G E N I

Global Environment for Network Innovations

GENI Control Framework Architecture

Document ID: GENI-ARCH-CP-01.4

October 20, 2008

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 GENI control framework architecture.

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 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.

1.2 Context for this Document

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

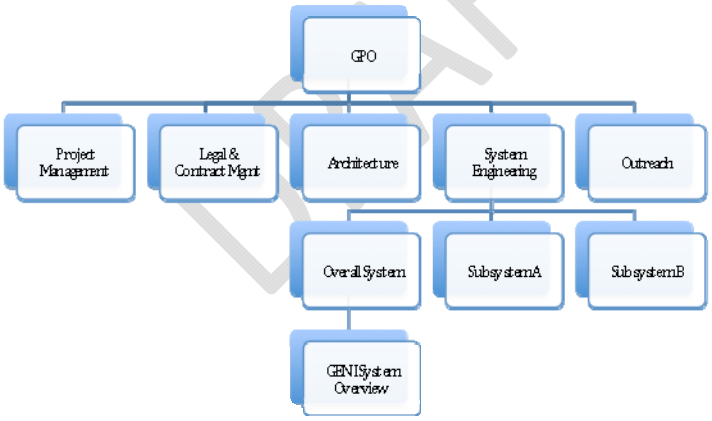

Figure 1-1. This Document within the GENI Document Tree.

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-01.4	GENI System Requirements, September 18, 2008 http://www.geni.net/docs/GENI-SE-SY-RQ-01.7.pdf
GENI-SE-SY-SO-01.5	GENI System Overview, September 19, 2008, http://www.geni.net/docs/GENISysOvrvw092908.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
N/A	"**GMC Specifications**," edited by Ted Faber, *http://www.geni.net/wsdl.php*, Facility Architecture Working Group, September 2006. http://www.geni.net/wsdl.php
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	Thomas Anderson and Michael Reiter, "**GENI Facility Security**," *GENI Design Document 06-23*, Distributed Services Working Group, September 2006. http://www.geni.net/GDD/GDD-06-23.pdf
N/A	Jim Basney, Roy Campbell, Himanshu Khurana, Von Welch, "**Towards Operational Security for GENI**," *GENI Design Document 06-10*, July 2006. http://www.geni.net/GDD/GDD-06-10.pdf
N/A	Grid references: http://www.globus.org/alliance/publications/papers.php#Security%20Components
N/A	A National Scale Authentication Infrastructure, by Ian Foster, et.al http://www.globus.org/alliance/publications/papers/butler.pdf

N/A	Security Grid Services, by Ian Foster, et.al http://www.globus.org/alliance/publications/papers/GT3-Security-HPDC.pdf
N/A	A Multipolicy Authorization Framework for Grid Security, by Ian Foster, et.al http://www.globus.org/alliance/publications/papers/IEEE_NCA_AGC.pdf
N/A	X.509 Proxy Certificates for Dynamic Delegation, by Ian Foster, et.al http://www.globus.org/alliance/publications/papers/pki04-welch-proxy-cert-final.pdf
N/A	Globus references: http://www.globus.org/
N/A	Globus Toolkit Version 4: Software for Service Oriented Systems, 2006, by Ian Foster. http://www.globus.org/alliance/publications/papers/IFIP-2006.pdf
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
N/A	Sharing Networked Resources with Brokered Leases, 2006, by David Irwin, Jeffrey Chase, et.al. http://portal.acm.org/citation.cfm?id=1267377
N/A	ORCA Technical Note: Guests and Guest Controllers, 2008, by Jeff Chase
N/A	PlanetLab references: http://svn.planet-lab.org/
N/A	ProtoGeni references: https://www.protogeni.net/trac/protogeni
N/A	ORCA references: http://nicl.cod.cs.duke.edu/orca/
N/A	ORBIT references: http://www.orbit-lab.org/wiki/WikiStart
N/A	DETER references: http://seer.isi.deterlab.net/

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
	9/25/08	H. Mussman	Started draft, and incorporated material from SFA document.

01.1	10/3/08	H. Mussman	Completed early draft, for limited review.
01.2	10/10/08	H. Mussman	Added updated figures, plus some updates to text.
01.3	10/17/08	H. Mussman	Added and updated figures and text, to reflect comments from reviews of early draft; ready for distribution to WG as a DRAFT for discussion.
01.4	10/20/98	H. Mussman	Added urls for references, and ORCA implementation text.

2 GENI Overview

The Global Environment for Network Innovations (GENI) is a suite of experimental network research infrastructure now being planned and prototyped. GENI prototyping is sponsored by the National Science Foundation to support experimental research in network science and engineering.

As envisioned in these community plans, this suite will support a wide range of network science and engineering experiments such as new protocols and data dissemination techniques running over a substantial fiber-optic infrastructure with next-generation optical switches, novel high-speed routers, city-wide experimental urban radio networks, high-end computational clusters, and sensor grids. All infrastructure are envisioned to be shared among a large number of individual, simultaneous experiments with extensive instrumentation that makes it easy to collect, analyze, and share real measurements.

Core concepts for the suite of GENI infrastructure feature.

- **Programmability** – researchers may download software into GENI-compatible nodes to control how those nodes behave;
- **Virtualization and Other Forms of Resource Sharing** – whenever feasible, nodes implement virtual machines, which allow multiple researchers to simultaneously share the infrastructure; and each experiment runs within its own, isolated slice created end-to-end across the experiment's GENI resources;
- **Federation** – different parts of the GENI suite are owned and/or operated by different organizations, and the NSF portion of the GENI suite forms only a part of the overall "ecosystem"; and
- **Slice-based Experimentation** – GENI experiments will be an interconnected set of reserved resources on platforms in diverse locations. Researchers will remotely discover, reserve, configure, program, debug, operate, manage, and teardown distributed systems established across parts of the GENI suite.

3 GENI System Overview

This section provides a technical system overview of the current state of GENI design, showing all the major parts of the GENI suite and how they fit together. It is abstracted from the GENI System Overview document [http://www.geni.net/docs/GENISysOvrvw092908.pdf].

3.1 Major Entities and their Relationships

Figure 3-1 presents a block diagram of the GENI system covering the major entities within the overall system. Optional but desirable parts of each entity are shown "grayed-out." This section provides a top-level overview of these entities and their relationships. They are defined in a very general way to permit the widest possible range of technologies, usage, and operational models.

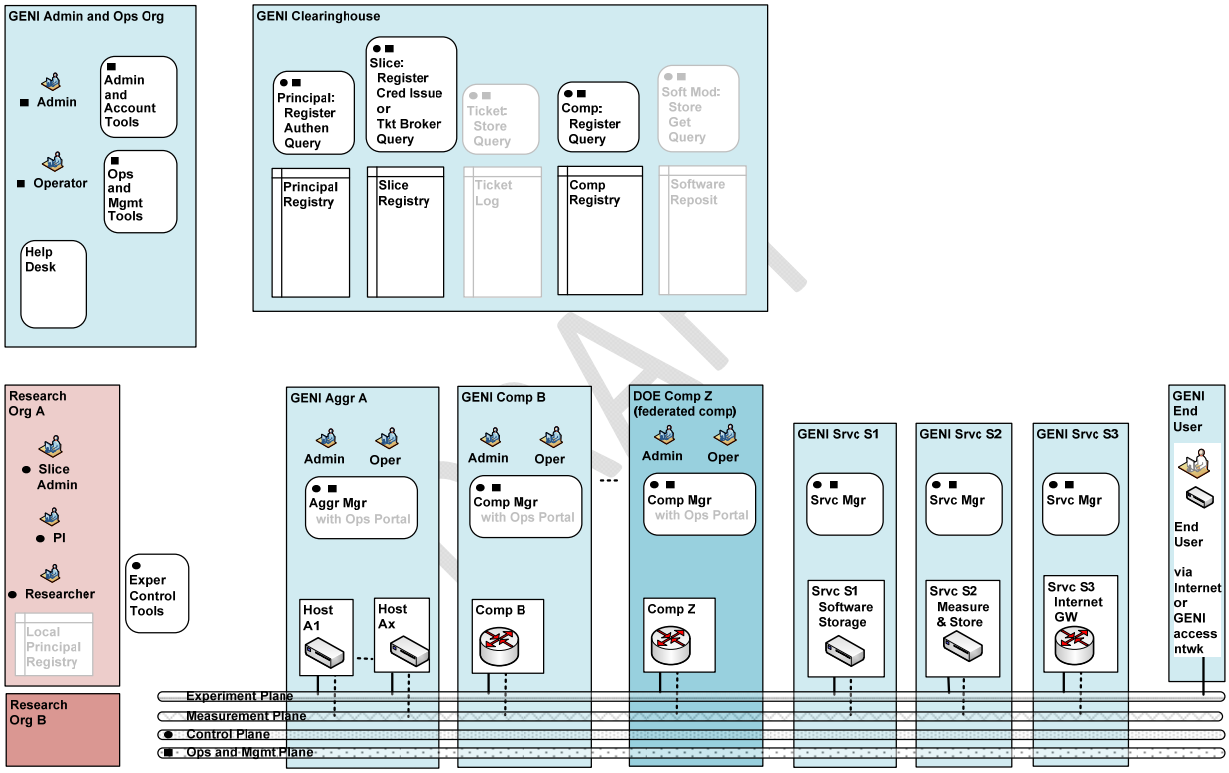

Figure 3-1. GENI Block Diagram.

3.1.1 Clearinghouse

The **clearinghouse** includes **principal, slice, and component registries**, with related services. It is likely that the clearinghouse will maintain registry transaction logs to allow for later troubleshooting and system utilization studies. In addition, the clearinghouse may contain the following optional entities: a **ticket log** and/or a **software repository**, with related services.

The **principal registry** holds a record for each GENI-associated researcher, PI, administrator, operator, etc. (or a pointer to a record in another trusted component registry, such as that of a trusted research organization). Each principal record includes: a global name, contact information, authentication key (or a pointer to it in another trusted registry), roles, and status (active, suspended, etc.). This registry includes services for: principal registration and management; principal

authentication; and related queries. In particular, the principal registry includes records for of all researchers who have permission to establish slices, each vouched for by an associated research organization.

The **slice registry** holds a record for each slice, equivalent to a "bank account" for that slice in that it authorizes access to each "account owner". Each slice record includes: the responsible organization (e.g., the slice administrator) and its permissions; associated principals (e.g., researchers) and their individual permissions; and the slice status (active, suspended, etc.). This registry includes services for: slice registration (creation) and management; issuing slice credentials; brokering tickets; and making slice queries.

The **component registry** holds a record for each affiliated substrate component or aggregate, possibly via a pointer to a record in another trusted component registry. Each record includes: the responsible organization (i.e., management authority); associated principals (e.g., operators) and their individual permissions; the interface(s) to query for available resources (e.g., on the component manager); other contact information; and (optionally) policy to be applied to use of this component or aggregate. This registry includes services for: registration and management; and related queries. Thus, this registry provides records for all components or aggregates that have agreed to participate in experiments that utilize this clearinghouse, entered by the owners/operators of the components. Aggregates may choose to have records for each constituent component.

A **ticket** is a "sliver record" that specifies the resources that a component allocates (or promises to allocate) to a given slice. Depending upon the approach used to obtain a ticket, the clearinghouse contains a credential issuing service or a ticket broker service associated with the slice registry. A **credential issuing service** in the clearinghouse issues a credential (a cryptographically-signed certificate) to a researcher, who then uses the credential to obtain a ticket directly from a component or aggregate. A **ticket broker service** in the clearinghouse (or elsewhere) brokers the initial ticket for a researcher from a component or aggregate and can apply clearinghouse policies to ticket allocation decisions. An example of such a policy might be an upper bound on the duration of tickets for certain classes of users (e.g. undergraduates) on some components.

A **ticket log** holds copies of the initial ticket (sliver record) and subsequent updates, or their equivalents. This could allow administrators and operators to find and manage all slivers related to a particular part of GENI, e.g., to find all slivers associated with a set of slices. A ticket log could provide useful diagnostic information for the control plane (e.g., for troubleshooting slice establishment problems or associating network events with slice activities). Additionally, it could allow administrators to track usage patterns and forecast growth. This log would include a service for queries. Ticket log information might also be useful to researchers and end-users, assuming appropriate security and privacy protection were available.

A clearinghouse can "federate" with another clearinghouse, recognizing the other's slices, principals, and components based on some negotiated policy. Therefore, clearinghouses will have **federation interfaces**. This is discussed further in Section 3.2.

3.1.2 Aggregates and Components

The GENI suite is expected to include many different **aggregates** and **components**.

Researchers use GENI by acquiring **resources** from aggregates and components through the GENI control framework. Resources may be virtualized or real and may be on a single component or require coordination from multiple components within an aggregate.

The **component manager** provides the interface to the control framework, manages resource allocation, and – using internal communications – configures components to provide **slivers**.

When components are organized into aggregates, an **aggregate manager** provides the above functions plus any needed organization of components to provide resources that span multiple components. For example, a network might be treated as an aggregate that provides Ethernet VLANs.

The aggregate/component manager may apply local access control or resource allocation policies.

Many aggregates or components will be "**federated**", i.e., owned and/or operated by different organizations, but nonetheless affiliated with the GENI clearinghouse.

Most aggregates/components provide an **operations portal** to export operational data to GENI O&M. This data, provided in a standardized format, can be used to provide help-desk services, maintain high-level views of system status, and identify network events (e.g., failures or attacks). The portal also permits privileged access by GENI O&M for diagnostic and management purposes (such as requesting shutdown of slivers associated with an out-of-control slice). A privileged access path is typically provided so that an operator can bypass a congestion or failure in the control plane.

3.1.3 Research Organizations, Researchers and Experiment Control Tools

GENI also includes **research organizations** (primarily universities) including **researchers** with associated **experiment control tools**. Optional **local principal registries** in a research organization keep track of that organization's own researchers and their roles (lab director vs. first year grad student), linkages between the organizations researchers and those at other organizations, and so forth. It is these organizations that know whether Researcher X is a graduate student, whether he/she is currently working on a given experiment for Professor Y, or whether he/she has left the organization (for example). Research organizations will typically include the following roles: **slice administrators**, who are responsible for creating a slice record, and authorizing PIs and researchers to utilize the slice to setup and run experiments; **PIs**, who manage the use of a slice by researchers to setup and run an experiment; and **researchers**, who actually setup and run experiments.

Due to the complexity of dealing with multiple components (and other services) to setup and run an experiment, researchers may use **experiment control tools** which act as proxies for researchers. These tools can help with software package acquisition; resource discovery; resource reservation; slice setup and debugging; slice operations; experiment measurements; archiving results; and forensic recordkeeping. When slices are large, tools will be helpful for coordinated scheduling and sliver interaction. Experiment control tools are expected to use slice-specific state (such as slice credentials) and will usually be dedicated to one slice. The research organization, the GENI clearinghouse or a third party may host such tools.

3.1.4 Experiment Support Services

The GENI community is expected to develop a wide range of **experiment support services**. These services might include Software Storage services, for researchers to archive code, configurations and experiment results; Measurement Storage services, to make, gather and archive experiment

measurements; and Visualization Services, to provide ways to view relationships between experiment plane data flows and experiment users.

A research organization, the GENI clearinghouse or a third party may host experiment support services. Services may be implemented with interfaces to substrate components aggregates, or slices. A service might use a dedicated slice of its own to collect, organize, and deliver information for the service. Particular components and aggregates might even incorporate service controllers.

3.1.5 Opt-in End Users

GENI **Opt-in End Users** are those who choose to participate or "opt-in" to a GENI experiment, and become part of the slice. These users may access GENI through the general-purpose Internet, or through a network that supports the GENI natively (for example at a university LAN that is part of a GENI project). These users may have no understanding of GENI, but may be running an application or service that takes advantage of GENI resources. Including real-world users and traffic in GENI is key to providing the fidelity experimenters need in the GENI suite of infrastructures to make their experimental results potentially relevant to real-world networks. The number of opt-in users may easily exceed the number of research users. Unlike research users, the opt-in users may not be individually registered and authenticated in the GENI clearinghouse or aggregates, so experiments will need to provide recording, tracking, and security/privacy functions for their opt-in users when they operate on GENI.

3.1.6 Administration, Operations and Management Functions

Operations and Management are provided by people, tools, and services who administer and operate some or all parts of the GENI system, and in particular its clearinghouse. Operations and Management (O&M) for GENI will be distributed among many organizations and individuals, because of the collaborative nature of the infrastructure suites.

Operations and management tools help operators to manage the overall GENI system and its interfaces to other systems and to troubleshoot, resolve, and record issues in the suite of GENI infrastructures. Depending on the component or aggregate, the operators may be able to access a separate O&M plane or interface to debug, reset, and query parts of the infrastructure in ways that differ from the normal experimental accesses. Operators may also use specialized functions such as emergency shutdown that are not available to the community at large.

Administration tools help administrators to provide routine functions such as: authorizing new research organizations to the GENI infrastructure suite; registering new slices; registering new principals; registering new components; modifying or deleting existing registrations; and reporting on changes in registries. Accounting (for usage) functions may also be needed, particularly if GENI implements clearinghouse policies that require it.

Help desk tools support researchers, PIs, slice administrators, operators in setting up slices, running experiments and reporting and escalating problems.

3.1.7 Glossary

The following table provides somewhat more formal definitions of each of these entities with descriptions of how they inter-relate.

Entity	Explanation
Aggregate	An *aggregate* is an object representing a group of components, where a given component can belong to zero, one, or more aggregates. Aggregates can be hierarchical, meaning that an aggregate can contain either components or other aggregates. Aggregates provide a way for users, developers, or administrators to view a collection of GENI nodes together with some software-defined behavior as a single identifiable unit. Generally aggregates export at least a component interface, i.e., they can be addressed as a component, although aggregates may export other interfaces, as well. Aggregates also may include (controllable) instrumentation and make measurements available. This document makes broad use of aggregates for operations and management. Internally, these aggregates may use any O&M systems they find useful.
Clearinghouse	A *clearinghouse* is a, mostly operational, grouping of a) architectural elements including trust anchors for Management Authorities and Slice Authorities and b) services including user, slice and component registries, a portal for resource discovery, a portal for managing GENI-wide policies, and services needed for operations and management. They are grouped together because it is expected that the GENI project will need to provide this set of capabilities to bootstrap the infrastructure suite and, in general, are not exclusive of other instances of similar functions. For example, there could be many resource discovery services. There will be multiple clearinghouses, which will federate. The GENI project will operate the NSF-sponsored clearinghouse. One application of 'federation' is as the interface between clearinghouses.
Components	*Components* are the primary building block of the architecture. For example, a component might correspond to an edge computer, a customizable router, or a programmable access point. A component encapsulates a collection of resources, including physical resources (e.g., CPU, memory, disk, bandwidth) logical resources (e.g., file descriptors, port numbers), and synthetic resources (e.g., packet forwarding fast paths).
Owners / Management Authorities	GENI includes *owners* of parts of the network substrate, who are therefore responsible for the externally visible behavior of their equipment, and who establish the high-level policies for how their portion of the substrate is utilized. A *management authority* (MA) is responsible for some subset of components, aggregates, or services: providing operational stability for those components, ensuring the components behave according to acceptable use policies, and executing the resource allocation wishes of the component owner. (Note that management authorities potentially conflate owners and operators. In some cases, an MA will correspond to a single organization, in which case the owner and operator are likely the same. In other cases, the owner and operator are distinct, with the owner establishing a "management agreement" with the operator.)

Entity	Explanation
Portals	A *portal* denotes the interface—graphical, programmatic, or both—that defines an "entry point" through which users access GENI. A portal is likely implemented by a combination of services. Different user communities can define portals tailored to the needs of that community, with each portal defining a different model for slice behavior, or support a different experimental modality. For example, one portal might create and schedule slices on behalf of researchers running short-term controlled experiments, while another might acquire resources needed by slices running long-term services. Yet another portal might be tailored for operators that are responsible for keeping GENI components up and running.
Resource	Resources are abstractions of the sharable features of a component that are allocated by a component manager and described by an RSpec. Resources are divided into computation, communication, measurement, and storage. Resources can be contained in a single physical device or distributed across a set of devices, depending on the nature of the component.
Substrate	GENI provides a set of physical facilities (e.g., routers, processors, links, wireless devices), which we refer to as the substrate. The design of this substrate is concerned with ensuring that physical resources, layout, and interconnection topology are sufficient to support GENI's research objectives.

Interface	Description
Measurement Plane	Configuration for measurement infrastructure; management of collected data.
Control Plane	Resource discovery, reservations, and release; slice control (e.g., experiment start and teardown); some debug.
Experiment Plane	Experiment data flow; "in-band" debugging; experiment control.
Operations and Management Plane	Operational status data; privileged slice & component/aggregate control; network event reporting.
Opt-In	Interconnecting GENI to non-GENI networks over, e.g., IP, IP tunnels, conventional (wired or wireless) link protocols. GENI experiments may run just in GENI (e.g., an experimental service accessed by Internet users) or end-users may 'opt-in' to running experimental code on their end-system.

3.2 Federated Suites

Figure 3-2 provides a system diagram illustrating federation between one GENI clearinghouse and another. As a hypothetical example, it depicts federation between a US-based clearinghouse and a compatible framework in the European Union (EU).

The GENI design does not assume that it is federating with identical "GENI-like" systems. Instead it provides a relatively narrow, clearly defined set of interfaces for federation, and can federate with any entity that implements those interfaces. For clarity, however, we show a similar type of system with its own clearinghouse functionality, as well as administration and operations functions, researchers, and aggregates that can be allocated and programmed for experiments.

