From Kansei to KanseiGenie: Architecture of Federated, Programmable Wireless Sensor Fabrics

Mukundan Sridharan[1], Wenjie Zeng[1], William Leal[1], Xi Ju[2], Rajiv Ramnath[1], Hongwei Zhang[2], and Anish Arora[1]

[1] Dept. of Computer Science and Engg., The Ohio State University, Columbus, OH, 43210, USA
{sridhara,zengw,leal,ramnath,anish}@cse.ohio-state.edu
[2] Dept. of Computer Science, Wayne State University, Detroit, MI, 48202, USA
{xiju,hongwei}@wayne.edu

Abstract. This paper deals with challenges in federating wireless sensing fabrics. Federations of this sort are currently being developed in next generation global end-to-end experimentation infrastructures, such as GENI, to support rapid prototyping and hi-fidelity validation of protocols and applications. On one hand, federation should support access to diverse (and potentially provider-specific) wireless sensor resources and, on the other, it should enable users to uniformly task these resources. Instead of more simply basing federation upon a standard description of resources, we propose an architecture where the ontology of resource description can vary across providers, and a mapping of user needs to resources is performed to achieve uniform tasking. We illustrate one realization of this architecture, in terms of our refactoring the Kansei testbed to become the KanseiGenie federated fabric manager, which has full support for programmability, sliceability and federated experimentation over heterogeneous sensing fabrics.

Keywords: wireless sensor network, federation, fabrics, resource specification, ontology, experiment specification, GENI, KanseiGenie.

1 Introduction

Several edge networking testbeds have been realized during this decade, in part due to the recognition within the networking community that testbeds enable at-scale development and validation of next generation networks. The role of edge networks —and edge networking testbeds— is likely to only increase, given the growth of wireless networks of sensors, vehicles, mobile communicators, and the like. In this paper, we focus our attention on an emergent architecture for next generation Wireless Sensor Network (WSN) testbed federation.

WSN Testbeds. Many WSN testbeds are in use today, of which Kansei [3], Orbit [7], NetEye [4], and PeopleNet [8] are but a few. Two recent experimentation trends are worth noting: One, experiments are being repeated in multiple

T. Magedanz et al. (Eds.): TridentCom 2010, LNICST 46, pp. 155–165, 2011.

testbeds, to learn about (potentially substantial) variability of performance in different backgrounds, radio types, and size scales. And two, a number of experiments involve long running deployments—they often yield long lived sensing services—which in turn implies that testbeds are increasingly hosting concurrent experiments. These trends motivate the emergent requirement that testbeds need to be *federations of programmable fabrics*.

By programmable WSN fabrics, we mean that individual sensor arrays offer not just resources on which programs can be executed, they also provide network abstractions for simplifying WSN application development and operation. Examples include APIs for scheduling tasks, monitoring system health, and in-the-field programming and upgrade of applications, network components, and sensing components. Fabrics can also support and manage the concurrent operation of multiple applications. Figure 1 compares the traditional WSN model with the emerging fabric model of WSNs. By federated WSN testbeds, we mean multiple WSN testbeds that are loosely coordinated to support geographically and logically distinct resource sharing. A federation provides users with a convenient, uniform way of discovering and tasking desired WSN resources.

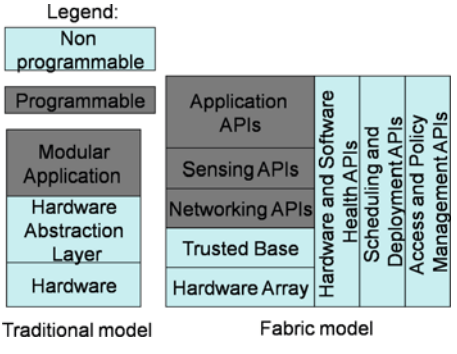

Fig. 1. Traditional and Fabric model

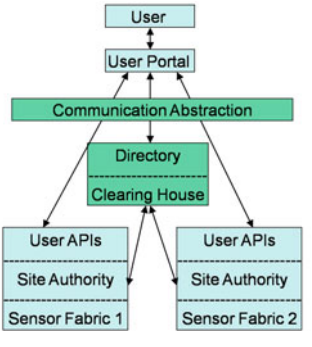

Fig. 2. Federated fabric/GENI model

GENI. The Global Environment for Network Innovation project [2] concretely illustrates an architecture where WSN fabrics are a key component. GENI is a next-generation experimental network research infrastructure currently in its development phase. It includes support for control and programming of resources that span facilities with next-generation fiber optics and switches, high-speed routers, city-wide experimental urban radio networks, high-end computational clusters, and sensor grids. It intends to support large numbers of users and large and simultaneous experiments with extensive instrumentation designed to make it easy to collect, analyze, and share real measurements and to test load conditions that match those of current or projected internet usage.

