Using Semantic Web Description Techniques for Managing Resources in a Multi-Domain Infrastructure-as-a-Service Environment

TR-13-02

Yufeng Xin (RENCI), Ilia Baldine (RENCI),
Jeff Chase (Duke University) and Kemafor Anyanwu (NCSU)

April 2013

RENCI Technical Report Series
http://www.renci.org/techreports

Using Semantic Web Description Techniques for Managing Resources in a Multi-Domain Infrastructure-as-a-Service Environment

Yufeng Xin (RENCI), Ilia Baldine (RENCI),
Jeff Chase (Duke University) and Kemafor Anyanwu (NCSU)

April 16, 2013

Abstract

This paper reports on experience with using Semantic Web technologies for managing multi-domain networking infrastructure-s-a-service (IaaS) testbed. An OWL ontology based on newly-created vocabularies was used to model multi-layer network providers with common base classes for fundamental cyber-resources, and adaptation functions from resources at one layer onto resources of the same base class at the layer below. Extended SPARQL path queries supported by GLEEN were used to support topology embedding and resource provisioning for creating connected arrangements of compute, storage and network resources gathered from multiple resource providers.

The context for the work is the use of the semantic models in ORCA - the control software for ExoGENI, a new testbed funded through NSFs GENI project. ExoGENI is a multi-domain cloud testbed with a high degree of control over networking functions, including links within each domain and dynamic inter-domain links over national circuit fabrics. The paper describes how the semantic network models enable ExoGENI to instantiate on-demand virtual topologies of virtual machines linked by on-demand circuits and segments for a variety of applications ranging from networking experiments to high-performance computing.

Introduction

IaaS (Infrastructure-as-a-Service) cloud services are evolving rapidly beyond their initial form as simple virtual machine services (e.g., Amazon EC2). Cloud providers increasingly integrate platform capabilities with various programming models (PaaS or Platform-as-a-Service), along with "network-as-a-service" (NaaS) [12] capabilities to allocate and configure virtual network topologies connecting edge resources in multi-tenant environments. Services may be composed together to form connected arrangements of virtualized infrastructure comprising compute, storage and network resources, assembled form multiple infrastructure providers. This process is referred to as *topology embedding*, i.e. finding isomorphic or homeomorphic mappings between the available resource network graph assembled from multiple providers and the request network graph provided by the user.

Semantic web technologies provide the necessary flexibility of expression and the readily available tools for querying and inference that simplify the typical problems encountered in the process of embedding topologies. Path query tools make it possible to operate on path abstractions in constructing higher-level complex embedding algorithms. Inference allows to generalize certain algorithms making them insensitive to specific resource types involved in the request.

As a motivating example, consider complex scientific applications that require a diverse set of re-

sources assembled together as elements of a computational workflow that may need to run multiple steps on different resources in order to achieve the final result with intermediate results transported between different computational resources. Performance isolation offered by a NaaS system capable of creating the necessary virtualized and individually tuned infrastructure rapidly on demand represents a significant advance on the current state of the art in computational infrastructure. As an example, Figure 1 shows the infrastructure to support a workflow modeling the electronic properties of a catalyst molecule involved in a chemical process which converts CO2 molecules from air and water molecules into methane under sunlight. Methane can then later be used as fuel. This work is part of a larger US DOE Solar Fuels initiative. This example requires the orchestration of resources from multiple computational and network resource providers in order to complete its steps. We study this example in more detail in Section 0.1.

We envision a future cloud ecosystem for supporting such applications in which multiple provider domains offer various cloud infrastructure services, both on a commercial basis and for non-commercial use by specific user communities, e.g., for research and educational use. This vision requires a rich means to represent resources and reason about them. Using these models, providers can advertise their resources and customers can describe desired resources. Also, declarative models can describe infrastructure at physical and virtual layers, or at multiple logical layers, for use by increasingly sophisticated algorithms to configure, program, and adapt the resources to user or application requirements. This paper describes our experience using Semantic Web technologies and extended SPARQL queries supported by GLEEN for modeling networking resources to support user specification of the infrastructure necessary to support their needs, which may include computational workflows as above or other applications that benefit from virtualized infrastructure-as-a-service.

We are working toward this vision in the ExoGENI/ORCA project. ORCA (Open Resource Control Architecture [10, 5]) is a multi-domain orchestration software framework for providers of cloud and network infrastructure. It is an architecture to orchestrate a collection of independent virtual infrastructure providers through their native IaaS interfaces. The ORCA software ecosystem is part of US National Science Foundation's GENI project and Cyberinfrastructure programs.

ExoGENI testbed funded by US NSF uses ORCA software to orchestrate resources across multiple sites in the US, located on university campuses, with each site contributing cloud resources to the testbed. Network providers like NLR and Internet2 offer bandwdidth-on-demand channels between the sites. ExoGENI allows users to create custom topologies using resources from multiple federated providers. In our view this infrastructure is tremendously useful as a tool for computational sciences, where performance isolation and customization abilities of the system make it possible to achieve science results faster and more conveniently. When completed, ExoGENI will span close to 20 US university campuses and labs.

In ORCA we use semantic network models to describe pools of resource infrastructure, cloud services and to allocate virtual resources from resource pools, and virtual networks. Our premise is that rich semantic models are the key to future development of networked clouds. They serve as a foundation for emerging declarative models to program networks and networked systems, and to control both the provider networks and the hosted customer environments (virtual networks).

