Design Notes On Using the GENIAPI to Build a TIED Federation Plug-in for the PlanetLab Control Framework

Ted Faber, USC/ISI

John Wroclawski, USC/ISI

Version 1.0

October 6, 2010

Corresponding to GENIAPI version 1.0.

Table of Contents

1 Introduction

This document continues our assessment of the emerging GENIAPI standards and implementation as a target plug-in API for the TIED/DETER[1] control architecture, and as a general GENI interoperability framework[1]. This version 1.0 of the document corresponds to GENIAPI version 1.0[2]. This document clarifies some of our earlier comments on how control frameworks and aggregates interrelate based on feedback from mailing list discussions, and discusses the limitations of the GENIAPI with respect to a future TIED/PlanetLab plug-in and by extension other GENIAPI consumers.

In our assessment of the GENIAPI as a unified framework – one that would support a unified PlanetLab/ProtoGENI TIED plug-in – we found several areas where the API needs to be modified or specified in order to make significant code sharing possible. The representation of resources needs to expose the slicing mechanisms used by the control framework; PlanetLab slicing and ProtoGENI slicing are fundamentally different and used for different kinds of experiments. The API must provide access to monitoring and layout tools within the control frameworks; many key PlanetLab tools are inaccessible to a plug-in. Finally interfaces to slice authorities – the entities that create the slice abstraction – must be created; PlanetLab and ProtoGENI have different authorization models with respect to slice creation and these interfaces need to provide authorization enforcement points for each as well as a common work flow for slice operations.

The TIED federation framework has been used successfully to construct experimental environments that span multiple dissimilar facilities. As one part of this process, a federated experiment environment is collaboratively constructed and then instantiated. In the instantiation phase, each facility maps the necessary control actions, resource access permissions and principal identities into local actions, access controls and identities. Facilities do this by implementing the required mapping functions and interfaces in the form of a *plug-in*, which is written to a standard specification[3]. TIED/DETER currently supports plug-ins, and thus can create federations across, the DETER testbed, Emulab[4] systems, ProtoGENI[5], deterministic-resource networks controlled by DRAGON[6], and a number of other resource categories.

This federation system has developed in parallel with the Slice-based Federation Architecture[2] (SFA) [7] originally outlined by members of the network testbed community under the auspices of the GENI Planning Group, and further developed by the community under the umbrella of the GENI Program Office and the GENI program.

The emerging GENIAPI[2][8] is a derivation and standardization of concepts from the SFA and the existing implementations of these concepts in the individual GENI control frameworks,

[1]The TIED/DETER control architecture is an evolving testbed control and federation architecture developed under the dual auspices of the DETER Cybersecurity Testbed project and the TIED GENI project. We occasionally refer to this architecture as either the "TIED" or the "DETER" architecture in cases where the dual TIED/DETER formulation is excessively awkward and not needed for clarity.

[2]Initially referred to as the "Slice-based Facility Architecture".

notably the ProtoGENI[5] and PlanetLab[9] efforts. The goal of the GENIAPI is to standardize an API for interoperability between existing and future control frameworks, components, and aggregates. Practically, the design and standardization aspects are balanced against documenting what the initial implementations have done and incorporating their insights. The GENIAPI aggregate manager interface, which we describe in more detail below, has been implemented to varying degrees in PlanetLab, ProtoGENI, and OpenFlow[10][11].

The purpose of this document is to describe the issues we encountered in assessing the usefulness of the GENIAPI in facilitating a PlanetLab plug-in for TIED. This document builds on our earlier assessment of the GENIAPI as TIED interoperability layer[1], beginning with an assessment of the goals and architecture of the GENIAPI.

1.1 Interrelating Control Frameworks and Aggregates

In order to understand and evaluate the GENIAPI we must understand how it relates to the SFA and GENI interfaces as a whole. The GENIAPI is an interface between a *control framework*, which implements the slice abstraction, and the experimenters that use it. The GENIAPI AM is the interface between those experimenter (or their agents) and resource *aggregates* that control federated resources that share an owner. *Slices* are collections of resources that the control framework acquires from resource aggregates. A collection of resources allocated within a single aggregate is called a *sliver*. At the request of a researcher, a control framework will allocate a slice and populate it with slivers from one or more aggregates. A primary goal of the GENIAPI is that aggregates can interoperate with multiple control frameworks[12].

Our earlier discussion treated control frameworks as monolithic, but decomposing the framework is helpful in understanding the design choices. In Figure 1 the control framework is broken into its constituents. The agent provides an interface to the experimenter as well as connecting to the GENIAPI-managed aggregates (AM) and the slice authority. The aggregates support the GENIAPI Aggregate Manager interface, used to advertise and allocate resources. The *slice authority* presents the slice interface. Specifically, the slice authority is responsible for creating and naming slices that span multiple aggregates. It is responsible for the bookkeeping necessary to create a single abstraction from a collection of resources.