Figure 2 depicts the GENI architecture from a usage perspective. In a nutshell, GENI consists of three entities: Researchers, Clearinghouses and Sites (aka resource aggregates). The Clearinghouse keeps track of the authenticated users,

resource aggregates, slices, and reservations in the federation. A Researcher (interacting typically via a specially designed Portal) queries a Clearinghouse for the set of available resources at one or more Sites and requests reservations for those resources that she requires. To run an experiment, she configures the resources allocated to her slice, which is a virtual container for the reserved resource, and controls her slice through well-defined interfaces.

Overview of the paper. A federation of WSN fabric testbeds needs to address two core issues: One, an efficient and flexible method for resource description, discovery and reservation. And two, a convenient, uniform way of tasking and utilizing federation resources. While standardizing resource descriptions would simplify addressing these two issues, we find that the diversity of sensor characteristics and the lack of a compelling standard model for describing wireless networks complicate the federation of WSN fabrics.

In this paper, we propose a software architecture that we call KanseiGenie for federating WSN fabrics that is compatible with GENI. KanseiGenie is based on the position that, on one hand, different WSN fabric aggregates can advertise resources based on different resource ontologies. On the other hand, users can obtain uniform experimentation support from a portal supporting a given federation. Central to the KanseiGenie architecture is a mapping between a uniform experiment specification and a non-uniform resource specification, which is handled by the portal.

The rest of the paper is organized as follows: In Section 2, we detail the requirements of each actor in a WSN federation. In Section 3, we discuss the need for Resource and Experiment Specification in federations and their design challenges. Then, we present the KanseiGenie architecture and its implementation in Section 4, and make concluding remarks in Section 5.

2 Requirements of Federated WSN Fabrics

As explained in Section 1, the federated WSN fabric model distinguishes three actors: the Site that owns WSN aggregate resources, the Researcher who deploys applications via a Portal, and the Clearinghouse (CH) that enables discovery, management, and allocation of resource. In this section, we analyze actor-specific requirements, to make user experimentation easy, repeatable, and verifiable.

2.1 Clearinghouse Requirements

Broadly speaking, a Clearinghouse has two functions: One, identification and authentication of various actors in the system (the details of which are out of the scope of this paper and hence will not be discussed here); And two, resource discovery and allocation using a resource description language. The resource description language should be feature rich and extensible to capture the underlying heterogeneity of WSN fabrics and their constraints. A CH design should be efficient and robust to manage multiple Site Resource Specifications, large number of resources and their lease states. In a global infrastructure such

as GENI, a CH might have work in both master-slave and peer-to-peer modes with others CHs. A Portal/Researcher might request resources from multiple CHs directly or a single one which in turn communicates with other CHs.

2.2 Site Requirements

To support federated experimentation, a Site needs to support sliceability, programmability, experimentation services for the resources it controls. In the GENI model of experimentation, each Researcher owns a virtual container, aka a Slice, to which resources can be added or removed, and experiments deployed. It follows that a Researcher should have secure intra-slice communication that is isolated from other slices. To enable multiple slices to co-exist on the same fabric, sliceability may require Sites to virtualize resources both at the node and network level. For example, memory, processing time, or communication channels on the same device/network may have to be shared between slices. The challenge in virtualization is to provide as much control to the users (as low in the network stack as is possible) while retaining the ability to share and safely recover the resource. Virtualization in WSN fabrics is nontrivial, sometimes even impossible, given the limited resource on sensor nodes and the interference caused by concurrent wireless communications. Sites are also expected to provide programming services that reliably deploys applications composed by Researchers on the WSN fabric, testbed status and experiment execution monitoring, logging, trace data injection, and workflow control services.

2.3 Portal Requirements

A Portal in our architecture is a way to provide the user with a standard set of tools that can be used to exploit the federated resources. We believe that the Portal has an important role in networking sub-domains like WSN, where a high level of domain knowledge is needed to exploit the resources efficiently. A Portal needs to provide an uniform resource utilization framework. The framework should also allow the Researcher to select one or more of the standard User Services provided by the Site/Portal to be instrumented on the slice. This is challenging since the federation may consist of fabrics with a great variety of available platforms, sensors, radios, operating systems and libraries. For instance, while some sensor platforms such as mica family and TelosB are programmed on bare metal, others such as iMote2, Sunspots, and Stargates host their own operating system. The execution environments in these platforms vary from a simple file download and flash reprogramming, to command line interfaces and virtual machines.

