Strawman GENI API for Enhanced Modularity

Ted Faber, USC/ISI

Version 1.1

March 7, 2011

Table of Contents

1 Introduction

This document contains our recommendations for overhauling the exisiting GENI interface to improve interoperability between control frameworks. We proceed by starting from the published GENIAPI[1] for aggregate managers and the ProtoGENI slice authority interface[2] and both adding necessary interfaces and revisiting existing interfaces in light of interoperability.

Our guiding principle in modifying and creating the interfaces is to require fairly minimal interfaces between the constituents of control frameworks, the slice authority that binds resource allocations together for administration and safety, and the aggregate managers that control resources. These have relatively clear tasks, but features have crept into the interfaces to support identifying the owners of resources. This design is mindful of both the goal of connecting information about users to their slices and about minimizing the detailed interconnections between the providers of that information and the control framework components.

The interfaces are designed to connect resources to slices and slices to users at clear articulation points. Where possible, the various entities know where to find more information, but are insulated from the contents and formats of that information. A slice authority knows which aggregate managers have contributed to a slice and how to name the allocations attached, but not each resource. Similarly, a slice authority has a contact point for the party responsible for a slice, but may not know if it links to a web page describing the owner or an e-mail address monitored by them. A slice manager decides if a link to the responsible party is adequate not by parsing the link, but by a trust relationship with the provider of the link, expressed through credentials.

The rest of this section outlines the control framework components in more detail and articulates our position on clearinghouses. The rest of the document defines our proposals for the interfaces. This is a starting point for further interface discussions.

1.1 Interrelating Control Frameworks and Aggregates

The GENI API 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[3].

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.

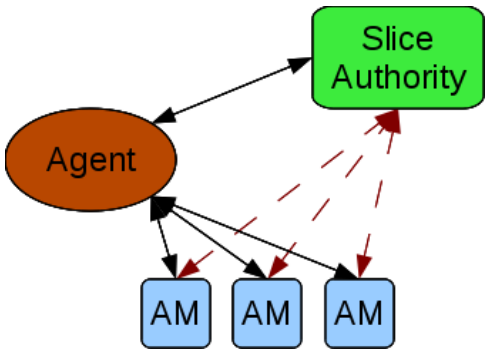

Figure 1: Control Framework

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 aggregate managers have contributed resources to the slice.

We further note that one system may play each of the roles in the architecture, and perhaps others besides. One purpose of standardizing these interfaces and functions is so that we have a common vocabulary to describe such hybrid or multipurpose systems.

The current GENI API 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.

Those missing standards, in particular the missing interface between aggregate managers and slice authorities, are barriers to resource providers making equipment available to multiple control frameworks. This document proposes interfaces for that purpose.

The discussion above assumes one layer of aggregate managers, all visible to the user. It is more likely that some aggregate managers will actually be managers of aggregate managers, recursively allocating resources that a user's agent is unaware of on the user's behalf (Figure 2). Initially, aggregate managers were called component managers and expected to control single pieces of equipment. The control framework prototypes generally use a single manager to control multiple pieces of equipment in this manner.

Prototype control framework developers have also found that aggregate managers are a useful place to encapsulate aggregate-specific decision making. The ProtoGENI prototype understands the internal state of the ProtoGENI aggregate and will select resources on behalf of an agent. Generalizing this by allowing aggregate managers to coordinate aggregate managers will improve scalability of the system. Such a smart aggregate manager is represented in Figure 2 by the AM/Agent oval.

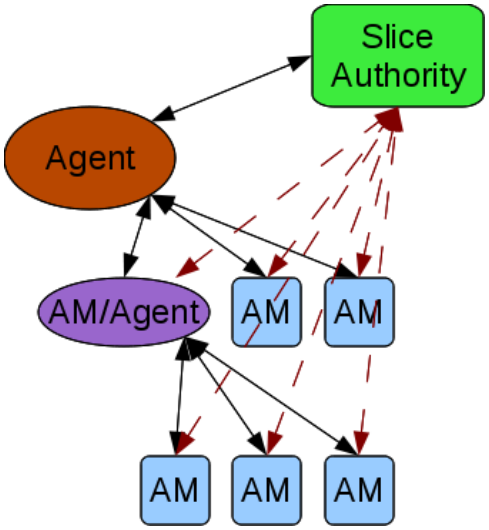

Figure 2: Recursive agents

Taking this one step further, the AM/Agent in Figure 2 need not control any resources of its own; it could be purely an embedding service or broker for other aggregates. To enable that, this document lays out both the credential delegation strategies and the aggregate manager/slice authority interfaces necessary to support it.