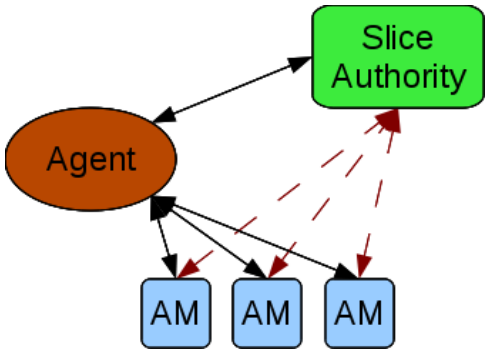

Figure 1: Control Framework

PlanetLab and ProtoGENI both implement this model. In PlanetLab, the agent is more monolithic, and it is addressed the same way when talking to different aggregates; in ProtoGENI, the existence of the various aggregates is exposed more directly to the user.

In both cases the slice authority and the aggregates share some state about the evolving slice – for example what AM's have contributed resources to the slice. The path for that information is underspecified; one can imagine that the agent passes information about successful allocations to the slice authority, or that a channel between the AM's and the slice authority exists (the dashed lines in Figure 1).

The current GENIAPI specifies the interface between the agent and the aggregates. Other interfaces are defined by local implementations – PlanetLab and ProtoGENI each have one documented between agent and slice authority, though neither describes the interface from aggregate manager to slice authority. Neither system requires the agent to act as a conduit for this information.

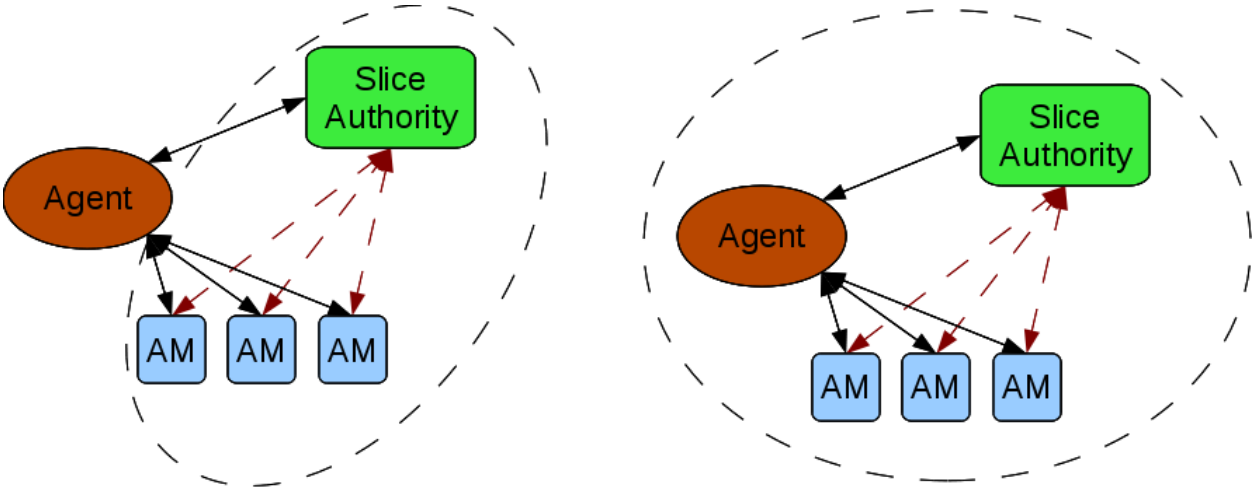

Figure 2: The Agent and the Control Framework

Depending on how they are used and what information they are privy to, agents may either be part of the control framework or users of those interfaces. An agent may embody some experimenter knowledge by exporting an interface that represents GENI resources in ways that a specific population understands as an *external agent*. Conversely an *internal agent* represents knowledge specific to the control framework, for example, how best to lay out a given experiment using the framework's resources, and may export an enhanced version of the GENIAPI interface.

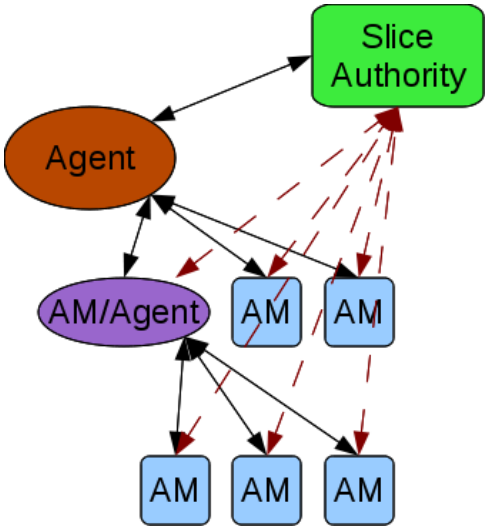

Figure 3: Recursive agents

External agents sit outside the GENIAPI-defined control framework as shown on the left side of Figure 2, while internal agents are part of that framework. Multiple external agents may independently operate on the same control framework, one external agent may compose experimental environments from many control frameworks.

The two sides of Figure 2 are not mutually exclusive. External agents can call into a control framework that has internal agents acting as agents, slice authorities and aggregates. This is the case we described somewhat opaquely as the recursive case in our earlier discussion[1]. An example of this is shown in Figure 3.

