Cyber Scientific Test Language

Peter Haglich[1], Robert Grimshaw[2], Steven Wilder[2],
Marian Nodine[3], and Bryan Lyles[4]

[1] Lockheed Martin Advanced Technology Laboratories, Virginia Beach, VA
peter.haglich@lmco.com
[2] Lockheed Martin Advanced Technology Laboratories, Cherry Hill, NJ
{robert.s.grimshaw,steven.m.wilder}@lmco.com
[3] Telcordia Technologies, Austin, TX
nodine@research.telcordia.com
[4] Telcordia Technologies, Piscataway, NJ
jblyles@acm.org

Abstract. The Cyber Scientific Method (CSM) formalizes experimentation on computer systems, hardware, software, and networks on the U.S. National Cyber Range. This formalism provides rigor to cyber tests to ensure knowledge can be shared and experiment results can be viewed with confidence, knowing exactly what was tested under what conditions. Cyber Scientific Test Language (CSTL) is an ontology-based language for CSM experiments. CSTL describes test objectives, statistical experiment design, test network composition, sensor placement, and data analysis and visualization. CSTL represents CSM experiments throughout their lifecycle, from test design through detailed test network description, instrumentation and control network augmentation, testbed buildout, data collection, and analysis. The representation of this information in a formal ontology has several benefits. It enables use of general-purpose reasoners to query and recombine test specifications for rapidly building an experiment network and testbed on the range. Additionally, it facilitates knowledge management and retrieval of test procedures and results.

Keywords: Ontology, Computer Hardware and Software, Experimentation, Statistical Design of Experiments.

1 Introduction

This paper presents the ontology-based Cyber Scientific Test Language (CSTL) for the Cyber Scientific Method (CSM). The CSM was developed in conjunction with the Lockheed Martin team's implementation of the U.S. Defense Advanced Research Projects Agency (DARPA) National Cyber Range (NCR) program. CSM provides a discipline for rigorous experimental design in the performance of tests of computer hardware, software, systems, and networks. The CSM and CSTL enable the formal description of the system under test (SUT), the test instrumentation and environment, experiment assumptions and objectives, and test results. CSTL serves as a lingua franca for cyber test representation throughout the entire experiment planning, execution, and analysis cycle. During experiment planning, CSTL helps the Test Scientist

L. Aroyo et al. (Eds.): ISWC 2011, Part II, LNCS 7032, pp. 97–111, 2011.
© Springer-Verlag Berlin Heidelberg 2011

to develop a statistical design for the experiment. As the experiment is executed, range software uses the CSTL test specification to automatically build the testbed and to perform the experiment via the insertion of test events. After the test has been conducted, experiment results expressed in CSTL are processed and are available for later review.

CSTL consists of a modular ontology library expressed in the OWL 2 Web Ontology Language [1]. We chose OWL 2 for several reasons, including our team's familiarity with OWL and OWL editing tools. We also wanted to leverage OWL 2 language features such as property chains. Finally, because NCR is a multi-year project we wanted to start with the newest version of OWL.

Our design approach for CSTL was driven by the overall system-engineering plan for NCR. Our ontology development was grounded in the need to express information related to test design, execution, and analysis and testbed buildout and sanitization. We validated our design through the representation of examples in CSTL, reviewed by subject matter experts and NCR component developers. Later in the ontology development phase we instituted a configuration control board to review and approve proposed changes to CSTL.

The major components of CSTL cover:

- **Experimental Design:** This segment of CSTL describes ideas used in designing an experiment, including the statistical design. Topics covered here include objectives, hypotheses, experiment type, treatments, factors, and response variables.
- **Test Planning and Administration:** A small portion of CSTL provides terms to describe some of the administrative details of planning an experiment, such as facility scheduling and experiment sponsorship.
- **Network Descriptions:** An important module within CSTL describes computer networks in terms of their topology. These terms include links, nodes, and subnets.
- **Asset Descriptions:** Terms for representing hardware and software assets form a major coherent and independent module within CSTL. This asset description ontology is explained more fully in [2].
- **Test Execution:** CSTL supports the automated transition of test specifications to a testbed during experiment buildout via transformations to the Network Simulator (ns-2) language [3]. Furthermore, CSTL describes automatic network traffic generation and test event execution subsystems for use during the performance of the experiment.
- **Data Analysis and Visualization:** A segment of CSTL is used to represent the instrumentation of the testbed with sensors and the associated metrics to be computed. This segment also describes the data archiving and analysis process.
- **Templates:** In cyber testing it is often desirable to define the SUT using multiple clones of identical individuals. CSTL provides a method for defining and instantiating templates to provide this capability. Templates are also used in incorporating predefined control structures such as traffic generation into the testbed.