The rest of the paper is structured as follows: in Section we provide some background on ORCA and ExoGENI; in Section we motivate and describe the NDL-OWL ontology we created to describe the various IaaS compute; in Section we describe the application of various models within ORCA framework as applied to ExoGENI testbed, we then describe the above scenario in which these components come together to support a real application and, finally, conclude with the description of potential future directions for our work in Section 0.1.

ExoGENI and ORCA

ORCA is a federation architecture for combining many providers into a unified service. ORCA

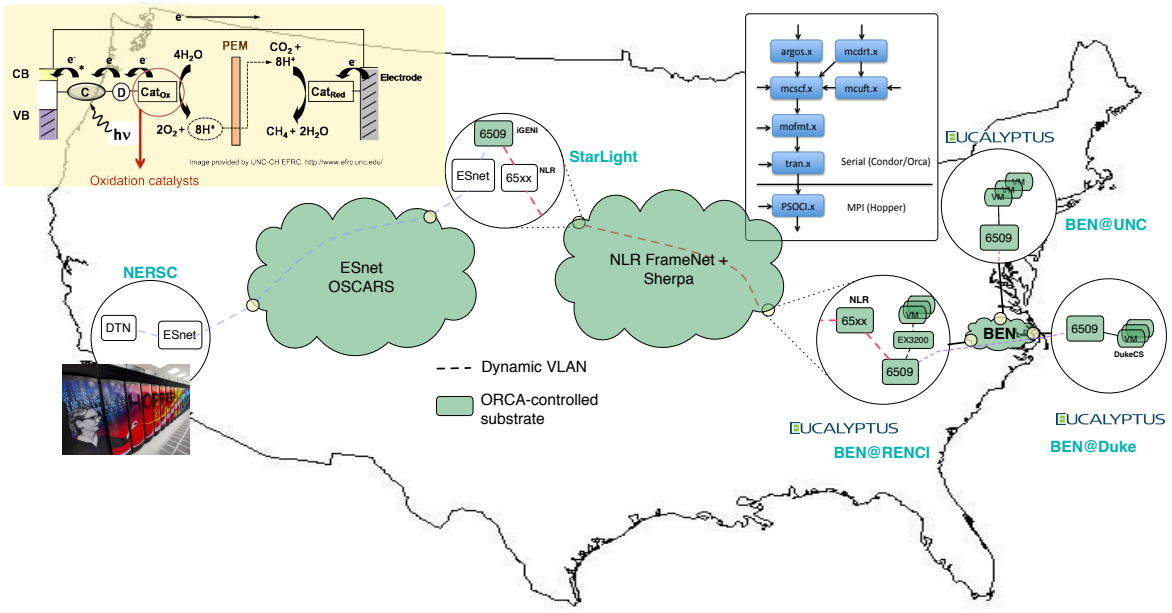

Figure 1: SC11 demo. Chemical process (upper-left), workflow steps (upper right) and topology.

software agents representing customers, resource providers, and brokers exchange information in the form of semantic models. Each provider offers an API to allocate, instantiate, configure, and use the resources that it controls (its *substrate*).

Providers may use various virtualization technologies or other means to manage their resources as pools of independently allocatable units. These virtual resources are called "slivers" to distinguish them from the underlying substrate resources. A *sliver* is any virtual resource element that can be named, instantiated, managed, and released independently of other slivers. The providers offer an infrastructure (IaaS) service, in which slivers correspond to instances of fundamental and well-understood cyber-resources, such as virtual machines, bandwidth-on-demand channels, storage volumes, and virtualized portions of switches or routers.

Slivers may be linked together in various ways to create built-to-order environments that host an activity such as an application or service. These linked sets of slivers hosting an activity are called *slices*. ExoGENI slices are isolated: they interact with the outside world through controlled interfaces, and the virtual resources in a slice may have defined quality-of-service properties for improved performance isolation.

Built-to-order virtual network slices can implement routing overlays using IP or other packet-layer protocols. Testbed users may deploy custom node operating systems with alternative networking stacks into their nodes, and use programmable datapaths and/or virtual routers to implement new network services at the cloud edge and at network intersection points. Scientific applications may create custom topologies out of necessary compute and data resources and execute their workflow in this dynamic and elastic slice infrastructure.

The initial ExoGENI deployment includes several kinds of aggregates offering network services, each represented by an ORCA Aggregate Manager (or AM for short):

- **Cloud sites.** A cloud site AM exposes a slivering service to instantiate virtual machines (VMs) on its hosts and virtual links (VLANs) over its internal network.
- **Bandwidth-on-demand services.** For these

services, the AM invokes a provider's native provisioning APIs to request and manipulate circuits. We have developed ORCA plugins for NLR's Sherpa FrameNet service and the OSCARS circuit reservation service used in ESNet and ION [2]. ORCA can also provision complex connections [4] on a multi-layer metro network called BEN (Breakable Experimental Network) connecting several of the ExoGENI sites.

Each virtual link instantiated from these aggregates appears as an atomic link in the slice's virtual topology. At the layer below, the aggregate may perform internal stitching operations to construct a requested virtual pipe or segment from multiple stitched links traversing multiple substrate components within the aggregate's domain. A virtual link may even traverse multiple providers if the host aggregate represents a multi-domain circuit service, such as ION.

Semantic web representations are pervasive within this infrastructure. The controller actor, representing the users of the system relies on semantic web representations of desired slice topologies as well as abstracted representations of substrate provided by the substrate owners, also expressed using semantic web mechanisms, in order to plan resource allocation steps. Multi-layered circuit planning in BEN AM is based on semantic web representations of resources and physical and virtual connection paths, we call NDL-OWL. Compute resources in cloud AMs are represented using an ontology of compute elements that we have designed.