Given this decomposition of the control framework, one formulation for the interoperability goals of the GENIAPI is to support situations like the one pictured in Figure 4. An external agent collects resources from two control frameworks into a single experiment. As pictured, the external agent coordinates two slices to make this happen. One could imaging taking one control framework's slice authority out of the equation, but this would require one control framework's slice authority to understand information about slice composition from other framework's aggregate managers. This will not happen without mechanisms for AM's to represent their allocations to foreign slice authorities.

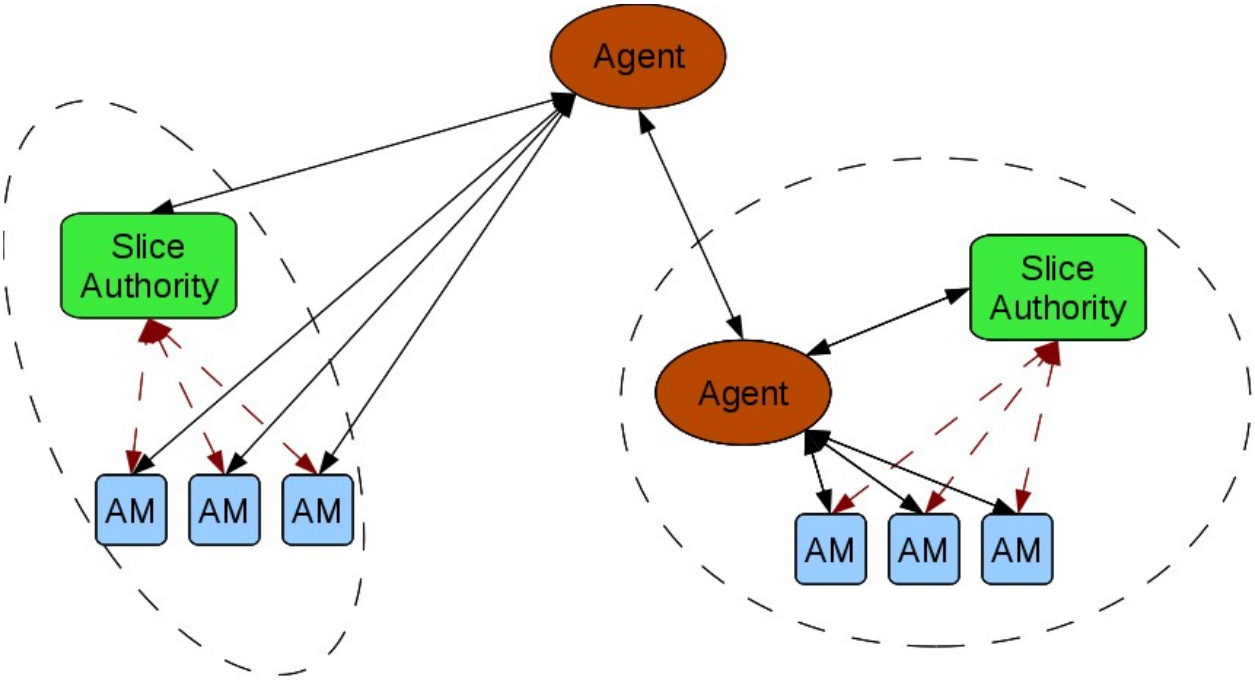

Figure 4: Interoperating Control Frameworks

The two observations we would like to highlight are:

- The design of smart agents inside control frameworks may be simplified by having them export the aggregate manager interface, assuming it is defined with that in mind
- The interfaces to the slice authority (agent/slice authority and AM/slice authority) must be defined.

The confusion about terminology and usage implies that there may be confusion in the GENI community about the role of the aggregate manager interface. Though the broad boundaries of the control framework as a whole, and the components of it are generally agreed upon, exactly what functions are required and how they will be structured continues to evolve.

This document proceeds by describing TIED plug-ins in Section 2 and the GENIAPI in Section 3, before discussing the specific lessons learned from evaluating the GENIAPI as a basis for a PlanetLab plug-in in Section 4. Sections 2 and 3 largely recapitulate the similar descriptions in our earlier discussion[1], with updates on changes since that document appeared. Our findings are summarized in Section 5.

2 TIED Federation and Plug-Ins

We briefly outline the architecture and goals of the DETER Control Architecture (DCA). The design document for the ProtoGENI plug-in[15] discusses this in more detail.

The DETER Control Architecture, on which DETER and TIED are based, supports as its basic abstraction a substantial generalization of the Utah Emulab[4] model of experiments and experiment creation. Some key properties of this model are as follows:

- Experiments consist of logical *nodes*, which may model many sorts of computing and communication resources, interconnected by abstract *links* and/or *LANs* to form a network topology in which the experiment is carried out. In the original Emulab model, nodes are implemented primarily using general-purpose computers, while interconnections are created using virtual networks implemented by off-the-shelf Ethernet switches. Recent advancements in DETER and Emulab extend this model somewhat.

