G E N I

Global Environment for Network Innovations

ProtoGENI Control Framework Overview

Document ID: GENI-SE-CF-PGO-01.4

February 27, 2009

Prepared by:
The GENI Project Office
BBN Technologies
10 Moulton Street
Cambridge, MA 02138 USA

Issued under NSF Cooperative Agreement CNS-0737890

TABLE OF CONTENTS

1 Document Scope

This section describes this document's purpose, its context within the overall GENI document tree, the set of related documents, and this document's revision history.

1.1 Purpose of this Document

This document provides an overview of the ProtoGENI control framework being implemented for Spiral 1.

It is a DRAFT, to be used for discussion in the GENI Facility Control Framework working group.

Some of the material in this document is drawn from the GENI System Requirements document.

Some of the material in this document is drawn from the GENI System Overview document.

Some of the material in this document is drawn from the GENI Control Framework Requirements document.

Some of the material is drawn from a draft "Slice-based Facility Architecture (SFA)" document, dated August 8, 2008, and edited by Larry Peterson. In particular, Larry Peterson and the others who contributed to the SFA document are recognized for their significant contributions, including: Scott Baker, Ted Faber, Jay Lepreau, Soner Sevinc, Stephen Schwab, Leigh Stoller, Robert Ricci, and John Wroclawski.

Much of the material in this document was provided by Rob Ricci and Leigh Stoller, both directly and via the ProtoGENI wiki. Thank you!

1.2 Context for this Document

Figure 1-1. below shows the context for this document within GENI's overall document tree.

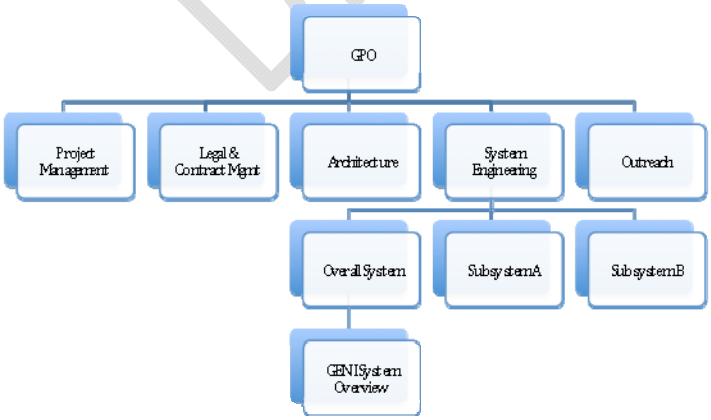

Figure 1-1. This Document within the GENI Document Tree.

1.3 Related Documents

The following documents of exact date listed are related to this document, and provide background information, requirements, etc., that are important for this document.

1.3.1 National Science Foundation (NSF) Documents

Document ID	Document Title and Issue Date
N / A	

1.3.2 GENI Documents

Document ID	Document Title and Issue Date
GENI-SE-SY-RQ-01.9	GENI System Requirements, January 16, 2009 http://groups.geni.net/geni/attachment/wiki/SysReqDoc/GENI-SE-SY-RQ-01.9.pdf
GENI-SE-SY-SO-02.0	GENI System Overview, September 19, 2008, http://www.geni.net/docs/GENISysOvrvw092908.pdf
GENI-SE-CF-RQ-01.3	GENI Control Framework Requirements, January 9, 2009, http://groups.geni.net/geni/attachment/wiki/GeniControlFrameworkRequirements/010909b%20%20GENI-SE-CH-RQ-01.3.pdf

1.3.3 Standards Documents

Document ID	Document Title and Issue Date
N / A	

1.3.4 Other Documents

Document ID	Document Title and Issue Date
GDD 06-10	"Towards Operational Security for GENI," by Jim Basney, Roy Campbell, Himanshu Khurana, Von Welch, GENI Design Document 06-10, July 2006. http://www.geni.net/GDD/GDD-06-10.pdf
GDD 06-23	"GENI Facility Security," by Thomas Anderson and Michael Reiter, GENI Design Document 06-23, Distributed Services Working Group, September 2006. http://www.geni.net/GDD/GDD-06-23.pdf
N/A	"GMC Specifications," edited by Ted Faber, Facility Architecture Working Group, September 2006. http://www.geni.net/wsdl.php
GDD 06-24	"GENI Distributed Services," by Thomas Anderson and Amin Vahdat, GENI Design Document 06-24, Distributed Services Working Group, November 2006. http://www.geni.net/GDD/GDD-06-24.pdf
GDD 06-38	"GENI Engineering Guidelines," edited by Ted Faber, GENI Design Document 06-38, Facility Architecture Working Group, December 2006.

	http://www.geni.net/GDD/GDD-06-38.pdf
GDD 06-42	"Using the Component and Aggregate Abstractions in the GENI Architecture," by John Wroclawski, GENI Design Document 06-42, Facility Architecture Working Group, December 2006. http://www.geni.net/GDD/GDD-06-42.pdf
N/A	"Slice Based Facility Architecture," v1.10, August 8, 2008, by Larry Peterson, et.al. http://groups.geni.net/geni/attachment/wiki/GeniControlBr/v1.10%20%20080808%20%20sfa.pdf
N/A	"Large Scale Virtualization on the Emulab Network Testbed," June, 2008, by Mike Hibler, et.al. http://www.cs.utah.edu/flux/papers/virt-usenix08-base.html
N/A	"Implementing the Emulab-PlanetLab Portal: Experience and Lessons Learned," December, 2004, by Kirk Webb, et.al. http://www.cs.utah.edu/flux/papers/portal-worlds04-base.html
N/A	"Decentralized Trust Management," 1996, by Matt Blaze, et.al, AT&T Research. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.44.6276
N/A	"Compliance Checking in the PolicyMaker Trust Management System," 1998, by Matt Blaze, et.al, AT&T Research. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.2525
N/A	"The Role of Trust Management in Distributed Systems Security," 1999, by Matt Blaze, et.al, AT&T Research. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.7726

1.4 Document Revision History

The following table provides the revision history for this document, summarizing the date at which it was revised, who revised it, and a brief summary of the changes. This list is maintained in reverse chronological order so the newest revision comes first in the list.

Revision	Date	Revised By	Summary of Changes
01.1	12/5/08	H. Mussman	Completed draft, for limited review, based on material provided by Rob Ricci and Leigh Stoller.
01.2	12/15/08	H. Mussman	Updated draft to follow structure from Control Framework Requirements document.
01.3	1/14/09	H. Mussman	Updated with small changes.
01.4	2/27/09	H. Mussman	Updated after discussion with Rob Ricci and to include new content from ProtoGENI wiki.

2 GENI System Overview

2.1 Major Entities and their Relationships

Figure 2-1 presents a block diagram of the GENI system covering the major entities within the overall system. Optional (but desirable) parts are shown "grayed-out." See the GENI System Overview document at http://www.geni.net/docs/GENISysOvrvw092908.pdf for more details.

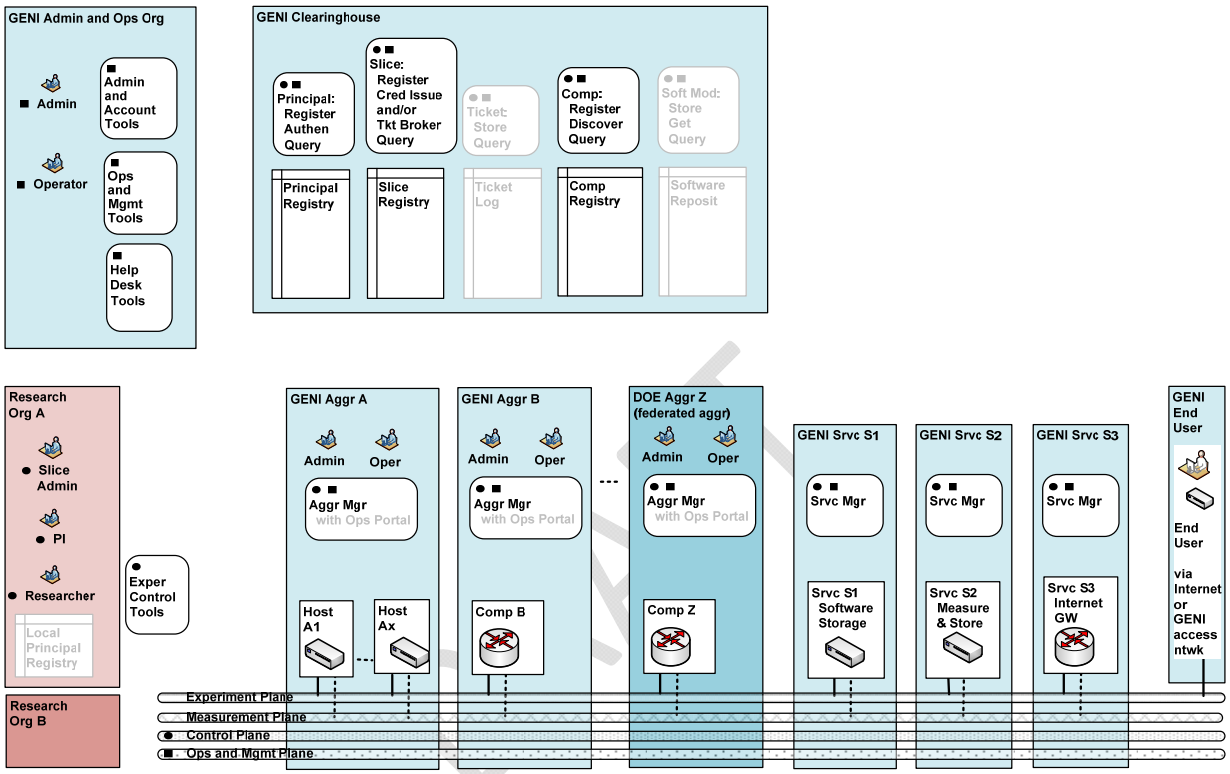

Figure 2-1. GENI System Diagram.

2.2 Federated Suites

Figure 2-2 provides a system diagram illustrating federation between one GENI suite and another. As a hypothetical example, it depicts federation between a US-based GENI suite and a compatible suite in the European Union (EU).