Using semantic network models for network description

ITU-T G.805 standard [11], *Generic functional architecture of transport networks*, provides an abstract informational model for a connection-oriented transport networks. Transport networks carry multiple types of traffic, including Internet traffic, however unlike the Internet, they provide capabilities for provisioning bandwidth-on-demand in the form of channels at potentially different layers (optical, ethernet and so on). Connections at different layers within transport networks have server-client relationships with server layer connections serving as an envelope for one or more client connections. As an example, multiple Ethernet VLANs (virtual links) can be carried inside a single optical wavelength. Certain types of networking equipment are capable of *adaptations* from one layer to another, i.e. accepting one or more client connections and multiplexing them onto one or more server connections at a lower layer.

The ITU data model defines three types of components: *Topological components* : access group, subnetwork, link, layer network. These concepts correspond to the node, link, subgraph, and graph concept in graph theory. *Process function components* : adaptation and termination, which describes the adaptation relationship between layers and the termination capability in the server layer. *Transport components*: link connection, subnetwork connection, and network connection, which can be used to describe the new transport entities formed after certain configuration actions, such as temporary connections created on demand from multiple concatenated links.

The pioneering work in this space was done by the developers of NDL or *Network Description Language* [1, 14] in creating a model for describing optical transport networks. The ontology model in NDL followed a modular and hierarchical approach of G.805 and used RDF language, in which a number of abstract classes and properties based on the G.805 are defined. The fundamental components in NDL include the *Interface* class, *Adaptation* class to define the adaptation relationship between layers, *ConnectTo* predicate to define the connectivity between *Interface*, and *switchedTo* predicate to define the cross-connect between interfaces within a switching matrix of a network element.

NDL-OWL

In ORCA we extended NDL to define a more powerful ontology using OWL (Ontology Web Language). We call our extended ontology suite *NDL-OWL*.[1]. NDL-OWL adopts key NDL concepts and extends

[1] `http://geni-orca.renci.org/owl contains the class and property hierarchies that define the structural part of NDL-OWL ontology`

them. It replaces RDF with the OWL standard to specify certain relationships among predicates, taking advantage of stronger class properties and inferences offered by OWL, which enable us to specify key network management functions as simple SPARQL queries over NDL-OWL models. It adds a new class hierarchy and predicates to describe edge resources such as compute servers, virtual machines, cloud services, and block storage servers. The ultimate goal of this process is to create a representation language that is sufficiently powerful to enable generic resource control modules to reason about substrate resources and the ways that the system might share them, partition them, and combine them. Some key distinctions of NDL-OWL from NDL:

Multiple models: NDL defined a single model for representing the state of a transport network that was suitable for multi-layered path-finding. In ExoGENI we need to not only define the state of the substrate, but using the same mechanisms must also allow users to express the desired topology of their slice in a *request*, be able to describe the virtual slice topology given to the user in a *manifest* and allow various elements of ORCA exchange *abstracted substrate definitions* of individual domains to support topology embedding. To support these requirement NDL-OWL defines multiple such models shown in Table 1. The difference in models lies in the use of top-level classes (e.g. Reservation or Manifest to distinguish e.g. a request from a manifest), while the majority of classes related to resource descriptions are reused across different models.

Generalized label concept: A *label* is an entity that distinguishes or identifies a connection among others sharing the same communication channel (a wavelength, a VLAN, etc). In a layer 1 network, the resource label usually corresponds to the physical channel ID, *e.g.,* a particular fiber in a conduit, a wavelength in the ITU-T grid, or time-slot in the SONET or OTN frame. The label range is fixed and a particular physical channel has a fixed capacity. We extend the NDL *Label* class to associate the capacity and QoS characteristics with a transport entity so that it becomes the information hook. Two properties, *availableLabelSet* and *usedLabelSet*, are defined to track the dynamic resource allocation.

Connection and subnetwork mapping: A virtual connection provisioned from multiple domains and lower-layer links can itself become a link that can be concatenated with other links at the same layer to create even more complex connections (and graph homeomorphisms). These two constructs allow defining the network topology recursively, thus supporting virtualization within virtualization. More importantly, these concepts can be conveniently mapped into the *subGraph* concept in semantic web.

Transitive and inverse properties: Major properties in NDL model define the client-server relationship between resources. *inverseOf* property axiom in OWL automatically helps infer the relationship in one direction based on the definition in another direction. For example, *hasInterface* and *interfaceOf* are *inversive* as are *adaptation* and *adaptationOf*. We use the *Transitive* axiom to define the connectivity property and adaptation property. These features are very useful in implementing a path finding algorithm in multiple ways. Within a single layer A *connectedTo* B and B *connectedTo* C can help infer that A *connectedTo* C from the transitivity of the *connectedTo* predicate. Further, if all necessary pairs of connection points are connected, an end-to-end path can be directly inferred. Notably, in the case of multi-layer path finding, the *adaptationOf* property is also needed in order to infer proper adaptations between layers.

NDL-OWL schemas are subdivided into core and technology-specific ontologies. Core ontologies: *collections.owl* describes useful collection types, *layer.owl* describes basic network concepts and adaptations, *domain.owl* describes basic concepts for domain representation. Technology-specific ontologies describe various types of substrate: *dtn.owl, itu-grid.owl* for optical networks, *ethernet.owl* for IEEE 802.3 and related ethernet networks, *ip.owl* for IPv4, *compute.owl* - the basic compute ontology and its extensions for Amazon EC2 *ec2.owl* compute element types. Finally there are several GENI-specific ontologies that describe unique GENI resources (*planetlab.owl* for PlanetLab resources, *protogeni.owl* for ProtoGENI resources and several others).