3 Specification Languages

To use WSN fabrics, a Researcher queries the CH to discover what resources are available, selects a set of resources and obtains a lease for them, and configures the resources and User Services needed for the experiment. Each of these steps

needs a flexible, feature-rich, and extensible language to convey the goals of the Researcher to the system. We refer to the language used to publish, query, request, and allocate resources as the RSpec language, and the language used to configure resources and script workflow as the ESpec language.

3.1 Resource Ontologies for Resource Description

Resources available at multiple Sites are usually different from each other. Although describing resources in terms of a taxonomical flat-schema could hide minor differences by shoe-horning similar resources into the same category, building an exhaustive schema for WSNs, which will be conformed to by every Site, Clearinghouse, and Portal, is both difficult and sometimes impossible. The current GENI proposal [1] mitigates this problem somewhat with a partial standardization approach, a uniform core RSpec and multiple domain-specific extensions. Although this solution in principle is better than a single standardized RSpec, agreement on the extensions is difficult in domains like WSNs. Roscoe in [13] lays out the drawbacks of this approach. A key challenge in using a standardized specification is anticipating in advance all of the possible things that must be expressible. He argues that the resource request instead should be viewed as a *constraint satisfaction problem* and not a simple database query. For example, in a wireless network, while a node and channel might be two separate resources, it is usually impossible to allocate just the node but not the associated channel, or vice versa. It is better to explicitly capture such dependencies and relationships in the RSpec which would lead to a better resource allocation.

Thus a RSpec language should be able to specify constraints and handle multiple decentralized specifications. One way of doing this is to represent resources in terms of ontologies, using a resource description language such as NDL [5], which could describe not just the resources but the also constraints. With this approach, in order to communicate, two entities need only to agree on the language in which resources are specified, but not on the ontology.

Using Multiple Resource Ontologies for WSN Resources. Our primary motivation to support multiple ontologies in WSN federation is the difficulty in specifying things *uniquely*. We offer two examples cases. The first pertains to definitions, or rather the lack of it in the WSN space. There is no agreements on even simple terms like *nodes* and *sensors*. More over, defining something as either an allocatable *entity* or as its *property* could differ significant between Sites, depending on what platforms they provide. For example, is a channel an allocatable resource or simply a property of a radio?

Our second example deals with the difficulty of specifying wireless network topologies. A network topology in a WSN can be specified either implicitly or explicitly. An example of implicit specification would be specifying the transmit power level of each node. An example of explicit specification would be specifying the number and/or the list of neighbors for each node and letting the fabric choose the transmit power level and other radio parameters. A Site provider could choose one or other while defining the Site ontology, depending on the

hardware platform. One cannot force a Site provider to adopt an ontology which is not compatible with the hardware.

A question that directly follows from our argument for multiple ontologies is, why not simply use a union of them? This approach could result in RSpecs that cannot be instantiated by the Sites. For example, consider the two ways of (as discussed above) specifying a network topology; Even if a hardware could support both specifications, a single RSpec, which uses both implicit and explicit specification cannot be instantiated by the Site. Thus the union approach could lead to the risk of producing consistent RSpecs.

Given the intense debate in the WSN community, forcing Sites to use a single ontology is likely to throttle innovation and will result in a needlessly bulky ontology that is not easily extensible. Since most Researchers are expected to interact only through a Portal (or two) of their choice, we envision that Portals will serve as the unifying agent for resource specifications. Since all Sites will use the same language for their descriptions, the Portal can map the different ontologies used by the Sites to a single ontology to be used by Researchers. We note that there are several extant techniques and tools to map and align ontologies [9,12].

3.2 Experiment Specification and Work Flow Control

One of the roles of a Portal is to enable uniform experimentation across all federation fabrics. One way of satisfying this requirement is by providing an Experiment Specification (ESpec) language that enables Researchers to configure slices in a generic manner. Intuitively, besides the resource that the experiment is to be run on, an ESpec should also include the selection of User Services that is required by the experiment. WSN applications typically run in multiple well defined phases, with each phase involving a possibly different configuration. In addition to declarative elements, the experiment specification language also includes procedural descriptions (or workflow elements). This enables iterative experimentation, where a researcher programs repeated experiments, and the configuration of each experiment depends on the outcome of the previous ones. ESpec thus provides Researchers with a flexible and feature-rich way of interacting with resources, rather than just a GUI or a command line interface. They become particularly relevant for future scenarios where applications will primarily involve machine-to-machine, as opposed to human-to-machine, interaction. The idea then is to standardize the Experiment Specification language and not the format of interaction.

