OS is available with a certain set of features, and the API libraries do not have to take into account the complexities of APIs that cross machine or OS boundaries.

Developers of network-centered services, rather than OS-centered services, do not have this luxury; we have a significant set of issues facing us today because of the fundamental fact that "the network" is not a single machine, or a homogeneous set of machines, but a heterogeneous and widely distributed set of services.

The conventional method for developers of network services today is to use either a technology specific to the language of preference, RMI for Java, .NET Remoting for .NET for C#, etc., or if trying to be "language neutral" picking a CORBA ORB or a Web Service technology like SOAP or REST. These choices are fine until the requirements of the application cannot accept the limitations of the remoting technology. If the application needs to work on non-Microsoft platforms, .NET Remoting is out (unless, of course, you can use the Mono implementation of .NET, but that brings with it other challenges). If the need is to support access from languages other than Java, then RMI is out. If the need includes support for real-time, asynchronous communication, or symmetric two-way communications, the Web services technologies must also be rejected.

For other classes of applications, there are simply no “standard” choices left. The developer is forced to drop down to a network protocol level and invent a new messaging system for their needs. Building a protocol by hand is hard; building a messaging system is also hard. This dramatically increases the barrier to entry for new, useful applications that leverage network-services.

An orthogonal problem exists when supporting more than one transport technology is required of the application, e.g. HTTP/SOAP and HTTP/REST or HTTP/SOAP and a proprietary service protocol. This is also burdensome to the developer because now two or more distinct technologies must be used to expose the same interface. This typically means the development and maintenance of parallel implementations of the service using the technologies native to the transport mechanism. Often the result here is that one interface is the complete interface, while others suffer from various levels of partial or out-of-sync implementation.

What if this was the reality instead: every interface to a network service could be had via a single, common API technology that 'just works' in every major language (C#, Java, Python, Ruby, C or even Javascript in a browser). What if this technology could produce the native stub code needed to do the networking and message passing (much like Web Services). Then the developer could concentrate on the business logic of the application or service rather than the idiosyncrasies of the network plumbing.

As a language and transport independent network API generator, Etch can provide programmers with a consistent API model to program against while giving them the ability to redeploy into a variety of languages or transports at runtime (per developer/customer choice). So, one may use the same API implementation to send messages using an XML coding on a stream protocol in Java, or binary coding wrapped in reliable UDP in C#, or a shared memory queue on a router backplane in C, or even Python over SOAP. One could, in fact, support all at the same time, and any others that you care to implement or find, as long as you support the required semantics of the API.

It all comes down to this: developers should not have to care about the implementation language or platform of the service nor what the transport is to get there, as long as basic semantics are honored, and these should be no more or less than the semantics of your programming language of choice. Further, a user requirement about specific protocols should not require rewriting of application logic to make it fit into some arbitrary framework scheme or container.

Current Status

Meritocracy

Etch was conceived by Scott Comer and Louis Marascio. As Scott finished the development of the core compiler and first transport implementation, others have made various contributions to the project: James Dixson and Shawn Dempsey have worked on the build environment; Manoj Ganesan has worked on a Ruby binding; James Dixson on the Python binding; and James deCocq on the C binding; Manoj Ganesan and Gaurav Sandhir did primary work on C# and maintenance work all around. J.D. Liau has been instrumental in ideas and maintenance. Hung Nguyen has created the Windows installer using NSIS and Seth Call is working on a JavaScript binding with JSON transport for thin clients.

Community

Etch solves problems lots of projects have. Any project that has a need to define multiple services in a consistent way, or expose services on the network to a variety of languages or platforms can benefit from Etch as technology.

Core Developers

The core developers are all listed in the initial committers list later in this proposal.

Alignment

The compiler code is in Java, but the technology is language- and protocol-agnostic and suitable for many different projects, including non-Java. The compiler makes use of Apache Ant for orchestrating the build, and Apache Velocity for code generation.

Known Risks

Orphaned Products

We are all quite committed to Etch and the development of an Etch community. Etch is a core component of shipping Cisco products and will only grow over time.

Our employer is also committed to the success of the technology, allowing us to continue to invest our time in support of Etch development as well as committing to Etch technology as a key component in multiple products.