- Experiments are generally isolated from one another, but make use of support services provided by the testbed infrastructure, such as file systems shared between experiment nodes and an event delivery system that enables loosely coordinated changes to the state of experimental nodes.

- There is a general communication path between experiment nodes and and testbed servers that can be used to remotely access the experiment interactively or programmatically. Depending on experimenter's comfort and familiarity with testbed services, either standard testbed services or more ad hoc systems may be used to carry out experimental procedures.

The DETER Control Architecture represents and implements a continuing evolution of this basic testbed model.

Given this environment, the basic process of creating an experiment (slice) in the TIED/DETER environment consists of three steps:

1. Acquiring access to individual testbeds consistent with local access control policies.
2. Allocating necessary resources to the experiment using local allocation strategies.
3. Forming the shared experimental connectivity and composing the required experiment services.

Each of these functions is implemented within the control architecture of the DETER/TIED system. Core elements of the DCA are shown in Figure 5 below.

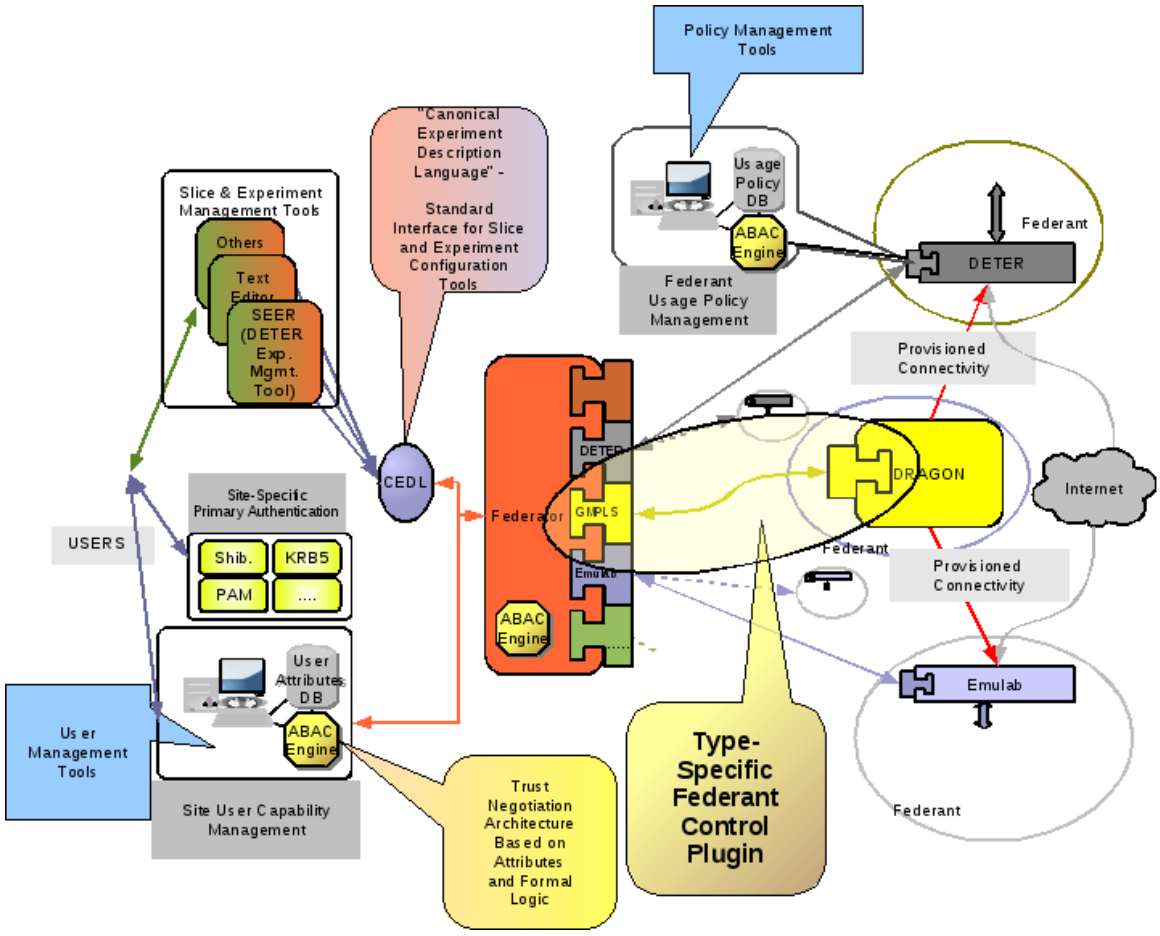

Figure 5: The DETER Control Architecture

Central to the DCA architecture is the *federator*. The federator and its control language, CEDL, serve as a "narrow waist" within the design, providing a unifying functional layer between, on the one hand, a broad range of specialized tools for user interaction and experiment configuration, and on the other, the interconnected resources of multiple physical facilities with different resources and capabilities. The federator acts as the interface between users who are creating and controlling an experiment (slice) and the various federants who have supplied the resources that constitute the slice. It presents users with a single interface for control, but translates the creation of sub-experiments/slivers into the configuration system of the local resource owners.