4 KanseiGenie

KanseiGenie is a refactoring of the Kansei testbed, to support a GENI compatible federation of geographically separated Sites, each hosting one or more WSN fabric arrays. Each sensor device is represented as a Component that defines a uniform set of interfaces for managing that device. An Aggregate contains a set of Components of the same type and provides control over the set. (In WSN

experiments, Researchers normally interact with fabric arrays through the aggregate interface rather than individual component interfaces). An Aggregate also provides other internal APIs needed for inter-component interactions.

4.1 Architecture and Implementation

In keeping with the GENI architecture, KanseiGenie consists of actors for a Site, a Clearinghouse, and a Portal. The current implementation of federation consists of a Site at The Ohio State University, which has four different sensor fabric arrays, and a Site at Wayne State University, which has two different sensor fabric arrays. The Sites and the Portal (which is hosted at Ohio State) run the KanseiGenie software developed at Ohio State. One of the Clearinghouse functions, namely resource management, is implemented using ORCA [6].

KanseiGenie Site. A KanseiGenie Site has four components: Aggregate of Aggregate Manager (AAM), the Web Service Layer (WSL), the individual Component Managers (CM) for each device type, and the Orca Site Authority module.

Aggregate of Aggregate Manager. Given that each fabric array is an aggregate, the KanseiGenie Site Authority (SA) is conceptually an Aggregate of Aggregate Managers that provides access to all the arrays. AAM is responsible for implementing the fabric APIs. AAM provides an AM interface for each sensor array through parameterization. Externally, AAM (i) administers usage of the resource provided by the Site according to local resource management policies, (ii) provides the interface through which the SA advertises its shared resource to one or more authenticated CHs and, (iii) provides a programming interface through which Researcher (via the Portal) can schedule, configure, deploy, monitor and analyze their experiments. Internally, the AAM provides mechanisms for inter-aggregate communications and coordination. The fabric APIs provided by an AM are organized into the four functional planes, namely, Resource/Slice management APIs, Experimentation APIs, Operation & management APIs, and Instrumentation & Measurement APIs.

Web Service Layer. WSL provides a wrapper for AAM and acts as a single-point external interface for the KanseiGenie SA. The WSL layer provides a programmatic, standards-based interface to the AAM. We utilize the Enterprise Java Bean framework to wrap the four functional GENI planes.

Component Manager. Each sensor device in KanseiGenie has its own Manager (although for some primitive devices such as motes the Manager is itself implemented on other more capable devices). The Component Manager implements the same APIs as that of AAM and is responsible for executing the APIs on the individual devices. Currently KanseiGenie supports Linux-based PCs/Laptops (Redhat and Ubuntu), Stargates, TelosB, and XSMs. CMs for Imote2 and SunSpots are under development.

KanseiGenie Portal. The Portal contains a suite of tools for the life cycle of an experiment. It provides an easy interface for experiment specification; at present, this is a user-friendly GUI; a user programmable interface is under development. It automates tasks for resource specification creation, requesting, and subsequent deploying of the experiment, for all Sites in the federation. It uses the Orca Slice Manager (explained below), to reserve resources requested by the Researcher. Once the reservation is done, it interacts with the AAM web services interface to configure and run experiments. Of course, a Researcher could directly program against the AAM web interfaces to gain more fine-grained control of experiments, i.e., write his own portal as need be. The Portal is implemented using the PHP programming language.

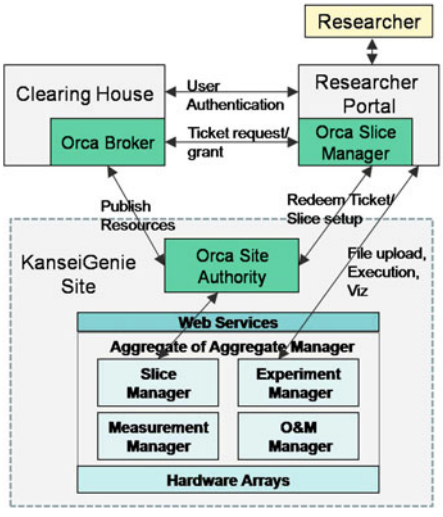

Fig. 3. KanseiGenie Architecture

KanseiGenie Clearinghouse. CH has two main functions; (i) Resource management: CHs are responsible for managing the resources on behalf of Sites. They keep track of resources and their leases. We use ORCA [6] (see below for more details) Resource Broker to implement our CH. (ii) Identity, Authentication and Trust: CH is also responsible for maintaining the overall security of the system. They authenticate/issue credentials for Sites, Portals and Researchers. In a federation, CHs implement trust-chaining to authenticate Researchers and Brokers from other domains. KanseiGenie, consistent with the GENI/ORCA effort, plans to use Shibboleth [10] for this purpose.