Figure 2 shows a simple connection between two servers that can be part of a resource provider's topol-

Substrate Description model	The detailed resource and topology model that is used to describe a physical substrate including network topology and edge resources (compute and storage).
Substrate Delegation model	The abstract model to advertise an aggregate's resources and services externally. This model allows multiple abstraction levels, as different domains may want to expose different levels of resource and topology description of its substrate.
Slice Request model	The abstract model to represent user resource requests. A typical request might be a *bound* or *unbound* virtual topology with specific resources at the edges, generated by some experiment control tool.
Slice Reservation model	The abstract model used by ORCA broker actors to generate resource tickets. Each ticket contains information on one or more resources allocated from a specific cloud site named in the ticket.
Slice Manifest model	The abstract model to describe the topology, access method, state, and other post-configuration information of the slice *as built*. It contains detailed information about instantiated resources and access methods for tools or users to interact with them.

Table 1: ORCA resource models

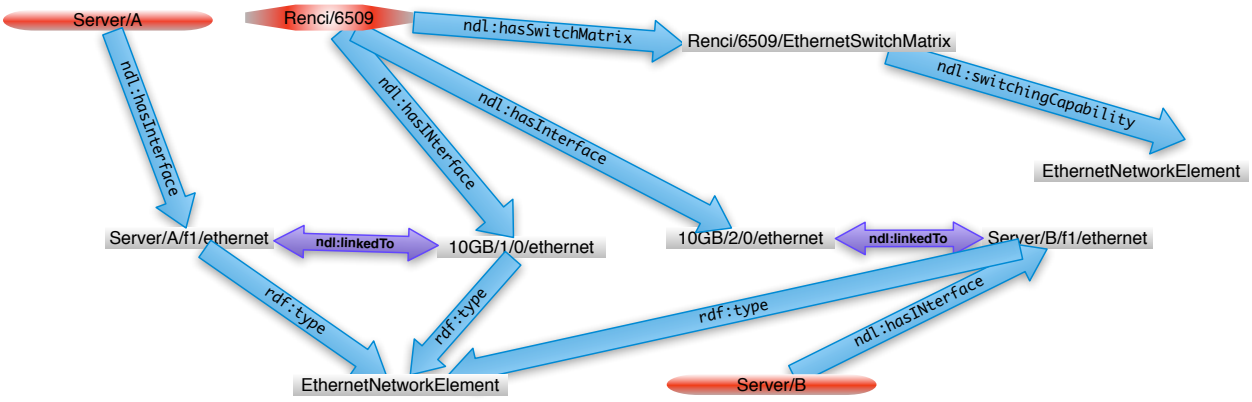

Figure 2: Simple connection between two servers.

ogy. Server/A and Server/B are unique URIs denoting individual servers which both belong to a subclass of *ComputeElement* class, itself a subclass of a NetworkElement top class (not shown). Renci/6509 is an Ethernet network switch (we know this because it *hasSwitchingMatrix* with *switchingCapability* of *EthernetNetworkElement*), which has interfaces (in this case ports) that connect to interfaces (also ports) on both servers. NDL property *hasInterface* allows to associate individual *NetworkElements* (nodes, links are all NetworkElements) with their interfaces. Interfaces, in turn can be *linkedTo* each other.

Figure 3 shows an example of a manifest model (a manifest model is returned to the user to show exactly which resources were provisioned based on the user request). The shaded portion represents the original request for 1 day for a cluster of 5 virtual machines (VMs) allocated from two cloud sites at Duke University (dukevmsite/Domain) and RENCI (rencivmsite/Domain). The request specifies that the VMs should be connected to each other with a common VLAN (connection/1). The provisioned resources are shown in the unshaded portion and represent the actual VMs provisioned based on the request, as well as the VLAN (vlan/1). The main instance of the manifest is Manifest/1 - of top-level class Manifest. It has a single element NetworkConnection/1 which contains the end points (the virtual machines) and the VLAN interconnecting them. Manifest instance (Manifest/1) is linked to the originating request (Reservation/1) via *manifestHasReservation* property. Similar to the substrate model shown in