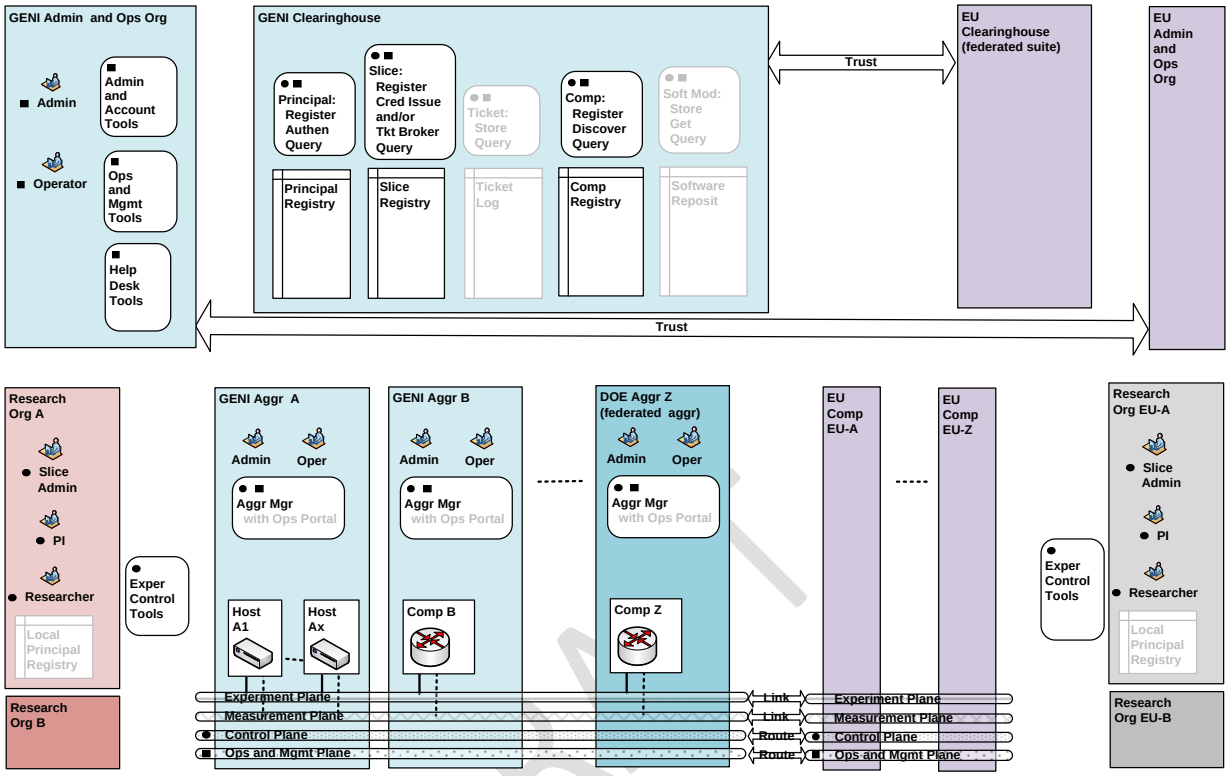

Figure 2-2. System Diagram with Federated Infrastructure Suites.

2.3 Slices

Figure 2-3 shows two researchers from different organizations managing their two experiments in two corresponding slices. Each slice spans an interconnected set of slivers on multiple aggregates and/or components in diverse locations. Each researcher remotely discovers, reserves, configures, programs, debugs, operates, manages, and teardowns the "slivers" that are required for their experiment. Note that the clearinghouse keeps track of these slices for troubleshooting or emergency shutdown.

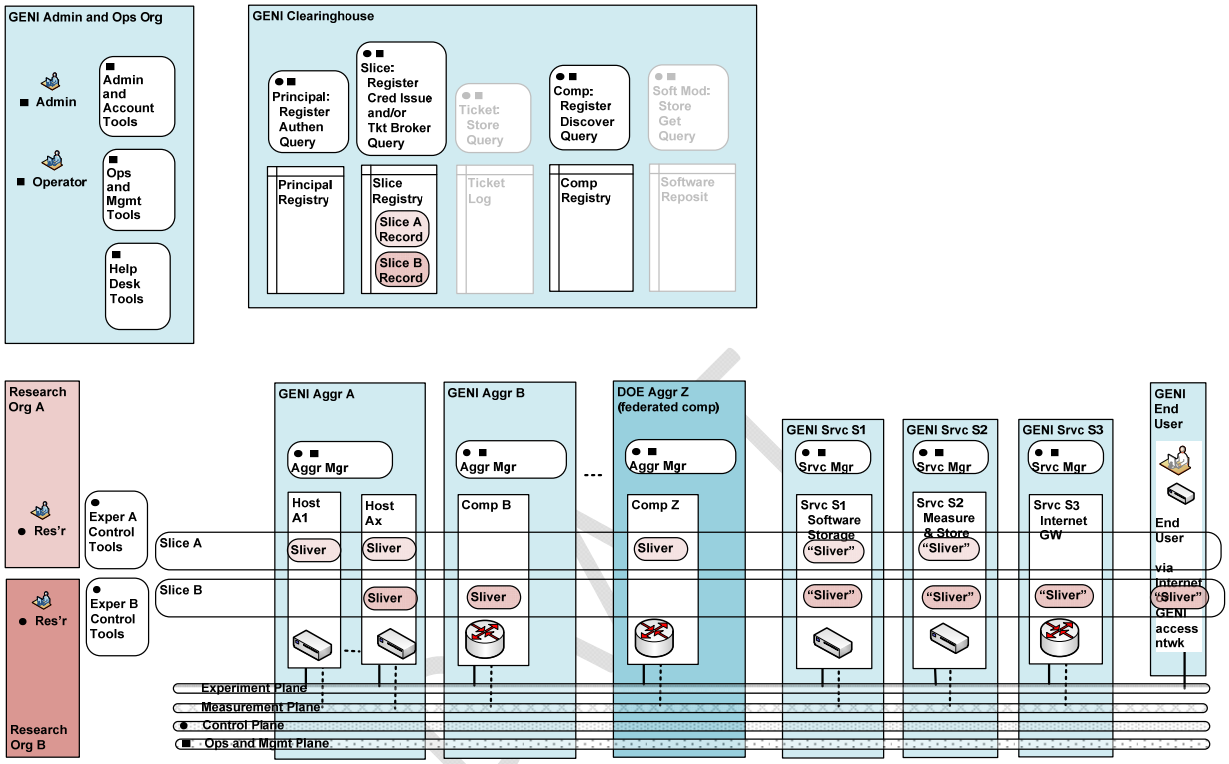

Figure 2-3. Two GENI Slices.

An aggregate manager a) interacting with the researcher (or her proxies) via the control plane and b) configuring the devices over internal interfaces establishes Slivers. Components may be virtualized, and can thus provide resources for multiple experiments at the same time, but keep the experiments isolated from one another. In addition, each slice requires its own set of experiment support services. Furthermore, as shown in Slice B, "opt-in" users may join the experiment running in a slice, and thus be associated with that slice.

3 GENI Control Framework Overview

3.1 Definition

The GENI control framework is defined in the GENI Control Framework Requirements document at http://groups.geni.net/geni/attachment/wiki/GeniControlFrameworkRequirements/010909b%20%20GENI-SE-CH-RQ-01.3.pdf

It includes the following clearinghouse entities in a GENI suite:

- Principal registry and related services.
- Component registry and related services.
- Slice registry and related services.
- An optional ticket log and related services, for holding "sliver records", used in administering and managing the GENI suite.
- An optional software repository, for holding software objects that are required to administer, operate or manage the GENI suite.

It includes the following entities associated with each aggregate or component that is providing experiment resources in a GENI suite:

- An aggregate manager and related services.
- An optional component manager and related services, for components that are part of an aggregate.
- An optional broker service and related services, that typically functions as an aggregate-of-aggregates manager in the GENI suite.

It includes the following entities associated with a principal who is utilizing, administering or managing experiment resources in a GENI suite.

- A principal acting from a server utilizing a browser client and/or a set of helper tools
- A principal service acting on behalf of a principal , utilizing a browser client and/or a set of helper tools, that appears as a principal in the GENI suite.

The GENI control framework defines:

- Interfaces between all entities.
- Message types including basic protocols and required functions.
- Message flows necessary to realize key experiment scenarios.

3.2 Requirements

The GENI control framework requirements are presented in the GENI Control Framework Requirements document at http://geni.bbn.com:8080/docushare/dsweb/Services/Document-1234 .

3.3 Implementation Approach for Spiral 1 Prototypes

Five control framework implementations are being developed for Spiral 1 prototypes, based on the following systems and software packages:

- PlanetLab, a system that allows researchers to conduct experiments on hosts located at various sites; see http://svn.planet-lab.org/ and http://groups.geni.net/geni/wiki/PlanetLab .
- ProtoGENI, which is based Emulab, a system that allows researchers to conduct experiments on hosts and other equipment located in various sites; see https://www.protogeni.net/trac/protogeni and http://groups.geni.net/geni/wiki/ProtoGENI .
- ORCA resource allocation software; see http://nicl.cod.cs.duke.edu/orca/ and http://groups.geni.net/geni/wiki/ORCABEN .
- ORBIT, a system that allows researchers to conduct experiments on a federated arrangement of wireless networks, utilizing peridically disconnected resources; see http://www.orbit-lab.org/wiki/WikiStart and http://groups.geni.net/geni/wiki/ORBIT .
- TIED, the Trial Integration Environment in DETER, a system that allows researchers to conduct experiments on a federated arrangement of hosts located in various sites; see http://seer.isi.deterlab.net/ and http://groups.geni.net/geni/wiki/TIED .

Each control framework will be used for one cluster of projects, and typically provides the clearinghouse and a reference implementation of the aggregate manager for its cluster. Researchers in a given cluster will typically be able to conduct experiments only on the prototype aggregates and components within that cluster.

At the completion of Spiral 1, these control framework prototypes will be compared and evaluated.

For Spiral 2 prototyping during the following year, improvements and/or consolidations are expected. Useful features in one control framework may be adopted by another. A particular project may choose to move from one control framework to another. And, it is also possible that two (or more) control frameworks may merge, or possibly just federate.

Sections 4 though 11 present an overview of the ProtoGENI-based control framework implementation.

4 ProtoGENI Control Framework Structure

A block diagram of the ProtoGENI control framework structure is shown in Figure 4-1, which includes:

- A Clearinghouse which includes a Registry for Principal, Slice and Aggregate/Component Records, a set of common Registry Services, and certain specialized Principal, Slice and Component Services.
- Slice Authority Services, that are currently provided within existing Emulab Sites.
- Aggregates, each of which includes a ProtoGENI Aggregate Manager and zero, one or many Components. Each of the current aggregates comprise all of the hosts and other resources accessible within an Emulab Site.
- Principals, including Researchers and their associated Slice Managers.