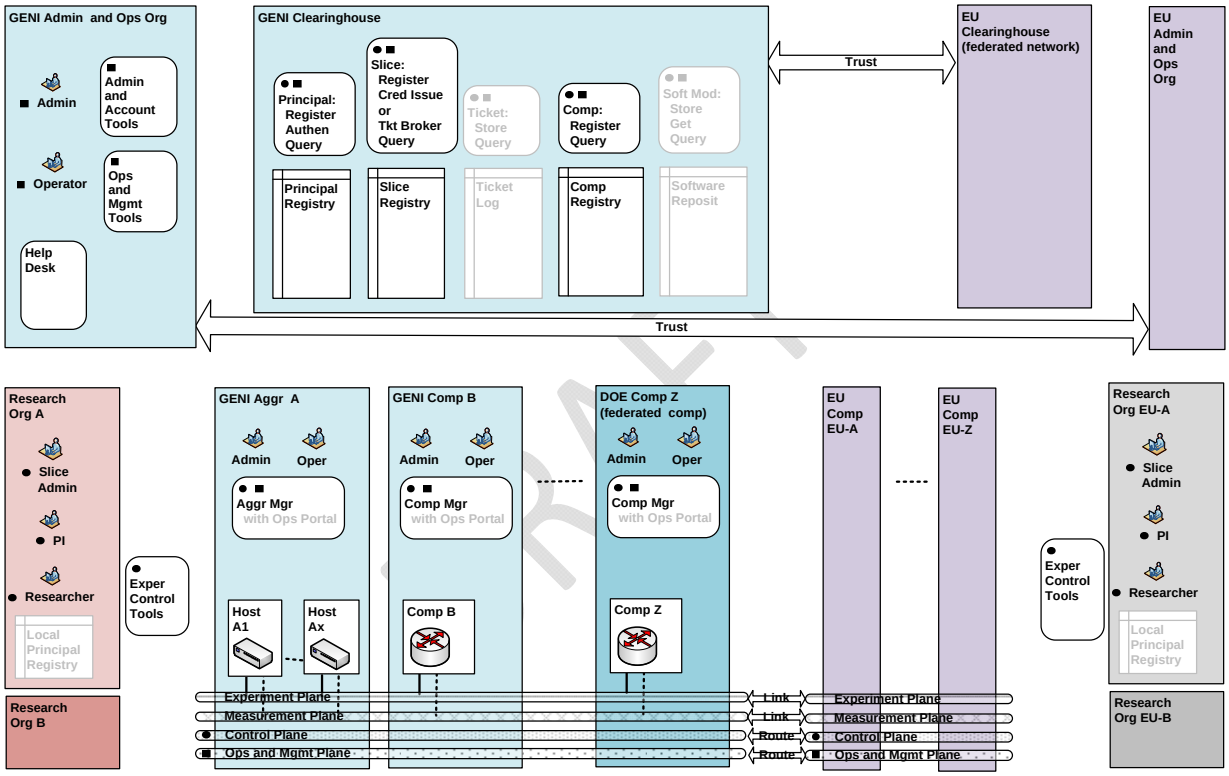

Figure 3-2. System Diagram with Federated Infrastructure Suites.

It is important to point out that several different kinds of interconnections are required for federation – for example, not only must the experiment control frameworks interoperate, but O&M systems must also interwork (even if only at the telephone and email level between human operators). And of course the various interconnection planes must also be compatible between at least some of the aggregates on either side.

Federation	Resource *federation* permits the interconnection of independently owned and autonomously administered facilities in a way that permits owners to declare resource allocation and usage policies for substrate facilities under their control, operators to manage the network substrate, and researchers to create and populate slices, allocate resources to them, and run experiment-specific software in them.

3.3 Slices

Figure 3-3 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.

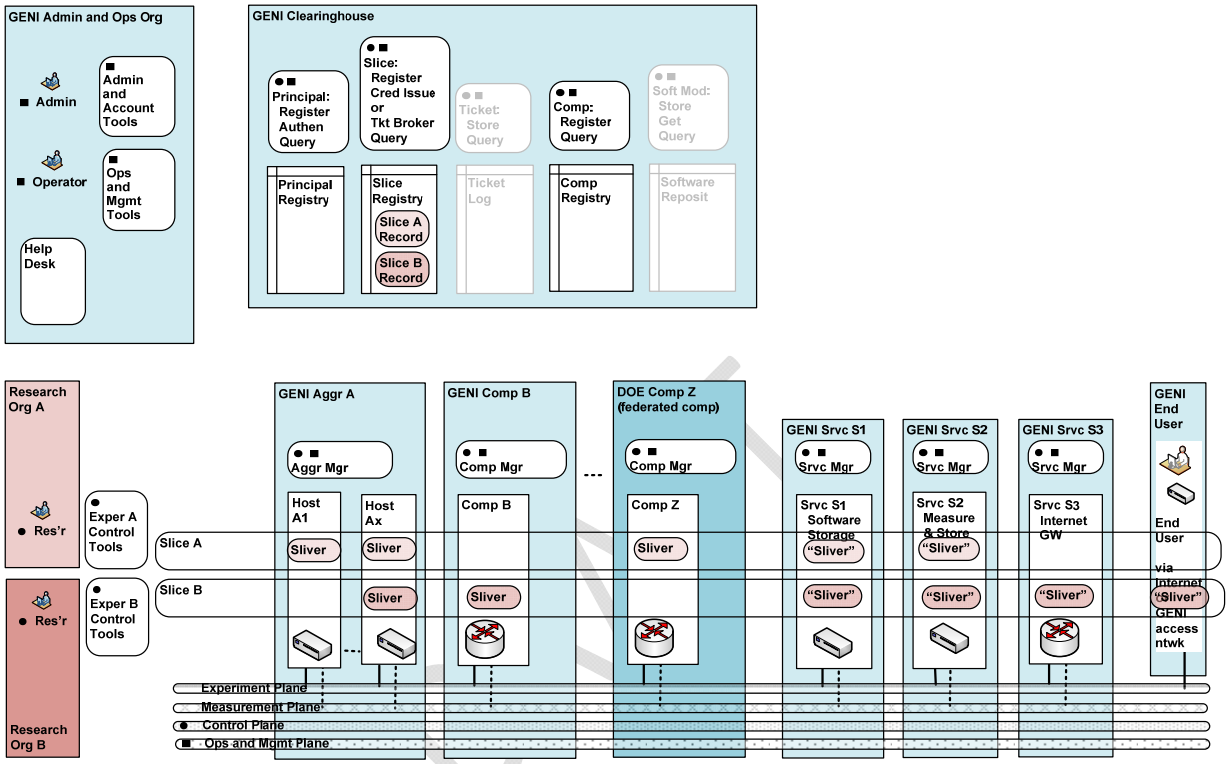

Figure 1-3. Two GENI Slices in a System Diagram.

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.

Experiment	An experiment is a researcher-defined use of a slice; we say an experiment runs in a slice, or in multiple slices since slices can be composed or interconnected. Experiments are not slices. Many different experiments can run in a particular slice concurrently or over time.

Sharing	Wherever possible, GENI components should support multiple concurrent experiments. We refer to this as making components and aggregates *sharable* (or sometimes "sliceable"). Different strategies may be needed to share components based on the nature of the technologies. This can be done by a combination of virtualizing the component (where each user acquires a virtual copy of the component's resources), or by partitioning the component into distinct resource sets (where each user acquires a distinct partition of the component's resources).
Slices	From a researcher's perspective, a *slice* is a substrate-wide network of computing and communication resources capable of running one or more experiments or a wide-area network service. From an administrator'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. A slice is defined by a set of slivers spanning a set of network components, plus an associated set of users that are allowed to access those slivers for the purpose of running an experiment on the substrate. That is, a slice has a name, which is bound to a set of users associated with the slice and a (possibly empty) set of slivers.
Slivers	It must be possible to share component resources among multiple users. This can be done by a combination of virtualizing the component (where each user acquires a virtual copy of the component's resources), or by partitioning the component into distinct resource sets (where each user acquires a distinct partition of the component's resources). In both cases, we say the user is granted a *sliver* of the component. Each component must include hardware or software mechanisms that isolate slivers from each other, making it appropriate to view a sliver as a "resource container."
User Opt-In	An important feature of GENI is to permit experiments to have access to end-user traffic and behaviors. For examples, end users may access an experimental service, use experimental access technologies, or allow experimental code to run on their computer or handset. GENI will provide tools to allow users to learn about an experiment's risks and to make an explicit choice ("opt-in") to participate.

3.4 Key Operational Scenarios

The following sections describe, in very high-level form, how the entities shown above interact in a number of key operations that will be performed on the infrastructure suite:

- Setting up a GENI suite
- Running an experiment
- Recovering from crashes
- Operating and managing a GENI suite

3.4.1 Setting up a GENI Suite

First, a *clearinghouse is established*. It is a set of high-availability software services managed by an operational staff.

Second, one or more *aggregates register* with the clearinghouse. This is a trust relationship – they must be certain that the clearinghouse is who it claims to be, and will behave in a responsible fashion, and the clearinghouse must have similar faith in the aggregate's operators. Since this forms a chain of trust upon which GENI will rely, some form of mutual authentication will be used. (For example, aggregates might include managed systems of computer clusters, regional optical networks, or metro wireless networks.)

Third, the registered *aggregates publish their resource information* to the clearinghouse. The exact information is currently undefined, but probably includes lists of resources and up-to-date schedules for resource availability. This information will change continually, particularly if an aggregate belongs to multiple clearinghouses, so the aggregate is responsible for keeping its clearinghouse(s) up to date.

Fourth, *research organizations register* with the clearinghouse. Again, this is a trust relationship – they must be certain that the clearinghouse is who it claims to be, and will behave in a responsible fashion, and the clearinghouse must have similar faith in the research organization. (For example, research organizations might include university computer science departments.)

Fifth, *researchers register with research organizations* on the basis of existing or planned experiments. In essence, the research organization vouches that a particular experiment is indeed being planned or conducted, and that this particular researcher is authorized to manipulate the slice that contains that experiment. Note that the clearinghouse itself does not need to authorize individual researchers: that function is carried out by the research organizations. (For example, a graduate student at University X might register with University Y's research organization to join a collaborative experiment run by a principal investigator at Y.)

Sixth, the *NetSE Council establishes policy* about who may access which resources, and under which constraints. This policy is codified into a rule set, and instantiated within the clearinghouse, where it governs all subsequent resource requests.

Seventh, the *clearinghouse federates with other clearinghouses*. Again this is a trust relationship, although its details are unclear at present.

At the end of this stage:

- The clearinghouse contains linkages to its trusted aggregates and trusted research organizations, NetSE Council-specified policy for resource allocation, and linkages to federated clearinghouses.
- The aggregate contains linkages to one or more trusted clearinghouses, and is periodically publishing up-to-date views and schedules for its resources to these clearinghouses.

- The research organization contains linkages to one or more trusted clearinghouses, and lists of authorized researchers for various experiments.

3.4.2 Running an Experiment

Now that the infrastructure suite is up and running, a researcher may perform an experiment. The steps below review the basic steps in this process, and are summarized in Figure 1-4.

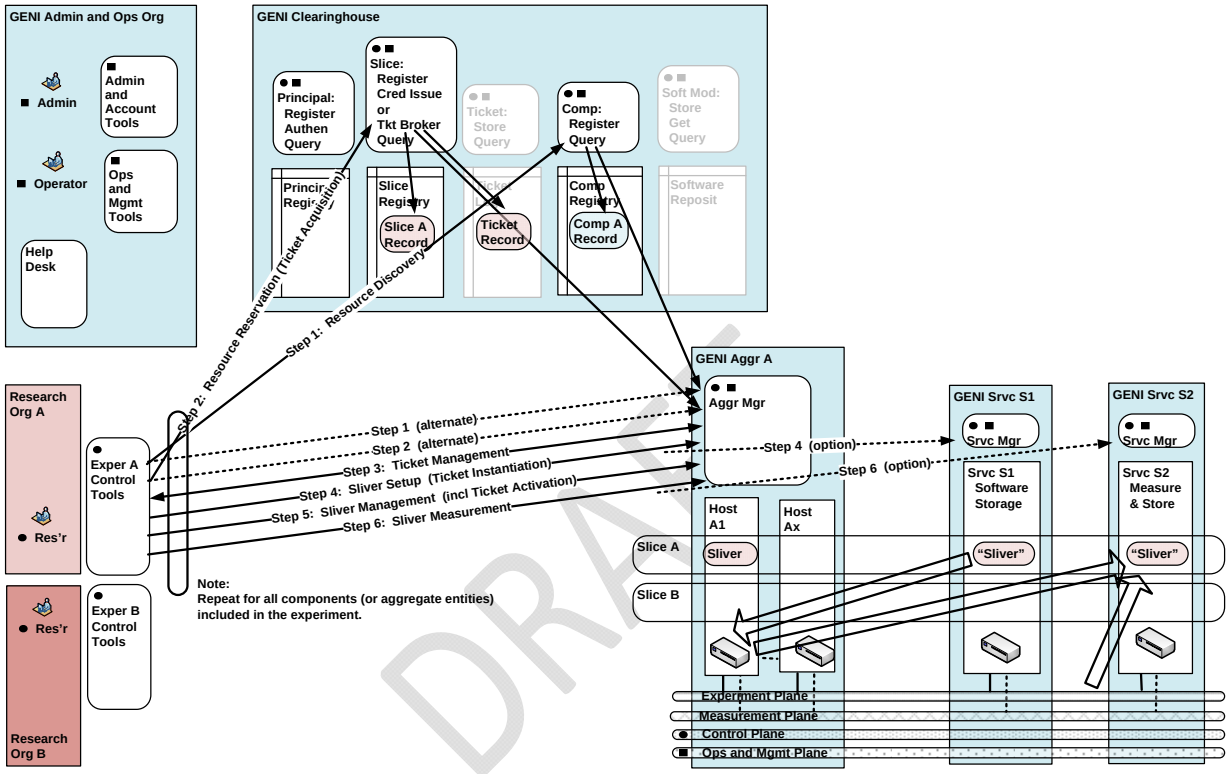

Figure 1-4. Steps to Setup and Run an Experiment

First, the researcher *acquires credentials* from her research organization. (The research organization must be registered with a GENI clearinghouse in order for these credentials to be useful.)

Second, the researcher sends her credentials to a GENI clearinghouse and requests a slice identifier from the clearinghouse. The clearinghouse validates that this research organization is indeed trusted and provides the researcher with a *globally unique slice identifier*.

Third, the researcher queries the clearinghouse (or other portal) for available resources. The clearinghouse *provides her with resource information*, including current views and projected schedules. This information comes from the lists provided by the clearinghouse's registered aggregates, possibly filtered (according to policy rules) to restrict what she is allowed to see. (Note that restricting resource visibility at the discovery stage is likely to be largely a convenience (to prevent users from asking for resources they won't be permitted to obtain) as the policy enforcement is expected to take place at the

aggregate/component level.) Included in this information are high-level resource descriptions and contact points (e.g., aggregate managers) for making reservations.

Fourth, the researcher contacts each aggregate manager for a resource of interest with detailed queries about available resources. She presents credentials issued by the clearinghouse that allow her to request resources. The aggregate manager may apply locally defined policy based on her credentials (or other parameters) that will constrain the types and amount of resources the researcher can obtain. The aggregate manager responds with RSpec *describing available resources*.

Fifth, the researcher, perhaps using helper tools, makes a *resource reservation* contacting the credential issue service in the clearinghouse to acquire a "signed slice credential" (Certificate). This is then used to get a ticket from the manager of the aggregate/component. The researcher submits a description of the resources desired based on the advertised RSpec, the start time and duration of the reservation, and the Slice ID the resources are to be bound to. The manager can apply its policy and decide whether to issue the ticket or not, considering the requesting researcher and slice.

Alternately, the researcher may use a ticket broker service in the clearinghouse before contacting the aggregate manager). The ticket broker service can apply clearinghouse policy, i.e., whether the researcher gets the requested resource, considering both the requesting researcher and slice and chosen aggregate/component resource. The aggregate manager can then apply its own policy, considering the requesting researcher and slice. The ticket broker service may drop a ticket record in the ticket log, to be used later in forensic searches.

The aggregate manager then marks these resources as booked for that period, and publishes an updated schedule to their clearinghouses.

Sixth, when it is time *setup the sliver*, i.e., ticket instantiation, the researcher sends the ticket back to the aggregate manager and the resources are made available. If topologies need to be created within the aggregate, or other composite forms of action performed (such as starting measurement devices), they occur at this time.

Seventh, the researcher *downloads software images* into her resources, and starts them running. She then debugs them by mechanisms TBD, and collects measurements by mechanisms also TBD.

Eighth, the researcher may choose to *make changes to the resources used* for additional experiment runs. If subsequent *ticket management* (revisions and status) for an existing sliver is needed, the researcher goes directly to aggregate manager to make changes. Alternatively, the aggregate may need to contact the researcher to inform her of component-driven changes to the reservation. For example, if there is a failure.

Ninth, the slice is torn down and its *resources freed*. There can be many triggers for this action, including researcher request, expiration of the slice lifetime, revocation of a researcher's credentials, management decision, etc. This action is coordinated by the clearinghouse, which instructs each aggregate to free up the associated resources, and then records the slice as "ended" in its log files.

3.4.3 Recovering from Crashes

Because GENI is a distributed system, it is possible that some parts of GENI will crash and restart while others will keep running. Loss of state synchronization can be an issue in such cases. GENI should avoid such problems, for example by defining a single, authoritative source for each type of shared, distributed state within the overall system.

Here we consider several scenarios; as design progresses, a thorough analysis will need to be performed. We assume that each important service is provided by replicated software / hardware, but

still wish to assure ourselves that GENI will operate correctly through complete failure of such replicated services.

- If a clearinghouse crashes, it must preserve its trust relationships on stable storage; they can be restored in a straightforward manner after it restarts. Similarly if it is the authoritative source of slice information, it must be very careful to preserve it on stable storage; an alternative scheme might be to tag this information with revision numbers and distribute it more widely through the system. It will relearn current resource availability from the ongoing (soft state) publications from its registered aggregates. [Don't know about federation information.]
- If an aggregate manager crashes, it must preserve its trust relationships on stable storage; they can be restored in a straightforward manner after it restarts. It should also store schedules for future resource booking on stable storage so they can be retried in case of restart. It seems safest to reset all its resources to a known good state, though this might not be required; if it does so, this will remove the running code for all experiments in all slices within this aggregate. When the manager is up and running, it must publish its latest views of resources and schedules to the clearinghouses with which it has registered.
- If research organization crashes, it must preserve its trust relationships on stable storage; they can be restored in a straightforward manner after it restarts. It must also store lists of authorized users on stable storage. [There seems to be a timing window here, if a researcher is revoked but the system crashes before anyone can tell the clearinghouse; it might be that her experiments will keep running although they should be shut down….]

3.4.4 Operating and Managing a GENI Suite

GENI Operations and Management (O&M) is a system-wide function that keeps GENI resources operating and manages GENI services. Many entities have needs of the O&M functions such as: researchers, the GENI Clearinghouse (which supports external interfaces to GENI O&M), other GENI-federated clearinghouses, university and industry network management (IT) organizations that support GENI users or GENI traffic, Internet Service Providers, organizations that provide aggregates of resources to GENI, and GENI policy makers, such as the National Science Foundation.

Trust relationships are required between GENI Ops and any cooperating or contracted organizations in order to maintain the integrity of GENI O&M functions. They also imply standard procedures and interfaces exist for any data or control signals exchanged between GENI Ops and the other organizations. Because very few O&M interfaces have yet been standardized in network engineering, GENI O&M will likely have to support many different types of procedures and interfaces, at least initially (as do today's Internet operators). Costs and risks can be reduced significantly if the GENI project advances standards and tools for O&M data exchange and joint management of resources.

GENI O&M will include many of the same functions performed on existing research networks: monitoring, event management, data archiving, managing planned and unplanned outages, network engineering and peering, and protecting GENI from external and internal threats. Because GENI includes slices, rather than individual resources or researchers as a fundamental managed unit, this affects all the "standard" functions to some degree, and also requires additional unique GENI functions (for example, mapping resources to slices and slices to researchers). Because a single GENI experiment can span the scope of control of several different O&M organizations, as well as several clearinghouses, aggregates, and networks that are not part of GENI, new O&M tools and procedures

may need to be developed. The emergency shutdown function that operates on slices, and may require cooperation among several clearinghouses and aggregates to work successfully, is another example of a new GENI function. Policy implementation, which is complex even in networks owned by a single entity, will need careful definition and implementation to accommodate policies from multiple clearinghouses, and GENI's own policy bodies.

4 GENI Control Framework Architecture Overview

4.1 Basic Functions

The GENI Control Framework Architecture (CFA) must mediate the following activities:

- Allow operators to declare resource allocation and usage policies for substrate facilities under their control, and to provide mechanisms for enforcing those policies. The assumption is that there will be multiple owners and it will be a *federation* of these facilities that will form the entirety of the network.
- Allow operators to manage the network substrate, which includes installing new physical plant and retiring old or faulty plant, installing and updating system software, and monitoring the network for performance, functionality, and security. Management is likely to be decentralized: there will be more than one organization administering a disjoint collections of sites.
- Allow PIs to identify the set of researchers at their organization that be permitted to utilize the facility.
- Allow researchers to create and populate slices, allocate resources to them, and run experiment-specific software in them. Some of this functionality, such as the convenient installation of software, including libraries or language runtimes, may be provided by higher-level services.

4.2 Requirements and Goals

The CFA must provide the capabilities to allow the GENI suite to provide these core features:

- **Programmability** – researchers may download software into GENI-compatible nodes to control how those nodes behave;
- **Virtualization and Other Forms of Resource Sharing** – whenever feasible, nodes implement virtual machines, which allow multiple researchers to simultaneously share the infrastructure; and each experiment runs within its own, isolated slice created end-to-end across the experiment's GENI resources;
- **Federation** – different parts of the GENI suite are owned and/or operated by different organizations, and the NSF portion of the GENI suite forms only a part of the overall "ecosystem"; and
- **Slice-based Experimentation** – GENI experiments will be an interconnected set of reserved resources on platforms in diverse locations. Researchers will remotely discover, reserve, configure, program, debug, operate, manage, and teardown distributed systems established across parts of the GENI suite.

Gap: Other derived requirements and goals are TBD for the CFA.

4.3 Engineering Approach

To achieve these goals, the GENI project uses an engineering approach informed by the success of the Internet and the open source software movement:

- Start with a well crafted system architecture
- Build only what you know how to build
- Build incrementally
- Design open protocols and software
- Leverage existing technology

This approach is reflected in the CFA presented in the following sections. The current version of this document represents a first version of the CFA, documented here at the start of the Spiral 1 prototyping effort.

This first version of the CFA is based on GENI control framework architecture work to date, including:

- The early GENI planning effort, where the control framework plan is summarized in [http://www.geni.net/GDD/GDD-06-11.pdf].
- A web-services based approach, outlined in [http://www.geni.net/wsdl.php], edited by Faber at USC/ISI.
- The "Slice-based Facility Architecture" (SFA), a proposed architecture that provides GENI core features in a flexible manner, by a group of researchers working on the Spiral 1 control framework implementations based on PlanetLab, ProtoGENI (Emulab) and DETER; see [http://groups.geni.net/geni/attachment/wiki/GeniControlBr/v1.10%20%20080808%20%20sfa.pdf], edited by Larry Peterson.
- The GENI System Overview, as summarized in [http://www.geni.net/docs/GENISysOvrvw092908.pdf], prepared by the GPO.

The intent of this first version of the CFA is to:

- Separate the CFA into component parts, and address each part in some detail.
- Indicate the architectural options for each part, and the current choice(s).
- Indicate the features and/or parts that are not yet complete, and what needs to be done to complete them.
- Provide one (or more) "worked examples" to fully illustrate the architecture. In particular, various worked examples are included from the SFA, but they are not intended to be definitive.
- Provide a structure to summarize, compare and evaluate the five control framework designs being developed for Spiral 1 prototypes.

Once the five control framework designs being developed for Spiral 1 are underway, their experiences with implementation and operation can be used to refine the CFA, and this will lead to a second version of the CFA.