2.1 The Role of Plug-ins in the DCA

The previous section outlined the core elements of the DETER Control Architecture and described the central role of the federator in this architecture. In implementation terms, the federator is broken up into two parts, the *experiment controller*, which manages the slice or experiment as a whole, and an *access controller* for each testbed or facility that contributes resources. The access controller is responsible for translating the access control decisions from the global domain into local configurations, for using local resource allocation systems to bind resources to the experiment, and for configuring those resources to create the topology and relevant services for the experiment.

Because the function of a access controller is specific to the class of testbed or facility it is controlling, the access controller is implemented as a *plug-in*, with different plug-ins used to interface with different classes of facility. The plug-shaped icons in the figure show the location of these plug-ins within the DCA design. The figure shows that each plug-in supports two interfaces, a standard one to the federator and a testbed-specific one to the testbed itself.[3]

Each plug-in may be implemented and deployed in several configurations, depending on the operational and administrative requirements of the testbed being federated. The experiment controller and access controller may run on the same machine, with the access controller proxying requests back to the testbed it manages; the access controller may be co-located with the testbed and accessed remotely by the experiment controller; or the plug-in may run on a third machine unrelated to either the testbed it controls or the location of the experiment controller. One can think of these layouts as placing more or less functionality in each of the plug-ins in Figure 5.

Though each experiment/slice is controlled by one experiment controller, experiment controllers are fairly lightweight entities. They are responsible for splitting the experiment up between access controllers and managing the credentials of the researchers who are creating the experiments. None of these responsibilities require that the experiment controller run on testbed resources; experiment controllers that run on desktops and communicate with access controllers running on testbed nodes is a likely configuration.

3 The Slice-Based Federation Architecture and GENIAPI

The SFA document traces its history to the earliest days of the GENI development, and the architecture it describes reflects many of the fundamental ideas behind GENI. Its original goal was to define a minimal interface that conformant GENI implementations would export, and it has become a touchstone for the architecture in general.

The GPO-managed prototyping and implementation phase encouraged multiple implementations of the architecture defined in the forerunners of the current SFA document. These implementations were called *control frameworks*. Confronted with the loose definitions of core resource allocation and management functions, data structures, and authorization framework,

[3]In the figure it appears that plug-in code must be loaded in the federator code base itself, but this is a conceptual diagram rather than an implementation block diagram. What is important is the standard interface between federator and plug-in.

each control framework implementor made a set of design decisions and created a different system within the overall confines of the architecture. The most successful of these implementations, ProtoGENI and PlanetLab, extended their existing interfaces to include the SFA interfaces. Of course, the underlying resource models of these and other control frameworks strongly influenced how they interpreted and extended the SFA. While the implementations shared a spirit, they differ in significant details.

The GENIAPI project is an attempt to reunify the various control frameworks to the point where they can interoperate. The initial release of this effort is a specification and implementation of the aggregate manager API[16]. That implementation is not a strict implementation of the SFA aggregate manager interface (called the *slice interface* in the SFA document), and is, in many ways, an improvement. We briefly review the current SFA document and the GENIAPI interface.

3.1 The SFA Document

The SFA document defines the key entities, abstractions, and data flow for resource allocation in GENI. It continues to be regularly revised, and the current version includes notes from the authors[7].

The SFA lays out the basic players in the GENI ecosystem and then describes the key abstractions of *slice* and *sliver* that underly much of GENI discourse. The SFA describes a sliver as a collection of co-managed resources and slice as a collection of slivers and users bound to the collection. It describes the life cycle of a slice, all indicative of the slice's central role in experiment creation in GENI.

Components and *aggregates* are also defined as abstractions of co-managed resources that can be multiplexed (*sliced*). The component/aggregate distinction is somewhat slippery, and is based on whether one or more user-visible resources are managed by it. We generally use “aggregate” to refer to this abstraction.

Naming and identification of actors in GENI is discussed in its own section of the SFA document. A fairly intricate system of binding a principal to a public key, a universally unique identifier[17], and a lifetime is put forward. These identifiers are called GIDs. The authors note that none of the implementations have adopted this framework, and the section will be revised, though we will continue to use GID to indicate a principal identifier. The lack of agreement on naming and identity remains an interoperation problem.

The *Rspec*, *ticket*, and *credential* are laid out as fundamental data types. The Rspec is a resource specification used to request slivers and slices; a ticket is a signed Rspec bound to a GID and a unique identifier used to denote a reservation from an aggregate. Though it is recognized as a fundamental data structure, the Rspec's formats and requirements are not specified here.

Credentials are described as the binding of a particular privilege to a GID, and a set of privileges are defined. Section 8 of the SFA refines this definition, and we discuss the refined definition below.

Following this interfaces are defined, including the aggregate interface also defined and implemented by the GENIAPI. The interface describes the binding of slivers to slices, though the interfaces do not include an explicit parameter for the slice being bound. The binding is described in the functional descriptions. Aggregates are aware of slices and carry out the process of binding resources to them, which corresponds more closely to our recursive model.