Etch being orphaned is unlikely.

Inexperience with Open Source

The group of initial committers has had various levels of interaction with open-source communities. Most of us came into Cisco through the acquisition of Metreos in 2006. While at Metreos, Louis Marascio and several of us were active contributor's to the OpenH323 project. We worked through several bugs, submitted patches and even sponsored development. We have also made contributions to other projects (some accepted, some not) on a much smaller scale over the years, QDox, Maruku, Capistrano, OpenGatekeeper, and Mono.

Homogeneous Developers

Etch has been completely developed by Cisco employees, therefore all of the initial committers to the project are affiliated with Cisco.

Etch has just recently been made publicly available. First in binary form in May 2008 as part of a Cisco product and in open source form in July 2008.

Reliance on Salaried Developers

It is expected that Etch development will be done both on salaried time and volunteer time. Cisco is highly interested in the success and development of an Etch community. At this time, Etch is a core component of shipping Cisco products and is likely to grow over time. Cisco in interested in investing time to support Etch development and use it as a key component in multiple products. It is also expected that non-Cisco developers will become interested in Etch.

Relationships with Other Apache Products

Etch currently depends upon these other Apache projects: Velocity, Maven and Ant.

We expect that as Etch becomes available, it will be seen as a very compelling technology and others will begin to depend upon it.

A Excessive Fascination with the Apache Brand

We believe Etch offers much to the Apache brand. We could easily, with the backing of Cisco, take a more independent route and support Etch directly without the Apache foundation. But after much consideration, we truly believe that would be the wrong approach for this technology.

As a technology, we believe Etch is very much a kindred spirit of the other software infrastructure technologies currently part of the Apache community: Ant, Velocity, Derby, and others. The technological niche of Etch--platform and language agnostic service definition and binding-is a technology that can be appreciated across a broad range software domains.

It is our view that Apache is simply the most appropriate community for the kind of technology Etch represents.

Documentation

Public documents are available. All documentation can be found here http://developer.cisco.com/web/cuae/etch .

Initial Source

Etch has been in development at Cisco since Jan-2007. The system was designed from the beginning to be open-sourced. We consider Etch to be at release 1.0 and ready for production use.

We continue to develop on Etch aggressively and a continually adding tests and documentation to accompany the code, in particular around Etch's unique pluggable architecture.

The compiler and language bindings for Java and C# are working and functional. Etch will be included in shipping Cisco products in Sept-2008 as a core technology component.

The language bindings for JavaScript, Python and C are in development. The Etch development home page is currently hosted a Cisco's developer portal: http://developer.cisco.com/web/cuae/etch . Full source and binary distributions are available there including access to our public subversion repository.

Source and Intellectual Property Submission Plan

Apache would receive all source and documentation under the Apache Corporate Contributor agreement. Cisco is the only license holder.

External Dependencies

Java, JavaCC and Velocity are core dependencies of the compiler. The Java language binding depends only on Java.

Ant and Maven are used by the build system.

For the other language bindings we have the following compile/link dependencies:

C# - Microsoft .NET v2.0 (Mono compatibility coming soon)

Cryptography

Etch uses the native capabilities of Java and C# to support TLS as an option for the default Etch binary transport protocol.

Required Resources

Mailing Lists

- etch-private
- etch-dev
- etch-commits
- etch-user

Subversion Directory

https://svn.apache.org/repos/asf/incubator/etch

Issue Tracking

JIRA : Etch (ETCH)

Other Resources

None

Initial Committers

Gaurav Sandhir gsandhir at cisco dot com

J.D. Liau jliau at cisco dot com

James Dixson jadixson at cisco dot com

James deCocq jadecocq at cisco dot com

Rene Barazza rebarraz at cisco dot com

Scott Comer sccomer at cisco dot com

Seth Call secall at cisco dot com

Youngjin Park youngjpa at cisco dot com

Affiliations

All the initial committers are Cisco employees.

Sponsors

Champion

Niclas Hedhman (has offered to be Champion)

Nominated Mentors

Niclas Hedhman

Doug Cutting

Yonik Seeley

Sponsoring Entity

Incubator