1.2 The Slice Authority and Shutdown

One of the primary roles of the Slice Authority, if not the primary role of the slice authority is to coordinate information about what resources have been allocated to the slice in case the slice needs to be shut down. Slices and

allocations (slicers) are designed to fail safely, in that if both have lifetimes and must be repeatedly renewed, so the amount of time a misbehaving slice can wreak havoc is limited.

Emergency shutdown is the feature that a third party with appropriate credentials can request a slice be shutdown before it or its slivers expires. This is a best effort attempt to contact the aggregate managers that have contributed resources to the slice and effect the shutdown.

By “best effort” the designers may mean that all the aggregate managers are known and the attempt to shutdown will only fail if the slice authority is unable to contact the aggregate managers, or that the list of aggregate managers is not necessarily complete at all times. In this document we assume the former, and this assumption particularly motivates decisions in Section 2.2 about the order in which resources are bound to the slice and allocated to the user. We also recognize that looser semantics may be desirable. The decision on the semantics of emergency shutdown will play an important role in the interface.

1.3 Where's the Clearinghouse?: Identity and Accountability

Clearinghouse proponents may wonder where a clearinghouse fits into these interfaces. As specified so far, a clearinghouse provides three things to the control framework: a trust anchor for the identities of the principals, a trust anchor for some credentials, and a repository of information binding that identity to a real world point of contact. We believe that eliminating the first makes for better interoperability and scalability properties of the system and the interfaces in this document are specified to decouple the details of the latter two.

We continue to believe that tying a principal's identity to a trust root makes it more difficult to join the federation without improving the security. To that end we believe users should be represented by self-certifying identities – e.g., public/private key pairs.

As for trust, that is expressed in credentials. Trust generally has a basis established outside the mechanism used for access control, and any access control system must capture that. However, the interpretation and basis of trust may take many forms. Rather that tie resource allocation to the hierarchy of trust represented by a clearinghouse, we present a credential structure that supports a hierarchy based at a trusted root as well as more distributed forms of access control reasoning.

Being able to convert from a self-certifying identity to data that links the identity to a real-world entity is important to many users and enablers of GENI. This facility can be used to direct queries about actions taken by that user, or to be used in building real-world trust relationships. The first would provide a location to direct queries about an oddly behaving slice, the second might influence a principal's willingness to issue credentials.

We leverage a trust relationship between an identity provider and a slice authority to allow a slice authority to accept a URI that links user information to slice information without specifying the format of the information collected. A credential from the GPO asserting that a user can be reached by a given link assures the slice authority that the information linked meets GPO standards.

The data one might collect and export about a given principal varies widely, however, and overspecifying that data can lead to fewer, less scalable clearinghouses. Because the specific data at a clearinghouse is not of interest to the resource allocation players, the interfaces here link to the data's location without constraining its content.

1.4 Acknowledgments

Though my name is on the document here, it has benefited from the feedback many people who have reveiewed early copies. Robert Ricci, Steve Schwab, Aaron Falk, Tom Mitchell, Aaron Helsinger, and John Wroclawski will all see their ideas in this document in some form. The inevitable misrepresentation of their ideas is my fault.

2 The Slice Authority Interface

The key undefined interfaces are those between the agent and aggregate managers and the slice authority. The section lays out the requirements for the slice manager and proposes the interfaces.

2.1 Requirements

A slice is the GENI abstraction of resources being used together in an experiment. Knowing which aggregates are contributing to a slice is most urgent when an experiment is misbehaving and needs to be shut down, as discussed in Section 1.2. Having the full list of sliver identifiers and aggregate manager contact points makes it possible to attempt to forcibly shut down the slice. In the event it cannot be shut down, the timeout mechanism will eventually terminate the slice.

When resources are allocated from an aggregate manager to a user they are also bound to a slice. Both bindings must be in place for the shutdown process above to work, so an aggregate manager must only return resources to the user after the binding has been accomplished. This interface guarantees that all resources allocated to a user have been registered with a slice authority, at least by sliver identifier, if not by the contents of the allocation.

Slices have lifetimes that can be renewed, so an aggregate manager must revoke sliver resources when the bound slice lifetime runs out. The existing aggregate manager interfaces include calls for this renewal process.