The last two sections of the SFA are somewhat different in tone from those described above. Section 7 titled "Authorization and Access Control," is fairly heavily annotated with disagreements. The authors agree that there are authorities responsible for authorizing access to slice interfaces and separate authorities controlling resource allocation, but there seems to be less agreement on how these authorizations are expressed. Discussions of group rights and individual attributes are included.

The final section, "SFA Authorization Using Registered Capabilities" seems to be an example of realizing the architecture, but there is no such explicit statement. The section introduces a new architectural element – the *registry* – and expands the definition of credentials. This separation of slice manipulation from earlier allocation mechanisms is closer to our two-level model.

A registry contains information bound to GIDs, including human-readable-names (HRNs). The HRN defines a set of entities responsible for validating the identity of the principal having that GID. This separation of validation information from the identity is somewhat confusing, and at odds with the earlier interface definitions. Other data is maintained in the registry, including real-world information about principals and slice information. Interfaces and privileges for accessing the registry are defined.

The description of a credential is expanded upon, defining the credential as a binding between principal GID, object GID (a sliver, slice, or registry), the privileges authorized and their lifetime and an expression of how those rights might be delegated.

The final section's indications of a two-level model conflict with the earlier sections' implications that the aggregate manager binds slivers to slices as in a recursive model. This is one of the indications that the community has not formed a consensus around the overall role of the aggregate interface.

3.2 The GENIAPI Definition

The GENIAPI specification of the aggregate manager interface is an improvement on the SFA document in some respects. The various slice and sliver operations include explicit naming of the object to be manipulated rather than being part of the credential. At points in the interface where data structures are evolving or extensible by nature the GENIAPI adopted appropriate self-describing data structures to support evolution and extension.

The specification is a subset of both the SFA document and the various existing control framework implementations. For example, none of the calls for ticket manipulation exist, which may make writing resource brokers difficult.

The most recent specification lays out certificate and credential formats, but leaves Rspecs unbound[2].

This specification does not directly specify either Rspec or credential formats, although some commonality is essential for long term interoperability. While the code picks an identity format based on X.509 certificates, the documentation does not specify that format.

Finally, key operations like creating a slice or assigning credentials remain unspecified. This may be because such definitions are missing from the SFA, because the implementations have adopted different models. These missing interfaces may also reflect confusion between the two usage models we have posited.

4 GENIAPI support for a TIED/PlanetLab Plug-in

From the perspective of the GENIAPI and the model in Section 1.1, a TIED plug-in is an internal agent that exports TIED rather than GENI interfaces. The TIED experiment controller is an external agent in the position depicted in Figure 4, communicating with multiple systems. In a world where the GENIAPI was perfectly successful, a single TIED/GENI plug-in would work for any control framework. This section discusses the difficulties in creating such a GENIAPI plug-in across the PlanetLab and ProtoGENI systems; we believe these difficulties are relevant to many internal and external agents using the GENIAPI.

Because TIED was derived from an Emulab model of experimentation, its resource model is one of full control of isolated resources; PlanetLab's lineage is one of multiplexed resources embedded in an existing Internet. Any unified GENI plug-in will need to be able to determine the kind of slicing in use so that the plug-in can advertise the resources to TIED properly. In addition, we need to expand the TIED model to accommodate a broader set of resource sharing modes.

PlanetLab's tools seem to be generally designed for human beings to use, and therefore require some context and understanding to use effectively. There are other tools available to assist humans in understanding the PlanetLab system, but these are not available through the GENIAPI (or the PlanetLab SFA). As a result, laying out a PlanetLab experiment requires more investment from the TIED plug-in's logic and more developer effort. In general terms, even an external agent would benefit greatly from access to more information and services than the GENIAPI exposes.

Finally, PlanetLab gathers resources that are owned in a more federated way and distributed more broadly than ProtoGENI, which makes the internal permission structure and responsibilities of the owners different. Many of these authorization decisions will be made on operations yet to be standardized in the GENIAPI, and these differences show that its shape must accommodate many structures.

We discuss each of these differences below. Each section details the underlying issue further and suggests how it should affect the GENIAPI.

4.1 Resource Models

Currently different control frameworks tend to represent different views of what resources are interesting and how they are sliced between experimenters. This is natural and useful, but as

control frameworks begin to interoperate key assumptions about their resource models must be made visible by the API. ProtoGENI and PlanetLab provide instructive examples.

To acquire a resource in a ProtoGENI environment is to control it almost completely. Systems like ProtoGENI, including Emulab and DETER, are designed to provide isolation between experiments and to give the experimenter total control over the environment in which the experiment is being conducted and the data collected. Repeatability and control of experimental parameters drive the systems.

To acquire a PlanetLab resource is to acquire rights to use part of a resource located somewhere in the shared Internet. Experimenters have some control over where the systems are located and some say in the rights they can exert, but a PlanetLab resource is fundamentally shared. Controlled access to the shared public Internet are underlying goals, not hard resource requirements.