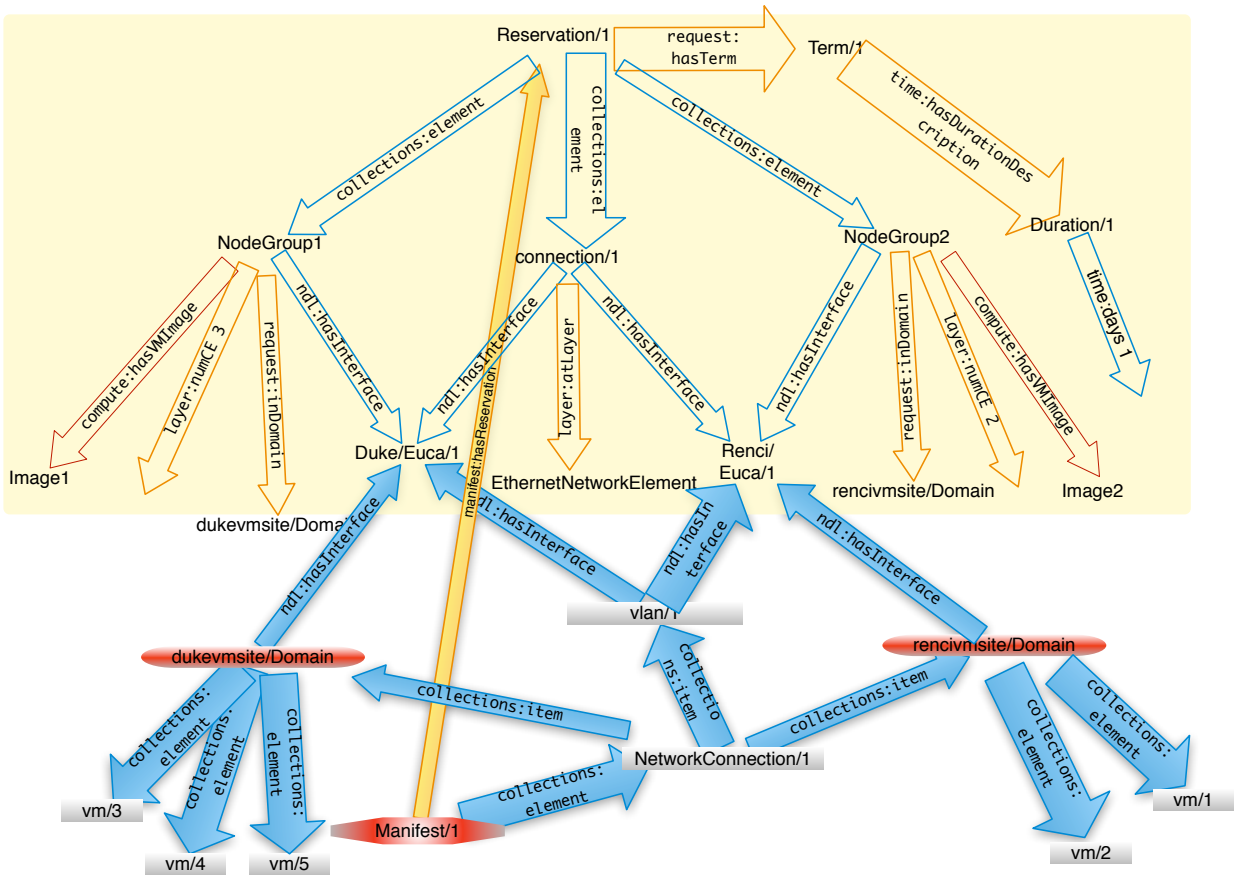

Figure 3: Example of a manifest model that incorporates the request for topology.

Figure 2 the *hasInterface* property links interface instances to the elements of the infrastructure (nodes and links) and the interfaces are linked to each other to indicate connectedness. The difference is in this case interfaces are virtual, not physical ports.

Layer Adaptations

In addition to network connectivity, a key concept NDL and NDL-OWL is *layer adaptations* between client layer and server layer described in the beginning of this section. Since lower layer server connections can be dynamic, creating a path at the higher layer may involve first constructing connections at lower layers. Thus a path between two nodes that presents itself as an Ethernet VLAN may in fact require an establishment of a DWDM path (a wavelength is provisioned) and a fiber path (fibers are interconnected) before the VLAN becomes active. This is referred to as a *multi-layered path*.

A *feasible* end-to-end multi-layered path must have compatible layer adaptations along the path within the network elements it traverses, thus significantly complicating the path-finding problem. Figure 4 inset shows an example of a simple multi layered connection offering VLAN Ethernet service between two PoPs (Points-of-Presence) in a multi-layered network. First a fiber path must be established, followed by a DWDM connection which is carried within the fiber, and finally one of the DWDM channels is used to carry the VLAN service. Adaptation functions within each PoP guarantee the ability to create ser-