4.4 Implementation Approach for Spiral 1 Prototypes

Five control framework implementations are being developed for Spiral 1, 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 project [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 project [ProtoGENI].
- ORCA resource allocation software; see [http://nicl.cod.cs.duke.edu/orca/] and project [ORCA].
- 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 project [ORBIT].
- 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 project [DETER].

See Sections 12 though 16, respectively, for current descriptions of each of the five control framework implementations.

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.

4.5 Federated Aggregates

A GENI infrastructure suite must include a wide variety of federated aggregates (and their included components) to provide a wide range of resources to the Researchers, and thus help meet the core GENI concept of federation. The federated aggregates are expected to be owned and operated by a wide variety of organizations, who have agreed to provide resources to the GENI infrastructure suite, and have them be allocated via the CFA. It is assumed that the federated aggregates can be used like any "native GENI" aggregates, but subject to the local policies implemented by their owners.

To federate an aggregate with the GENI suite requires its owner to:

- Connect its aggregate with other GENI aggregates via the data plane. Currently, this could include IP connectivity via the Internet or a dedicated virtual private network, and/or Ethernet (VLAN) connectivity via a combination of access and backbone networks.

- Establish an aggregate manager for interfacing with the control and ops&mgmt planes of the GENI suite. Currently, the control and ops&mgmt planes are expected to carried via a combination of the Internet and dedicated virtual private networks.
- Connect its aggregate with other GENI aggregates via the measurement plane in a manner TBD.
- Register their aggregate with the component registry in the GENI suite.
- Accept requests for and provide resources to GENI Researchers with appropriate credentials via the aggregate manager, following the CFA.

The aggregate manager provides the key bridge between the control plane of the GENI suite and the federated aggregate by:

- Interfacing with the control and ops&mgmt planes of the GENI suite, following the CFA.
- Interfacing with the local aggregate following its supported control and ops&mgmt interfaces.
- Providing a security dividing line between the GENI suite and the local aggregate, and establishing the necessary trust relationships.
- Providing resources from the local aggregate to GENI Researchers with appropriate credentials, following the CFA, and following locally established policies.
- Providing forensic and accounting information that can be accessed and utilized by both the GENI and local services that provide administration, operations and management.
- Providing for monitoring of events associated with the use of resources in the local aggregate, that can be used to trigger actions, including interventions by operations services.

The current CFA does address federated aggregates.

Federated aggregates are fully supported in the five control framework designs being developed for Spiral 1. Each subtending project in a cluster typically provides a federated aggregate to the project that provides the clearinghouse.

4.6 Federated Suites

It is also expected that the GENI infrastructure suite will federate with similar suites, where each suite has its own complete set of entities, including clearinghouse, aggregates and research organizations, but is independently owned and operated. A federated suite may or may not utilize the same control framework architecture as the GENI suite.

Federating two (or more) federated suites involves:

- Establishing a way to provide for global identities and names among all entities, principals, and services.
- Establishing trust relationships between the clearinghouses, and understanding how resource authorizations and policy mechanisms will be established for the federated environment.
- Allowing Researchers associated with either suite to utilize resources in either suite.

In the general case, federating suites is likely to be a complicated and difficult task, particularly if they utilize different control framework architectures.

Gap: The current CFA does not fully address the general case of federated suites.

However, it should be possible to more towards this in steps followwing this approach:

- Start with suites that utilize the same CFA.
- Establish aggregates in both suites that can operate both locally and are federated into the other suite.
- Establish trust relationships to allow Researchers associated with slices in one suite, to be recognized in the other suite and given resources.

Two of the control framework designs being developed for Spiral 1 (those based on ORBIT and DETER) are focused on working towards providing federated suites.

4.7 GENI Security Architecture

The CFA is highly dependent on a GENI Security Architecture (SA).

Early work on security is summarized in [http://www.geni.net/GDD/GDD-06-23.pdf] and [http://www.geni.net/GDD/GDD-06-10.pdf], and more work is being done during the Spiral 1 implementation period; see project [Security].

To be able to proceed with the CFA, the various security techniques are shown here, and the current choices are indicated. **Note: These will have to be reconsidered, and possibly changed, when the GENI SA is finally formulated.**

4.8 Resource Authorization and Policy Mechanisms using Trust Management

A principal function of the GENI control framework is to allow the authorization and assignment of resources from many GENI or federated aggregates to GENI Researchers following established policies. This involves the interaction of a variety of elements:

- The researcher
- The designated slice, where the 'slice record" can be considered analogous to a bank account
- The aggregate, including its resource availability and local policies.
- Policies that can be associated with another entity, such as the GENI clearinghouse or an intermediate broker.

Policies can be based on a variety of parameters, including:

- Researcher lineage and status
- Slice lineage and status
- Presence of electronic currency, i.e., "GENI tokens"
- Resource availability

To account for all of these elements and policies, a capable "trust management" system is required, that goes far beyond a simple access control list. See [http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.44.6276] for a general overview of trust

management systems, and [http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.7726] for the application of example systems to distributed systems.

In all cases, a decision to grant a resource (or not) is made regarding a request from a researcher to an aggregate.

In a simple case, supported by the current CFA, an aggregate can check the slice lineage of a request against a local list of supported slices.

However, the CFA should support a richness in resource allocation and policy mechanisms. In particular, there should be a way to include policies that are associated with a clearinghouse or an intermediate broker.

It is expected that various resource allocation and policy mechanisms will be explored in Spiral 1 implementations.

4.9 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. The final CFA must permit disconnected operation for such components.

4.10 Forensic and Accounting Information

It is important that the CFA provide a structure for gathering forensic and accounting information that can accessed and utilized by GENI (and other) services that provide administration, operations and management. This involves:

- Identifying the information and formats for records that need to be saved.
- Establishing log structures for these records, and functions to access these structures.

Forensic and accounting information has many uses, including:

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

Gap: The current CFA does not fully address gathering forensic and accounting information, and more work should be done in this important area as the CFA is firmed up.

It is not known what mechanisms for gathering forensic and accounting information are included in any of the Spiral 1 implementations.

4.11 Status Monitoring and Triggers for Intervention

It is important that the CFA provide a structure for the monitoring of events associated with the use of resources in the local aggregate; this can then be used to trigger actions, including interventions by operators and/or operations services. For example, a rogue traffic flow could trigger an emergency shutdown.

Gap: The current CFA does not fully address the mechanisms needed for monitoring, triggers and interventions, and more work should be done in this important area as the CFA is firmed up.

It is not known what mechanisms for monitoring, triggers and interventions are included in Spiral 1 implementations.

5 Control Framework Structure

A block diagram of the basic GENI control framework structure is shown in Figure 5-1, which includes:

- Principal, Component and Slice Registries, which are the key entities in the Clearinghouse, and which include, respectively the Principal, Component and Slice Authority Services.
- Aggregates, each of which comprise one or more Components.
- Principals, such as Administrators, Operators, PIs and Researchers.
- Services, including Experiment Control and Broker Services.

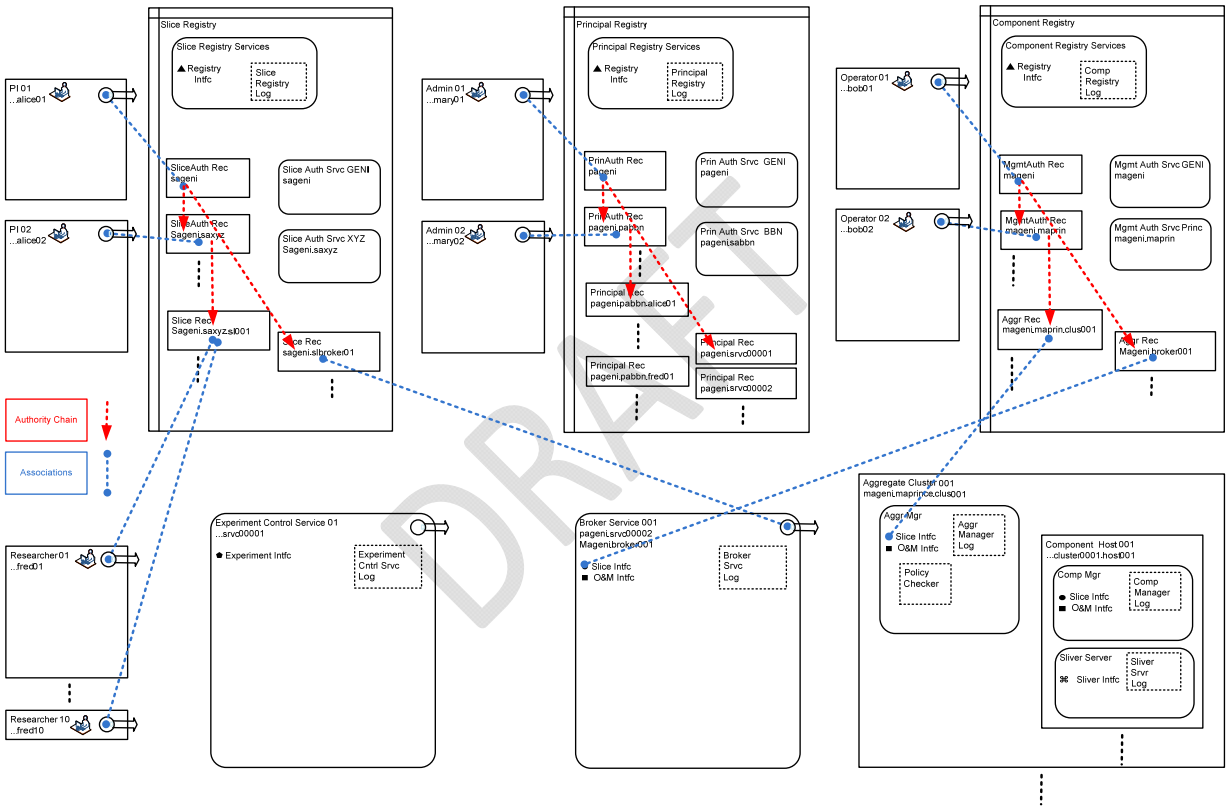

Figure 5-1. GENI Control Framework Structure

5.1 Registries and Interfaces

The registries hold all of the records plus associated services which are necessary for the operation of the GENI suite.

Although the three registries provide for different types of records and services, note that it should be possible to implement all of them on one underlying infrastructure, if that is desired.

The registries hold records that are generally static, changing only occasionally.

Each registry supports the Registry Interface, which is accessed by Principals and certain Services.

In each registry there is an Authority Chain that leads from a "root authority", e.g., Slice Authority GENI, down to all subtending records, both authority and object (principal, aggregate and slice) records. These authority chains are indicated in the figure using red dashed lines, with arrows in the down direction. The current view is that is done by:

- Including entries in an authority record that explicitly point down to a subtending authority record.
- Objects associated with a given authority include the global name of the authority in their global name, and are thus included in the same authority chain.
- An authority or object can find its way up the chain by checking its global name, which includes the name of the higher authorities.

In each registry there are also entries on records that create associations. These associations are indicated in the figure using blue dashed lines, with large dots on both ends.

The principal registry:

- Holds records for Principal Authorities, who are responsible for creating principal records and granting them proof of their identity for authentication.
- Holds the information necessary to associate Administrators with each principal authority record, to act for that principal authority.
- Holds records for Principals, who can then be given active roles in the GENI suite, such as Administrators, Operators, PIs and Researchers.
- Holds the information necessary to authenticate a Principal, and supports services to issue the necessary certificates and credentials to principals so that they can prove their identity.
- Includes an authority chain from the root principal authority record (pageni) down to all other principal authority and principal records.
- In the case where a service acts as principal, then it is given a principal record.

The component registry:

- Holds records for Management Authorities, who are responsible for creating aggregate records and who are responsible for managing aggregates and their components.
- Holds the information necessary to associate Operators with each management authority record, to act for that management authority.
- Holds records for Aggregates, each of which comprise one or more Components, that are able to contribute resources to the GENI suite.
- Holds the information necessary to locate an aggregate, and then discover its resources and status.
- Includes an authority chain from the root management authority (mageni) record down to all other management authority and aggregate records.
- In the case where a service acts as aggregate, then it is given an aggregate record.

The slice registry:

- Holds records for Slice Authorities, who are responsible for creating slice records and who are responsible for managing slices, including all of their slivers.
- Holds the information necessary to associate PIs with each slice authority record, to act for that slice authority.

- Holds records for Slices, each of which is equivalent to a "bank account" for that slice.
- Holds the information necessary to associate Researchers with a slice, who are then allowed to create the slivers that constitute the slice.
- Includes an authority chain from the root slice authority (sageni) record down to all other slice authority and slice records.

5.2 Distributed Registries

Alternately, a registry and its associated authority services may be decomposed into separate entities, which may be distributed on separate servers, and that can be deployed in different locations.

For example, Figure 5-2 shows how the slice registry and its associated services can be decomposed into two entities:

- Slice registry for Slice Authority SA_GENI, the "root authority".
- Slice registry for Slice Authority SA-GENI.SA_XYZ, which subtends the root authority.

Following the approach outlined above:

- An entry in the SA_GENI record points down to the SA-GENI.SA_XYZ record.
- The global name SA-GENI.SA_XYZ points up to the name of the higher authorities, e.g., SA_GENI, which also happens to be the "root authority".
- In both registries, slice records associated with the slice authority have global names that include the global name of the authority.

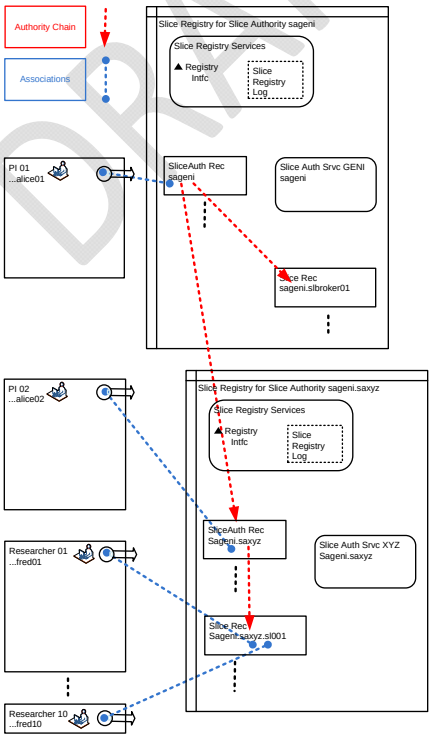

Figure 5-2. Distributed Slice Registry Structure

When there are distributed registries, queries concerning any authority or slice that are rooted in SA_GENI must be directed to the appropriate slice registry: SA_GENI or SA-GENI.SA_XYZ.

The current approach for doing this is to examine the name of the authority or slice to determine the responsible authority or authorities, i.e., SA_GENI or SA-GENI.SA_XYZ, and then formulate a URL that DNS resolves to the correct registry.

An alternate approach would be to direct all queries to the slice registry for the root authority SA_GENI, which would the return a redirection message to the slice registry for SA-GENI.SA_XYZ.

5.3 Aggregates and Interfaces

Aggregates, and the components which comprise them, are the primary building blocks of the GENI suite.

A component might correspond to an edge computer, a customizable router, or a programmable access point.

A component encapsulates a collection of *resources*, including physical resources (e.g., CPU, memory, disk, bandwidth) logical resources (e.g., file descriptors, port numbers), and synthetic resources (e.g., packet forwarding fast paths). These resources can be contained in a single physical device or distributed across a set of devices, depending on the nature of the component. A given resource can belong to at most one component.

It must be possible to multiplex (slice) component resources among multiple users. This can be done by a combination of virtualizing the component (where each user acquires a virtual copy of the component's resources), or by partitioning the component into distinct resource sets (where each user acquires a physical partition of the component's resources). In both cases, we say the user is granted a *sliver* of the component. Each component must include hardware or software mechanisms that isolate slivers from each other, making it appropriate to view a sliver as a “resource container.”

A sliver that includes resources capable of loading and executing user-provided programs can also be viewed as supporting an *execution environment* on a Sliver Server.

Each aggregate is controlled via a Aggregate Manager (AM), which exports two well-defined, remotely accessible interfaces: the Slice Interface and the Ops&Mgmt Interface.

Components within an aggregate may include their own Component Managers (CMs), which can also export the Slice and/or the Ops&Mgmt Interfaces. The location of these interfaces is typically returned to a Researcher after the AM typically assigns the component.

If a component includes a Sliver Server, then the location of the Sliver Interface on the sliver server is returned to a Researcher after the AM typically assigns the component.

5.4 Principals

A Principal is a user playing one (or more than one) of the following roles in the GENI suite:

- Administrators, who act for a principal authority and are responsible for principal records and the authentication of principals.
- Operators, who act for a management authority and are responsible for creating aggregate records and who are responsible for managing aggregates and their components.
- PIs, who act for a slice authority, and are responsible for slice records and who are responsible for managing slices, including all of their slivers.

- Researchers, who utilize the GENI suite for running experiments, deploying experimental services, measuring aspects of the platform, and so on.

5.5 Services and Interfaces

A Service can be deployed in the GENI suite to assist in the setup and running of experiments, or in the setup and running of the GENI suite. Services that may be deployed include:

- Experiment Control Service
- Broker Service
- Administration Service(s)
- Operations and Management Service(s)

Although optional, it is expected that an Experiment Control Service will typically be deployed in a GENI suite to assist the Researcher in setting up and running experiments, particularly in the challenging task of managing a slice that includes slivers on many aggregates.

An Experiment Control Service supports an Experiment Interface that is typically accessed by a Researcher.

Because an Experiment Control Service acts like a Researcher that is a principal, it requires a principal record to receive an identity that can be authenticated.

However, an Experiment Control Service typically acts for a given Researcher, and receives its ability to request resources, etc., from the Researcher. Thus, an Experiment Control Service does not typically need to be associated to a slice.

A Broker Service is optional in some Spiral 1 implementations, and required in other Spiral 1 implementations.

A Broker Service supports a Slice Interface and optionally an Ops&Mgmt Interface.

5.6 Slices

From a researcher's perspective, a *slice* is a substrate-wide network of computing and communication resources capable of running an experiment or a wide-area network service. 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.

A slice is defined by a set of slivers spanning a set of network components, plus an associated set of researchers that are allowed to access those slivers for the purpose of running an experiment on the substrate. That is, a slice has a name, which is bound to a set of users associated with the slice and a (possibly empty) set of slivers.

There are three unique stages in the lifetime of a slice:

- Registered: the slice exists in name only and is bound to zero, one or more researchers;
- Instantiated: the slice is instantiated on a set of components, and resources have been assigned to it;
- Activated: the slice is activated, and is providing resources to a researcher for their experiment.

A slice has to be registered and bound to a set of users before it can be instantiated, and it must be instantiated before being it can run code or be accessed by a user.

Slices are registered in the context of a *slice authority* that takes responsibility for the behavior of the slice. A slice is registered only once, but the set of users bound to it can change over time.

Instantiating a slice effectively configures the slice on a set of components; this step can be repeated multiple times. In fact, instantiating often involves two sub-steps: a slice is first instantiated on a set of components with only best-effort resources assigned to it, and later provisioned with additional (perhaps guaranteed) resources, for example, for the duration of a single experiment.

Activating a slice means that all associated slivers have been activated, and are ready to provide resources. An experiment or service can then ``run in'' a slice, utilizing its slivers. Multiple experiments can be run in a single slice. For each run, the experiment may change parameters but leave the slice configuration (instantiation) unchanged, or it may change either the set of components or the resources assigned on those components, or both. How rapidly a slice can be reconfigured to support a new experiment depends on the implementation of the instantiation and provisioning operations.

5.7 Control Communications Flows and Mutual Authentication

The current approach to control communication flows in the GENI suite is summarized by:

- Principals and services that periodically connect and communicate with registries, aggregates and services via defined interfaces and APIs.
- Since the aggregates in particular are expected to be widely distributed, and connections are made over an IP network that may be the open Internet, it is important that communications be secure, and that the principals be properly authenticated.
- Authentication of a principal at an interface must indicate who is connected, and this information may also be used for access control.
- Many of the communications flows utilize a web services model, with SOAP plus http(s) arranged for mutual authentication.
- Mutual authentication is done with certificates held on both sides which can be exchanged, and verified using a limited set of root certificates.

The current approach to authentication of registries, aggregates, principals and services in the GENI suite is summarized by:

- One Public Key Infrastructure (PKI) that covers all GENI suite registries, aggregates, principals and services.
- This PKI provides all necessary certificates, and allows verification to be done using a limited number of root certificates.
- Security problems can be expected to occur when a principal loses their certificate, or it is compromised; these must be detected, and managed.
- Thus, this PKI must support a Certificate Revocation List (CRL) (or equivalent) to be able to respond to compromised certificates, etc.

However, maintaining one PKI for a GENI suite is not likely to be practical or appropriate since:

- Supporting a large PKI is complicated and relatively expensive.
- Associated institutions may well support their own PKI systems, or other authentication mechanisms, such as Kerberos, etc.
- Accommodating existing authentication systems from associated institutions is a step towards federated GENI suites.

Gap: Extending the CFA to accommodate such alternative authentication arrangements is important.

One example is the authentication infrastructure developed by the GRID community; see [http://www.globus.org/alliance/publications/papers/IEEE_NCA_AGC.pdf] However, the GRID community has solved a different problem: permitting a researcher to log-in to a wide variety of resources, managed by a wide variety of institutions, each of which control accounts and apply local policies. The GENI suite is different in that there is always an aggregate manager that serves a dividing line between the GENI environment and the local environment, and there is no need to permit the log-ins required in the GRID community.

Furthermore, most communication flows are expected to follow a web services model, and various authentication extensions have been developed in that model. For example, a web server may authenticate a principal by utilizing the Shibolleth system to access a remote authentication system and query it with SAML 2.0 messages; see [http://www.globus.org/alliance/publications/papers/butler.pdf].

[What can be done as a first step?]

5.8 Authorization via the Exchange of Tokens

The current approach to authorization in the GENI suite is to

- Utilize the exchange of tokens (called credentials, tickets, etc.).
- The tokens are used to permit access to registries and authority services.
- The tokens are used to mediate the resource authorization, plus assignment and management.
- The tokens must be signed (certified) by the appropriate authorities and objects (principals, aggregates and slices) to give them some intrinsic value.
- This is the approach that was developed in a trust management system; see [http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.7726].
- This is the approach taken in the SFA; see [http://groups.geni.net/geni/attachment/wiki/GeniControlBr/v1.10%20%20080808%20%20sfa.pdf].
- This is the approach taken in the research leading to the ORCA implementation; see [http://portal.acm.org/citation.cfm?id=1267377].
- It is clearly a very flexible approach.
- It permits the involved entities to be widely dispersed and independently operated, and it should help provide for “disconnected operation”.

The current approach to certifying and verifying tokens in the GENI suite is summarized by:

- To certify a token (credential, ticket, etc), 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.
- The authority or object that receives this token can then verify it using a limited set of root certificates.