These characterizations are, to some extent, generalizations. There are facilities in TIED/DETER for expanding controlled experiments by using shared Internet connectivity. There are systems in PlanetLab for acquiring dedicated internode capacity and guaranted resources. But the two systems solve two different experiment layout problems. DETER/Emulab/ProtoGENI is analogous to a virus lab or a particle accelerator where the environment is as controlled as possible. PlanetLab is a way to make excursions looking for new network species or to deploy prototypes into an uncontrolled environment.

For an agent accessing the resources of one of these two control frameworks, the problem is that the only way to determine the kind of resources in the aggregates knowledge outside the advertisement – a framework supports PlanetLab slicing if it is PlanetLab. Strictly speaking, even parsing the advertisement requires such knowledge because the various control frameworks use different Rspec formats for resource advertisements.

Agents that want to speak across multiple frameworks, like a TIED GENIAPI plug-in, will need both a standard format, or small set of formats for advertisement and reservation. Those formats should capture information about slicing model and information about what phenomena are important in real world selection. For example, are nodes able to communicate over the Internet? Are they required to do so? Do they have a meaningful physical location? A location in some network coordinate system? Can they sense data in the real world? Actuate devices in the real world?

As we have discussed before, the formats of the various Rspecs are not very different[1], but in this case the closeness in format hides underlying differences in how resources are sliced and how they relate to the world. While experiments are carried out inside one control framework with a consistent resource model, these assumptions are unimportant because they are never violated. When aggregates are shared across multiple control frameworks or agents manipulate multiple control frameworks, these assumptions need to become visible constraints.

4.2 Experiment Layout

Constructing the experiment topology from the resources allocated in a control framework requires knowledge of the resources and their interconnection properties. Control frameworks

tend to develop tools to help with or perform topology layout fairly early in the control framework's lifetime. Access to relevant information can be important to both internal and external agents that perform layout and access to layout tools that operate in a framework is key to external agents that want to offload part of the layout task. This section contrasts the tools available in PlanetLab and ProtoGENI and describes the ramifications on the GENIAPI design.

The PlanetLab configuration tools reflected through the SFA and the GENIAPI are the core tools of PlanetLab itself, and tend to reflect their heritage of human-designed experiments. Node locations are given by strings that are both human-readable and human-meaningful. Users unfamiliar with the names of cities and states in certain parts of the world may have difficulty deciding which nodes are best suited for their experiment.

To use the VINI system connect nodes using dedicated capacity, similar human interaction is required. An experimenter must gather the VINI connection points and assemble the experiment with both the location requirements above and the VINI interconnectivity information.

While this basic information is available through the GENIAPI interfaces, the client of those interfaces, for example a TIED plug-in, must create the layout from that information itself.

PlanetLab users who go outside the SFA have more tools at their disposal. The SWORD[18] resource discovery system is a sophisticated resource discovery system that effectively performs advisory experiment layout based on constraints. The CoMon monitoring system[19] provides current and past data on the performance of slices and nodes in PlanetLab that can be used to guide experiment layout. Other similar tools appear in PlanetLab with regularity, but are generally not accessible from the GENIAPI interface.

Our point is not that PlanetLab's tools are less accessible or complete than ProtoGENI's tools. ProtoGENI's resource model requires layout tools to be useful at all while PlanetLab's requires transparency. Our point is that the advertisement interface and sliver allocation interfaces should be wide enough to pass relevant information to external agents as well as to internal agents. ProtoGENI's layout tools are less configurable than the Emulab tool chain on which they are based partially to present a simplified interface. PlanetLab's more interesting tools are inaccessible. While part of this is a matter of allocating developer resources, part of it is making sure that the GENIAPI supports communicating with such entities.

4.3 Slice Creation and Authorization

Perhaps surprisingly, different control frameworks ascribe different qualities to the fundamental slice abstraction. In the case of ProtoGENI and PlanetLab, these differences are small to outside users, but may shape the slice authority interface. This section describes the differences in resource ownership that give rise to different authorization structures for slice creation.

In ProtoGENI and related systems creating an experiment/slice is a fairly common, low-overhead operation with minimal repercussions. The resources are all contained in a single physical area and experiments that misbehave can be stopped with relative ease.

PlanetLab is more of a true federation; though nodes are administered centrally, a misbehaving node that becomes isolated from the central authority on the network can only be controlled by a

set of human agents who physically control the machines. These agents, PI's in PlanetLab parlance, are more trusted than even the most trusted ProtoGENI users, because PlanetLab depends on them for correct operation. These differences are reflected in the authorization structure of PlanetLab that are loosely captured in the GENIAPI.

In ProtoGENI or TIED/DETER, the right to create slices is almost a given for an experimenter. Experimenters can be barred from doing so, but the perception is that it is a right that must be removed rather than granted. Tying the slice creation right to resource provision as PlanetLab does, results in a more controlled propagation of that right.

Without the interface to the slice authority, it is unclear what policies can be enforced on what aspects of slice creation and user management. Again, any agent such as a TIED plug-in remains tied to the non-standard mechanisms here. The more these diverge, the more difficult it will become to unify them in the future.