Beyond shutting down a slice it is also useful to be able to connect these resources to a human agency. Experience in the prototypes has shown that some experiments behave in ways that their hosting institutions did not expect, but that are not immediately disruptive to service. In these cases, the most useful thing for the GENI infrastructure to do is to provide the interested party with a way to contact a human running the experiment and discuss the issues. The tightness of the binding between that contact point and a particular person will vary according to the policies of the slice authority. In some cases an e-mail address will be sufficient and in others some more detailed identity will be required.

There are other benefits to aggregating slice information. With more information, slice authorities can track utilization, display experiment information and provide other services, but the information required for each slice is:

- The set of aggregate managers and slivers that contribute to the slice in the event of a shutdown
- A contact point for resolving questions and problems with the experiment

2.2 Proposed Interface: Slice Authority/Aggregate Manager

The proposed interface has one essential interface, BindSliver, that is called by an aggregate manager on a slice authority. The result of a successful call is that the slice authority has recorded the attachment of a sliver to the given slice, knows how to contact the aggregate making the allocation and has received the credentials necessary to shut it down.

The call may fail because the user requesting the action did not have rights to the slice, the aggregate manager did not have the right to make the binding, or for policy reasons encoded in the slice authority. For example, a slice manager may allow all resources for a slice to be from classified aggregates or all from unclassified aggregates but no mixing. Such a policy can only be enforced by the slice authority, as aggregates do not communicate about slices.

The interface is:

BindSliver(credentials, slicename, slivername, proposeddeadline, contactpoints, authoritycredentials, resources)

A successful call returns a lifetime for the binding and a justification for the decision (assuming an ABAC-like system that can deliver such a thing), an unsuccessful call returns an error code, including any authorization justification (failed proof).

Details about the arguments:

- Credentials: a mix of user and aggregate credentials that authorize the binding (see Section 2.2.1).
- Slicename: the name of the slice to bind.
- Slivername: the name of the sliver to be created.
- Proposed deadline: the amount of time the aggregate is willing to commit resources. The slice authority may reduce this time in its reply.
- Contact points: One or more URIs where the slice authority can initiate a shutdown of this sliver.
- Authority credentials: zero or more credentials granted to the slice authority so that it may call shutdown on the sliver. These are not used in determining access for the BindSliver call, but are held by the slice authority in the event it needs to call shutdown on this sliver.
- Resources: a (possibly empty) RSPEC describing the resources in the sliver. A slice authority may or may not require data about the particular resources being allocated to a sliver and an aggregate may or may not be configured to share that information. The two must be able to come to an agreement, perhaps through multiple attempted BindSliver calls, before the call succeeds.

2.2.1 Authorization

Binding resources to slices is a complex action in terms of authorization. The slice authority is being asked to accept a binding based on its trust of two principals: the aggregate manager and the user/agent making the allocation.

The user's right to bind resources from the slice is based on that user's relationship to the slice authority. In a simple case, the slice authority passes a (delgatable) credential to the user conferring the right to bind resources to the slice when the user creates it. When each request for a sliver is made, the user delegates that authority to the aggregate who makes the call. The slice authority gets to confirm each binding, and confirms at that point that the delegations have been made in agreement with the slice authority's policy.

The slice authority also cares about the credentials of the aggregate manager. The slice manager is depending on the aggregate manager to administer the resources in accord with the slice authority's policies – for example, following GENI semantics – and will demand some assurance of that. This may be an explicit delegation of trust from the slice authority or more indirect evidence, e.g., a certificate from an auditor.

Exactly how the slice authority interprets those credentials is a matter of its policy, but credentials from both parties can play a role.

2.2.2 Lifetimes

There are several different expressions of lifetime that need to be taken into account when allocating resources to a slice. Specifically:

- The time for which a credential that enables allocation is valid
- The time for which a credential that enables binding a resource to a slice is valid

- The time for which a resource is allocated
- The time for which a resource is bound to a slice

It is tempting to base allocation or binding times on the lifetimes of the credentials, but we argue that credentials must explicitly specify lifetimes of slices and allocations. This allows an instructor to issue a student a credential that allows the student to create slices that last for one day throughout the semester. The credential expires at the end of the semester and can be reused until then, but each use of the credential creates a slice that lasts one day.

Because the system does not use credential lifetimes, the requested lifetimes need to be explicit in the binding call (and in the CreateSliver/RenewSliver calls). From the explicit lifetimes allowed, the aggreagte manager can determine how long the allocation and binding will be made. Allocation time and binding time do need to be equal, because allocated resources must always be bound for the fail-safe properties to hold.