ORCA-based Resource Management System. ORCA consists of 3 sub-entities, each one correspondingly embedded into the three KanseiGenie actors (Portal, Site and Clearinghouse) respectively, to implement the resource management function. (i) The ORCA Slice Manager interacts with the Researcher and gets the resource request, forwards it to the ORCA Broker and gets the lease for the resources. Once a lease is received, the Slice Manager, forwards it to the Site Authority to redeem the lease. (ii) The ORCA Site Authority keeps inventory of all the resources that need to be managed. It delegates these resources to one or more Brokers, which in turn lease the resources to Researchers through the Slice Manager. (iii) The ORCA Broker keeps track of the resources delegated by various Site Authorities. It leases resources (if free) to the ORCA Slice Manager on request. A number of different allocation policies can be implemented using a policy plug-in.

4.2 Portal-Based Federation in KanseiGenie

Apart from being the single point of access for a Researcher, the Portal plays an important role in KanseiGenie federation architecture. The Portal has three important functions in federation, namely Resource Specification Mapping, Experiment Specification Mapping, and Federated Slice Stitching. KanseiGenie has thus far chosen the Portal as the main federating agent in the system. This design suits the view that a Portal realizes *application domain specific support*, and that for different domains, different Portals may suit. In other words, we view the KanseiGenie Portal as suitable for WSN experiments (and perhaps only some classes of WSN experiments) as opposed to sufficing for all GENI-Researcher needs.

In this view, Clearinghouses are treated as being generic rather than domain specific. Roles which are less domain specific, e.g. embedding Resource Specifications or slice stitching, can be moved from Portals to CHs (or even to Sites) assuming the method of stitching desired is communicated to them. Now, should CHs evolve to become domain specific, they may import more roles from Portals. Taken to the extreme, this would suggest that a top level CH be directly or indirectly capable of unifying all resources in GENI.

Resource Specification Mapping. As explained in Section 3 our position is that Sites may use their own Resource Specification dialects (ontologies). To provide a unified experience to the Researchers, we put the onus of interpreting multiple ontologies on the Portal. The Portal discovers resources from multiple sites, understands the Resource Specification ontologies and remaps the different ontologies into a unified ontology at the Portal. The current research [12,11] suggest that this remapping of ontologies can be done automatically with very high probabilities for related ontologies. However, we do the mapping manually since it is an one-time operation per Site. After a resource request is created (in the Portal ontology), it first gets *translated* into the individual Site's ontology and then sent to the CH to obtain the lease.

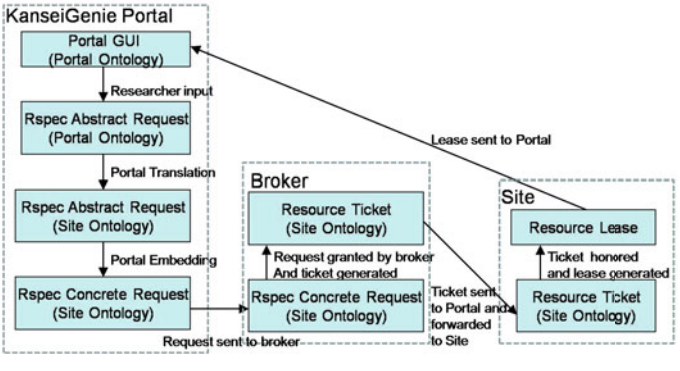

Fig. 4. Resource Specification mapping, translation and lease generation

In the current implementation, the Portal is also responsible for RSpec *embedding*), i.e. the embedding of an abstract RSpec into a concrete RSpec. It is often convenient for a Researcher to request networked resources in an abstract manner. For instance, requesting a 5-by-5 connected grid with 90% link delivery radio is much easier than pinpointing specific sensor devices that match the required topology. Since the resources published at the CHs are specified concretely, a Portal needs to convert or embed (many be with the help of the Site) the abstract RSpec into an concrete RSpec. The RSpec generation, translation, embedding, ticketing, and lease generation process is shown in Figure 4.