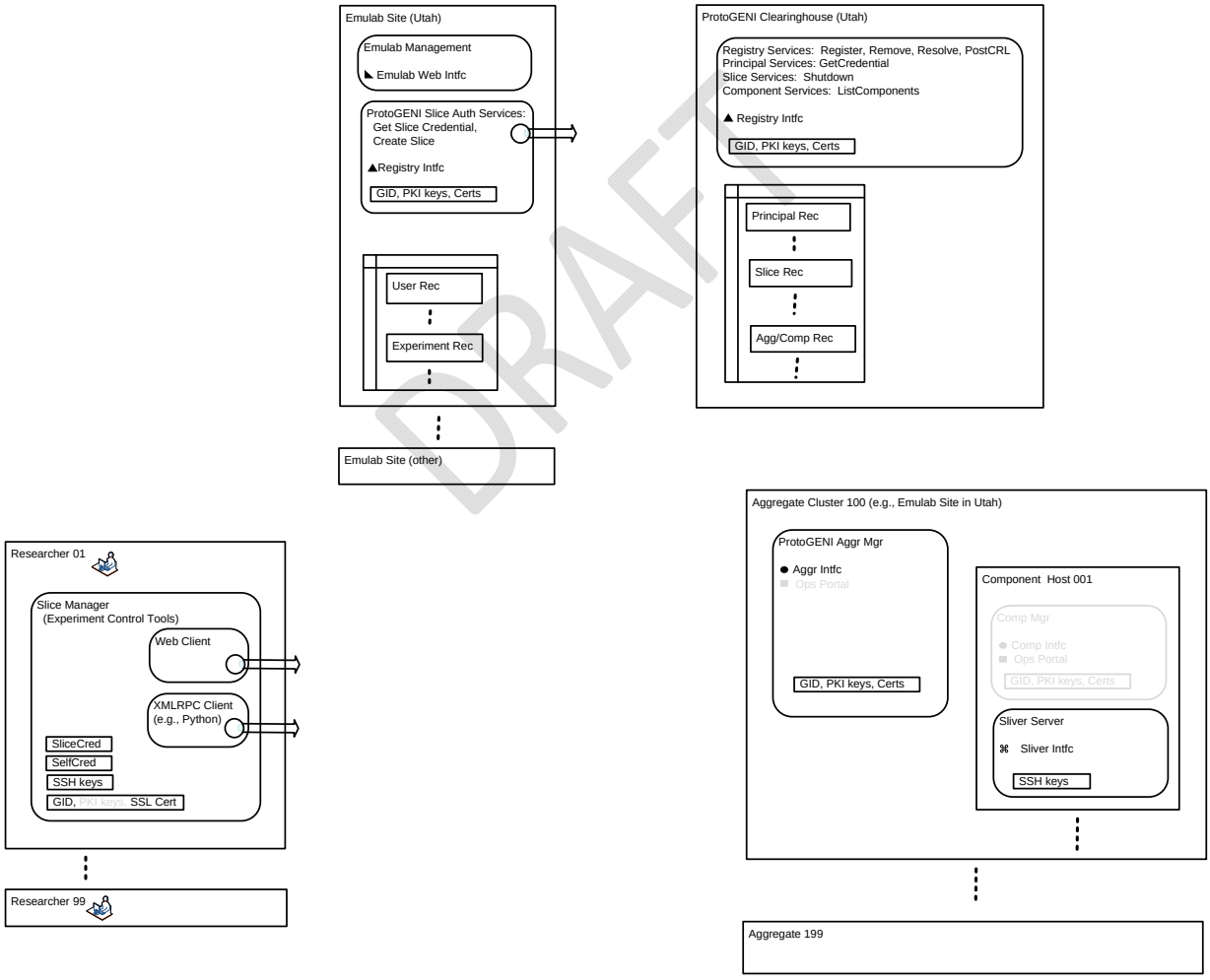

Figure 4-1. ProtoGENI Control Framework Structure.

The ProtoGENI control framework structure is based on the "Slice-based Facility Architecture" (SFA) summarized in http://groups.geni.net/geni/attachment/wiki/GeniControlBr/v1.10%20%20080808%20%20sfa.pdf , but it includes a number of engineering decisions that are consistent with a "prototype" implementation. Note that the SFA is based on earlier GENI efforts, including: "GMC Specifications," "GENI Distributed Services," and "GENI Engineering Guidelines."

For a current view of the ProtoGENI design, see the ProtoGENI wiki at http://www.protogeni.net/trac/protogeni and http://www.protogeni.net/trac/protogeni/wiki/BecomingAProtoGENISite

4.1 Registry

ProtoGENI implements a centralized clearinghouse and Registry that is co-located with the Emulab site in Utah; for more information, see http://www.protogeni.net/trac/protogeni/wiki/ClearingHouseDesc .

The Registry exports a Registry Interface.

Per the SFA, each registered item has a Global Identifier (GID) that includes a UUID and a Global NAME (GNAME). The UUID is a random number, generated following X.667 (RFC4122), that is guaranteed to be unique. Once assigned to an item, its UUID should never change. The registry Resolve() and GetCredential() functions take either a GNAME or a UUID as the target of the lookup operation.

Principals (users) and Slices are registered in the Clearinghouse by the Slice Authorities at each Emulab instance.

Question: Aggregates (currently a federated Emulab site) are registered by?

Operationally, the Utah clearinghouse does a limited set of tasks:

- Allows registration, deletion, and resolution of various principle objects. Currently limited to slices, users, slice authorities, aggregate managers, and components.
- Provides a list of all known aggregate managers. The information returned includes the URL a client can contact to get a list of currently available resources.
- Exports a centralized "emergency shutdown" facility that can be used to terminate a misbehaving slice.
- Assists in the registration of new federates as they come on-line.
- Collects and distributes Certificate Revocation Lists from all of the federates. These CRL's are generated periodically by the federates so that others in the federation can be made aware of users that have been terminated or who have had to generate a new certificate (GID).

Other items of note:

- Only one instance of the clearinghouse is in operation. While the architecture eventually calls for multiple clearinghouses, there is no support in the current code for federating multiple clearinghouses together.

- Everything is stored in a mysql database.
- The location (URL) of the clearinghouse is hardwired, in the sense that the initialization script that federates run, downloads the GID (certificate) of the clearinghouse from Utah. Inside that certificate is a URL (in the Subject Alternative Name of the certificate).
- Most operations are restricted to known Authorities (SA,CM); mere users cannot access the clearinghouse, except where noted below.

The current implementation of the clearinghouse is highly dependent on the Emulab source code. A number of Emulab libraries and support systems are used by the clearinghouse, but in general it is not really necessary to have an actual Emulab testbed running in order to run an instance of the clearinghouse. A future project is to provide an installation target for the Emulab code that would install the bare minimum needed to run a clearinghouse.

As noted above, mere users do not generally talk to the clearinghouse. Rather, known Slice Authorities and Component Managers are the only ones that can register and delete entries, as well as resolve records. Authorities can get a credential to access the clearinghouse in one of two ways;

- Request a credential via XML-RPC (see below). The credential is signed using the clearinghouse's GID (certificate).
- Generate a self-signed credential. In this case, the authority uses its own certificate (GID) to sign the credential. This is the same certificate that the authority then uses to establish the SSL connection. The credential is accepted because the signature is valid, and because clearinghouse has that GID in its database.

The clearinghouse is accessed through XML-RPC over SSL.

Clients supply a client certificate as part of SSL connection establishment. This certificate must be signed by an authority known to the clearinghouse (these authorities correspond to the federates).

The clearinghouse's URL is: https://www.emulab.net/protogeni/xmlrpc/ch .

The Clearinghouse is somewhat simple at this point, mostly because the size of the federation is small, as are its needs. Items that need to be addressed are:

- There is no web interface to see what is in the Clearinghouse. Admin people must use mysql commands to look at the tables.
- There is no up to date view of slices that are instantiated (slivers).
- No global view of what resources are currently available.
- No expiration of registry entries and no renew.

4.2 Aggregates and Components

A ProtoGENI aggregate may comprise all of the resources running in an Emulab site, including cluster nodes, switches and other resources.

An Aggregate Interface is exported from an aggregate comprising an Emaulab site, but individual resources such as nodes and switches do not export a component interface; all operations on individual resources are via the aforementioned Aggregate Interface. See Figure 4.2 and http://www.protogeni.net/trac/protogeni/wiki/ComponentManagerAPI

Private communication channels are then invoked to handle any operations required on individual resources.

Question: How will aggregates be built that do not include an Emulab site?

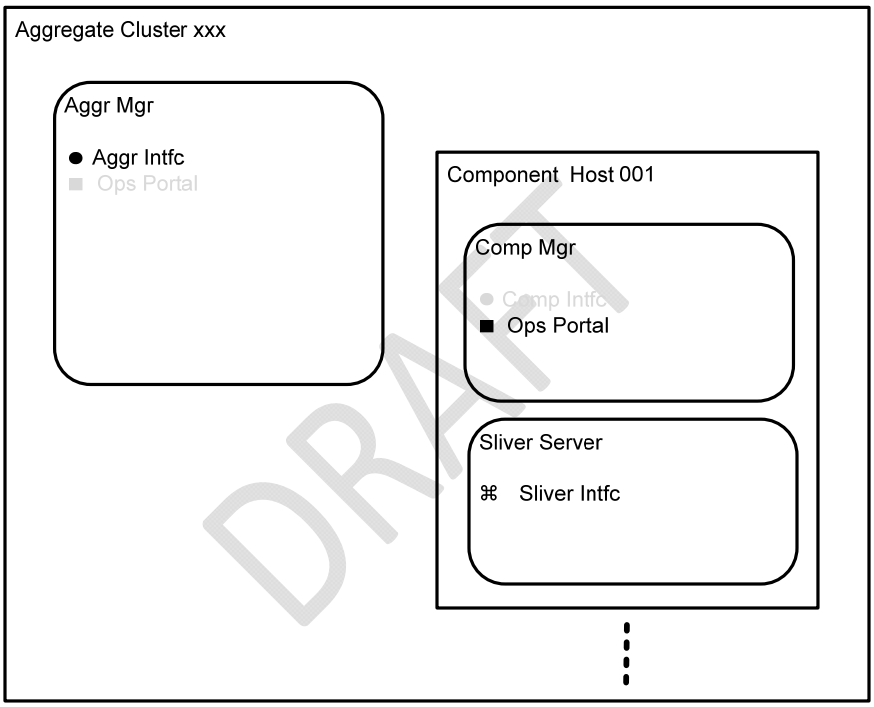

Figure 4-2. An Aggregate and Its Internal Structure.

4.3 Principals

ProtoGENI principals (users) include:

- Researchers, who utilize the ProtoGENI suite for running experiments, deploying experimental services, etc.

Question: What other principals are defined in GENI?

Each Researcher uses an associated Slice Manager to setup and run experiments. It includes:

- A web client.
- An XML_RPC client.

- An SSH client.

See Figure 4.1.

4.4 Services

(Not yet defined in ProtoGENI.)

4.5 Slices

From a Researcher's perspective, a slice is an interconnected set of reserved resources, or slivers, on heterogeneous substrate aggregates (components). Researchers can remotely discover, reserve, configure, and program, debug, operate, manage, and teardown resources on these aggregates (components) to setup, utilize and then teardown the slivers necessary to complete an experiment. See Section 2.3.

From an Operator's perspective, slices are the primary abstraction for accounting and accountability—resources are acquired and consumed by slices, and external program behavior is traceable to a slice, respectively.

Question: Does a ProtoGENI slice map to an Emulab experiment when an aggregate comprises an Emulab site?

4.6 Control Message Flows

Control message flows in the ProtoGENI control framework are summarized by:

- Researchers that periodically connect and communicate with Registries and Aggregates via defined interfaces and APIs so that they can acquire the resources necessary for them to setup and execute experiments. See Figure 4.3.
- In these transactions, Aggregates supply resources, and Researchers consume resources.
- Since the Aggregates are expected to be widely distributed, and connections are made over an IP network that may be the open Internet, the control message flows must be secure, and the Researchers must be properly authenticated and authorized.