2.2.3 Unbinding Resources

A sliver will disappear when its slice expires or when the user calls DeleteSliver on it at the aggregate manager that owns it. It is not strictly necessary to unbind the resources from the slice at this point but there may be performance and operational advantages to doing so. We propose the simple interface:

UnBindSliver(credentials, slicename, slivername)

Unlike BindSliver and CreateSliver, a DeleteSliver (or Shutdown) can return success even if the UnBindSliver call is unsuccessful or not attempted.

2.3 Proposed Interface: Slice Authority/Agent

We believe much of the slice authority/agent interface at ProtoGENI is reasonable and useful.[2]We propose making some parameters explicit that are currently implicit as well as excluding some operations from the required interface.

2.3.1 More Explicit Parameters

The primary operation a user requests from a slice authority is slice creation. In the ProtoGENI authority this is accomplished by the Register operation. We propose changing the signature of that operation to look like:

Register(credentials, contact_uri)

The operation should return an identifier, a lifetime, credentials that allow the user to bind resources and renew the slice, and a set of contact addresses where aggregates can bind resources to the slice.

The credentials passed in to the operation are used to determine if the user can create the slice at all. They may be as simple as a direct statement from a clearinghouse that the slice authority trusts saying to trust the user or may be more diffuse. In addition, the credentials should prove that the user has a right to use the contact URI.

The contact URI is a point of contact that could be an e-mail address, a web page at a clearinghouse with more detailed information, or some other form of contact information. The slice authority makes the decision about whether to allow it based on the policy and credentials. For example, if the GPO provides a credential saying a given user has the right to use a particular GPO-managed URI as a contact point and the authority has a policy of trusting the GPO about contact URIs, the call can continue.

That explicit policy decision allows a clean break between the collection of personal/contact data managed by clearinghouses and and the administration of resources carried out by slice authorities and aggregate managers. Rather than try to constrain the semantics of the contact URI, we convert it to a trust relationship. A credential issuer that validates poor contact URIs can become untrusted, or if bound in contract, subject to real world penalties.

2.3.2 Simpler Semantics for GetCredential

In keeping with a minimalist view of what a slice authority must support, we believe that GetCredential should pass any credentials issued by this slice authority to the principal. These may include credentials that empower the caller to renew a slice, delegate authority, etcetera. These credentials are all available other ways – calling Register or RenewSlice – but it provides a way for failed agents or users who have lost credentials for existing slices to restore credentials.

For simplicity, just the first, parameterless version needs to be supported.

2.3.3 Trimming Extra Operations

We advocate removing three operations in the current ProtoGENI slice authority interface, BindToSlice, GetKeys, and DiscoverResources. We believe that BindToSlice is better handled with more expressive credentials and that DiscoverResources and GetKeys belongs outside the narrow scope of the slice authority. We believe that BindToSlice should disappear altogether, but that the other two simply not be required of slice authorities.

BindToSlice associates a user with a slice in a way that allows the slice creator to bestow its rights on another user. We believe that the right place to do this is by credential delegation, either using current GENI credentials or ABAC credentials. BindToSlice does not have any way to do partial delegation, there is no clear mechanism for getting the necessary credentials for resource allocation to the user being attached to the slice. Overall, we think this interface raises more problems than it solves.

GetKeys retrieves user data from the slice authority – specifically authentication keys for resource access that are included in CreateSliver calls on aggregate managers. We think that the slice authority should not hold that data, and that GetKeys or its equivalent should happen elsewhere. More precisely, we think that a slice authority should not be *required* to support this function.

Similarly, a slice authority itself holds no resources, so should not be required to support DiscoverResources; a particular slice authority may make the interface available as a convenience.

2.3.4 Remove as an Optimization

Slices disappear if not renewed, so the Remove call is mostly an optimization for the slice authority. ProtoGENI currently implements it this way, and we support the usage.

3 Conclusions

We have presented our ideas for a reshaping and specification of the GENI AI+PI for interoperability between control frameworks. These are initial steps, so there is not much to conclude.

Again, this work benefited from the feedback of the folks in Section 1.4, and is a starting point for further discussion.

References

[1]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

[2]ProtoGENI Slice Authority, http://www.protogeni.net/trac/protogeni/wiki/SliceAuthorityAPI.

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