Engineering Guidelines

GDD-06-38

GENI: Global Environment for Network Innovations

December 1, 2006

Status: Draft (Version 0.1)

Note to the reader: this document is a work in progress and continues to evolve rapidly. Certain aspects of the GENI architecture are not yet addressed at all, and, for those aspects that are addressed here, a number of unresolved issues are identified in the text. Further, due to the active development and editing process, some portions of the document may be logically inconsistent with others.

This document is prepared by the Facility Architecture Working Group.

Editors:

Ted Faber, *USC/ISI*

Contributing working group members:

John Wroclawski, *USC/ISI*

Larry Peterson, *Princeton University*

Tom Anderson, University of *Washington*

The work is supported in part by NSF grants CNS-0540815 and CNS-0631422.

Revision History:

Version	Changes log	Date
v0.1	Original version posted	12/01/06

Table of Contents

1. Introduction

This document motivates and explains several engineering decisions made by the designers of the GENI Management Core (GMC). An engineering decision is a choice of a particular implementation technology or framework, not a decision regarding the architecture of the GMC itself. For example the decision to create entities in the GMC called components and users that communicate with each other via remote procedure calls(RPC) is a design decision. The decision to use the World Wide Web Consortium's Web Services design and tools to implement those RPCs is an engineering decision.

The next section describes the guiding principles used to make the engineering decisions and the remainder of the document explains how those principles were applied to specific decisions in the GMC design.

2. Engineering Principles

In addition to offering guidance to design and resource allocation decisions, we identify a set of engineering design principles that we consider essential to the successful construction of GENI. Many of these are related to the significant role software development is expected to play in GENI.

- **Start with a well-crafted system architecture**. The more complex the factorization of the system into a set of component building blocks, the greater the risk that the inter-dependencies among components will become unmanageable. The success of the Internet itself can be traced in large part to the fact that its architecture allowed components to evolve independently of each other. The GENI architecture is guided by the same design principle, whereby independent technologies can be plugged into the management framework with virtually no dependency on each other, and independent distributed services to be developed without heavy-weight coordination.

- **Leverage existing software**. While some aspects of GENI will need to be implemented from scratch, we expect to be able to leverage significant amounts of existing software. It is essential that we take advantage of such software, and to the extent possible, do so in a way that allows us to also leverage the support systems already in place to keep this software up-to-date. Even adapting, rather than directly using an off-the-shelf software package takes time, and raises the question of who now supports the modified package. Similar arguments favor commercially available hardware.

- **Build only what you know how to build**. Because software is plastic, there is a tendency towards feature creep; it is easier to specify the features a system “must” have, than it is to make those features work together. Left unchecked, this can result in systems that are simply too complex to work. There will be those who will complain that we are doing too little, beyond what we already understand. Our answer is, exactly, but the *synthesis* of these elements is revolutionary.

- **Build incrementally, taking experience and user feedback into account**. It is a well known result of computer science research that in software or hardware construction efforts, errors are cheapest to fix when they are caught early. The best way to do that is to put the system into active use at the earliest possible moment, gain live experience with the system, and incrementally evolve the system based on what you learn.
- **Design open protocols and software, not stovepipes**. A huge point of leverage for us, versus other examples of large scale software systems construction, is that the users of the facility—the computer science research community—are themselves capable of fixing and enhancing the system, if we give them the right tools. This is unique to the case where we build systems for ourselves, versus building systems for other people; project meltdown is much more likely if the result is take it or leave it. We aim to build a system that continues to evolve in meaningful ways after GENI construction is complete. All of the successful examples of large-scale systems being successfully delivered by the computer science research community have the property that they continued to be modified by their user community, well after initial delivery.

3. Web Services

The current GMC design makes extensive use of the World Wide Web Consortium(W3C)'s Web Services(WS) framework.[w3c] The engineering decision to use that framework is based on its interoperability, the significant existing implementation activity, and its modularity and extensibility.

From within the framework, we have made specific choices where the framework allows multiple options. For example there are several data schema specifications that are compatible with Web Services, and we chose to use XSD 1.1[xsd1,xsd2]

This section is broken into a discussion of the motivation for adopting the Web Services framework in general and detailed explanation of individual technology decisions.