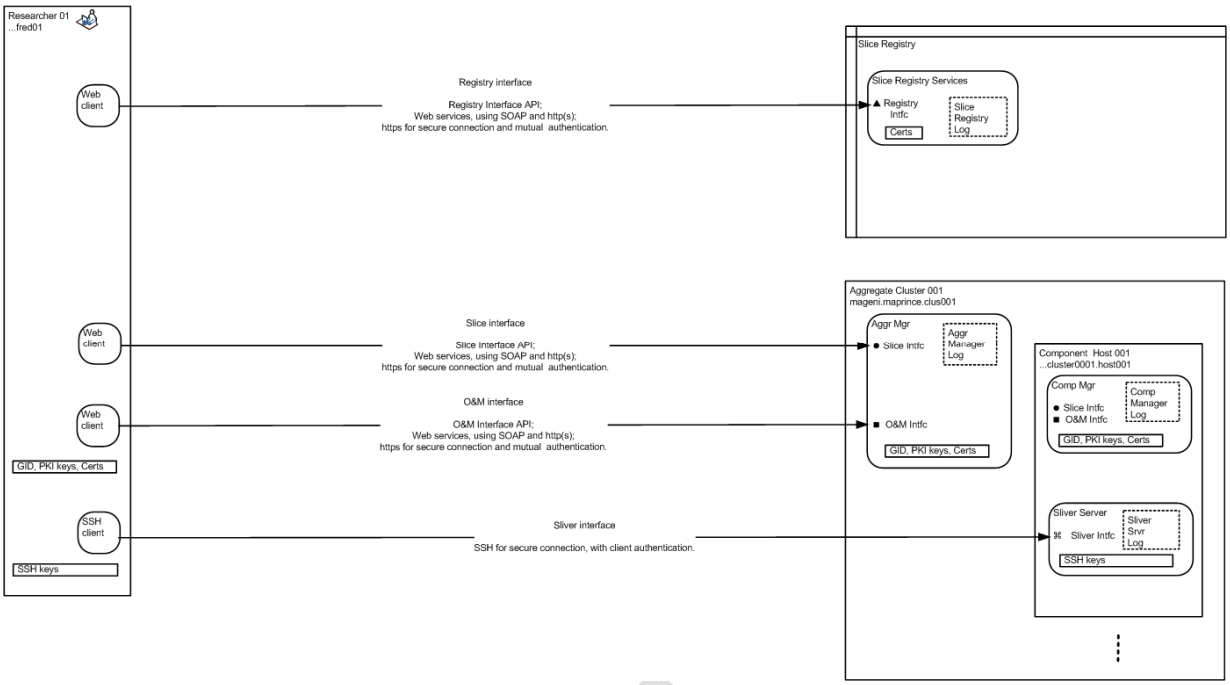

Figure 4.3 – Researcher to Registry and Aggregate Message Flows
(Changes required in figure)

4.7 Registry and Aggregate (Component) Interfaces

In ProtoGENI, communications between a Researcher using their Slice Manager and the Registry and Aggregate (Component) Interfaces follows a web services model. See Figure 4.3. The current decision is to utilize:

- A custom API for each of these interfaces which utilizes web service messages.
- The Researcher (principal) is on the client side, and uses an xmlrpc client to talk to the servers, such as python.
- The Registry and Aggregate (Component) Interfaces are on the server side.
- Web service messages that utilize XML_RPC and http(s) protocols for messages.
- A WSDL that specifies the message structures.
- An XSD that specifies the data structures.
- http(s) utilizes TLS to provide a secure connection.
- http(s) provides for client authentication at the server by using a root certificate.

Question: Is http(s) used to provide for server authentication at the client?

4.8 Ops and Mgmt Interfaces

(Note yet defined in ProtoGENI.)

4.9 Sliver Interfaces

In ProtoGENI, a "sliver" can consist of local cluster nodes, including vlan links between them, cluster nodes at different sites with GRE tunnels between then, and PlanetLab nodes.

Unlike Emulab experiments, raw ProtoGENI nodes are currently just that, raw. None of the Emulab setup is done on those nodes, like experimental interfaces initializing interfaces with IP addresses, building accounts for other project members, starting programs automatically with the event system, etc. This will eventually be supported in the "cooked" interface.

Once a Sliver has been created on a component, a Sliver Interface is typically provided on the component to allow the Researcher to program, configure and operate a server within the underlying component, here designated the Sliver Server. The Sliver Server may be a physical server, or a virtual machine. The Sliver Server may represent a component (host) itself, or a part of the component, e.g., a controller acting as a protocol engine.

Typically, a Researcher on their server connects with the Sliver Server via the Sliver Interface using a Secure Shell (SSH) login. See Figure 4.3.

An SSH login provides for almost complete control of the Sliver Server within the component, and it is important that that server by well isolated from other slices to prevent security breaches.

Client authentication is provided in an SSH login. The Sliver server permits an SSH connection for a client holding a private key matching a public key held in a local folder. Both public and private keys are held in the SSH client, and the public key is loaded into the Sliver Server using a call to the Aggregate Interface.

The UpdateSliceAttributes() function at the Aggregate Interface allows for the specification of user keys, init scripts, and other as yet undefined items, to be associated with a slice on a component. The call can be made prior to any RedeemTicket() or InstantiateSlice() calls (before a sliver exists on the component) so that keys and the like can be pre-staged.

UpdateSliceAttributes(Credential, AttrNames[], AttrValues[]);

where AttrNames is currently one of "keys" or "initscript". When specifying keys for a slice, the value is another array that allows multiple keys to be associated with multiple researchers, thus binding a set of researchers (and their keys) to the slice, on that component.

4.10 Identification

Per the SFA, each registered item has a GENI Identifier (GID) that includes a Human Readable Name (HRN) and a UUID.

The UUID is a random number, generated following X.667 (RFC4122), that is guaranteed to be unique.

The registry Resolve() and GetCredential() functions take either a HRN or a UUID as the target of the lookup operation.

A GID (GENI Identifier) is simply an SSL certificate with a Distinguished Name (DN) field that conforms to a set of conventions. There is also some additional info in the Subject Alternative Name part of the certificate.

Stored in the DN field of the certificate are the user's: UUID ; Human Readable Name (HRN); and their email address.

Fundamentally, it is the UUID that determines which user; a user may regenerate their certificate, but they will have the same UUID embedded in it.

The HRN is included for compatibility with other systems, but within ProtoGENI it is not used for identification (although it can be used in Resolve() operations).

Each of other types of principle objects in ProtoGENI also has a unique UUID and thus a certificate (GID) associated with it. In most cases these certificates are used for identity purposes, not authentication (as in an SSL session). For example, when constructing a credential for a slice, the target of the credential is the GID for the slice. More on this in the credential section below.

4.11 Authentication

The current approach to the authentication of ProtoGENI entities is summarized by:

- One Public Key Infrastructure (PKI) that covers all ProtoGENI registries, slice authorities, aggregates and researchers (principals).
- This PKI provides all necessary certificates, and allows verification to be done using a limited number of root certificates.
- Certificate Revocation Lists (CRLs) are provided by the entities.

See https://www.protogeni.net/trac/protogeni/wiki/AuthImpl

ProtoGENI employs a "web of trust" model in which all of the members of the federation hold copies of all root keys (trust anchors). Since the number of trusted "roots" will be small at first, we exchange root SSL certificates out of band, and populate a certificate directory that can be used for verifying client certificates when they are presented.

Each member of the ProtoGENI federation is itself an Emulab installation. Part of the installation process for each federate is the generation of a self-signed certificate authority certificate, that serves as the "root" for that Emulab. This root certificate is subsequently used to sign, among other things, the certificates of its users.

Each user has an encrypted SSL certificate issued by their home Emulab that authenticates them to the entire federation, by virtue of the fact that the certificate is signed by one of the federation members (trust anchor).

This certificate becomes their ProtoGENI identity; it allows them to initiate an SSL session with any of the RPC servers in the federation. Since all of the federates maintain a cache of the root certificates, the signatures can be trivially verified at any federate.

In short, SSL's built in client verification is used to step back through the chain of issuers, which in the current prototype is usually of length one. As a result, ProtoGENI does not currently use or implement GID chains.

Note that we will support GID chains in the future, so that we conform to the SFA spec and so that we can inter-operate more easily with other GENI Spiral 1 implementations. The current approach was taken so that we can develop the other parts of ProtoGENI, while we wait for the specification of GIDs to be finalized.

CRLs (Certificate Revocation Lists) are supported by ProtoGENI. Each member of the federation runs a cron job that generates a CRL once a day, and posts that CRL to the Clearinghouse. The

Clearinghouse collects the CRLs and combines them into a single bundle and posts them for all of the federates. The CRLs allow the federates to be notified in a timely manner of users that are no longer allowed to use the ProtoGENI APIs (say, if a user account is terminated). CRLs also allow users to change their certificates; the old certificate is revoked so that it can no longer be used.

4.12 Authorization

The current approach to authorization in the ProtoGENI suite is summarized by:

- The exchange of credentials, including tickets, that mediate resource authorization and assignment by Aggregates, plus resource control in Aggregates.
- These credentials are signed (certified) by the appropriate authorities (slice) and objects (aggregates and slivers) to give them some intrinsic value.
- To certify a credential (ticket), an authority or object signs the token using its own private key, which is then followed by signatures from its responsible authorities, up to the root authority. In the current implementation, there is always only one signature.
- The Public Key Infrastructure (PKI) that is used to authenticate principals, etc., provides all of the keys and other structure to sign and verify credentials.
- The Aggregate that receives this credential can then verify it using a limited set of root certificates.
- This follows the approach that was developed in "trust management systems" and in early GENI designs.
- This follows the SFA; see http://groups.geni.net/geni/attachment/wiki/GeniControlBr/v1.10%20%20080808%20%20sfa.pdf .

Currently, ProtoGENI has implemented a few simple policies that federates (Aggregates) can apply to remote users.

- Individual nodes may be prevented from being viewed and allocated by remote users
- All access for remote users can be turned off (eg. during times of high local need, such as near the end of a semester)
- The total number of nodes allocated to remote users can be limited

These are by no means a complete set, and it's possible for federates (Aggregates) to implement their own policies if they wish.

4.13 Credentials

A ProtoGENI credential is the basis for granting privileges to users, where privileges take the form of a small XML document describing what actions may be taken by a particular user (Researcher) towards a specific ProtoGENI object (Aggregate). Each credential has an owner and a target.

In the current implementation, the set of privileges roughly corresponds to the allowed API calls.

For example, a Researcher requires a Credential to access an Aggregate via the Aggregate Interface to have resources authorized and assigned to create slivers, and to control slivers. Once a Researcher presents a credential to an Aggregate Manager, the Aggregate Manager then decides whether to authorize the requested function, or not.

Slivers in ProtoGeni are first class objects; every sliver is controlled via a credential that is created when the sliver is instantiated. RedeemTicket() and InstantiateSlice() both return a credential to the caller. While a credential can be requested via the MA interface, we feel that to be an unnecessary additional step.

A ProtoGENI credential as currently defined is shown in https://www.protogeni.net/trac/protogeni/attachment/wiki/Authentication/credential.rnc and includes:

```
## A credential granting privileges or a ticket.
credentials = element credential {
   ## The ID for signature referencing.
   attribute xml:id {xs:ID},
   ## The type of this credential. Currently a Privilege set or a
Ticket.
   element type { "privilege" | "ticket" | "capability" },
   ## A serial number.
   element serial { xsd:string },
   ## GID of the owner of this credential.
   element owner_gid { xsd:string },
   ## GID of the target of this credential.
   element target_gid { xsd:string },
   ## UUID of this credential
   element uuid { xsd:string },
   ## Expires on
   element expires { xsd:dateTime },
   ## Privileges or a ticket
   (PrivilegesSpec | TicketSpec | CapabilitiesSpec),
   ## Optional Extensions
   element extensions { anyelementbody }*,
   ## Parent that delegated to us
   element parent { credentials }?
}

SignedCredential = element signed-credential {
   credentials,
   signatures?
}
```

The current implementation of credentials uses the XML Digital Signatures format, and the XMLSEC library to generate and verify the signed document.