Experiment Specification Mapping. Researchers use the ESpec language to script an ESpec which is fabric/platfrom neutral. The Portal maps the ESpec created onto Site/fabric specific experiment manager APIs. Thus, a Researcher may "stitch $fabric_1$ *slice* and $fabric_2$ *slice*", "inject data on $fabric_1$ *slice* from $file_1$", etc., without worrying about the details of $fabric_1$ *slice* and $fabric_2$ *slice*; She may likewise repeat the same experiment on different fabric slices easily. In KanseiGenie, we attempt to use the same APIs for different aggregates whenever possible; nevertheless, whenever the notion of a service (say logging on a Virtual Machine fabric versus a Mote fabric) is different between fabrics, the complexity of mapping the configuration onto the corresponding APIs is handled by the Portal.

Federated Slice Stitching. When conducting a federated experiment, a Researcher expects seamless communication between the resources in the federated slice, which means that sub-slices from different Sites needs to be stitched together to form a federated slice. While individual Sites provide the stitching services as part of the AAM, the Portal possesses the knowledge for implementing stitching (such as VLAN numbers, IP addresses, ports, web URLs, etc.). Note that multiple types of stitching might be needed (and possible) depending on the sensing fabrics involved and their capabilities, e.g., it is easy to stitch a federated slice consisting exclusively of virtual machines connected by wired virtual LANs, while it is much harder to stitch two wireless network slices to create a single federated wireless slices.

5 Conclusion

In this paper, we described the KanseiGenie software architecture for federated WSN fabrics. We argued for letting WSN fabrics choose their own Resource Specification ontologies and resolve the diversity in resource specifications by letting the Portal map the Site specific description to a local ontology with which the Researchers can interact. We also illustrated the need for a Experiment Specification language to enable Researchers to uniformly interact with multiple WSN fabrics in the federation; Experiment Specifications further enable scripted experimentation and complex staging between fabrics. As KanseiGenie grows to accommodate other sites, it remains to be seen whether a need to develop other Portals will emerge or some of the federation functionality in the Portal will migrate to Clearinghouses/Sites.

Finally, we share here a few of the lessons from our experience with federated testbed design. (i) RSpec and ESpec design, their completeness and adaptability is very important for the growth and research in testbed federations. (ii) A powerful Portal, such as in our design, is not always a necessity, but definitely a blessing in highly domain-specific federations such as WSN. The domain knowledge needed to exploit WSNs is very high and a fully functional Portal is a very useful tool for Researchers. (iii) The design and implementation of the Aggregate and Component Managers is a non-trivial aspect of WSN fabric design. The AMs needs to make the least assumptions about platform support, while simultaneously providing non-trivial guarantees to the Researcher. (iv) A significant amount of work is involved in maintaining a healthy testbed. A systematic approach to fault-detection and correction, and an autonomous health monitoring system are an essential part of any long living WSN infrastructure.

References

1. GENI draft, `http://groups.geni.net/geni/attachment/wiki/GEC2/TF_RSpec.pdf`
2. GENI: Global environment for network innovation, `http://www.geni.net`
3. Kansei wireless sensor testbed, `http://kansei.cse.ohio-state.edu`
4. NetEye wireless sensor testbed, `http://neteye.cs.wayne.edu`
5. Network description language, `http://www.science.uva.nl/research/air/projects/ndl/`
6. Open resource control architecture, `https://geni-orca.renci.org/`
7. Orbit network testbed, `http://www.orbit-lab.org/`
8. PeopleNet mobility testbed, `http://peoplenet.cse.ohio-state.edu`
9. Protege: Ontology editor-knowledge framework, `http://protege.stanford.edu/`
10. Shibboleth: Software package for identity and authorization management, `http://shibboleth.internet2.edu/`
11. Johnson, H.L., Cohen, K.B., Hunter, L.: A fault model for ontology mapping, alignment, and linking systems. In: Pacific Symposium on Biocomputing (2007)
12. Noy, N.F., Musen, M.A.: Prompt: Algorithm and tool for automated ontology merging and alignment. In: Proc. of the 7th National Conference on Artificial Intelligence, pp. 450–455. AAAI Press/The MIT Press (2000)
13. Roscoe, T.: Languages not Formats: Tackling network heterogeneity head-on. In: Proc. of 3rd the Workshop on Future Directions in Distributed Computing (2007)