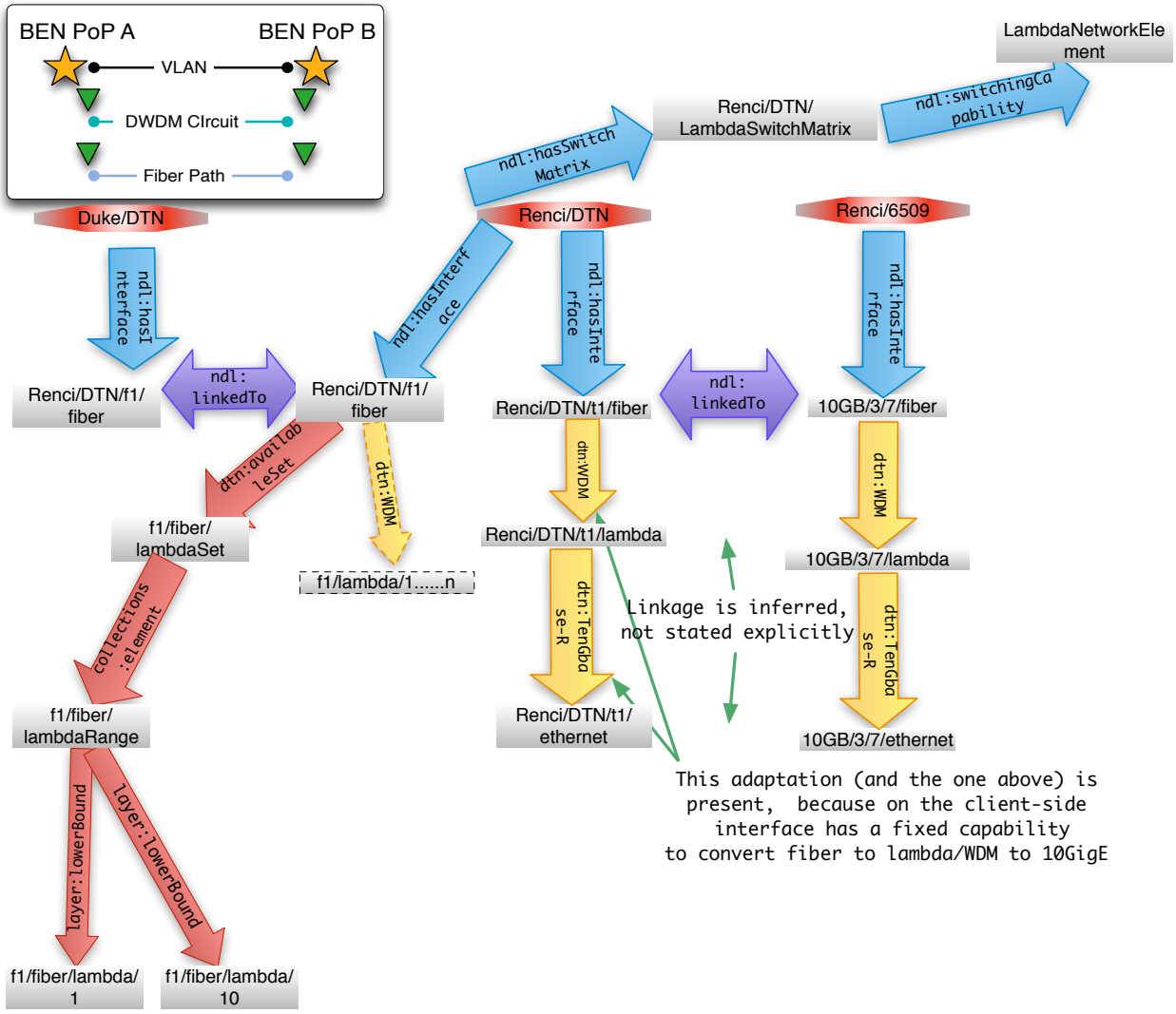

Figure 4: A cross-layer connection using adaptations.

vice primitives in this case in a server-client fashion, where higher-level connections are clients of the lower-level server connections.

It is important to note that whether two devices are connected in the multi-layer setting cannot be solely decided by the connectivity (*connectedTo* or *linkTo* properties) between their interfaces. The *adaptation* property has to be taken into consideration to guarantee the compatibility along the path. Therefore a valid path between two devices normally goes through a number of RDF triples with combinations of the following properties: *hasInterface*, *adaptation*, *connectedTo*, *linkTo*, *adaptationOf*, *interfaceOf*.

Figure 4 shows part of the NDL-OWL model reflecting a multi-layered connection. Network elements called RENCI/6509 (an Ethernet switch at BEN PoP B/RENCI) and RENCI/DTN (a DWDM transport node at BEN PoP B/RENCI) are physically linked to each other over fiber interfaces RENCI/DTN/t1/fiber and 10GB/3/7/fiber. These

interfaces due to available adaptation properties *dtn:WDM* (adaptation from fiber port to DWDM) and *dtn:TenGbase-R* (adaptation from DWDM to Ethernet) have additional interfaces at DWDM layer (RENCI/DTN/t1/lambda and 10GB/3/7/lambda) and Ethernet layer (RENCI/DTN/t1/ethernet and 10GB/3/7/ethernet), which allow the system to determine the existence of an Ethernet connection between those two elements. The figure only shows one half of the connection at BEN PoP B, with a similar setup at BEN PoP A/Duke. Inference is used to determine the presence of the Ethernet connection based on the base *ndl:linkedTo* property between the fiber interfaces.

Semantic Queries

In networking, one of the most interesting graph patterns is to find if a path exists between two end-points, *e.g.,* , two resources in the RDF data set or two nodes in the RDF nodes. In a simple case, a *path* query can be created by a conjunction of a list of consecutive triples, $P =< (x, p_1, o_1), (o_1, p_2, o_2), ..., (o_{(k-1)}, p_k, o_k) >$, such that the object of a triple is the subject of its succeeding triple except for the last triple.

This is possible only when we know the sequence of resources on the path exactly ahead of time. However, in general it is not known. In provisioning network services, we usually face two *path* problems: (1) given a source node, find all the connected destination nodes; (2) given two nodes, find a valid path, *i.e.,* our task is actually to identify the sequence of resources and the intermediate hops along a valid path. We note that a *path* usually consists of a sequence of segments that has the same topological structures, *i.e.,* the same subgraph pattern in the RDF data set. The basic SPARQL graph pattern can only query for the possible path segment between two neighboring nodes with known adaptation and connectivity relationships. What we need is to express a path pattern recursively using a regular path pattern.