A credential is signed using the XMLSIG specification, located at http://www.w3.org/TR/xmldsig-core/ . To facilitate delegation, a credential can have an optional chain of parent credentials, each one signed. An original (un-delegated) credential will not have a parent. A credential is delegated by creating a new credential, setting the parent to the original credential, and then signing the entire blob. The credential can then be verified by reversing the operation, verifying the signature at each level. An existing tool called xmlsec1 (http://www.aleksey.com/xmlsec/) is used for the verification. A secondary step is then used to ensure that the rules of delegation were not violated (ie: an inner credential does not include a privilege that an outer credential did not allow to be delegated).

Each credential has its own UUID to allow for easier bookkeeping, and for simple revocation. A UUID is also useful when supporting unmediated splitting of tickets; the CM needs to be able trace back the split ticket to the original ticket.

We currently feel (although not very strongly) that delegation should be at the individual privilege level, not at the credential level, except when the credential is really a ticket.

Credentials are extensible, using the "extensions" field above. No format is defined as of yet, but the intent is to be able pass along data in the credential that has been signed along with the rest of the credential data. When delegating a credential, the extension data must remain unchanged.

As an example, consider the user Joe who has created a slice nicknamed MySlice. When Joe asks his Slice Authority to create this new slice, a new credential is formed that includes, among other items:

```
Joe's GID (UUID, HRN, email)

MySlice's GID (UUID, HRN, email)

A list of tokens

A digital signature
```

Joe's GID is the one that is maintained by his home Emulab. The GID for MySlice is a brand new certificate which includes a new UUID assigned to MySlice. The list of tokens defines what operations (destroy, modify, etc) Joe may take with respect to MySlice. Lastly, a digital signature that covers all of this information. The entire credential, still an xml document, is returned to Joe so it can be used in subsequent operations. The next thing Joe might do is destroy the slice. To do that, Joe will initiate an SSL session with the Slice Authority RPC server, passing the credential to the DeleteSlice() operation. The operation succeeds because the credential is valid (the signature verifies) and Joe is the user in the credential.

Regarding the last statement, the signature verifies because any unauthorized changes would invalidate the digital signature. In addition, only Joe can use this credential because only Joe has the pass phrase to his personal certificate, that he uses to initiate the SSL session with the RPC server. Thus, the certificate (GID) associated with the client of the session is the same as the GID contained in the credential.

Please see the credential specification for more details.

4.14 Tickets

In ProtoGENI, a Ticket is a specific kind of credential. See Section 4.13.

Tickets are a variant of a credential, that include a description of the resources being promised by an Aggregate in the form of an RSpec.

As with credentials, tickets includes the GID of the user and the GID of the slice to which the ticket refers. Thus, a ticket grants the holder a "promise" to assign the set of resources in the RSpec, to the slice that is referenced in the ticket. Once the ticket is redeemed, it is effectively useless in subsequent operations.

Credentials and tickets both include an expiration date. This expiration is currently unused in credentials, but is set to a very short time (on the order of minutes) for tickets. Component Managers will not honor a ticket that has expired; a new ticket must be requested, although the same resources may no longer be available.

4.15 Resource Specification (RSpec)

A *resource specification* (RSpec) describes a component in terms of the resources it possesses and constraints and dependencies on the allocation of those resources.

The ProtoGENI rspec is described at https://www.protogeni.net/trac/protogeni/wiki/RSpec . It is based on the ptop/vtop format designed for assign. This format is fairly simple, and has the advantage of being in production use for 8 years now in Emulab. This format does have some parts that are specific to Emulab and assign: these will simply be ignored to begin with, and we will remove them from the RSpec or implement them as extensions as we move to a full GENI RSpec.

The RSpec schema is given in the "RelaxNG Compact" syntax. The advantage of doing this is that RNC is very human-readable and flexible, and can be translated to the W3C XML schema, which is more widely supported, but is incredibly hard to read/write by hand.

Key features of the ProtoGENI RSpec :

- Identifying resources
  - All nodes and links are identified by a UUID
  - There is also a "human readable" name field to aid readability
- Nodes
  - Node have types
  - Virtualization technology is included as a field
- Links
  - Links are point-to-point: LANs and other "full connectivity" environments (such at the Internet), a "LAN node" is created, and all members are linked to it.
  - Links have bandwidth, a type, etc.
  - Links endpoints reference Interfaces on Nodes
- Interfaces
  - Endpoint of a link
  - Named by node, plus an opaque interface name
  - *In progress*: Interfaces will be first-class entities, declared as part of the component they belong to
- Metadata
  - A "valid until" field
  - A "generated" time

Common topological elements, used by the ProtoGENI RSpec can be found at http://www.protogeni.net/trac/protogeni/attachment/wiki/RSpec/top.rnc

Schema for most elements in the ProtoGENI RSpec 'body' can be found at

http://www.protogeni.net/trac/protogeni/attachment/wiki/RSpec/protogeni-rspec-common.rnc

Per the ProtoGENI implementation, an RSpec can include a request for a link between two nodes, or a LAN between a set of nodes. ProtoGENI supports independent control of these links and LANs, and even the individual interfaces on nodes attached to the links. As a result, ProtoGENI treats these sub resources the same as slivers, creating credentials that the caller can use to control them. In addition to the defined sliver operations in the API, ProtoGENI will export additional APIs that are specific to these other resources.

Since Emulab clusters exporting the ProtoGENI APIs, are exporting n Aggregate Interface, the resulting sliver (credential) might encompass a collection of resources (nodes and links). To operate on an individual piece of that sliver, such a link, it must be possible to extract a credential that refers to that link. ProtoGENI provides an extraction operation that takes a credential and a GNAME, and returns a new credential for that specific resource. For example, to extract a credential for a link, you would use the extraction operation on the credential for geni.emulab.slice0.link0, to get a credential for an interface, geni.emulab.slice0.node0.eth0.

5 Principals in the ProtoGENI Control Framework

5.1 Identification

Each principal (user) in ProtoGENI, such as a Researcher, has a unique GENI Identifier (GID) that includes a Human Readable Name (HRN) and a UUID.

See Section 4.10.

Question: Who creates the UUID, and names the Researcher (principal)?

5.2 Registration

Each principal (user) is registered within the ProtoGENI suite, so that they can be identified and authenticated.

See Figure 5-1 for the steps in registering a Researcher.

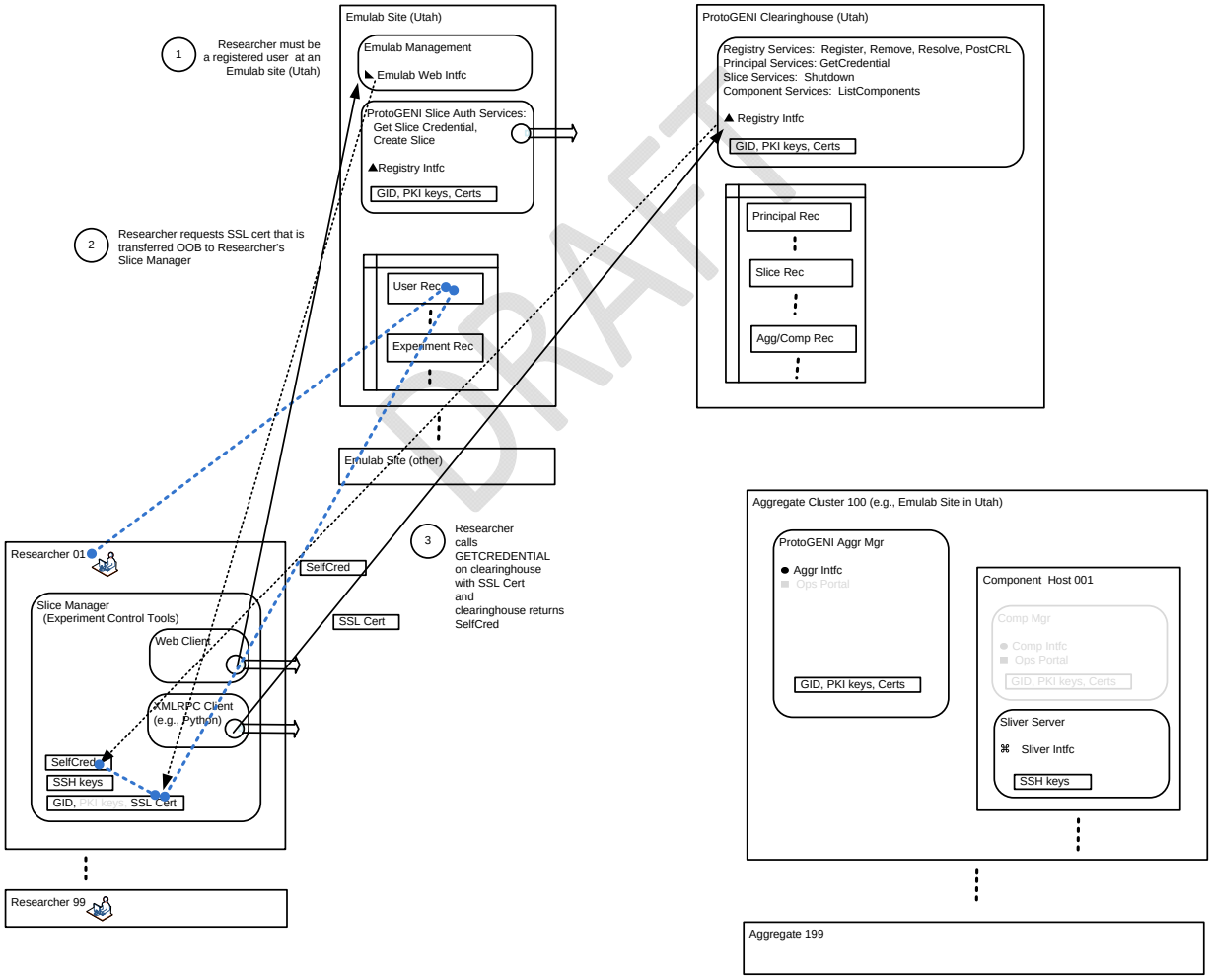

Figure 5.1 – Researcher Registration

Step 1: The Researcher must be a registered user at an Emulab site, e.g., the Utah Emulab site.

Note: Each Emulab site has Projects, Groups and Members.

- There are currently approx 587 Projects at the Utah Emulab site. Each Project has one Project Leader, who can delegate their responsibility.
- Each Project has one or more Groups. Groups within a Project are numbered starting from 0 for the Default Group, then 1, 2, etc. In some cases, a Project corresponds to a class, and then Groups are used to differentiate between class teams.
- Experiments belong to a Group within a Project.
- Members are associated with each Project. Each Member has a Name, UserID and a Role. Roles include: project_root; group_root; local_root_; user.