3.1 Using Web Services

The Web Services framework consists of standards and implementations of a services architecture designed to allow fairly aribtrary services interoperate. It has largely grown from the desire of developers to provide more sophisticated and extensible services available using the basic HTTP/TCP access methods that are in broad use by the World Wide Web. The Web Services community covers developers interested in most aspects of service provision, from those interested in providing support for reasoning about services and interactions to those interested in insuring interoperable data formatting in individual requests. Few, if any, WS adopters use all of the existing or proposed technologies. Following our principle to "build only what we know how to build," we have primarily adopted the directly implementable parts of the WS framework, as described in detail below.

The WS effort defines a useful, cohesive implementation architecture without being unduly restraining. We describe WS as a framework, because there are many efforts operating under

the WS umbrella. These efforts are bound together through a common model of services and common implementation strategies – e.g., XML data encoding. We find that loose organization attractive in that we can leverage the efforts that simplify GMC development and deployment without being forced into accepting parts that are unattractive. The loose agreement on model and interfaces mirrors the "well-crafted architecture" requirement that we have for both the GMC and as a guiding engineering principle. Other service provision architectures, e.g, CORBA or DOM, are less flexible.

WS is also attractive because there is significant momentum among developers, especially among open source developers that GENI development can leverage. The specifications continue to evolve in response to developer experience and freely available and distributable tools abound. Many major commercial entities make their services available via WS, which offers the opportunity for the GMC or hosted experiments to make use of those services easily. Amazon and Google both export WS interfaces for developers. We have been able to install WS-based free software implementations of simple GMC interfaces on commodity wireless routers without significant difficulty, which provides that both the specifications and implementations are mature and easily adopted.

The WS standards are evolving and open allowing GENI to influence those standards based on our experience, allowing our "build incrementally" strategy to feed back into the standards as well as the GMC design. The W3C is not as open as the IETF is, but few organizations are. Though any standardization body will have its frustrations, one hopes that a major engineering undertaking like GENI would be listened to at the W3C.

In summary, the WS framework offers the opportunity to leverage significant development effort from a broad community while enhancing our ability to build incrementally. The overall architecture is well-defined and compatible with the GMC. The designs and implementations are open enough to integrate with our own open development methodology and the framework is factorable enough that we can use only the parts we need to build what we know how to build.

The remainder of this section discusses specific choices made in factoring the WS framework to implement the GMC.

3.2 Technology Decisions

We have made decisions to use WS facilities to describe services and interfaces as well as adopting specific interfaces for resource discovery services. Even within that decision are other key decisions about data representation.

3.2.1 Web Services Description Language (WSDL)

The Web Service Description Language version 1.1 (WSDL)[wsdl] is a W3C standard for describing services. It describes the contents of messages exchanged between participants, the mechanism used to encode those messages for transmission, and the pattern of communication those messages encode. It is a widely-deployed, fairly sophisticated RPC description language

that may grow into a more general communication description language. In its widely deployed versions it is capable of expressing the GMC interactions that we have specified.

WSDL is prevalent enough today that Google and Amazon both export WSDL descriptions of their services. Command-line unix tools exist to access WSDL services. Tools to go from WSDL specifications to communication code in a variety of programming languages from C to python abound. Adopting it puts a variety of fast prototyping tools into developers' hands and simplifies interoperability.

In addition to simple RPC specification, WSDL 1.1 also can encode other simple communication patterns, e.g, logging data or monitoring streams. It also describes a set of failure modes distinct from simple RPC messages. Even in WSDL 1.1 there is the notion of extending services from previous declarations. All of these are attractive features that we expect to see built on in future versions.

There are some other choices for service/RPC description. Evolving versions of WSDL are more expressive and offer object-oriented extensibility mechanisms, but implementations are less prevalent and few offer the new features. The attraction of WSDL 1.1 is that there are many interoperable tools now.

There are also WSDL-based systems that include the stateful modeling of the endpoints, e.g. ws-resource and its associated services.[ws-resource] There are two primary reasons not to adopt ws-resource: the extensions are implemented fewer places and the additional utility of representing internal object state is not compelling in implementing the GMC.

3.2.2 XML Schemas (XSD)

An XML Schema[1,2] describes a data structure and its encoding into XML. The most common way to encode message contents in WSDL is to use XSD, and the GMC adopts it. In addition to its common use and availability in tools, XSD is somewhat more rich in its description of data structures and their interrelations than competitors are. XSD can express uniqueness of fields and use of one field to key another field. While that expressiveness is helpful for implementations that understand it, implementations that do not can discard the extra information. This allows GMC developers to express these dependencies in their data descriptions.