The key for the GENIAPI here is to standardize the slice authority interface(s). Certainly no single internal agent codebase can be expected to manipulate multiple control frameworks when the code must be specialized for how each handles slice allocation and binding of users to slices. The contrast between ProtoGENI's simple, loose slice creation requirements and PlanetLab's tighter constraints shows that the operations on slices need to be carefully defined. The GENIAPI operations serve both as a procedural breakdown of tasks and as a set of authorization enforcement points.

The current state of slice state as directly manipulated registry entries both leaves the information required for slice implementation very loose, and fails to clearly identify the objects about which access control decisions must be made. The two different focuses of the PlanetLab and ProtoGENI slice authorization systems offer interesting test cases of the evolving interfaces.

5 Summary

This document describes our position that the GENIAPI standardization effort is a valuable step if control frameworks are to easily interoperate. Looking at the possibility of using this interface across control frameworks has illuminated three key areas for improvement.

- Advertisements of resources should include information about the slicing model used to allocate them.

- Aggregate Manager interfaces should be made wide enough to include specialized control framework dependent tool access in a sufficiently standard way for agents to use it.

- Interfaces to the Slice Authority need to be defined with an eye toward both standardizing workflow and providing authorization enforcement.

We believe that the role of the current API in connecting control frameworks to aggregates is somewhat hazy and have tried to capture the ongoing discussion about the shape control

frameworks take. We have outlined those models and plan to continue to push the community for clarity on these issues.

References

[1]Ted Faber, John Wroclawski, "Preliminary Review of the GENIAPI as Control Framework Interoperability Architecture and TIED Federation Plug-in Candidate," *http://groups.geni.net/geni/attachment/wiki/TIED/TIED_GENIAPI_v1.2.pdf*

[2]The GENI Project Office, "Aggregate Manager API, v 1.0, 1 Sept 2010, http://groups.geni.net/geni/attachment/wiki/GeniAggregateManagerApiDoc/GENI-SE-CF-AMAPI-01.0.pdf

[3]DETER, "The DFA Plug-in Architecture," *http://fedd.isi.deterlab.net/trac/wiki/FeddPluginArchitecture*, 2010.

[4]Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad Mac Newbold, Mike Hibler, Chad Barb, Abhijeet Joglekar, "An Integrated Experimental Environment for Distributed Systems and Networks," *Proceedings of OSDI,* (October 2002).

[5]ProtoGENI, http://www.protogeni.net/trac/protogeni/wiki.

[6]Thomas Lehman, Jerry Sobieski, Bijan Jabbari, "DRAGON: A Framework for Service Provisioning in Heterogeneous Grid Networks," *in IEEE Communications Magazine,* Vol. 44, no. 3, March 2006.

[7]Larry Peterson, Robert Ricci, Aaron Falk, Jeff Chase, *Slice-Based Federation Architecture*, July 2010, *http://groups.geni.net/geni/wiki/SliceFedArch*.

[8]The GENI API, *http://groups.geni.net/geni/wiki/GeniApi*.

[9]Larry Peterson, Soner Sevinc, Scott Baker, Tony Mack, Reid Moran, Faiyaz Ahmed, *PlanetLab Implementation of the Slice-Based Facility Architecture,* Version 0.07, *http://svn.planet-lab.org/attachment/wiki/WikiStart/sfa-impl.pdf*, Sept. 2009.

[10]OpenFlow, *http://www.openflow.org*, 2010.

[11]Guido Appenzeller, "OpenFlow Demos at GEC8," *http://www.openflowswitch.org/wp/2010/07/gec8/*, 2010.

[12]Tom Mitchell, *GENI Aggregate Manager API*, (slides), http://groups.geni.net/geni/attachment/wiki/Gec8ControlFrameworkAgenda/GEC8-Mitchell-AggregateManagerAPI.pdf, 2010.

[13]How to use ProtoGENI, http://www.protogeni.net/trac/protogeni/wiki/Tutorial, 2010.

[14]Users Guide to the SFI, http://svn.planet-lab.org/wiki/SFAGettingStartedGuide, 2010.

[15]Ted Faber, John Wroclawski, *TIED Testbed Control Framework: Plug-in Design Document*, Feb 2010, *http://groups.geni.net/geni/attachment/wiki/TIEDProtoGENIPlugin/TIED_CF_plugin_design_spec_v1.0.pdf* .

[16]*GENI Aggregate Manager API*, *http://groups.geni.net/geni/wiki/GAPI_AM_API*, 2010.

[17]*International Standard "Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components"* (ITU-T Rec. X.667, ISO/IEC 9834-8, 2004.

[18]Jeannie Albrecht, David Oppenheimer, David Patterson, and AminVandhat, "Design and Implementation Trade-offs in Wide Area resource Discovery," *ACM Transactions on Internet Technology*, 8(4), September 2008.

[19]KyoungSoo Park, Vivek Pai, "CoMon: A Mostly Scalable Monitoring System for PlanetLab," *ACM Operating Systems Review*, 40(1), January 2006.