Question: Must a ProtoGENI Researcher have a particular role in the Emulab DB?

Step 2: The Researcher requests an SSL cert from the Emulab site, using the sites web interface; it is then transferred out-of-band (OOB) to the Researcher's server, for use by the Researcher's Slice Manager

Step 3: The Researcher calls the GetCredential function at the Clearinghouse Regsitry interface, presenting its SSL certificate to allow client authentication; the clearinghouse returns a Self Credential. The owner of this SelfCredential is the Researcher; the target of this SelfCredential is the clearinghouse.

Question: Does this step leave a record with the clearinghouse registry?

Question: What about other ProtoGENI principals?

5.3 Authentication

Each ProtoGENI principal (e.g., Researcher) is authenticated using their SSL certificate. See Section 4.11.

5.4 Privileges and Roles

As indicated in Section 4.13, each ProtoGENI principal (e.g., Researcher) receives a set of privileges that roughly correspond to the allowed API calls, via a credential.

Question: Does the Self Credential hold privileges for the ProtoGENI principal (e.g., Researcher)?

6 Aggregates and Components in ProtoGENI Control Framework

6.1 Identification

Each Aggregate in ProtoGENI has a unique GENI Identifier (GID) that includes a Human Readable Name (HRN) and a UUID.

Question: Who creates the UUID, and names the Aggregate?

Question: What about Components?

6.2 Registration

Each Aggregate must be registered within the ProtoGENI suite, including pointers (URLs) to the Aggregate.

Question: Who does this? How is it done?

6.3 Resource Allocation

By registering an Aggregate in the ProtoGENI suite, the administrator (owner) of the aggregate indicates that they are willing to allocate resources to experiments in the ProtoGENI suite.

However, the information currently contained in the Aggregate Record does not provide any information on the resources that can be utilized in ProtoGENI.

7 Slices in ProtoGENI Control Framework

7.1 Identification

Each Slice in ProtoGENI has a unique GENI Identifier (GID) that includes a Human Readable Name (HRN) and a UUID.

7.2 Registration

In ProtoGENI, a Slice is registered when a Researcher presents a Self Credential (and an SSL certificate) to a Slice Authority (SA) service in an Emulab site, and requests a Slice Credential. See Figure 7-1.

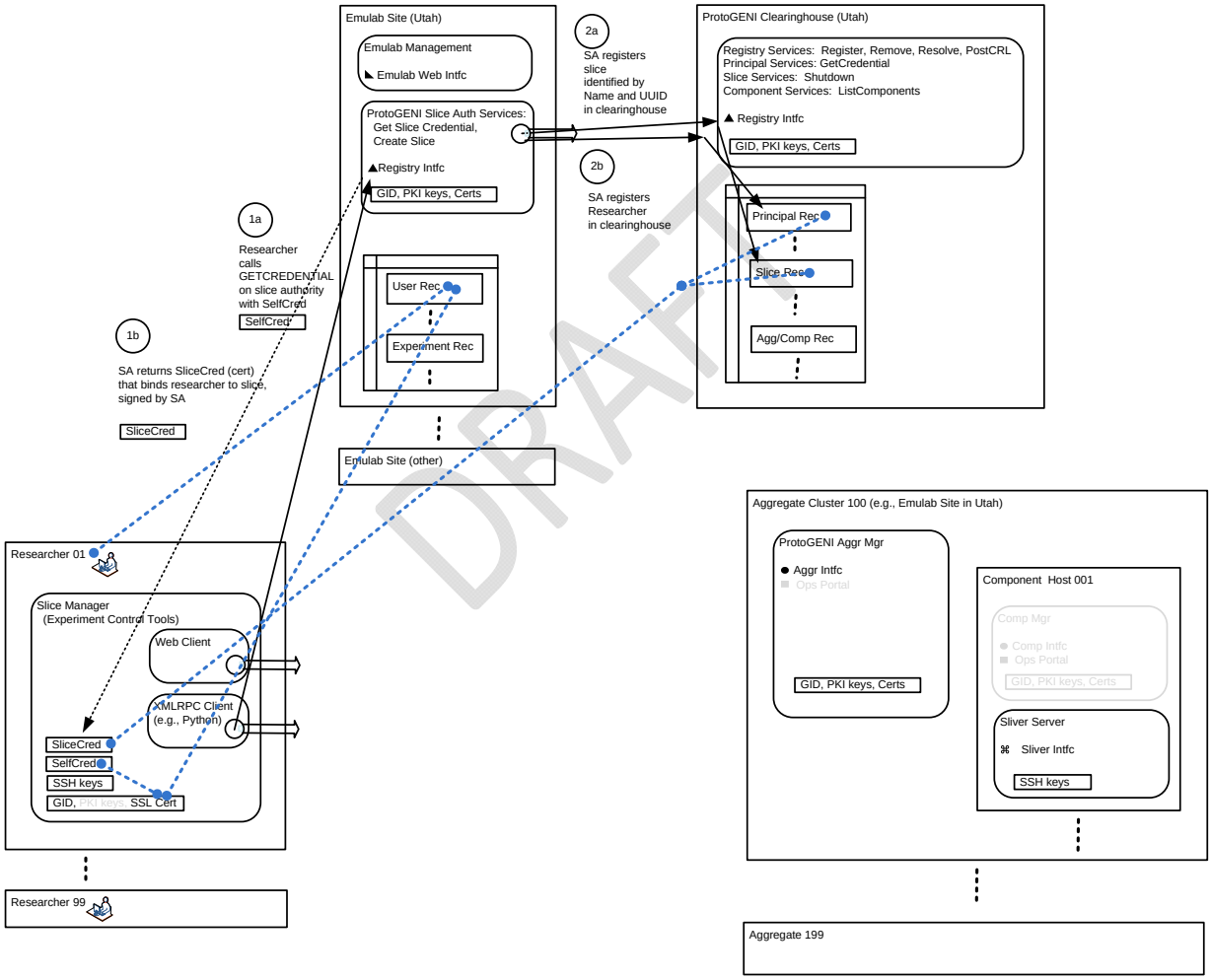

Figure 7-1. Slice Registration and Slice Credential Issue.

Step 1: The Researcher calls the GetCredential function at a Slice Authority (SA) Service Registry interface, and presents its SelfCredential (and an SSL certificate).

See http://www.protogeni.net/trac/protogeni/wiki/SliceAuthorityAPI for more information on the SA API.

Question: How does the SA, based on SSL cert and the SelfCredential, decide that it should issue a slice? Does it always issue a requested slice?

Step 2a: The SA service creates the slice, identified by UUID and HRN, and registers it in the ProtoGENI clearinghouse.

Question: Who creates the UUID and names the Slice?

Step 2b: The SA service registers the Researcher (principal) in the ProtoGENI clearinghouse.

7.3 Credential Issue

Step 1b: The SA service returns a SliceCredential that binds Researcher to Slice, and is signed by the SA.

Note that there is a `BindToSlice` call that binds a local user to a slice so that the user may manipulate the slice. This call can be invoked by any user with a valid slice credential. The newly bound user can then request a slice credential using the GetCredential() above, providing the user with all of the same rights as the original slice creator. Note that this call is provided in lieu of credential delegation, which has not been implemented yet.

8 Experiment Setup in ProtoGENI Control Framework

The ProtoGENI control framework provides all of the basic functions necessary for a GENI researcher to setup an experiment.

8.1 Resource and Topology Discovery

The ProtoGENI control framework allows a Researcher, using the Component Registry, to discover all of the resources available to them from the Aggregates associated with the ProtoGENI suite. See Figure 8-1.

ProtoGENI uses a very simple resource discovery model. There are two phases:

- Phase 1: The Researcher gets a list of all physical resources - whether or not they are currently available. The idea is that this operation does not need to be done often - possibly only once, or possibly weekly, monthly, etc. This information is cached.
- Phase 2: The Researcher gets a list of which resources are currently available. This phase does *not* return details about the physical hardware, simply a "bit" indicating whether each component is available for use or not. This step in context-sensitive: ie. the query needs to be made with the identity of the Researcher gets who will be requesting tickets, as the (aggregate) components might make different policy decisions depending on who is asking.

Because all components are uniquely identifiable with UUIDs, it is easy to combine the results of the second query with the results of the first, and feed this in to the slice embedding service.

The idea is to use optimistic resource reservation, as Emulab does: make the availability query right before running the slice embedder, then attempt to get tickets for the components that were chosen. If any have become unavailable in the meantime, simply hold on to the tickets you were able to grab, and run the slice embedder again to replace the unavailable ones.

Question: How can they discover their interconnection topology?

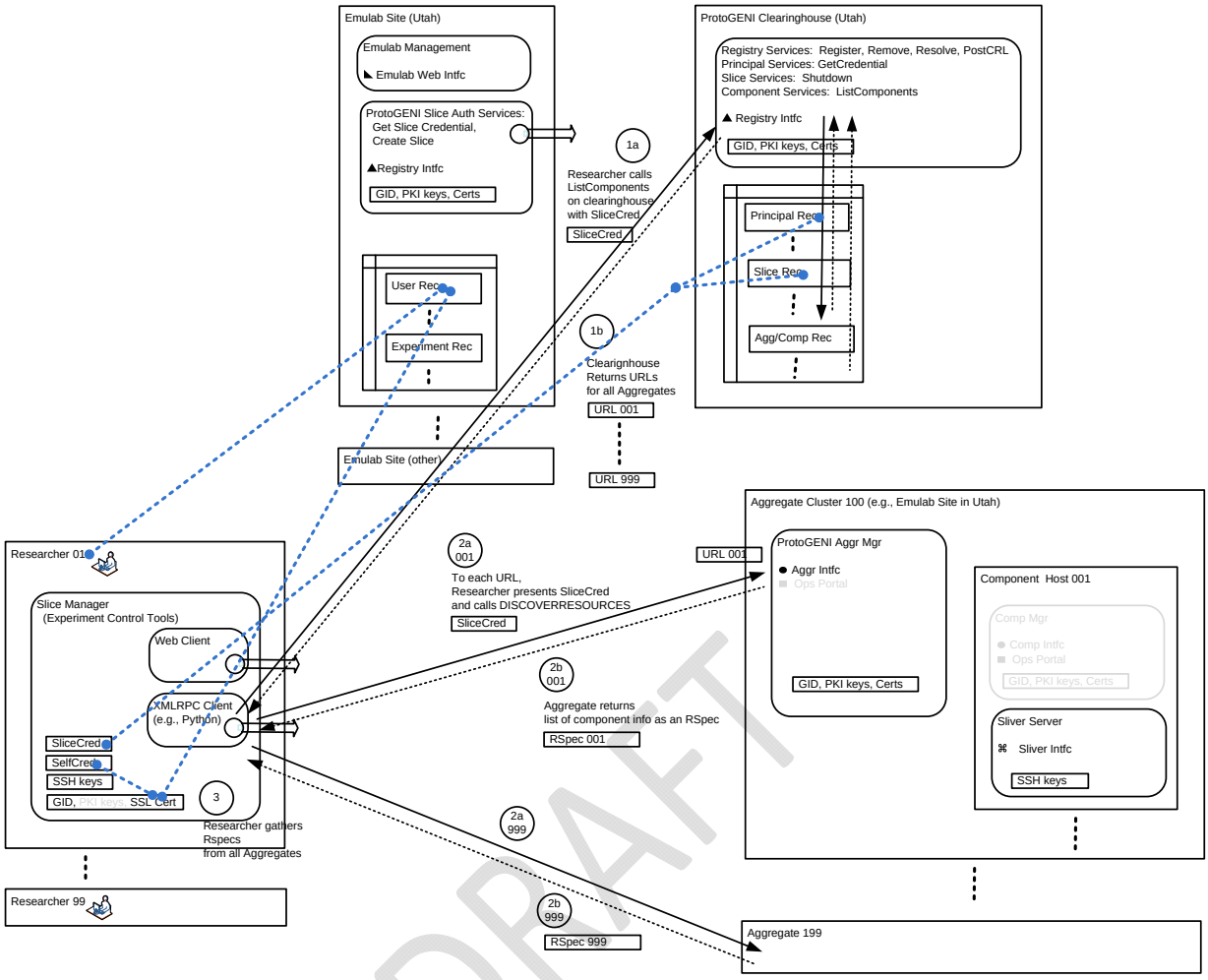

Figure 8-1. Resource Discovery.