GLEEN [7] is a regular path expression library plugin to the Jena ARQ package. It supports the regular expression operators like '?' (zero or one), '*' (zero or more), '+' (one or more), '—' (alternation), and '/' (concatenation). It was designed to find the path pattern between two entities in the medical ontology so that a simplified view can be generated out of the complicated class and property hierarchy. It defines two types of query constructs that can be directly applied to a triple pattern of the SPARQL query body. In both cases, the path expression is formed by a number of properties recursively using above regular expression. The *gleen:OnPath* is used to find all the objects that are connected to the subject via the defined path expression. A triple pattern which uses this construct has the following form:

subject gleen:OnPath (pathExpression object)

For our purposes, the following simple query will return all the network devices that are reachable from a specific *source* device via one or more hops:

```
Select ?destination
Where {
    source gleen:OnPath
    (([ndl:hasInterface]+/[ndl:connectedTo]+/[ndl:interfaceOf]+)+
    ?destination ).
}
```

This pattern, however, only returns the destination objects without revealing how the paths are traversed. The second construct *gleen:Subgraph* is defined to accomplish this and can be applied to the SPARQL triple pattern in the following way: *(inputSubject pathExpression inputObject) gleen:Subgraph (outputSubject outputPredicate outputObject) .* The three arguments in the object position triple must be unbound and are the variables to be answered by the query. In this way, all intermediate resources along with the path edges connecting them are obtained for path between *inputSubject* and *outputSubject* via the *pathExpression*.

```
Select ?a ?b ?c
Where {
    (source
    ([ndl:hasInterface]+/[ndl:connectedTo]+/[ndl:interfaceOf]+)+
    destination).
    gleen:Subgraph (?a ?b ?c)
}
```

Because there is no way to know which layer a dynamic connection may go through, a valid path query needs to try every known possibil-

ity of adaption and de-adaptation in the path segment. If there are n adaptation possibilities, the path expression pattern can be constructed by the unions of n basic expressions in the form of $([ndl : hasInterface] + /[aAdaptation] * /[ndl : connectedTo] + /[aAdaptationOf] * /[ndl : interfaceOf]+)+$. We note *aAdaptation* and *aAdaptationOf* are variables that represent one of the n inverse property pairs for the adaptation function between two layers. When the objective is to find the shortest path, *Filter* clause has to be included to evaluate a path to make sure the length of a valid path is less than a value. Thus in our application domain the evaluation of shortest cross-layer path expression require *AND*, *FILTER*, and *UNION* operators.

In [13], the authors studied the computation complexity of evaluating different forms of the SPARQL graph query pattern. The evaluation can be done in polynomial time if the pattern only contains the *AND* and *Filter* operators. The evaluation becomes NP-Complete if *AND*, *FILTER*, and *UNION* operators appear in the pattern. If *OPT* operator is involved, the problem becomes a *PSPACE-complete* problem.

ORCA makes extensive uses of the regular SPARQL and GLEEN query facilities, which has greatly simplified the implementation of its topology embedding algorithms.

Domain Abstractions

Network providers and cloud providers typically do not expose their internal topology to outsiders. However, in order to support federated cross-domain provisioning, some limited information about the substrate and its properties must be exposed. This type of substrate or domain model abstraction, similar to database views, is key to achieving federated resource provisioning that satisfies complex multi-domain slice requests. It provides information to help compute inter-domain paths between different edge resource providers, while at the same time filtering the information exposed by transit network providers to limit the exposure of topological details of their networks. This approach puts an emphasis on the important tradeoff between the requirement for privacy of network providers and the optimality of the computed inter-domain paths. The more information the abstract representation exports, the more optimal the resource allocation process becomes, typically at the expense of privacy.

In general, the abstracted domain representation describes the virtual resources that it is willing to delegate. Each entity in a model has a global name given by a URI, so a model can conveniently represent linkages to other models by including links to named RDF nodes in descriptions of other domains. The control framework can either collect the domain abstract models from different providers and assemble them into an overall semantic web model (RDF graph), or treat the models as a distributed linked data semantic web database.

ORCA implementation of the abstraction process automatically derives the abstracted NDL-OWL model from the complete substrate description NDL-OWL model created by the domain owner. While multiple levels of abstraction are possible, ORCA supports the model similar to ATM PNNI, representing each domain as a single switching matrix with interfaces that indicate peering points with other domains. Several attributes, like available label sets and bandwidth are defined for the interfaces.

Applying semantic models to ExoGENI

NDL-OWL Models

As described in Section , NDL-OWL defines multiple models for describing the various types of resource descriptions used by ORCA within ExoGENI, as shown in Table 1. Figure 5 shows how the different models are exchanged between different actors. The models are generated dynamically and exchanged by various ORCA actors in the form of RDF-XML documents using an RPC interface, in a departure with the Linked Data approach common in the Semantic Web community.

Users submit their requests to the ORCA controllers in the form of a request model that contains the *Reservation* instance which includes a variety of elements describing the needed resources, their topol-

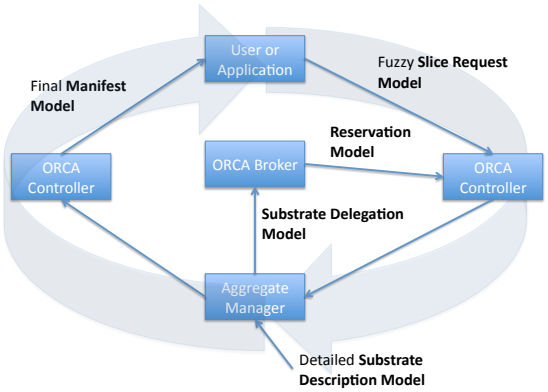

Figure 5: NDL-OWL models in ORCA.

ogy, the term of the request and other configuration properties, like link bandwidth and OS images to be used on the compute resources. Requests may not be fully specified with respect to binding resources to specific domains, thus leaving it up to the controller to decide which specific provider sites the embedding should take place at.