This additional expressiveness also argues for XSD as a l*ingua franca* of data description across GENI. Developers may choose to work in other data description languages, for example RELAXNG[6] or simple DTDs, that may be simpler for their use. Most, if not all, of these data description languages can produce XSD output. This is because their most complex specifications can be expressed in XSD, but the opposite is not always true. For simple specifications any of these descriptions can be used.

The combination of expressiveness and wide adoption argues for GENI's use of XSD for data description, including message format specification.

3.2.3 SOAP

WSDL supports encoding messages using a variety of systems from a literal encoding of the schema to other XML representations like SOAP and XML-RPC. Of the major contenders here, SOAP is the most complex, but also the most expressive. The expressiveness takes the form of additional addressing mechanisms and out-of-band data.

Basically most systems can cope with SOAP encodings of GMC requests and the expressiveness of SOAP leaves the system room to grow.

3.2.4 Higher-level WS Constructs

The Web Services framework includes higher level constructs that, as of this writing, have not been fully integrated into the GMC design but that seem strong candidates. The Universal Description Discovery and Integration[uddi] (UDDI) service provides a system to discover and acquire WSDL service descriptions. Though not a W3C standard, it is standardized by the Organization for the Advancement of Structured Information Standards (OASIS)[oasis]. By providing mappings from strings to services, UDDI could be an enabling technology for resource discovery and component naming. Additionally, for services that export WSDL representations, UDDI could be a basis for service discovery.

4. X.509 Certificates

The GMC assigns unique names and public keys to the various actors in the system. These unique names are a Universal Unique Identifier (UUID) [x667] bound to a public/private keypair. Given the heavy use of Web Services, it may be worthwhile to use the Uniform Resource Identifier namespace for UUIDs[RFC4122] to encode the UUID. By exchanging a challenge /response encrypted with the keypair an actor can prove it is the actor to which the name has been assigned. In order for this to work, a way to validate that the binding of keypair to UUID has been established by the GMC.

This binding of identifier to keypair is exactly the role of the X.509 certificate[x509,RFC3820] in the World Wide Web. Like the Web Services systems above, there are widely available implementations for a variety of platforms that are in daily use. It is in our best interest to use a well defined, existing solution here rather than to invent a new system to provide the same function.

References

[oasis] OASIS home page, http://www.oasis-open.org/home/index.php

{RFC3280] Russel Housley, Warwick Ford, Tim Polk, David Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate revocation List (CRL) Profile," *RFC 3280,* ISOC, April 2002.

[RFC4122] Paul J. Leach, Michael Mealling, Rich Salz, "A Universally Unique Identifier (UUID) URN Namespace," *RFC 4122*, ISOC, July 2005.

{relaxng] The RELAXNG home page, http://www.relaxng.org/

[uddi] OASIS UDDI Specifications Technical committee page, http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm

[wsdl] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, "Web Services Description Language (WSDL) 1.1," http://www.w3.org/TR/wsdl

[ws-resource] Ian Foster, Jeffrey Frey, Steve Graham, Steve Tuecke, Karl Czajkowski, Don Ferguson, Frank Leymann, Martin Nally, Igor Sedukhin, David Snelling, Tony Storey,William Vambenepe, Sanjiva Weerawarana, "Modeling Stateful Resources with Web Services", http://www-128.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf

[w3c] World Wide Web Consortium, http://www.w3c.org/2002/ws/

[xsd1] Henry S. Thompson, C. M. Sperberg-McQueen, Shudi (Sandy) Gao, Noah Mendelsohn, David Beech, Murray Maloney, "XML Schema 1.1 Part 1: Structures," http://www.w3.org/TR/xmlschema11-1/

[xsd2] David Peterson, Paul V. Biron, Ashok Malhotra, C. M. Sperberg-McQueen, "XML Schema 1.1 Part 2: Datatypes," http://www.w3.org/TR/xmlschema11-2/

[x509] "ITU-T Rec. X.509: Information Technology – Open Systems Interconnection – The Directory: Interconnection framework," June 1997.

[x667] "ITU-T Rec. X.667: Information Technology - Open Systems Interconnection - Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components", September 2004.