Step 1a: Researcher presents SliceCredential to the clearinghouse Registry and calls ListComponents.

Step 1b: The clearinghouse Registry returns a list of arrays, where each array contains the GID (certificate) of the manager. The URL and HRN can be extracted from the GID, but as a convenience the array includes these as well.

Step 2a: To the Component Interface URL on each Aggregate, the Researcher presents ita Slice Cred and calls DiscoverResources.

Step 2b: Each Aggregate returns a list of component info as an RSpec.

Step 3: The Researcher gathers RSpecs from all Aggregates.

8.2 Resource Sharing

In ProtoGENI, each aggregate provides for resource sharing among multiple researchers, referencing multiple slices, and assigns each researcher their own sliver or slivers.

8.3 Resource Authorization and Policy Implementation

The ProtoGENI control framework allows an Aggregate to authorize the assignment of resources to a Researcher referencing a particular Slice, following local policies. See Figure 8-2.

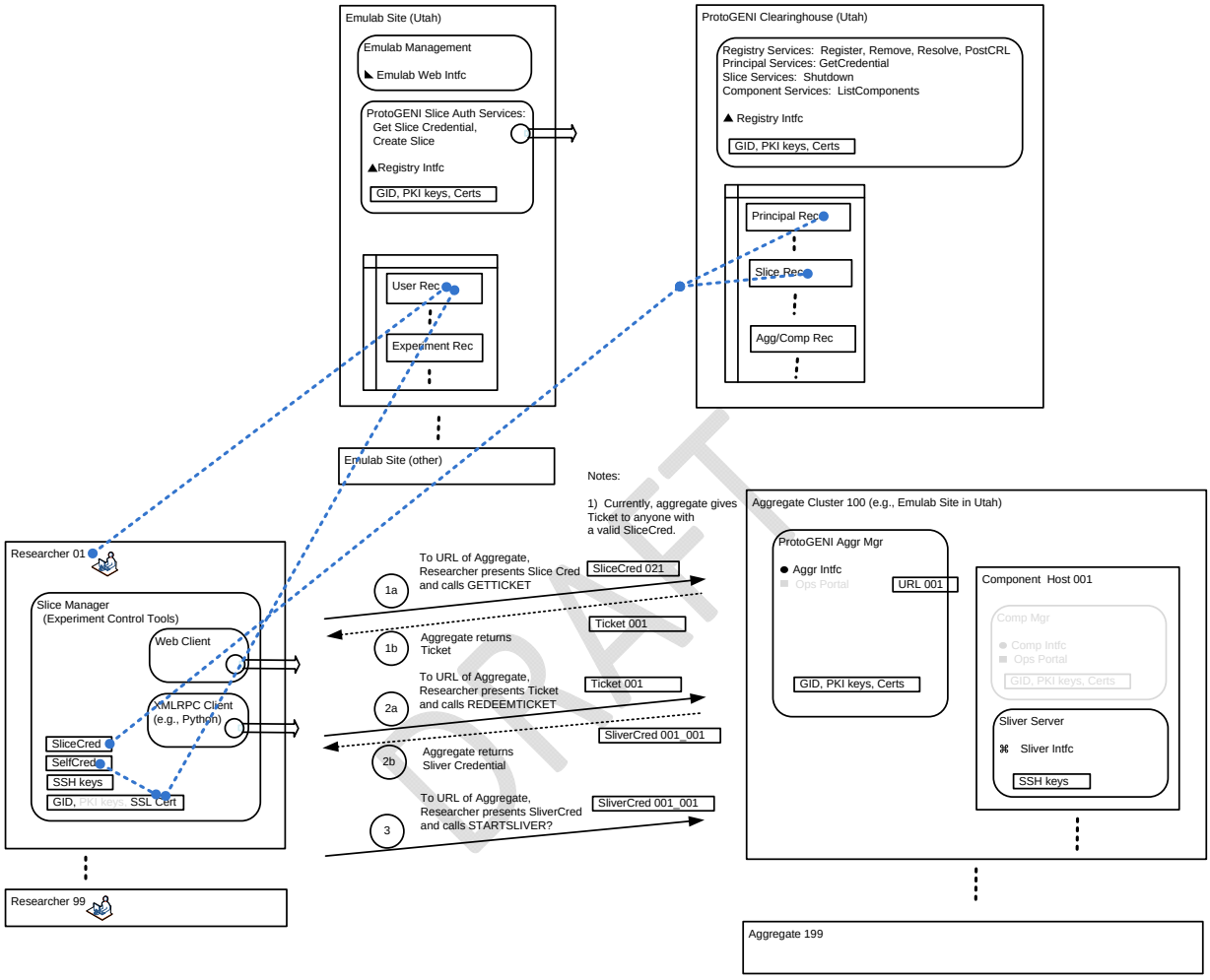

Figure 8-2. Resource Authorization and Assignment, plus Sliver Control

Step 1a: To Component Interface URL of Aggregate, Researcher presents Slice Credential and calls GetTicket function.

Question: Does this call include an RSpec?

Step 1b: Aggregate decides that it will authorize resources to this Researcher and the referenced Slice using its local policy, and then returns a Ticket to the Researcher.

Currently, an Aggregate gives a Ticket to anyone with a valid Slice Credential, i.e., there is an "always yes" local policy.

Currently, the Ticket specifies a starting time of now, and an expires time of "very short".

8.4 Resource Assignment

The ProtoGENI control framework allows an Aggregate to assign resources to a Researcher presenting a valid Ticket. See Figure 8-2.

Step 2a: To Component Interface URL of Aggregate, Researcher presents Ticket and calls RedeemTicket function.

Step 2b: Aggregate assigns resources in a Sliver to the Researcher and their Slice, and then returns a Sliver Credential, that the Researcher can use to control the Sliver. Currently resources are assigned per the Ticket, with a starting time of now.

Note that Slivers in ProtoGENI are first class objects; every sliver is controlled via a credential that is created when the sliver is instantiated. RedeemTicket() and InstantiateSlice() both return a credential to the caller.

While an aggregate is defined for a collection of components, it is not clear what a slice of an aggregate looks like: is it considered an 'aggregate sliver', or several slivers?

One of the key reasons this matters is in regards to what one gets back from a CreateSliver?() call on an aggregate - is it a credential for a single sliver? Is it a set of credentials for all slivers that got created by this call?

The answer we came up with is this: It returns a "handle" on the set of slivers created by the call. If one wants to operate on, or delegate rights to, all of these slivers at once, one can simply use the credential returned by the CreateSliver?() call. However, we will add a new call to the API for an aggregate component manager that, given such a handle, returns credentials for all of the slivers inside.

In this way, the simpleness of the common case (operating on all slivers we got in the aggregate) is preserved, but users can also "look inside" of the slivers they have been given from an aggregate, in a hierarchy that exactly mirrors the original aggregate.

8.5 Component Programming

The ProtoGENI control framework allows a Researcher to login to an assigned sliver (component), load code, and then boot it, etc. This is done via the Sliver Interface; see Section 4.13.

8.6 Disconnected Operation of Components

In a GENI suite, some of the components (such as wireless servers) will require “disconnected operation”, where they are controlled and polled in the short periods of time that they are connected to the suite.

(Note yet defined in ProtoGENI.)

8.7 Disconnected Operation of Researchers

In a GENI suite, some researchers, will connect to the GENI suite to setup an experiment, e.g., by reserving resources for use at a later time, and then will disconnect until they are ready to execute the experiment.

This is supported by the basic operations in ProtGENI.

8.8 Resource to Resource Connections

When a researcher has been assigned resources from two (or more) aggregates that must be connected together, the ProtoGENI control framework provides a way for the researcher to learn about the connection points, request the connections, following the necessary sequence, and receive a verification that the connection has been completed.

This follows the Emulab "virtual network" approach described in http://www.cs.utah.edu/flux/papers/virt-usenix08-base.html .

For example, an RSpec can include a request for a link between two nodes, or a LAN between a set of nodes. ProtoGENI supports independent control of these links and LANs, and even the individual interfaces on nodes attached to the links.

To accomplish this, ProtoGENI treats these sub resources the same as slivers, creating credentials that the caller can use to control them. In addition to the defined sliver operations in the API, ProtoGENI will export additional APIs that are specific to these other resources.

8.9 Setup Verification

When a researcher has been assigned resources on GENI (or federated) aggregates for an experiment, the control framework shall provide a way for the researcher to ask the aggregates to verify the setup before it is time for the experiment to start.

(Note yet defined in ProtoGENI.)

9 Experiment Execution in ProtoGENI Control Framework

9.1 Experiment and Sliver Control

When a Researcher, associated with a designated slice, has been assigned resources on aggregates for an experiment, the ProtoGENI control framework provides a way for the researcher to control the slivers in the aggregates using commands appropriate to the nature of the sliver. For example, start, stop and reboot for a process running on a host. Or, connect, disconnect and loopback for a path in a network. See Figure 8-2.

Step 3: To Component Interface URL of Aggregate, Researcher presents SliverCred and calls StartSliver, or other appropriate command.

All of the available functions are detailed at:

http://www.protogeni.net/trac/protogeni/wiki/ComponentManagerAPI

9.2 Experiment Data Collection and Management

GENI will provide for experiment data collection and measurement, both locally within aggregates (components) and globally in designated measurement services. It is expected that large data files will be gathered by these services, and that they will need to be transferred to a software repository and/or an experiment analysis service after an experiment.

To accomplish this, the control framework should provide the mechanism(s) to allow a researcher to transfer large software records between components, software repositories, etc. For example, the control framework could provide a file transfer service based on ftp.

(Not yet defined in ProtoGENI).

9.3 Forensic and Usage Data Collection and Management

Forensic and usage data from a GENI suite has many uses, including:

- Finding anomalies that indicate errors, faults, malicious activity, etc.
- Allowing help desk functions to be provided to researchers.
- Permitting proper administration and management of suite resources.
- Permitting financial accounting where necessary.

The control framework should provide a structure for collecting and managing forensic and usage data, including formats and log structures.

(Not yet defined in ProtoGENI.)