Resource providers represented by ORCA Aggregate Managers (AMs) delegate abstractions of their resources using abstract delegation models to the ORCA Brokers. Controllers query brokers for these models in order to perform topology embedding. Brokers issue tickets for resources which permit controllers to instantiate the necessary resources at selected AMs and stitch them together. The controllers return the manifest model, which merges the information about instantiated resources with the original request model, to the user to enable the user access to the newly provisioned virtualized infrastructure of the slice.

Running scientific applications on ExoGENI

In this section we return to the motivating example described in the introduction, which helps put the multitude of described mechanisms and technologies in perspective. At a recent SuperComputing11 conference ORCA linked private cloud resources in North Carolina's ORCA deployment to US DOE's *Hopper* super-computer to execute a complex computational workflow simulating the atomic behavior of a solar fuel catalyst. The workflow under the control of Pegasus [6] workflow management system started execution on a Condor [9] compute pool built out of cloud resources in NC and completed with a massive 3K-way MPI job on Hopper (see Figure 1). ORCA created a 'slice' of resources consisting of a collection of cloud resources from multiple sites in North Carolina (interlinked on-demand by BEN) and a dynamic QoS-provisioned link between NC resource and NERSC (the facility housing Hopper) composed of VLAN/MPLS segments from NLR Framenet and ESnet. As part of the workflow data from the first step of workflow was transported from NC to NERSC using this link in order to complete the second step.

In this demo each provider domain (individual cloud sites and network domains) operated using a substrate description constructed by the substrate owner using Protégé [3] based on NDL-OWL ontologies. Abstract delegation models were automatically generated by ORCA and delegated to a *broker* actor responsible for coordinating resource allocation.

The topology for the slice was described in NDL-OWL and submitted to ORCA controller as RDF-XML document. The validity of the request model was tested with a few rules made simple by the use of inference on properties like *rdfs:subClassOf*. The embedding of the request in specific ExoGENI sites was determined based on SPARQL queries on a graph model assembled from substrate delegation models of domains. GLEEN queries were used to determine the shortest path across multiple domains (namely BEN, NLR and ESnet) between North Carolina and California sites. Within BEN in North Carolina multi-layered connections were established based on paths computed using GLEEN queries that determined available adaptations between layers. Server-connections (DWDM and fiber) were created in response to the need to create a VLAN connections between the BEN sites.

Conclusions and Future Work

In this paper we presented an overview of implementation and use of OWL-based resource representa-

tion models in a multi-domain IaaS environment controlled by software framework called ORCA. The substrate for the deployment is an NSF GENI-funded testbed called ExoGENI currently under active deployment across the US. Our work demonstrates the practical uses and future potential of this type of knowledge representation approaches for active management of cyber-resources in a distributed environment on a global scale.

Our on-going and future work is concentrated on devising more powerful algorithms for embedding slices in substrate and supporting this work through more sophisticated tools developed within the semantic web community. The challenges we face lie in a number of dimensions:

Model churn: while the more common models frequently face the challenge of scale, they are updated relatively infrequently. In contrast NDL-OWL models are relatively small (on the order of thousands of triples), however they are constantly updated by adding and removing triples corresponding to provisioning and removal of new resources. Dealing with churn in our models represents a challenge in terms of efficient concurrent processing, where multiple agents may want to concurrently update the same model.

SPARQL performance: as noted in Section , SPARQL performance greatly varies depending on the types of statements involved. Due to NP-hard nature of the multi-layered path-finding problem [8], this complexity translates into the complexity of the SPARQL searches we need to perform. We're devising various heuristics that help deal with this problem in a suboptimal, but tractable way.

References

[1] Network Description Language. http://www.science.uva.nl/research/sne/ndl.

[2] OSCARS. `http://www.es.net/oscars/`.

[3] Protege Ontology Editor and Knowledge Acquisition System. `http://protege.stanford.edu/`.

[4] I. Baldine. Unique Optical Networking Facilities and Cross-Layer Networking. In *Proceedings of IEEE LEOS Summer Topicals Future Global Networks Workshop*, July 2009.

[5] J. Chase, L.Grit, D.Irwin, V.Marupadi, P.Shivam, and A.Yumerefendi. Beyond virtual data centers: Toward an open resource control architecture. In *Selected Papers from the International Conference on the Virtual Computing Initiative (ACM Digital Library)*, May 2007.

[6] E. Deelman, G. Singh, M. hui Su, J. Blythe, A. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 13:219–237, 2005.

[7] L. Detwiler, S. Dan, and B. James. Regular paths in sparql: Querying the nci thesaurus. In *American Medical Informatics Association Fall Symposium*, Washington DC, Nov. 2008.

[8] F. Dijkstra. *Framework for Path Finding in Multi-Layer Transport Networks.* PhD thesis, Universiteit van Amsterdam, 2009.

[9] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *Proceedings of the Tenth Symposium on High Performance Distributed Computing (HPDC)*, August 2001.

[10] D. Irwin, J. S. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. G. Yocum. Sharing Networked Resources with Brokered Leases. In *Proceedings of the USENIX Technical Conference*, June 2006.

[11] ITU-T. G.805 : Generic functional architecture of transport networks.

[12] OpenStack Cloud Software Quantum API. `http://wiki.openstack.org/Quantum`.

[13] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of sparql. *ACM Trans. Database Syst.*, 34(3), 2009.

[14] J. van der Ham, P. Grosso, R. van der Pol, A. Toonk, and C. de Laat. Using the network description language in optical networks. In *Tenth IFIP/IEEE Symposium on Integrated Network Management*, May 2007.