- One Public Key Infrastructure (PKI) that covers all GENI suite authorities, principal, services and aggregates, and provides all of the keys and other structure to sign and verify tokens.
- Currently, this PKI is the same PKI that is used to authenticate principals, etc., but it could be independent, especially as principal authentication is extended as noted above.
- Some security problems can be expected to occur when a token is lost, or it is compromised; these must be detected, and managed. Perhaps this needs to be done only for "high value" tokens.
- This PKI may need to support a Certificate Revocation List (CRL) to be able to respond to lost tokens, or perhaps there is a simpler mechanism that would be sufficient. For example, the authority or object that receives a token could check directly with the entity that issued the token.
- **Gap: Extending the CFA to support a Certificate Revocation List (CRL) to be able to respond to lost tokens, or an equivalent simpler mechanism, is important.**

5.9 Registry Records

The registries hold all of the records which are necessary for the operation of the GENI suite. The three registries provide for different types of records and services, but it should possible to implement all of them on one underlying infrastructure.

The registries hold records that are generally static, changing only occasionally.

Here are a set of example formats, based on the SFA, but with some modifications and additions. They are not meant to be definitive, but to illustrate some of the parameters that are required.

Principal Registry Records

Principal Authority Record
{
Type = PrincipalAuthority
GlobalName
Public Key
GID = (Type, GlobalName, PublicKey)
Expires= [expiration date and time for this record]
OrganizationName = [xxx]
ContactInfo = [xxx]
Administrators = [list of GIDs for associated principals]
SubAuthorites = [list of GIDs for subordinate authorities]
}

Principal Record
{
Type = PrincipalIndividual or PrincipalService
GlobalName

ObjectID (optional)
PublicKey
GID = (Type, GlobalName, ObjectID(optional), PublicKey)
Expires= [expiration date and time for this record]
PrincipalName = [xxx]
ContactInfo = [xxx]
Optional: One or more identifiers from other environments, e.g, local research organization.
}

Component Registry Records

Management Authority Record
{
Type = ManagementAuthority
GlobalName
Public Key
GID = (Type, GlobalName, PublicKey)
Expires= [expiration date and time for this record]
OrganizationName = [xxx]
ContactInfo = [xxx]
Operators = [list of GIDs for associated principals]
SubAuthorites = [list of GIDs for subordinate authorities]
}

Aggregate Record
{
Type = AggregateComponent or AggregateService
GlobalName
ObjectID (optional)
PublicKey
GID = (Type, GlobalName, ObjectID(optional), PublicKey)
Expires= [expiration date and time for this record]
OwnerName = [xxx]
ContactInfo = [xxx]
OperatorName = [xxx]
ContactInfo = [xxx]
PhysicalLocation = [xxx]
URIofSliceIntrfc = [xxx; and/or other contact point]
URIofOps&MgmtIntrfc = [xxx; and/or other contact point]
URIofResourceAvailabilityService = [xxx; and/or other contact point]
URIofStatusService = [xxx; and/or other contact point]
}

Slice Registry Records

Slice Authority Record
{
Type = Slice Authority
GlobalName
Public Key
GID = (Type, GlobalName, PublicKey)
Expires= [expiration date and time for this record]
OrganizationName = [xxx]
ContactInfo = [xxx]
PIs = [list of GIDs for associated principals]
SubAuthorites = [list of GIDs for subordinate authorities]
}

Slice Record
{
Type = Slice
GlobalName
ObjectID (optional)
PublicKey
GID = (Type, GlobalName, ObjectID(optional), PublicKey)
Expires= [expiration date and time for this record]
OwnerName = [xxx]
ContactInfo = [xxx]
Researchers = [list of GIDs for associated principals, individuals or services]
}

Note that we expect the information available in a registry to be relatively static. To learn more detailed and dynamic information about an aggregate, for example, one needs to call the URIs identified in the registry.

5.10 Global Names and Authority Chains

Chains of authorities that are responsible for an object (principal, aggregate or slice) are currently included in the CFA for these reasons:

- To be able to organize authorities (and their associated records) into groups that can be associated with separate (distributed) registries.
- To allow principals associated with an authority to act on all authorities and objects at or below their level, including modification of their records. For example, this gives an PI associated with the top-level or root GENI authority the ability to access the Slice Intfc on all aggregates and make changes on any slice, including shutting it down.
- To specify that any necessary certificate be signed by this chain, and thus those who receive the certificates need only have a root certificate from the top-level or root GENI authority to evaluate them.

The current choice is a hierarchical name space corresponding to a hierarchy of authorities that have delegated the right to create and name principal, component and slice objects. This hierarchy assumes a "GENI" top-level naming authority trusted by all entities, resulting in GlobalNames of the form:

GlobalName = top-level_authority . sub_authority . sub_authority . object_name

= authority_chain . object_name

where

authority_chain = top-level_authority . sub_authority . sub_authority

Example GlobalNames:

sageni.saxyz.sl001

pageni.pabbn.alice01

mageni.maprin.clus001

Thus, the chain of authorities for an object is included in its GlobalName, and is found by removing the object.name from the GlobalName.

And, when a certificate is required for the GID of an object, it is signed in turn by the responsible authorities, ending with the top-level or root authority. Thus, the chain of authorities is also indicated by the sequence of signatures on a GIDcert.

Within a registry, the current way to find the chains of authority is to examine the records and GlobalNames for all authorities and objects. Then:

- There are explicit entries in authority records for the GID(s) of subtending authorities.
- Removing the last name in the GlobalName of an authority record indicates the authorities above it, starting with the root authority.
- Removing the last name in the GlobalName of an object (principal, slice or aggregate) records indicates the authorities above it, starting with the root authority.

5.11 Global Identifiers and Authentication Information

The CFA defines a unique *global identifier* (GID) for each authority and object (principal, aggregate or slice). Specifically, each record holds these identifiers and authentication records:

Type = […]
GlobalName
ObjectID (optional)
PublicKey
GID = (Type, GlobalName, ObjectID(optional), PublicKey)

Expires= [expiration date and time for this record]

While the corresponding authority or object (principal, aggregate or slice) holds this information:

PrivateKey and PublicKey Pair

Type = […]
GlobalName
ObjectID (optional)
PublicKey
GID = (Type, GlobalName, ObjectID(optional), PublicKey)
GIDcert = [GID signed by responsible authorities, with expiration date and time]

In addition, a principal optionally holds one or more:

PrivateKey and PublicKey Pair for ssh access

Figure 5-3 shows how these records and information are held by authorities and objects.

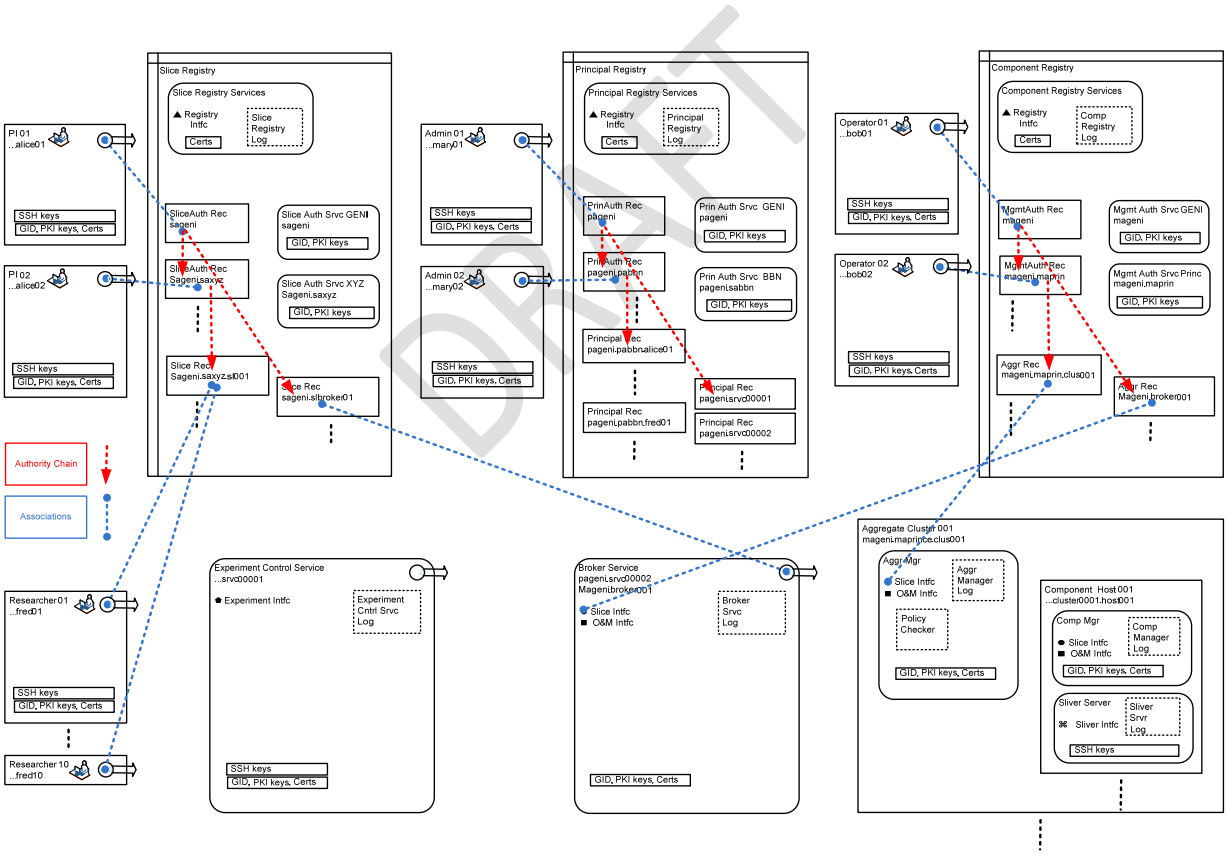

Figure 5-3. Identifiers, PKI Keys, Certificates and SSH Keys.

This apporach supports two kinds of authentication in the CFA for authorities and objects (principals, aggregates and slices):

- Mutual authentication over a secure channel, using a protocol such as TLS.

- Certification by authorities and objects (principals, aggregates and slices) of the tokens (credentials, tickets, etc) that are exchanged to authorize resource assignment, etc.

For mutual authentication over a secure channel, both objects hold GIDcerts, which can be exchanged, and verified using a limited set of root certificates.

For certification of a token (credential, ticket, etc) by an authority or object, it signs a token using its own private key, which is then followed by signatures from its responsible authorities, up to the root authority. The authority or object that receives this token can verify it using a limited set of root certificates.

This optional ssh key pairs support between a principal and an aggregate:

- Mutual authentication over a secure channel, using a protocol such as SSH.

Note that each authority and object object identified by a GID holds the private key corresponding to the PublicKey in the GID, thereby forming the basis for authentication.

The Expires field says how long the GID is valid; GIDs need to be "refreshed" periodically.

Gap: How is this refreshment to be done? Can the expires time be short enough that a Certificate Revocation List (CRL) does not need to be provided?

The GIDcert follows the x.509 format, and is signed by the responsible authorities, up to the top authority. Each authority is identified by its own GID, and thus any entity may verify a GID via cryptographic keys that lead back, up the chain, to a well-known root or roots.

It is important that each GID be unique. If there is no ObjectID, this means that each GlobalName must be unique, and each registry must enforce uniqueness when an authority or object is registered.

The SFA suggests utilizing an ObjectID that is a UUID. The UUID is a random number, generated following X.667 (RFC4122), that is guaranteed to be unique. Its format is:

128 bits, 16 bytes, arranged in groups of hex digits as: 8-4-4-4-12.

The SFA also suggests utilizing only the ObjectID (that is, the UUID) to identify an object and find its record in a system registry. However, this cannot (practically) work in a system with multiple, distributed, registries, since the ObjectID = UUID, by itself, being a random number, contains no information that can be used to resolve a query to a particular registry.

Thus, since we want to allow a system with multiple registries, we cannot use an ObjectID that is a UUID to guarantee a unique GID. It seems best that we rely on a unique GlobalName.

However, there are situation when an ObjectID could be useful, such as tagging packets with a number to identify the responsible slice. Then, it may be appropriate to include an ObjectID in these situations, but its format is TBD.

5.12 Associations and Principal Roles

As shown above, and in the SFA document, a registry entry associates a principal with an authority or object to give them a particular role:

- Administrator to Slice Authority
- Operator to Management Authority
- PI to Slice Authority

- Researcher to Slice

Based upon these roles indicated in the registry records, the identified principal can be issued a credential token that gives them the privileges associated with their role.

And, one principal can associated with more than one authority or object to give them multiple roles.

Alternately, other approaches are being implemented. Per the ProtoGENI implementation, a principal can receive a credential token directly from an authority that then allows them to play a particular role. For example, a principal could receive a credential from a slice authority that allows them to act as a PI, and a principal could receive a credential from a slice authority that allows them to act as a researcher. In this case, the slice authority may, or may not, record the resulting roles in the slice registry.

Extending this further, a principal could receive a credential from a slice authority that allows them to act as a PI, and in turn they could delegate the credential, with reduced privileges, to another principal, so that they could act as a researcher. In this case, the PI may, or may not, record the resulting roles in the slice registry.

6 Control Framework Interfaces

6.1 Overview

The CFA includes these interfaces:

- Registry Interfaces on registries, accessed via the Control Plane.
- Slice Interfaces on aggregates and broker services, accessed via the Control Plane.
- Ops&Mgmt Interfaces on aggregates and broker services, accessed via the Ops&Mgmt Plane.
- Experiment Interfaces on experiment control services, accessed via the Control Plane.
- Sliver Interfaces on sliver servers, accessed via the Control Plane.

All interfaces are accessed by a principal using their associated server.

Figure 6-1 summarizes the nature of these interfaces for the case where the principal is a Researcher.

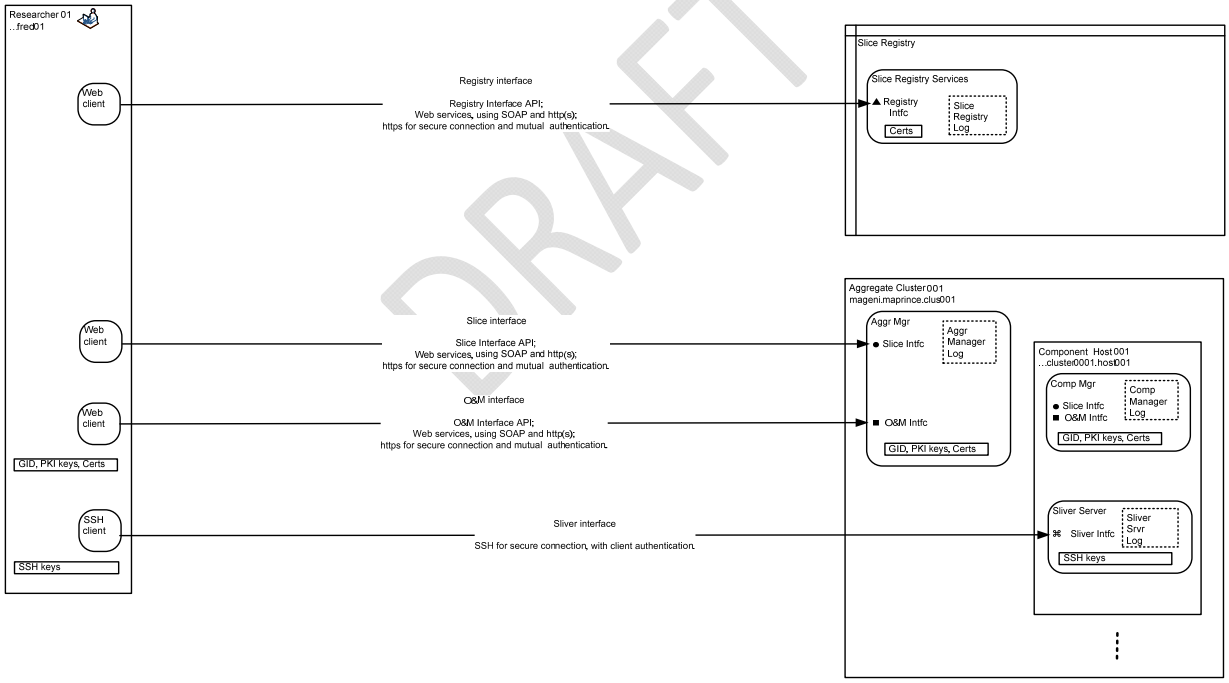

Figure 6-1. Registry, Slice, Ops&Mgmt and Sliver Interfaces.

6.2 Registry, Slice and Ops&Mgmt Interfaces

Communications between a principal's server and he Registry, Slice and Ops&Mgmt interfaces follows a web services model. The current decision is to utilize:

- A custom API for each of these interfaces which utilizes web service messages.

- The principal and their server is on the client side.
- The registry, slice and ops&mgmt interfaces are on the server side.
- Web service messages that utilize SOAP and http(s) protocols for messages.
- A WSDL that specifies the message structures.
- An XSD that specifies the data structures.
- https that utilizes TLS to provide a secure connection.
- http(s) that is used to provide mutual authentication.
- To permit mutual authentication, certificates in both servers can be verified by the other server using a root certificate.

This approach is widely deployed and well proven, which should facilitate the implementation of software modules to realize the CFA.

The current approach to communications between a Researcher and a Register Interface, including mutual authentication, is summarized in Figure 6-2 and here:

- A Public Key Infrastructure (PKI) covers all GENI suite registries, aggregates, principals and services and provides all necessary certificates.
- This allows verification to be done using a limited number of root certificates, which are loaded into all servers.
- Authentication of a principal at an interface indicates who is connected, which should be logged, and which can also be used for access control.

Gap: Is access control required, given that the API provides for authorization? If access control were required, how could it be done? Just check for the high level authorities?

- Security problems can be expected to occur when a principal loses their certificate, or it is compromised; these must be detected, and managed.
- Thus, the PKI should support a Certificate Revocation List (CRL), and the TLS server side should check this list during the authentication procedure. [If the CRL is checked, it will take time, and no response could delay communications indefinitely; is this OK?]
- One approach would be to have the appropriate principal registry support a CRL, that could be checked by the TLS server.

Gap: Is there another way to verify the identity of the researcher? Perhaps issue a chellenge that requires them to sue their PrivateKey, and then verify the response using their published PublicKey?

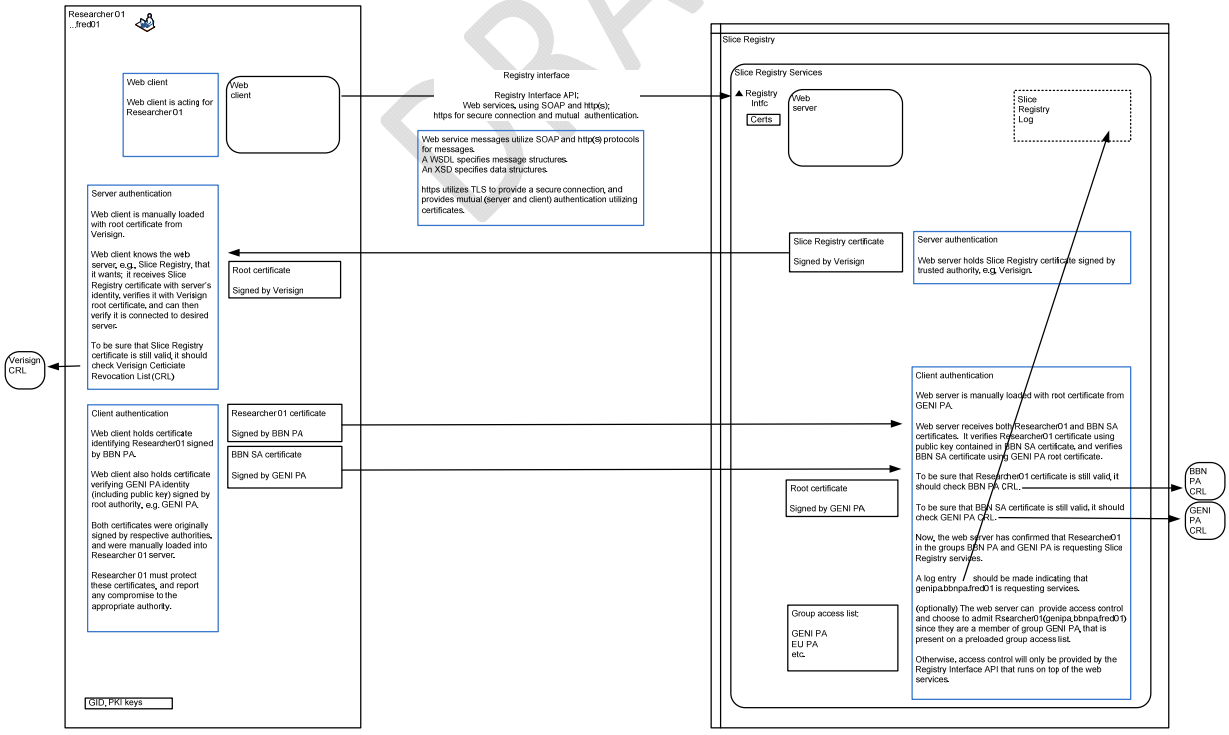

Figure 6-2. Communications from Researcher to Registry Interface, with Mutual Authentication.

The case where a researcher on their server is communicating with an Experiment Control Service on its own server, and the Experiment Control Service on its server is communicating with a register interface is shown in Figure 6-3. Note that it is the Experiment Control Service server that is being authenticated at the Registry Interface, and not the Researcher or their server.

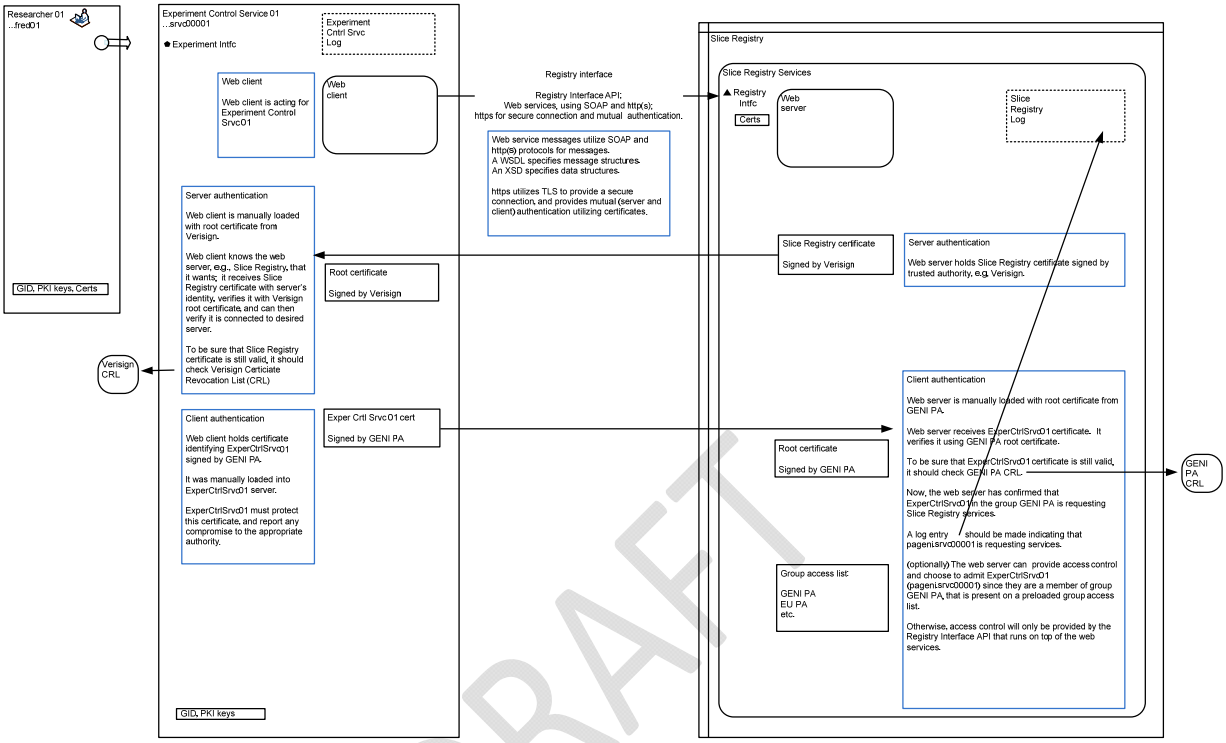

Figure 6-3. Communications from Researcher via Experiment Control Service to Registry Interface, with Mutual Authentication.