9.4 Experiment Status Events and Notifications

In a GENI suite:

- The control framework shall provide a structure for defining experiment status events, triggered by the use of resources in an aggregate or component, and ways to delivery notifications of these events to principals or entities.
- It shall be possible for these events to be defined by a researcher and/or by the aggregate or component administrator or operator.

For example, a network gateway may indicate that the following event has occurred: "traffic outbound to the Internet from slice 62 has exceeded it pre-determined threshold".

(Not yet defined in ProtoGENI.)

Question: What about the Emulab per-node watchdog process, with status message to Emulab central? See "monitor health" section in http://www.cs.utah.edu/flux/papers/portal-worlds04-base.html .

9.5 Experiment Status Commands and Responses

In a GENI suite:

- The control framework shall provide a structure for defining experiment status commands, and ways to deliver these commands to an aggregate or component, that responds with a change in the use of resources within the aggregate or component.
- It shall be possible for the responses to be defined by the aggregate or component administrator or operator, or by the researcher.

For example, a command may be sent to an aggregate "to shutdown all slivers in this aggregate associated with slice 62".

In ProtoGENI, a rudimentary form of emergency shutdown has been implemented. Anyone with a slice credential can contact the Slice Authority for the slice, and ask it to do the shutdown operation.:

```
int Shutdown(credential, uuid);
```

where credential is the credential returned by the GetCredential() call above, uuid is the uuid of the slice to be shutdown.

Currently, the Slice Authority calls the Clearinghouse, since only it knows about each Aggregate Manager. Since the Clearinghouse must contact each Aggregate Manager to tell it to shutdown the slice, this call will return immediately. There is currently no facility to find out if/when the shutdown has completed.

10 Federation in ProtoGENI Control Framework

10.1 Federated Aggregates and Components

A ProtoGENI control framework provides for the inclusion of a variety of federated aggregates (and their included components) to provide a wide range of resources to the Researchers.

Starting with a ProtoGENI cluster node and the Utah Emulab cluster node, it will add federated Emulab nodes at Carnegie Melon and Kentucky and thus realize a unified ProtoGENI suite.

Furthermore, the Utah Emulab cluster node already includes federated access to two PlanetLab Central (PLC) installations at Princeton and at Utah. This federation was done by implementing an Emulab to PlanetLab portal; see http://www.cs.utah.edu/flux/papers/portal-worlds04-base.html for more details.

10.2 Federated Suites

The control framework should provide for the federation of a GENI suite with other suites, where each suite has its own complete set of entities, but is independently owned and operated.

(Not yet defined in ProtoGENI.)

11 ProtoGENI Cluster C Spiral 1 Implementation

11.1 Start of Spiral 1

At the beginning of Spiral 1, the ProtoGENI Cluster C implementation includes a ProtoGENI cluster node and the Utah Emulab cluster node. See Figure 11-1.

Note that the Utah Emulab cluster node already includes federated access to two PlanetLab Central (PLC) installations at Princeton and at Utah. This federation was done by implementing an Emulab to PlanetLab portal; see http://www.cs.utah.edu/flux/papers/portal-worlds04-base.html for more details.

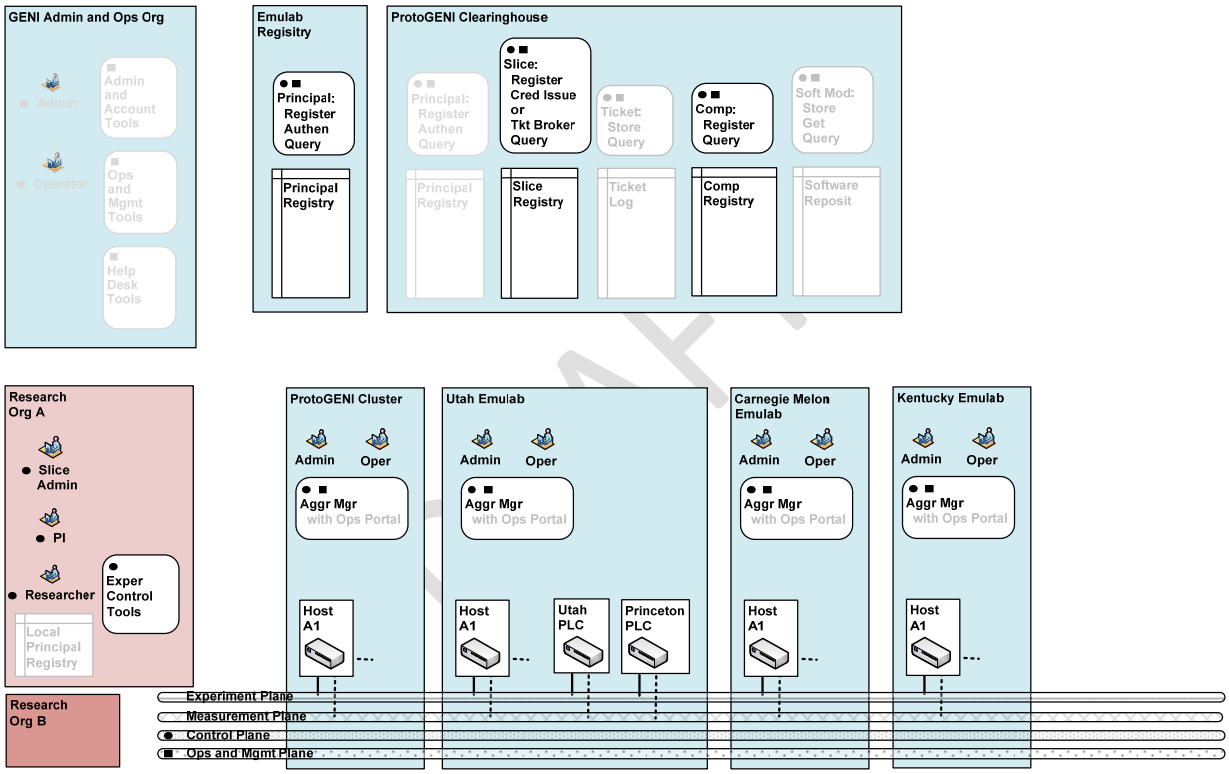

Figure 11-1. Start of ProtoGENI Cluster C Spiral 1 Implementation.

Next, it will add federated access to:

A programmable edge cluster, HomeNet?, and a wireless emulator at Carnegie Mellon University

Netlab at the University of Kentucky

Netlab at Georgia Tech

Schooner, part of WAIL at the University of Wisconsin-Madison

11.2 Completion of Spiral 1

Eventually, the following Cluster C projects will be integrated into the ProtoGENI implementation; see Figure 11-2.

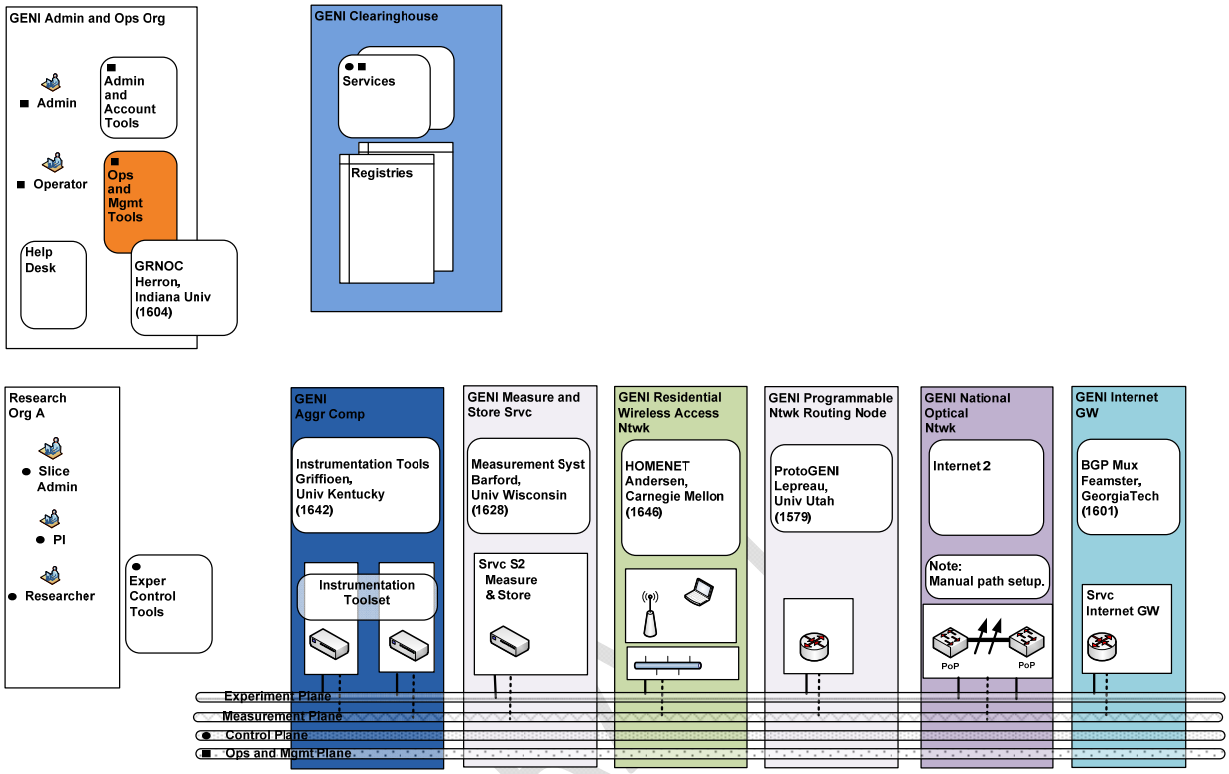

Figure 11-2. ProtoGENI Cluster C Implementation.

University of Kentucky: *Instrumentation Tools*

Tools for collecting and displaying measurements of hosts, such as traffic volumes and host load characteristics. Integration Plans

University of Wisconsin-Madison: *Measurement System*

Prototype of the GIMS measurement framework. A system for the capture and storage of packets, using capture devices separate from GENI components (eg. by employing optical splitters). An important part of this will be providing controlled access to measurements collected, possibly with anonymization. Integration Plans

Georgia Tech: *Virtual Tunnels*

Georgia Tech is a participant in the VINI project, which has as one of its goals improved network virtualization, allowing the creation of tunnels between sliced hosts, and allowing slivers more control over routing, etc. between their interfaces. Integration Plans

Georgia Tech: *BGPMux*

The BGP multiplexer can be used to share BGP feeds among researchers, for the purpose of advertising prefixes (a possible opt-in end user strategy) or getting vantage points into the global BGP topology. Integration Plans

Carnegie Mellon University: *HomeNet?*

HomeNet? will be a deployment of components throughout Pittsburgh on residential broadband (Cable and DSL), and will be equipped with 802.11 interfaces to create an urban wireless mesh. Integration Plans

Carnegie Mellon University: *WirelessEmulator?*

CMU's wireless emulator allows for the creation of controlled, repeatable wireless channel condition, under which real wireless hardware and protocols can be tested. Integration Plans

University of Massachusetts-Lowell: *Programmable Edge Node*

This project will produce a component that is sliced in a lightweight manner using the OpenVZ mechanism in the Linux kernel, and provides a highly-scalable set of virtual interfaces using a network processor. Integration Plans