6.3 Experiment Interface

This interface is TBD, but it is likely to follow a web services model, and be similar to the registry, slice and Ops&Mgmt interfaces.

6.4 Sliver Interface

When a principal on their server connects to the Sliver Interface on a sliver server, the purpose is to be able to program, configure and operate that server. This is necessary to meet the overall objective of programmability

To do this, this connectivity is provided with a Secure Shell (SSH) login, and that is expected to be the typical approach in the GENI suite.

A SSH login does provide for almost complete control of the sliver server, but it is important that that server by well isolated from other slices to prevent security breaches.

Mutual authentication is required in a SSH login, and various mechanisms are supported by SSH, including:

- Matched account logins.

- One-time passwords.
- Public and private key pairs.
- Kerberos-like arrangements.

The current choice is to utilize public and private key pairs, where the public key is loaded in to the server (sliver server, in this case) and both public and private keys are held in the SSH client.

Figure 6-4 summarizes how mutual authentication occurs in the case where a researcher on their server complete a SSH login into a sliver server.

This login requires that a public key for this researcher be loaded into the sliver server before the SSH login is done. The current approach is to have the researcher utilize the Slice Interface API on the component (or aggregate) to load the public key.

Note that other variations to these choices could work well: the sliver server could return a one-time password to the researcher via the sliver interface API.

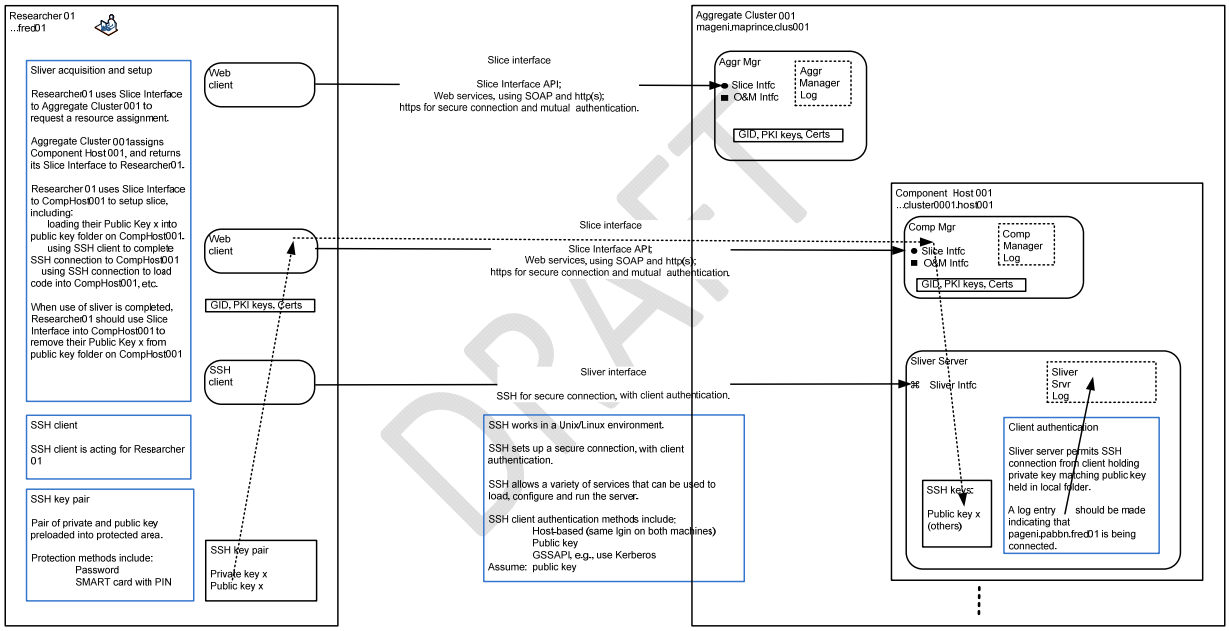

Figure 6-4. Communications from Researcher to Sliver Server Interface, with Mutual Authentication.

Figure 6-5 summarizes how mutual authentication occurs in the case where a researcher on their server is communicating with an experiment control service on its server, which in turn completes a SSH login into a sliver server. In this case, the public and private key pairs are held in the experiment control service server, and the public key must be loaded by the service. [But, how can the key pair be kept secure in the experiment control services server?]

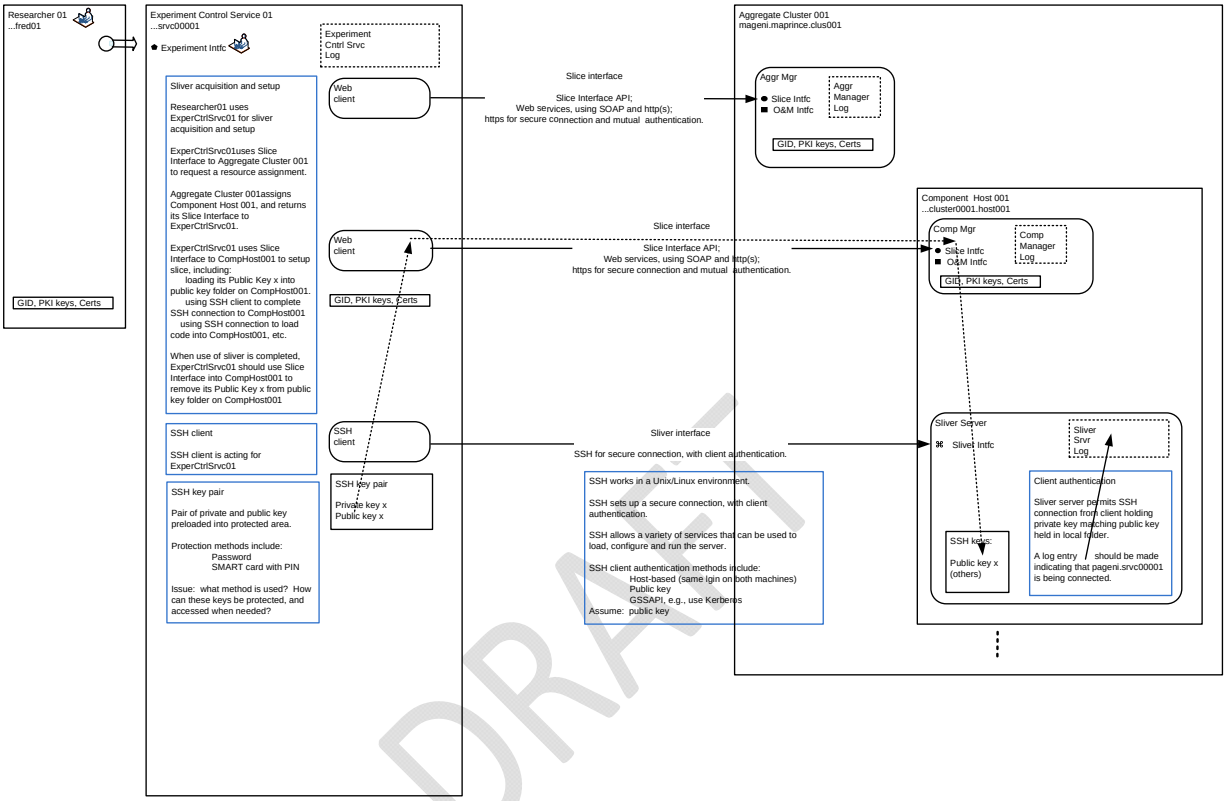

Figure 6-5. Communications from Researcher via Experiment Control Service to Sliver Server Interface, with Mutual Authentication.

7 Credentials

7.1 Overview and Bootstrapping

The current approach to authorization in the GENI suite is to utilize the exchange of tokens, including Credentials, following a trust management system.

A principal (or service) requires a Credential to access:

- Authority services and registry records via the Registry Interface
- Aggregates to authorize resource assignment via the Slice Interface
- Aggregates to manage resources, via the Ops&Mgmt Interface.

Each Credential must be signed (certified) by the appropriate authorities to make them valid, and to allow them to be verified.

Each Credential specifies one (or more) privileges that are granted to the principal (or service) that holds the credential.

The way for a principal (or service) to get a Credential is to execute a GetCredential operation at a Registry Interface to have the relevant authority service give them the desired credential. But, as indicated above, a principal (or service) requires a credential to access an authority service via a Registry Interface. How can this process be bootstrapped?

The current approach is summarized here and in Figure 7-1:

- A Principal assembles their basic identity (GID, including GlobalName and optional ObjectID) and authentication (PrivateKey and PublicKey pairs) records, and the administrator of one of their responsible principal authorities creates a Principal Record for them, and registers it in the Principal Registry.
- The Principal calls the Principal Registry Interface with a GetCredential request where both the calling and referenced objects are themselves, identified with their GID.
- The registry matches the GID with the existing Principal Record, and verifies their identity via the certificate carried over the SSL connection.
- Finally, the registry issues the Principal a "self credential", which can then be used to access a registry to get other credentials.

To summarize: A Principal gates a "self credential" from the Principal Registry, signed by the appropriate Principal Authorities.

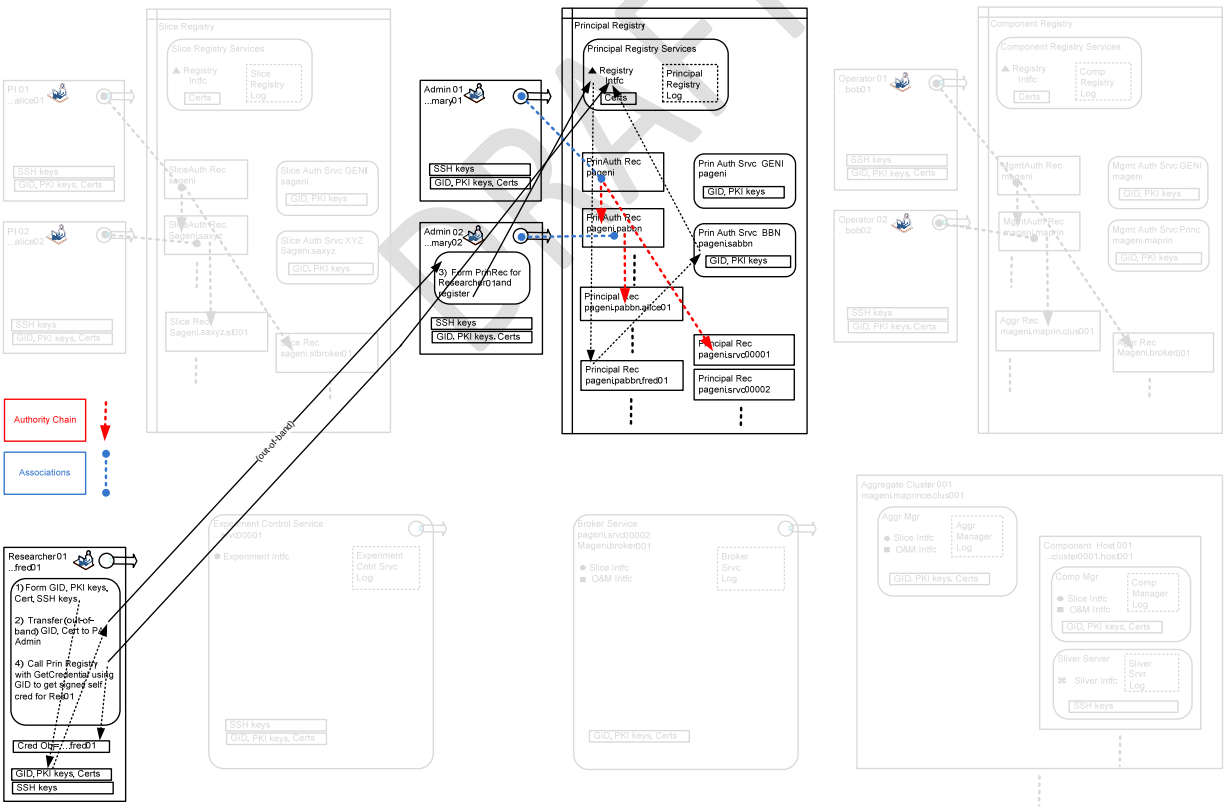

Figure 7-1. Principal Getting First Self Credential

Now that a Principal has a "self credential" identifying themselves, they can use it to request additional credentials at other registries. These other registries must recognize the signatures on the "self credential".

For example, following the SFA, a Researcher can use their "self credential" to call the Slice Registry Interface with a GetCredential request, with the referenced object being a slice to get a "slice credential" that will allow them to create slivers, etc.

In a similar fashion, an Administrator, PI or Operator can use their "self credential" to call the Principal, Slice or Component Registry Interfaces, respectively, to allow them to act as the respective authorities to register records, and manage slicers or aggregates.

Thus, in most situations, each Principal has at least two credentials: a self credential and a credential corresponding to their principal role. This is shown in Figure 7-2.

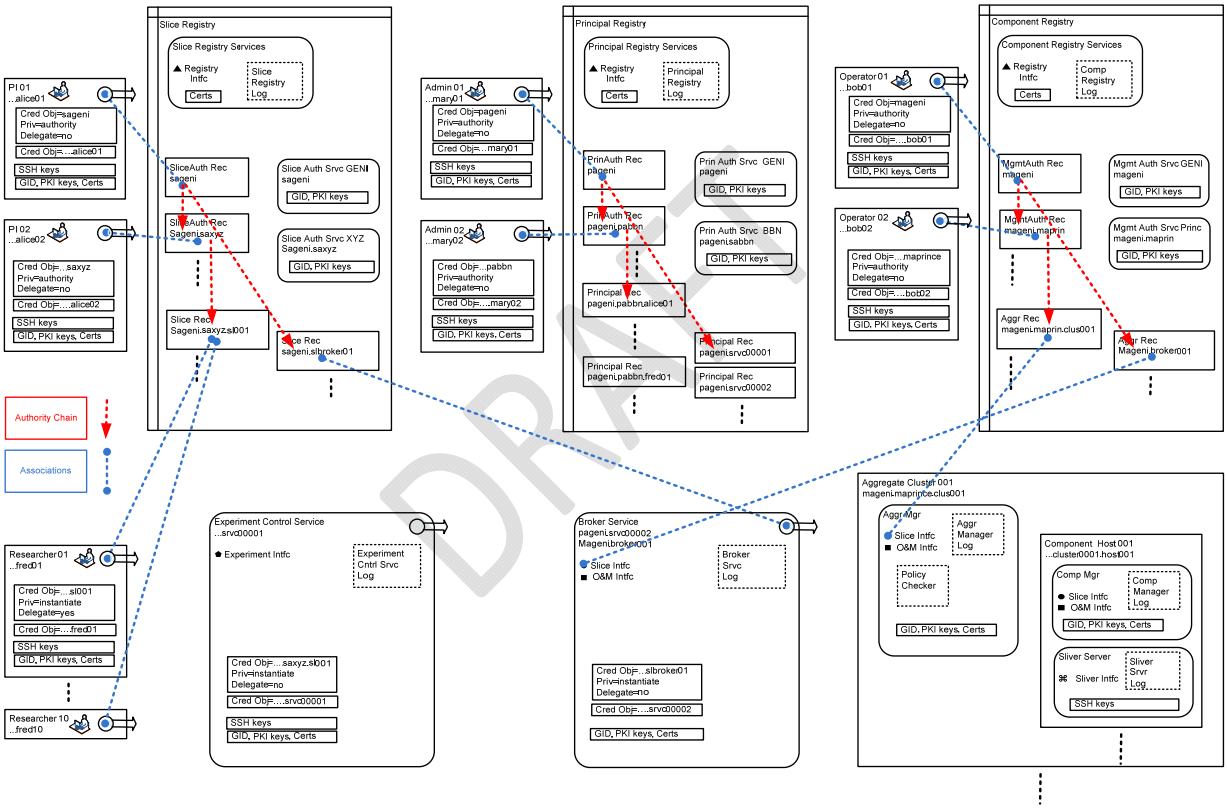

Figure 7-2. Principals and Services with Credentials

7.2 Credential Usage

The API calls to the Registry, Slice and O&M Interfaces typically take a credential as an argument.

A credential carries the rights issued to a particular principal. For example, a principal acting as a researcher is granted credentials that allow it to instantiate a slice in a set of willing components.

A credential must be sufficient for the called object to determine that:

- The credential is valid.
- The credential has not expired.
- It is being presented by the indicated principal, or on their behalf, and has not been stolen from the principal.
- The caller is allowed to invoke the specified operation.

The called object may verify that the credential is valid by verifying that the authorities who signed the credential are recognized and trusted, by checking its signatures.

The called object needs to check that a credential has not expired. If it is expired, it needs to reject the credential. Then the principal must go back and get a new credential, and present it to the called object. Note that setting the expiration time to a short interval is a way for the system to force each principal and their credentials to be periodically reauthorized.

Gap: How can the called object verify that the credential is being presented by the indicated principal, or on behalf of the indicated principal? In the case where it has been presented directly by the indicted principal, their identity can be checked by using the authentication mechanism. In the case where the credential is delegated, i.e., to an Experiment Control Service, there is no direct way to verify htat it is being presented on behalf of the indicated principal. However, the service can be authenticated, and perhaps the service can then be trusted to have received the credential in a secure manner from the principal.

The called object may decide that the caller is allowed to invoke the specified operation by using a local Policy Checker, to check parameters presented as part of the credential, plus parameters in the called object.

Gap: How are the policy checker rules written for a given aggregate, on behalf of its management authority? What is the format, i.e., pseudo code? How are these rules distributed to each aggregate manager?

Note that one key to federation is the ability of the called object to recognize and trust more authorities, and to make valid judgments about granting resources to a wider set of principals and slices.

7.3 Example from SFA: Credential Format

This example is from the SFA document; see http://groups.geni.net/geni/attachment/wiki/GeniControlBr/v1.10%20%20080808%20%20sfa.pdf].

A credential is given by the 5-tuple:

Credential = (CallerGID, ObjectGID, Expires, Privileges, Delegate)

Where:

- CallerGID identifies the principal to which the credential has been issued;
- ObjectGID identify the object for which the credential applies;
- Expires says how long the credential is valid;
- Privileges identifies the class of operations the holder is allowed to invoke; and
- Delegate indicates whether the holder is permitted to delegate the credential to another principal.

A credential is signed by the responsible authority, and similarly re-signed when delegated. Although not defined in this document, we assume a library routine that a user calls to delegate a credential to another principal. This routine must allow the holder of a credential to delegate a subset of the privileges it holds, as well as clear the Delegate field so that the credential cannot be re-delegated.

Each privilege implies the right to invoke a certain set of operations on one or more of the SFA interfaces. Privileges include:

Privilege	Interface	Operations
authority	Registry	*all*
refresh	Registry	Remove, Update
resolve	Registry	Resolve, List, GetCredential
pi	Slice	*all*
instantiate	Slice	GetTicket, InstantiateSlice, DeleteSlice, UpdateSlice
bind	Slice	GetTicket, LoanResources
control	Slice	UpdateSlice, StopSlice, StartSlice, DeleteSlice
info	Slice	ListSlices, ListComponentResources, GetSliceResources, GetSliceBySignature
operator	Management	*all*

7.4 Example from SFA: Policy for Issuing Credential

When a registry issues a credential to a principal, it grants privileges per its own policy, which in turn allows the principal to call certain operations on a selected interface.

Caller	Object	Privilege	Interface	Operations
Admins, PIs, Operators	Records of associated authority	authority	Registry	*all*
Principals	Own record	refresh	Registry	Remove, Update
Principals	All records	resolve	Registry	Resolve, List, GetCredential
Principals associated with SA	Slice associated with that SA	pi	Slice	*all*
Researcher associated with slice	Slice	instantiate	Slice	GetTicket, InstantiateSlice, DeleteSlice, UpdateSlice
Researcher associated with slice	Slice	bind	Slice	GetTicket, LoanResources
Researcher associated with slice	Slice	control	Slice	UpdateSlice, StopSlice, StartSlice, DeleteSlice
All principals	All slices and aggregates	info	Slice	ListSlices, ListComponentResources, GetSliceResources, GetSliceBySignature
Operator Associated with MA	Aggregates managed by that MA	operator	Management	*all*

7.5 Example from ProtoGENI: Credential Format

This example is from the current ProtoGENI implementation, as summarized in Section 13.

An ProtoGeni credential is currently defined as:

```
## 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.
   element type { "privilege" | "ticket" },
   ## 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 ),
   ## Optional Extensions
   element extensions { anyelementbody }*,
   ## Parent that delegated to us
   element parent { credentials }?
}

signatures = element signatures {
   element sig:Signature { ... }+
}
SignedCredential = element signed-credential {
   credentials,
   signatures?
}
```

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.

7.6 Example from ProtoGENI: Policy for Issuing Credential

The approach being utilized in the current ProtoGENI Implementation is summarized here:

- The slice authority maintains a list of principals who can be PIs.
- A Principal on the PI list requests and receives a slice credential directly from the slice authority.
- The slice credential contains a complete set of privileges.
- The PI forms other slice credentials with reduced sets of privileges, for delegation to principals that can be Researchers for this slice.
- The PI transfers these new slice credentials to the Researchers, who can then present them to aggregates to get resources.

8 Registry Interface API

8.1 Overview

The Registry Interfaces on all three registries are used to:

- Operate on the records contained within the registry.
- Query the registry to obtain information from its records.
- Issue credentials based upon information in the records, and have these credentials signed by the authority services contained within the registry.

8.2 Example from SFA: Supported Operations

This example is based on the SFA document; see [http://groups.geni.net/geni/attachment/wiki/GeniControlBr/v1.10%20%20080808%20%20sfa.pdf].

The registry interface supports the following six operations.

All require the caller to submit a Credential with the privileges to allow it call that operation. From Section xxx above, it can be seen that a Caller who is an Admin, PI or Operator has been granted an Authority privilege, which allows them to call any of these operations.

(no returned value)	Register (Credential, Record)
(no returned value)	Remove (Credential, Record)
(no returned value)	Update (Credential, Record)
Record =	Resolve (Credential, GlobalName, Type)
Record[ ] =	List (Credential, Type)
Credential =	GetCredential (Credential, GlobalName, Type)

The first two operations are used to register and un-register objects and principals, while the third operation is used to update information about an entry.

The fourth operation is used to learn the information bound to a given GlobalName.

The fifth operation is used to retrieve information about the set of objects managed by a given authority.

All operations are interpreted relative to a Credential that specifies the context (authority) in which the operation is applied. For example, invoking List with a Credential that specifies GlobalName=planetlab.princeton and Type=Slice returns all slices registered by the Princeton slice authority.

The final operation allows a principal to retrieve credentials corresponding to the named object. For example, a user might invoke GetCredential, giving his or her user credentials as the first argument, to retrieve the credentials associated with the named slice. The Type argument is used to differentiate among multiple records with the same name, so for Type=Slice, the return value is a “slice credential” that can subsequently be passed to the operations defined in the next section. Similarly, a call to

GetCredential with Type=SA returns a "registry credential" that can subsequently be used to operate on records belonging to the named authority.

8.3 Example from SFA: Registering a Slice and Associating Researchers

This example is based on the SFA document; see [http://groups.geni.net/geni/attachment/wiki/GeniControlBr/v1.10%20%20080808%20%20sfa.pdf].

Figure 8-1 summarizes the process where a PI registers a Slice in the Slice Registry and associates principals who become Researchers.

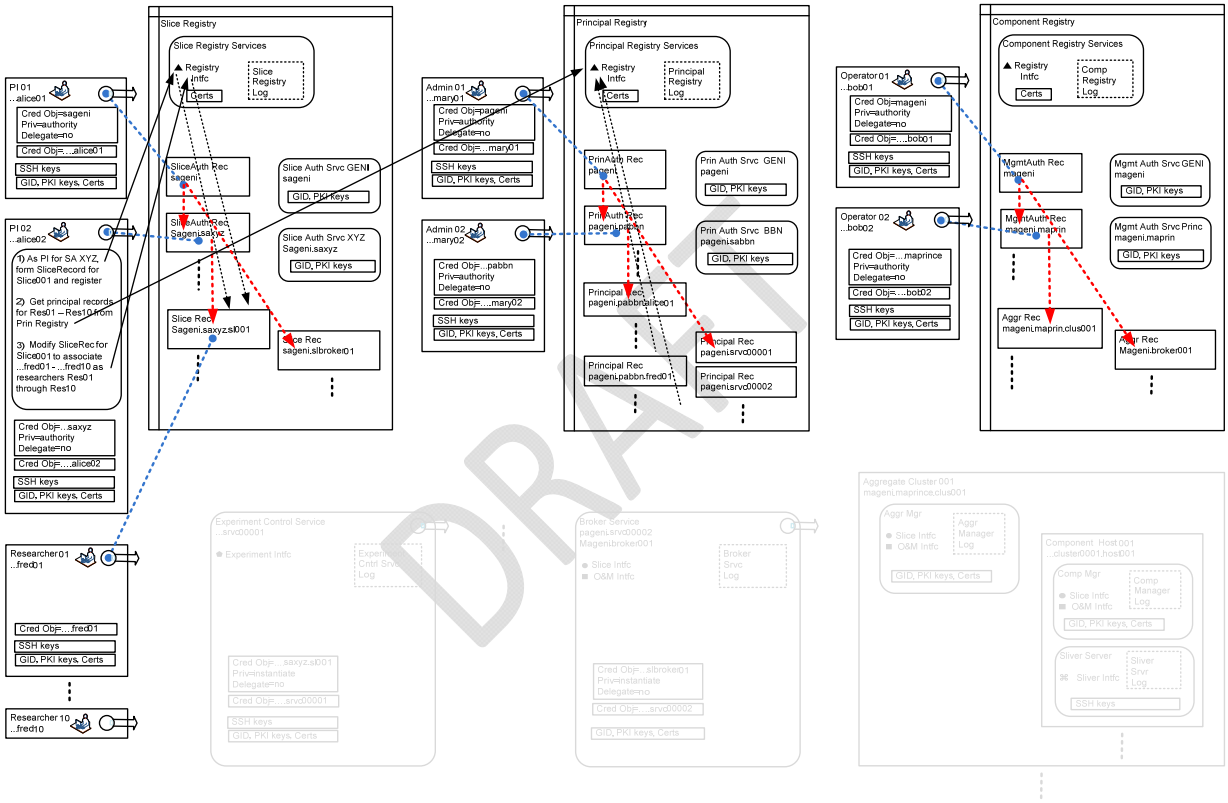

Figure 8-1. Registering a Slice and Associating Researchers

8.4 Example from SFA: Getting a Slice Credential for a Researcher

This example is based on the SFA document; see [http://groups.geni.net/geni/attachment/wiki/GeniControlBr/v1.10%20%20080808%20%20sfa.pdf].

Figure 8-2 summarizes the process where a Researcher gets a Slice Credential from the Slice Registry.

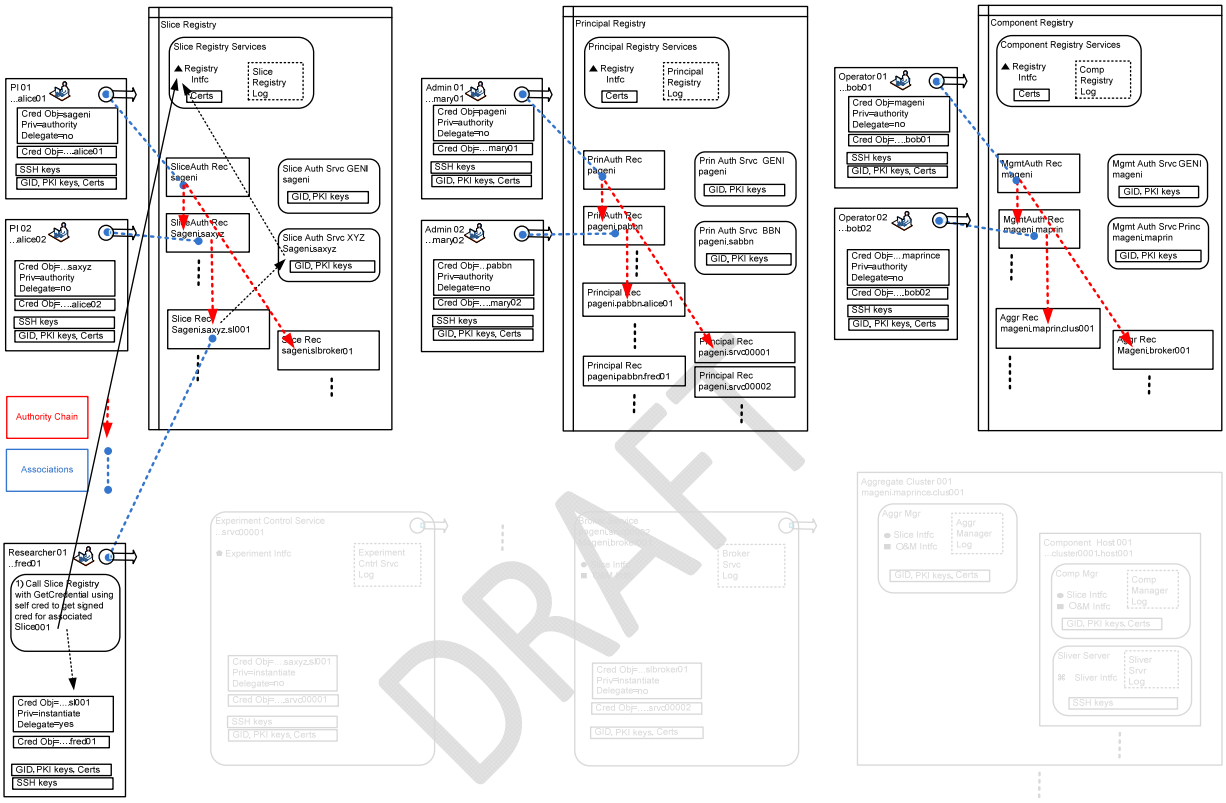

Figure 8-2. Getting a Slice Credential for a Researcher

8.5 Example from SFA: Registering an Aggregate

This example is based on the SFA document; see [http://groups.geni.net/geni/attachment/wiki/GeniControlBr/v1.10%20%20080808%20%20sfa.pdf].

Figure 8-3 summarizes the process where an Operator registers an Aggregate in the Component Registry.

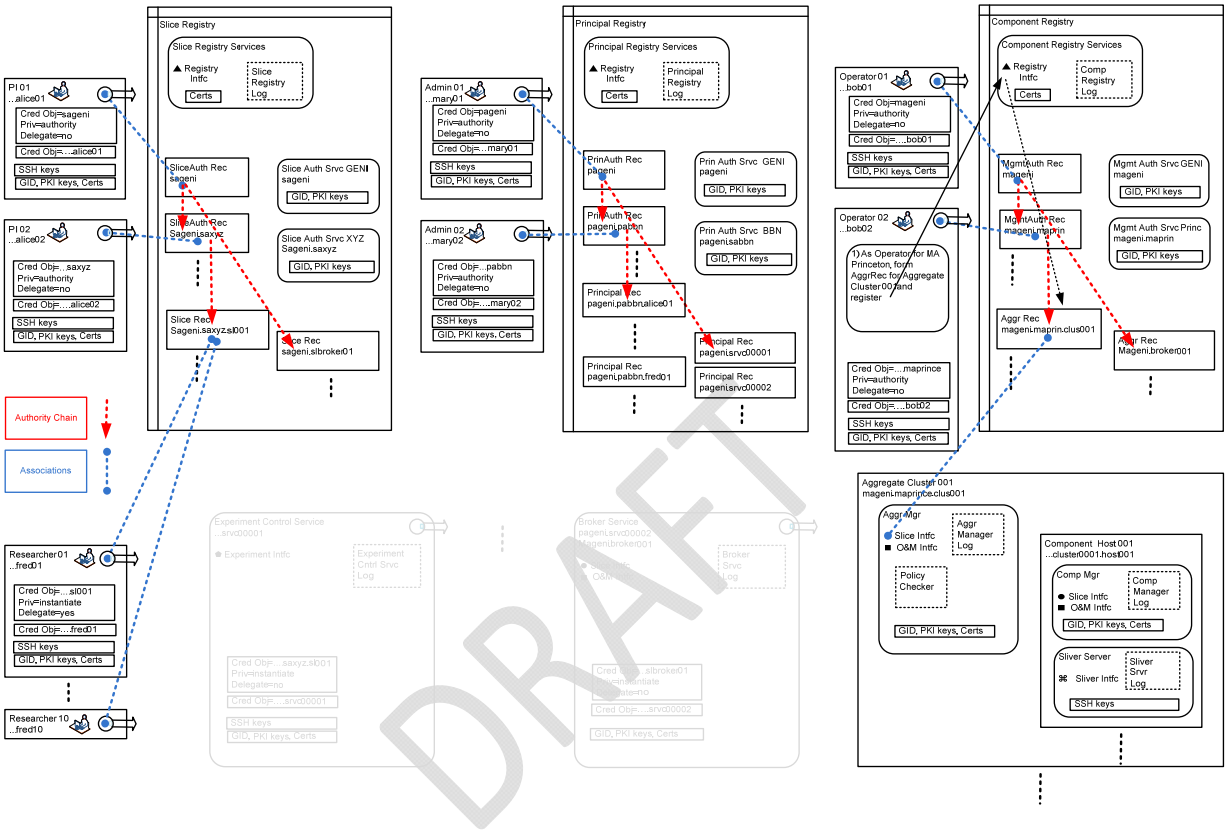

Figure 8-3. Registering an Aggregate

9 Token Flows for Resource Authorization

9.1 Overview

As indicated in Section 5.8, the current approach to authorization in the GENI suite is to

- Utilize the exchange of tokens (called credentials, tickets, etc.).
- The tokens are used to permit access to registries and authority services.
- The tokens are used to mediate the resource authorization, plus assignment and management.
- The tokens must be signed (certified) by the appropriate authorities and objects (principals, aggregates and slices) to give them some intrinsic value.
- This is the approach that was developed in a trust management system; see [http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.7726].
- This is the approach taken in the SFA; see [http://groups.geni.net/geni/attachment/wiki/GeniControlBr/v1.10%20%20080808%20%20sfa.pdf].
- This is the approach taken in the research leading to the ORCA implementation; see refs [http://portal.acm.org/citation.cfm?id=1267377].
- It is clearly a very flexible approach.
- It permits the involved entities to be widely dispersed and independently operated, and it should help provide for "disconnected operation".

Each of the pending Spiral 1 implementations supports a different set of token flows, and each is summarized in the following five sections.

9.2 Token flows in Spiral 1 Implementation based on PlanetLab and the SFA

Token flows for resource authorization, assignment and management in the Spiral 1 implementation based on PlantLab and the SFA are summarized in Figure 9-1.

In this case, the Researcher has a slice credential that has been delegated to the Experiment Control Service, which then acts on behalf of the Researcher to obtain resources from an aggregate.

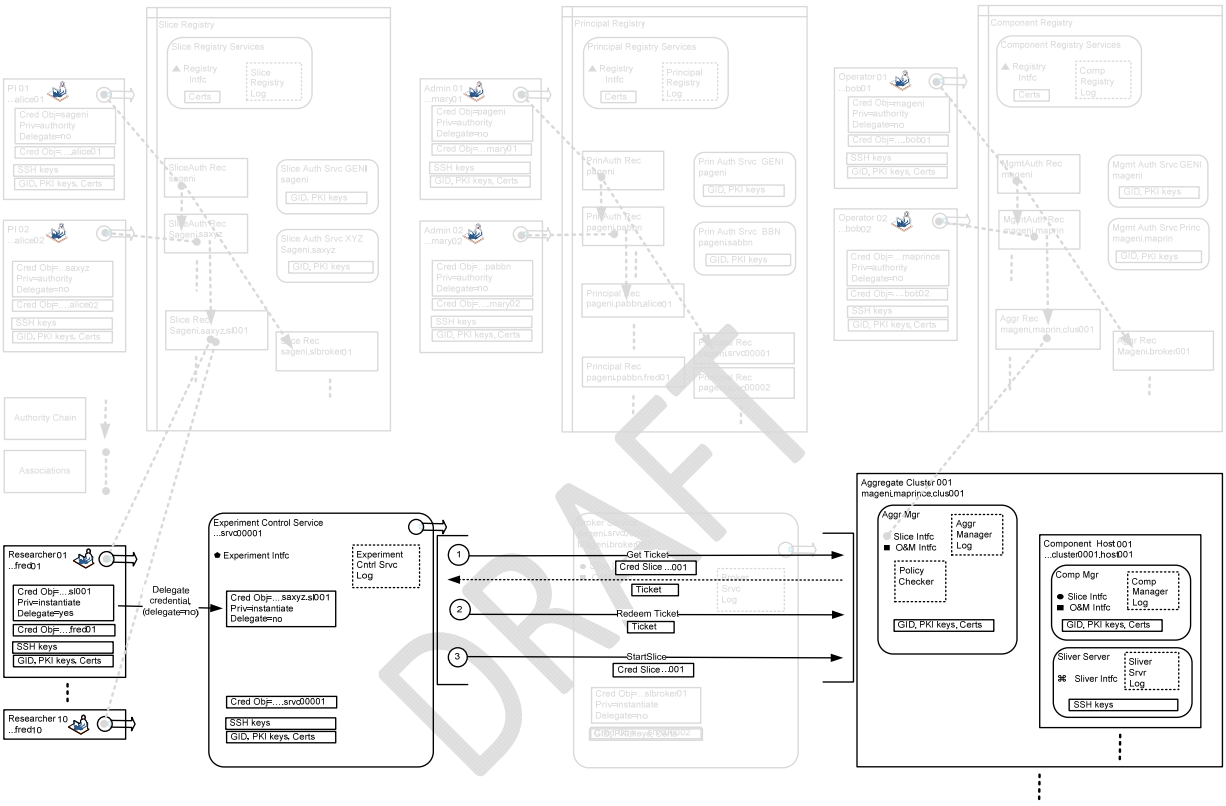

Figure 9-1. Token Flows for Resource Authorization, Assignment and Management per PlanetLab and the SFA

9.3 Token Flows in Spiral 1 Implementation based on ProtoGENI

Token flows for resource authorization, assignment and management in the Spiral 1 implementation based on ProtoGENI are summarized in Figure 9-2.

In this case, the Researcher has a slice credential that has been delegated to the Experiment Control Service, which then acts on behalf of the Researcher to obtain resources from an aggregate.

When the Ticket is redeemed, and the Aggregate Manager has assigned resources for the Researcher, a sliver object is created in the Aggregate Manager, and it issues a sliver credential, for use by the Researcher to start the sliver, etc.

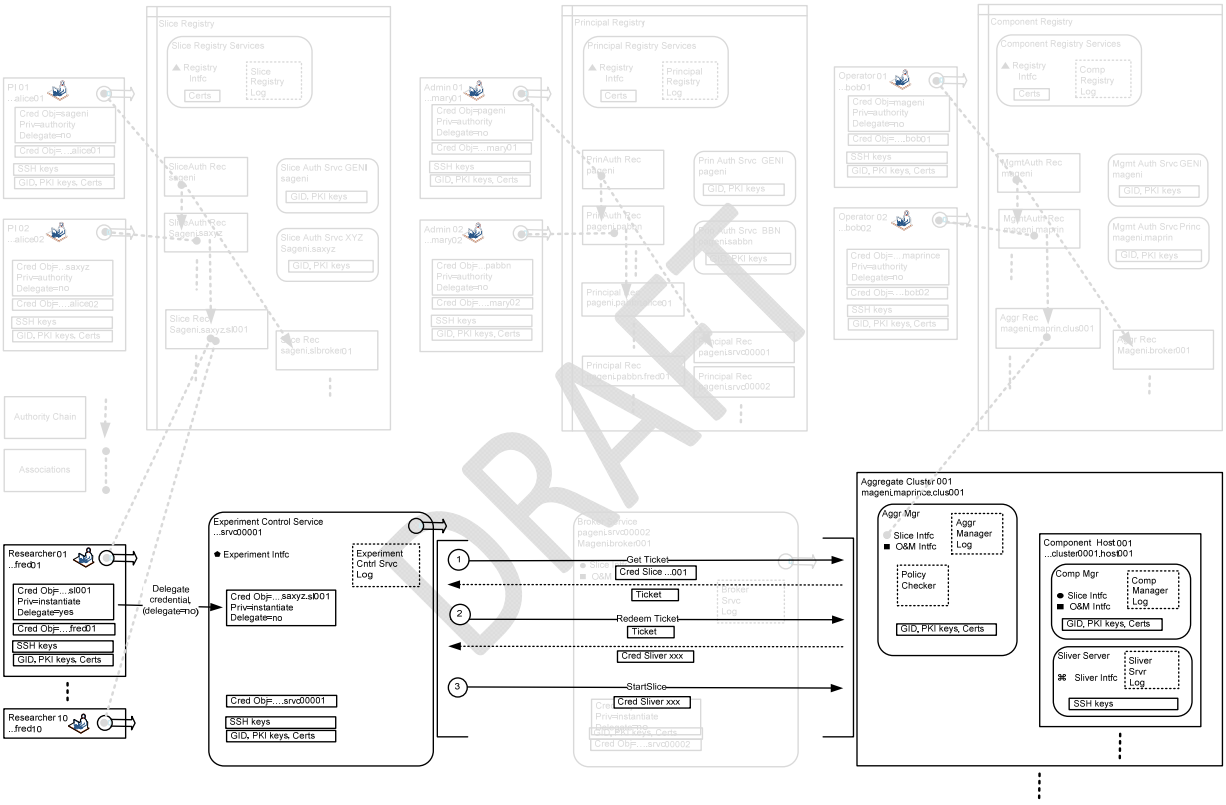

Figure 9-2. Token Flows for Resource Authorization, Assignment and Management per ProtoGENI

9.4 Token flows in Spiral 1 Implementation based on ORCA

Token flows for resource authorization, assignment and management in the Spiral 1 implementation based on ORCA are summarized in Figure 9-3.

In this case, the Researcher has a slice credential that has been delegated to the Experiment Control Service, which then acts on behalf of the Researcher to obtain resources from an aggregate.

As shown in ORCA [http://portal.acm.org/citation.cfm?id=1267377], there is a broker service that gathers tickets from aggregates, and delivers them to researchers via the experiment control service. After a ticket is redeemed at the aggragate manager, a firm assignment of resources is made, and a lease is returned to the Researcher/ Experiment Control Service.

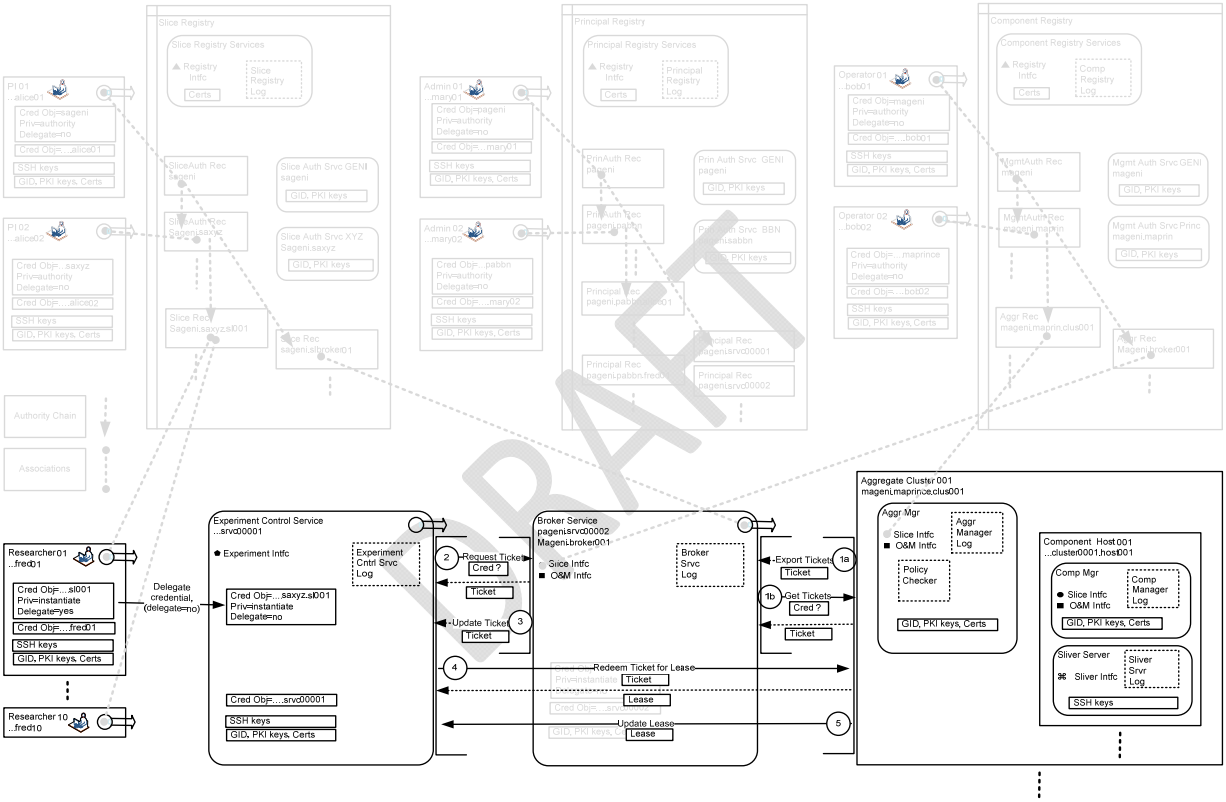

Figure 9-3. Token Flows for Authorization per Token Flows for Resource Authorization, Assignment and Management per ORCA

9.5 Token flows in Spiral 1 Implementation based on DETER

Gap: To be provided.

9.6 Token flows in Spiral 1 Implementation based on ORBIT

Gap: To be provided.

10 Slice Interface API

10.1 Overview

Once a slice has been registered with a trusted slice authority, any Researcher bound to the slice can retrieve a credential giving it the right to invoke the following operations on one or more aggregates to instantiate and activate the slice by establishing slivers on each aggregate.

10.2 Example from SFA: Supported Operations

This example is based on the SFA document; see ref [http://groups.geni.net/geni/attachment/wiki/GeniControlBr/v1.10%20%20080808%20%20sfa.pdf].

The slice interface on an aggragate supports the following operations.

All require the caller to submit a Credential with the privileges to allow it call that operation. From Section xxx above, it can be seen that a Caller who is an Admin, PI or Operator has been granted an Authority privilege, which allows them to call any of these operations.

10.2.1 Instantiating a Slice

A combination of four operations are used to instantiate a slice:

Ticket =	GetTicket (Credential, RSpec)
(no returned value)	RedeemTicket (Ticket)
(no returned value)	ReleaseTicket (Ticket)
(no returned value)	InstantiateSlice (Credential, RSpec)

A user invokes the first operation on a component to acquire rights to component resources. The returned ticket effectively binds the slice to the right to allocate on that component the requested resources. Whether or not the call succeeds depends on the local resources available on the component, and the resource allocation policy implemented by the component (on behalf of the component owner). The Credential parameter identifies the slice or slice authority requesting the resources, and indicates the period of time for which the slice's registration is valid; the component likely limits the returned ticket's duration accordingly. The Credential must include the instantiate or bind privilege.

Once a principal possesses a ticket, it can create a sliver on the component and bind new resources to an existing sliver by invoking the RedeemTicket operation. Creating a new sliver requires the instantiate privilege and augmenting an existing sliver with additional resources requires the bind privilege. The ReleaseTicket call undoes a ticket allocation.

Alternatively, a caller can embed a slice with a single InstantiateSlice call. This call is essentially equivalent to back-to-back GetTicket/RedeemTicket calls.

Note that RedeemTicket and SplitTicket (next section) are the only operations that do not take a Credential as an argument. Instead, both take a Ticket, which effectively plays the role of a credential in the sense that it says what set of resources the corresponding principal has the right to allocate or bind. A principal must have the instantiate or bind privilege to call GetTicket, but once a ticket exists, the principal to whom the resources are bound may call SplitTicket.

10.2.2 Provisioning a Slice

Three operations are used to manipulate the resources bound to a slice:

NewTicket = SplitTicket(Ticket, GID, RSpec)
(no returned value) LoanResources(Credential, GID, RSpec)
(no returned value) UpdateSlice(Credential, RSpec)

An entity that holds a ticket uses the first operation to split off a portion of the corresponding resources, effectively creating a new ticket. The GID parameter specifies the slice to which the ticket's resources are to be bound. Note that splitting a ticket requires calling the entity that originally issued the ticket, independent of how many times the ticket has previously been split. (In contrast, a credential can be delegated locally, without contacting the issuer of the credential.) This new ticket can be redeemed using the RedeemTicket operation (described above); resulting in either a new slice being instantiated on the component or additional resources being bound to an existing slice.

A slice uses the second operation to loan some of its current resources to the specified slice. A slice can learn its allocation on the component using the GetSliceResources operation (described below). Loaned resources are transferred from one slice to another without being encapsulated in a ticket.

A user invokes the third operation to request that additional resources—as specified in the RSpec—be allocated to the slice. Note that UpdateSlice and InstantiateSlice can be viewed as alternative name for the same operation: the former creates the slice if it does not already exist, while the latter updates the slice if it already exists.

10.2.3 Controlling a Slice

Component managers support four control operations:

(no returned value) StopSlice(Credential)
(no returned value) StartSlice(Credential)
(no returned value) ResetSlice(Credential)
(no returned value) DeleteSlice(Credential)

where the Credential parameter passed to all four operations identifies the slice being controlled. The first two operations stop and start the execution of an existing slice. The slice retains any acquired resources on the component, although a component that uses work-conserving schedulers is free to utilize those resources for the duration of the suspension. The slice should not expect the threads running in the slice to resume at the point the slice was suspended, as the implementation of StopSlice is free to kill all running threads, in which case, StartSlice effectively reboots the slice. However, the slice's on-disk state should remain unaffected by the operations. The third operation resets a slice to its initial state. This includes clearing any on-disk state associated with the slice. Thus, ResetSlice is effectively equivalent to deleting and re-creating the slice on the component, but without freeing the slice's resources. The fourth operation removes the slice from the component and releases all of its resources.

> **Note: Does a freshly instantiated slice/sliver start in the suspended state (and hence, one must invoke the StartSlice operation to "boot" it), or is each active sliver in a slice automatically booted when it is instantiated?**

Note that these operations might be invoked by a user responsible for the slice (e.g., a researcher associated with the slice with the slice or the PI that vouched for the slice), or by a user responsible for the component (e.g., an operator affiliated with the MA). In the latter case, the operator might not know that the slice exists on the component, but is terminating or suspending the slice on all components it

manages. This permits an operator to control a slice on all of the components it manages without the cooperation of a slice manager that knows all the components on which the slice has been embedded.

10.2.4 Slice Information

Components support three informational operations:

SlicesNames[ ] = ListSlices(Credential)

RSpec = ListComponentResources(Credential)

RSpec = GetSliceResources(Credential)

They are used to learn the HRNs for the set of slices instantiated on that component, the resources available on the component, and the set of resources bound to a particular slice, respectively. All three calls require a Credential, but for ListSlices and ListComponentResources, it is reasonable for components to return the requested information to any caller with a legitimate GID.

In practice, the ListComponentResources and GetSliceResources operations, in conjunction with GetTicket, can be used by a slice (or a slice manager running on its behalf) to (a) learn what resources are available on a given component, (b) request a collection of resources be allocated to the slice on that component, and (c) determine precisely what resources the component assigned to the slice. This sequence can be repeated to incremental acquire the desired resources.

Note that when ListComponentResources is invoked on an aggregate, the caller is able to learn the set of components available within that aggregate. This information is likely to be both more detailed and more dynamic than the component information available in a registry.

A fourth operation

SliceName = GetSliceBySignature(Credential, Signature)

where

Signature = (StartTime, EndTime, Protocol, SrcPort, SrcIP, DstPort, DstIP)

is used to learn the HRN for the slice that sent a particular packet onto the Internet. It is meaningful only on a component that is able to forward packets to/from the legacy Internet.

10.3 Tickets

A component signs an RSpec to produce a *ticket*, indicating a promise by the component to bind resources to the ticket-holder at some point in time. Such tickets are "issued" by a component, and later "redeemed" to acquire resources on the component. Tickets may also be "split," effectively passing resources from one principal to another.

The SFA defines tickets to include the following information:

Ticket = (RSpec, GID, SeqNum)

where RSpec describes the resources for which rights are being granted by the component; GID identifies the slice or slice authority to which rights to allocate the resources are being granted; and the SeqNum ensures that the ticket is unique. This information is signed by the component that issues the ticket.

The ProtGENI implementation defines tickets to include the following information:

```
## 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.
   element type { "privilege" | "ticket" },
   ## 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 ),
   ## Optional Extensions
   element extensions { anyelementbody }*,
   ## Parent that delegated to us
   element parent { credentials }?
}

signatures = element signatures {
   element sig:Signature { ... }+
```

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

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.

10.4 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 exact form of an RSpec is still being defined elsewhere [ref].

Per the SFA, each RSpec includes the following two fields:

(StartTime, Duration)

indicating the period of time for which the requested resources are desired (or granted resources are available). By default, StartTime=Now and Duration=Indefinite.

Per the ORCA implementation, (StartTime, Duration) should be included in the Ticket, but not in the RSpec.

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 addition APIs that are specific to these other resources.

The ProtoGeni rspec is based on the ptop/vtop format designed for Emulab's "assign" resource mapper. 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 ProtoGeni rspec.

Since Emulab clusters exporting the ProtoGeni APIs, are exporting an 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 geni.emulab.slice0.link0;

to get a credential for an interface, geni.emulab.slice0.node0.eth0.

10.5 Example from SFA: Getting and Redeeming Tickets

This example is based on the SFA document; see [http://groups.geni.net/geni/attachment/wiki/GeniControlBr/v1.10%20%20080808%20%20sfa.pdf].

Figure 10-1 summarizes the process where a Researcher via an Experiment Control Service instantiates a slice on an aggregate by using GetTicket call, followed by a RedeemTicket call.

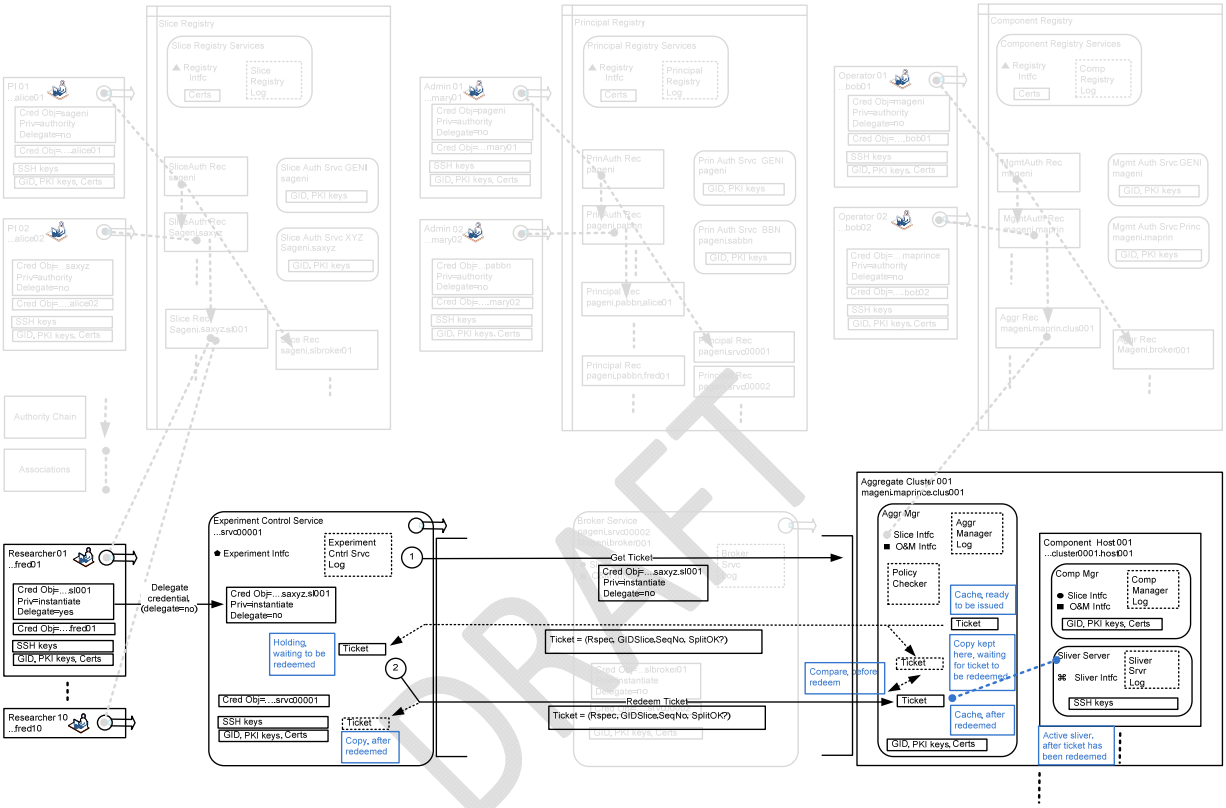

Figure 10-1. Getting and Redeeming Tickets

10.6 Example from SFA: Aggregate Applying Local Policy

This example is based on the SFA document; see [http://groups.geni.net/geni/attachment/wiki/GeniControlBr/v1.10%20%20080808%20%20sfa.pdf].

Figure 10-2 summarizes the process where an aggregate applies local policy to decide whether to issue a Ticket, and then later whether to redeem a Ticket.

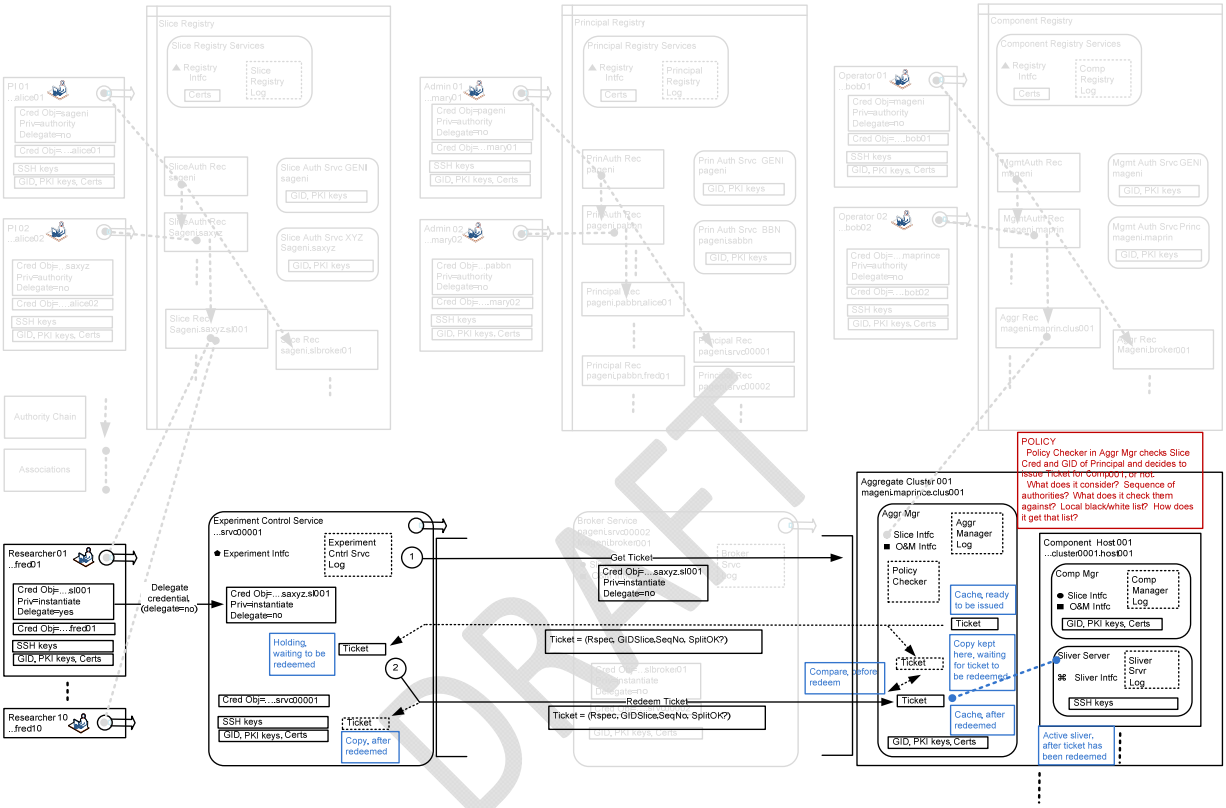

Figure 10-2. Policy Application when Redeeming a Ticket

10.7 Example from SFA: Activating a Sliver

This example is based on the SFA document; see [http://groups.geni.net/geni/attachment/wiki/GeniControlBr/v1.10%20%20080808%20%20sfa.pdf].

Figure 10-3 summarizes the process where a Researcher via an Experiment Control Service activates a slice on an aggregate by using the StartSlice call, etc.

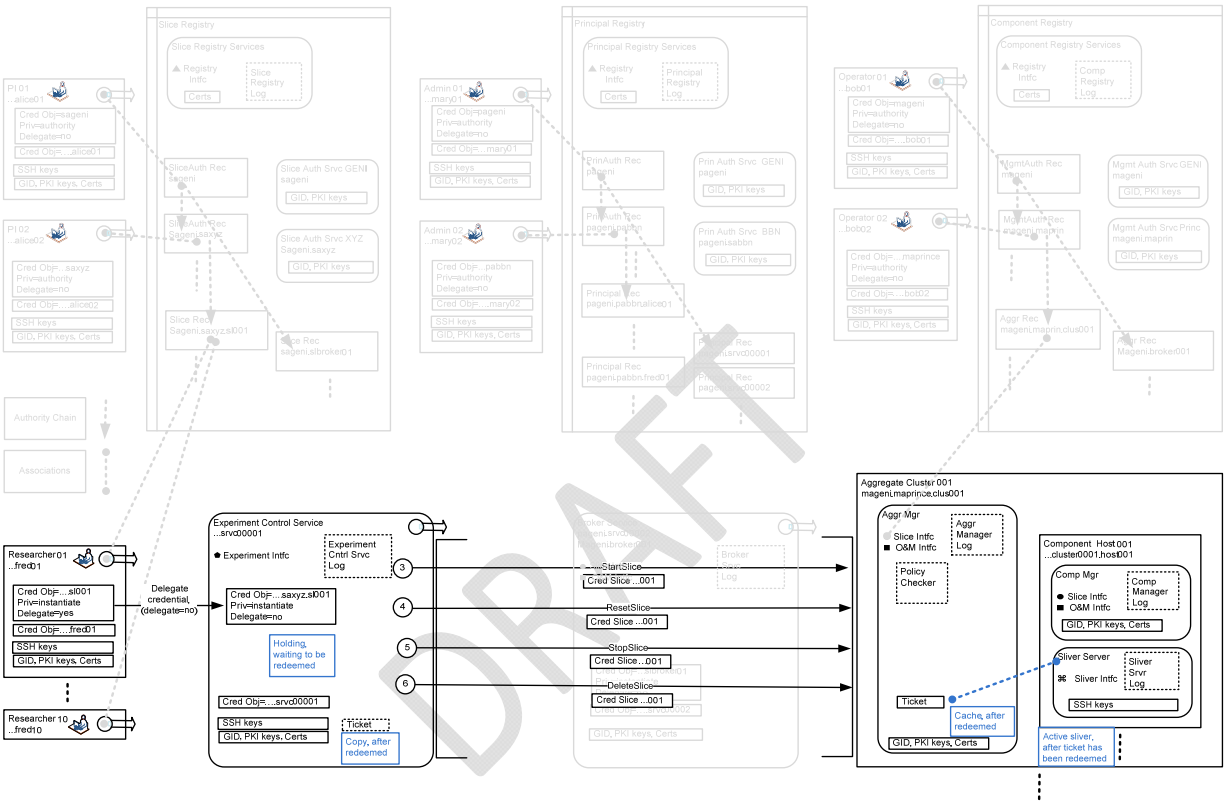

Figure 10-3. Activating a Sliver

11 Ops&Mgmt Interface API

11.1 Overview

An aggregate typically supports an Ops&Mgmt Interface, which is typically used by an Operator as associated in the GENI component registry to manage the aggragate. In addition, it is expected that there will be local operators for aggregates.

A component management interface is used to boot and configure components, bringing them into a state that they can support the slice interface. The interface is also used to bring the component into a safe state should the component be compromised. Both individual components and aggregates representing a set of components can be expected to support the management interface.

11.2 Example from SFA: Supported Operations

This example is based on the SFA document; see [http://groups.geni.net/geni/attachment/wiki/GeniControlBr/v1.10%20%20080808%20%20sfa.pdf].

The slice interface on an aggragate supports the following operations.

All require the caller to submit a Credential with the privileges to allow it call that operation. From Section xxx above, it can be seen that a Caller who is an Operator has been granted a Management privilege, which allows them to call any of these operations.

The management interface includes three operations:

SetBootState(Credential, State)

State = GetBootState(Credential)

Reboot(Credential)

The first operation is used to set the boot state of a component to one of the following four values: debug (component fails to boot, but should keep trying), failure (component is experiencing hardware failure, and so is taken offline until a human intervenes), safe (component available only for operator diagnostics), or production (component available for hosting slices). The second operation is used to learn a component's boot state and the third operation forces the component to reboot into the current boot state.

Note that we expect a given component (or aggregate) to support a much richer set of management-related (O&M) operations, effectively extending the required operations listed here. The management interface defines only the minimal set of operations **all** components (including aggregates and proxies) must support.

11.3 Example from SFA: Managing a Component

This example is based on the SFA document; see [http://groups.geni.net/geni/attachment/wiki/GeniControlBr/v1.10%20%20080808%20%20sfa.pdf].

Figure 11-1 summarizes the process where an Operator manages an aggregate using the Reboot call, etc.

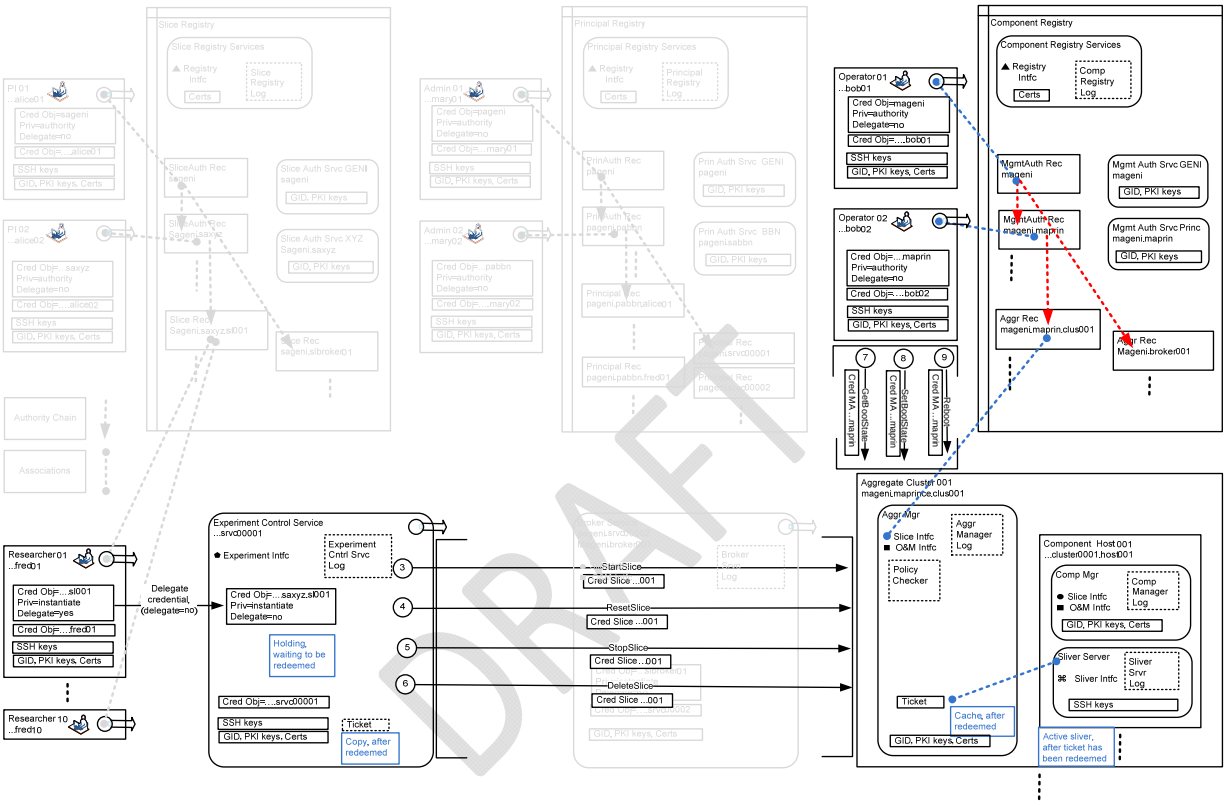

Figure 11-1. Managing a Sliver

12 Spiral 1 Implementation based on PlanetLab

(provided by Larry Peterson)

PlanetLab supports a prototype implementation of the abstractions and interfaces defined in this document. This section outlines a PlanetLab-centric "projection" of the slice-based facility architecture.

PlanetLab Central (PLC) bundles together an aggregate and a registry server. Individual PlanetLab nodes correspond to components. Both the PLC aggregate and each node component export the slice interface.[1]

The PlanetLab Consortium serves as a top-level slice and management authority. Sub-authorities correspond to member institutions, as well as federated partners. For example, planetlab.princeton.codeen is the human-readable name for the CoDeeN slice from Princeton, planetlab.vini.nyc.node1 is the HRN for a component in the VINI backbone, and planetlab.eu.inria is the HRN of a slice authority within the PlanetLab Europe sub-authority.

12.1 Engineering Decisions

As a working system, PlanetLab has made certain engineering decisions. This section outlines these decisions and their implications. We identify three key design decisions.

1. PlanetLab maintains all authoritative state at PLC. Individual nodes maintain only cached state that must be updated should the node fail and subsequently reboot. This means, for example, that any RedeemTicket or LoanResources operations invoked on a node must be re-invoked whenever the node reboots. Note that each node does have persistent storage that records certain information for the slices it hosts (e.g., the fact that the slice exists and is mapped to a particular virtual machine), but this state may become out-of-date during the time a node is down.
2. Nodes implicitly delegate control over their resources to PLC (the aggregate), which is responsible for implementing PlanetLab's resource allocation policy. As a consequence, the GetTicket, InstantiateSlice, and UpdateSlice operations succeed on PLC, but fail when invoked on individual nodes. Technically, these per-node invocations are return a "no available resources" message in response to requests to allocate resources since they have relinquished control over their resources to PLC. Individual nodes do, however, support the RedeemTicket and LoanResources operations, so it is possible to get a ticket from PLC and then redeem it on individual nodes. Both PLC and individual nodes support all other operations defined by the slice interface.
3. Tickets are idempotent. This means no matter how many times one redeems a ticket granting a slice 1Mbps of link bandwidth, for example, the slice is granted only 1Mbps of link bandwidth. In other words, tickets specify absolute resource capacity, rather than relative or incremental capacity. On the other hand, the LoanResources operation does increment a slice's resource allocation by the amount given in the RSpec.

PlanetLab's current resource allocation policy is fairly simple. Most slices are granted "best effort" resources by default. The policy recognizes only select slices as qualifying for guaranteed resources. One of these corresponds to the Sirius Reservation Service, which subsequently uses the LoanResources operation to grant other slices link and CPU guarantees for one-hour time slots.

[1] The GENI literature refers to a Clearinghouse, which can be viewed as a bundle of related software packages—e.g., an aggregate manager and registry server—and a "trust anchor." PLC can be viewed as an example GENI Clearinghouse on both counts.

PlanetLab supports an extensive O&M interface that goes well beyond anything defined in this document. This is a private interface known only to PlanetLab operators. One can view the management interface defined in Section 6.3 as a small subset of this PlanetLab-specific O&M interface that is common to all components participating in a federated slice-based facility.

12.2 Usage Scenarios

This section walks through a sequence of usage scenarios showing how PlanetLab might evolve to take advantage of the SFA to support both federation and third-party user services. Throughout this section, we use the notation outlined in Figure 8.1.

Figure 8.1: **Notation used throughout this section, including both interfaces and managers.**

Note we introduce two new constructs not defined elsewhere in this document. First, the *uber researcher interface* provides a high-level interface (possibly GUI-based) that researchers interact with to set up, control, and tear down their slices. This interface is not one of the standard SFA-defined interfaces, although it likely extends the slice interface. For example, it might allow users to manipulate graphical representations of their slices, it might iteratively discover and acquire resources, and it might help users steer the experiments running in those slices. Second, a *slice manager* is a module that manages slices on behalf of users. We assume it exports the uber researcher interface, and that it is the agent in the system that keeps track of where a given slice has been instantiated. It is not essential that either of these constructs exist. For example, users might manage their own slices by running a tool on their desktop that directly invokes operations on the slice interface exported by various aggregates and components. We believe, however, that one or more slice management services, each exporting an interface tailored to a particular user community, is likely to emerge. We represent this set with the SM module and uber researcher interface throughout this section.

Note that with the exception of the first scenario (vanilla PlanetLab), this section outlines the planned evolution of PlanetLab, not the state of affairs today. The subsequent scenarios (except for the one illustrated in Figure 8.4) correspond to configurations currently supported in PlanetLab, but using PlanetLab-specific interfaces rather than the SFA interfaces defined in this document.

12.2.1 Vanilla PlanetLab

The first scenario, depicted in Figure 8.2, corresponds to a simple deployment of PlanetLab, in which a trivial slice manager (SM), an aggregate manager (AM), and a registry (R) are all bundled in PLC, with each node running a component manager (CM). In all the examples presented throughout this section, we focus on the slice-related records in the registry. Component-related records are also recorded in the registry, but we do not illustrate how these records are used in the following discussion.

(Currently, PLC manipulates these records on behalf the constituent components, with PLC and the components communicating using a private interface.)

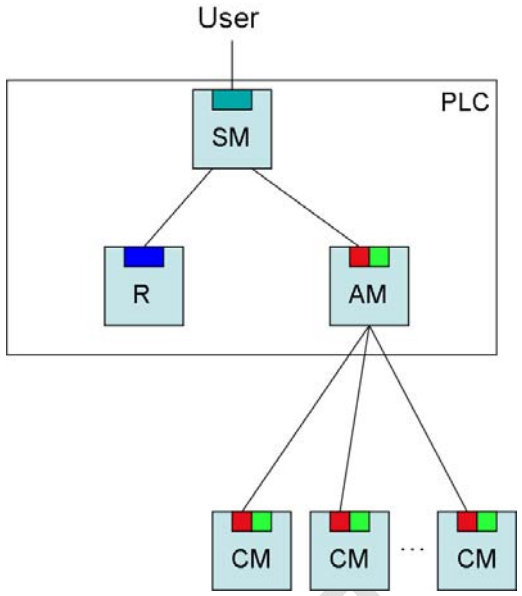

Figure 8.2: Vanilla PlanetLab, with bundled slice manager, registry, and aggregate manager.

In this example, users interact with the slice manager (using either a GUI or a programmatic interface) to create and control their slices. The slice manager contacts the registry to retrieve the necessary credentials, and then invokes the slice interface on the aggregate to create and control the slice. As is the common case in PlanetLab, the aggregate (rather than end users) interacts with the individual nodes. Note that the current implementation of PLC uses a private interface to interact with the individual components (although the components also export the slice interface to other clients).

12.2.2 Alternative Slice Manager

We can augment the simple scenario by allowing users to interact with alternative slice managers; in the example shown in Figure 8.3, one provided by Emulab. In general, users may employ any number of different slice managers, not just the simple one provided by PLC.

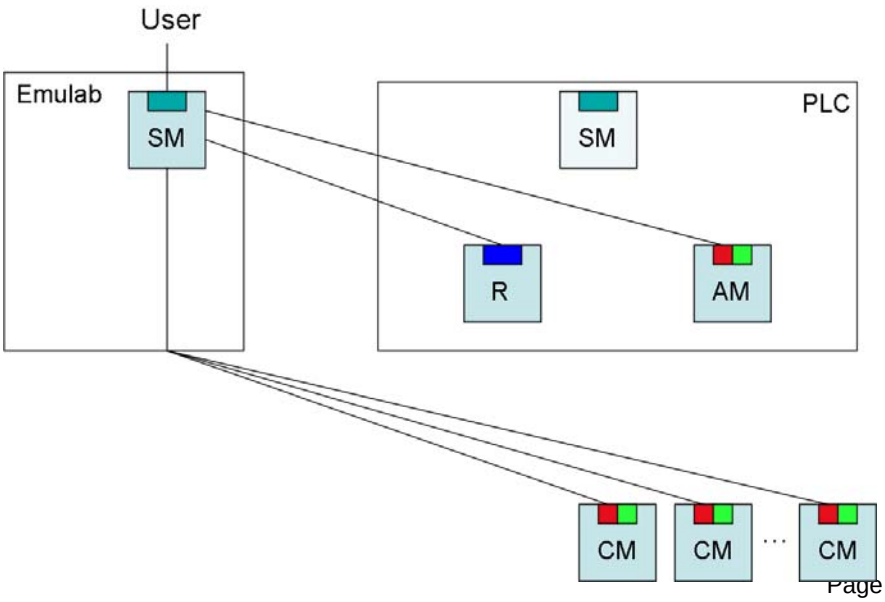

Figure 8.3: **PlanetLab nodes accessed from an alternative slice manager.**

In this scenario, the Emulab slice manager contacts the Planetlab registry to retrieve the necessary credentials. It then contacts the PlanetLab aggregate manager to retrieve a ticket for each slice it wants to instantiate. The Emulab slice manager then directly contacts the PlanetLab nodes to redeem these tickets, and later, to control the slices on those nodes. Because each node only caches slice-related state, the Emulab slice manager is responsible for ensuring that the slices it instantiates persist across node failures.

12.2.3 Common Registry

In another possible interaction between Emulab and PlanetLab, the Emulab slice manager may choose to trust users registered with PlanetLab—retrieving their credentials from the PlanetLab registry—but otherwise instantiate the slice purely on Emulab nodes. This allows Emulab to create experiments for PlanetLab users without requiring those users to separately registering with Emulab. This scenario is illustrated in Figure 8.4.

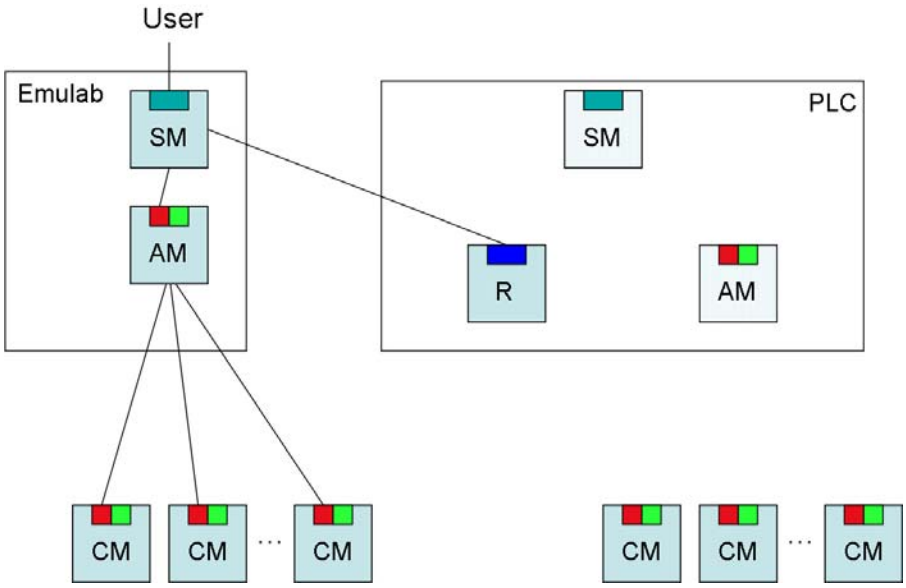

Figure 8.4: **Another testbed (Emulab) taking advantage of users and slices registered in the PlanetLab registry.**

12.2.4 Multiple Aggregates

The scenario depicted in Figure 8.5 spans multiple aggregates—PlanetLab and VINI—each responsible for its own set of components. That is, VINI and PlanetLab are distinct management authorities, each responsible for a distinct aggregate of components. In this case, VINI does not operate its own registry or slice manager, and PlanetLab's slice manger presents users with a unified view of all the components available on both systems, hiding the fact that its global view spans multiple aggregates.

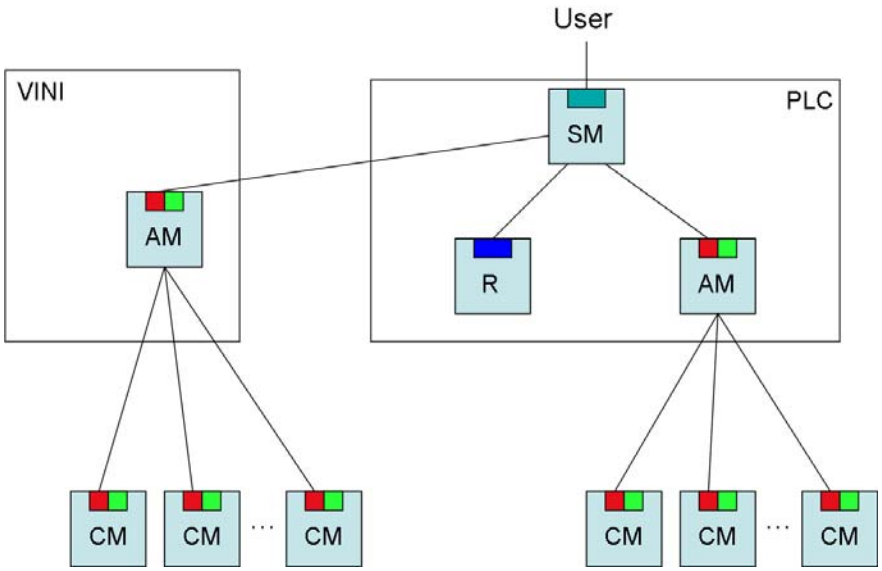

Figure 8.5: **VINI and PlanetLab represent independent aggregates (and corresponding management authorities), unified by a single slice manager.**

To create a slice, the PlanetLab SM would need to contact both available aggregate managers to learn about the available components. It would then present these components to the user in an SM-specific way. Once the user selects the set of components to be included in his or her slice, the SM would call the SR to retrieve the necessary credentials, and then invoke the InstantiateSlice operation on the respective aggregates to create the cross-aggregate slice.

12.2.5 Full Federation

Our final scenario, shown in Figure 8.6, involves symmetric federation between two autonomous aggregates, one representing PlanetLab Europe (PLE) and the other representing the rest of PlanetLab (PLC). Both systems support their own slice manager, registry servers, aggregate manager, and set of components. As in the previous scenario, users interact with their "local" SM, which creates and manages slices spanning both aggregates.

Although not explicitly depicted in the figure, the PLC registry points to the PLE registry. That is, registry records for the top-level PlanetLab authority, including the record for the EU sub-authority, are maintained in the PLC registry, while records associated with the EU sub-authority are maintained in the PLE registry server.

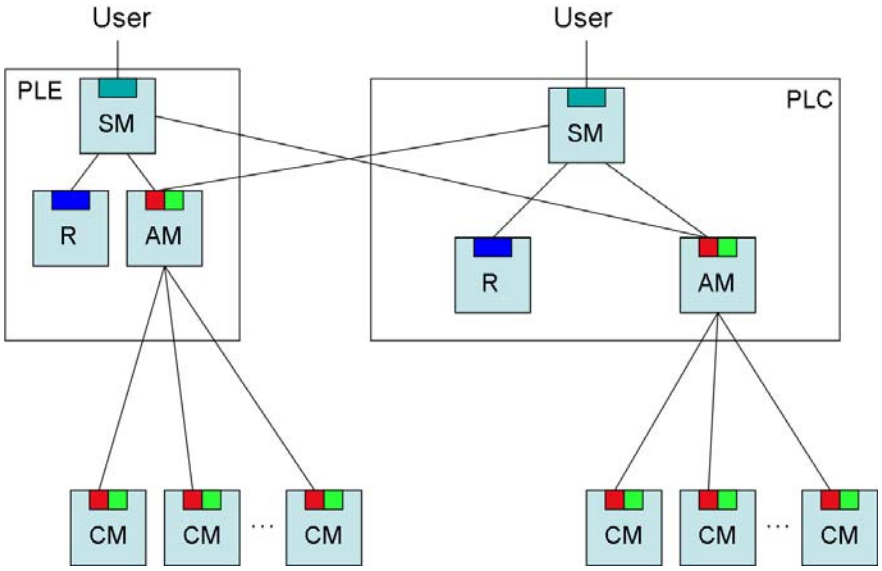

Figure 8.6: **Peer testbeds (PLC and PLE) federate their aggregates.**

13 Spiral 1 Implementation based on ProtoGENI (Emulab)

(provided by Leigh Stoller)

Emulab's ProtoGeni implementation strives to combine Emulab cluster nodes, Federated Emulab nodes, and other Geni resources, seamlessly into a ProtoGeni experiment. Towards this end, we have diverged slightly from the specification contained in this document. We have also made a number of engineering decisions that are consistent with a "prototype" implementation.

ProtoGeni, when running on an Emulab instance, is fundamentally an aggregate of resources, consisting of cluster nodes, switches and other resources within an Emulab installation. Both a registry and (aggregate) component interface are exported from Emulab clusters, but individual resources such as nodes and switches, do not export a component interface; all operations on individual resources are via the aforementioned CM at the cluster Private communication channels are then invoked to handle any operations required on individual resources.

ProtoGeni implements a centralized ClearingHouse registry on Utah's Emulab cluster. Users and Slices are registered in the Clearinghouse by the Slice Managers at each Emulab instance.

ProtoGeni employs a much simplified trust model that is more consistent with a prototype implementation. 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. 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 spec and so that we can inter-operate more easily with other Geni Sprial 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.

An ProtoGeni credential is currently defined as:

```
## 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.
  element type { "privilege" | "ticket" },
  ## 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 ),
    ## Optional Extensions
    element extensions { anyelementbody }*,
    ## Parent that delegated to us
    element parent { credentials }?
}

signatures = element signatures {
    element sig:Signature { ... }+
}
SignedCredential = element signed-credential {
    credentials,
    signatures?
}
```

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.

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

A new UpdateSliceAttributes() function has been added to allow 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.

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.

ProtoGeni supports a variety of hardware resources in addition to nodes. 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 addition APIs that are specific to these other resources.

The ProtoGeni rspec is based on the ptop/vtop format designed for Emulab's "assign" resource mapper. 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 ProtoGeni rspec.

Since Emulab clusters exporting the ProtoGeni APIs, are exporting an 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 geni.emulab.slice0.link0;

to get a credential for an interface, geni.emulab.slice0.node0.eth0.

A problem with the above relates to the use of GNAMES. A GNAME is defined to correspond to a chain of authorities, but a slice is not an authority, nor is a node. One possibility is to treat the last N components of the GNAME as non-authoritative when verifying the chain of credentials (verification stops when a component does not correspond to GID in the chain). But this approach needs more thought.

14 Spiral 1 Implementation based on ORCA (Shirako)

(provided by Jeff Chase)

Open Resource Control Architecture (Orca) emphasizes emphasizes modular policies for resource management and slice adaptation. Orca derives from the SHARP architecture for resource leasing contracts with accountable ticket brokering.

The core of the Spiral 1 implementation is a Java-based resource leasing toolkit called Shirako. Shirako is the basis for Java reference implementations of a GENI clearinghouse, aggregate managers, and programmable slice controllers. These actors in the GENI control framework match the respective Orca/Shirako actor roles: broker, domain authority, and service manager (guest controller). The actors run as SOAP servers exchanging digitally signed messages (WS-Security). Each actor has a web control portal interface for the user or operator.

We made several engineering decisions to simplify and focus the effort for the Spiral 1 prototype:

All components are aggregated and controlled by some aggregate manager. The aggregate manager also subsumes the role of Management Authority for the substrate resources that it controls. For Spiral 1 we plan to demonstrate aggregate managers for edge clusters, an optical connectivity provider (RENCI's Breakable Experimental Network, BEN), sensor testbeds, and a mobile testbed (DieselNet).

The edge clusters use virtual machine slivering (e.g., based on Xen hypervisor), in conjunction with a local virtual cluster manager (e.g., Eucalyptus). Node slivers (virtual machines) can boot any named image accepted by the local cluster manager.

The clearinghouse offers the optional ticket broker service, which issues all tickets to experiments. The strong broker role in the clearinghouse makes it possible to experiment with policies for coordinated resource allocation. All component aggregates delegate splittable tickets to the broker service, and attempt to honor any tickets issued by the broker.

The ticket broker service is the primary means to discover resources in the prototype, i.e., a slice controller discovers a component by requesting tickets for resources by attributes, and receiving ticket for component aggregates matching those attributes.

There is a single shared Slice Authority integrated with the clearinghouse. The component aggregate managers accept only slice identifiers endorsed by the designated slice authority, and they enable only the slice owners it endorses to operate on slices. The GNAME space is flat and names only slices.

The clearinghouse maintains a principal registry containing the public keys of identity providers and users registered by the clearinghouse operator. The clearinghouse accepts any user with a registered public key, or bearing an X.509 certificate endorsed by a registered identity provider.

The ticket broker policies arbitrate resource shares among user identities grouped by attributes. Resources include various sizes of virtual machines and connectivity options, e.g., between BEN sites.

Clearinghouse federation is not supported. Orca enables loose federations of substrate providers that delegate splittable tickets to a given clearinghouse for subsets of their resources over specific intervals of time, as directed by their operators. A given substrate provider may offer resources to multiple clearinghouses.

15 Spiral 1 Implementation based on DETER

Gap: To be provided.

16 Spiral 1 Implementation based on ORBIT

Gap: To be provided.