We will provide more detail on each of these topic areas.

2 Experiment Design and Planning in CSTL

A major goal for the CSM was the unification of test and evaluation methodology with computer and network system descriptions. Statistical experiment design formalizes the definition of an experiment through the identification of response variables, treatment variables, and other experimental factors. In the CSTL ontology library these concepts are captured in a separable design of experiments (DoEx) ontology module, *designofexperiments.owl*, which is applicable to other types of test and evaluation. Note that this segment of CSTL can be used to describe general statistical experiment design, even outside of the cyber domain.

In this section we will describe the major concepts in this module.

2.1 Experiment Types

The DoEx framework in CSTL includes ontology classes for four experiment types. The most general is the Discovery Experiment, in which testers observe the system under a variety of conditions. Discovery experiments often include humans in the loop. A second type of experiment is the Diagnostic Experiment, typically used for troubleshooting the behavior of a malfunctioning system or network. A third type of experiment is the Benchmark Experiment, whereby network or system performance characteristics are measured. The fourth type of experiment is the Statistically Designed Experiment. The remainder of this section will summarize the major concepts related to this type of experiment. The representation and process described herein is based on a Job Aid developed by Sandia National Laboratories [4] to assist researchers with DoEx.

2.2 Objectives and Hypotheses

At the start of experiment planning, it is good to state the objective or purpose of the experiment. Along with the objective, statistically designed experiments address a quantitative hypothesis based on a test statistic, which is a function of the experiment observations. In DoEx this hypothesis is expressed in terms of a null hypothesis and an alternative hypothesis. Most commonly, the null hypothesis represents the status quo and the alternative hypothesis represents the effect that is being investigated. For example, in an experiment to examine the effectiveness of an update to an anti-virus program, the null hypothesis could be that the new version has the same probability of allowing infection as the old version. An alternative hypothesis could be that the new program is 90% more effective than the old program.

The DoEx ontology includes classes for Objective, Hypothesis, and Test Statistic.

2.3 Treatments, Factors, and Response Variables

DoEx is designed to support the efficient planning of a set of experiment runs for data collection to determine the degree of evidence to reject the null hypothesis. This is

based on the value of a test statistic computed from observations of response variable values during repetitions of the experiment, or runs. For the anti-virus example, the test statistic might be the number of infections observed divided by the number of tests run on each version. Statistical tests can use the values of the test statistic for the original version and the updated version in order to recommend rejection of the null hypothesis in favor of the alternative hypothesis.

The response variables are assumed to have a dependency on one or more input variables or experimental factors. In fact, the objective of the experiment is usually related to determining the response of the system to a subset of the input variables, known as treatments. The other input variables are also important, and need to be controlled or measured for later analysis.

In the cyber domain, these input variables represent system or network characteristics (e.g., available bandwidth), capacity (e.g., number of connected hosts), or configurations (e.g., which variant of Windows is installed on experiment hosts). In our anti-virus example, the fundamental treatment is the version of the anti-virus program used. An example of a non-treatment experimental factor to be controlled in this scenario is the service pack level of the underlying host operating system.

2.4 Designs

At an early stage of designing the experiment, the test scientist(s) may use a brainstorming process to identify potential treatment and factor variables for inclusion in the experiment, along with response variables. In CSTL we refer to this identification of variables as a proto-design. After further consideration the scientist will establish a final set of response, treatment, and factor variables in the experiment design. CSTL allows the scientist to tag the variables considered during brainstorming as being accepted for inclusion in the final design or as being excluded. This allows a later review of the test to determine which variables were considered but not included. For each accepted treatment and controlled factor variable, he will also specify the potential setting values. In our example there are only two setting values for the anti-virus treatment {'New Version', 'Old Version'}. If the underlying operating system is Windows XP there might be four setting values for service pack level {'SP0', 'SP1', 'SP2', 'SP3'}. A set of treatments or controlled factors with their setting values is called a treatment combination [6]. For example, combining the new version of the anti-virus software with Windows XP SP1 would produce one of the eight possible unique treatment combinations. Each run in the experiment is associated with a treatment combination. In the DoEx ontology the Experiment Design individual is linked to its response, input, and factor variables and its treatment combinations.

The top-level concepts in the DoEx ontology are shown in Fig. 1.[1]

[1] The figures in this paper are concept maps of ontology classes (rounded rectangular nodes), properties (labeled links), and individuals (rectangular nodes). These concept maps were developed using CMap Tools [5].

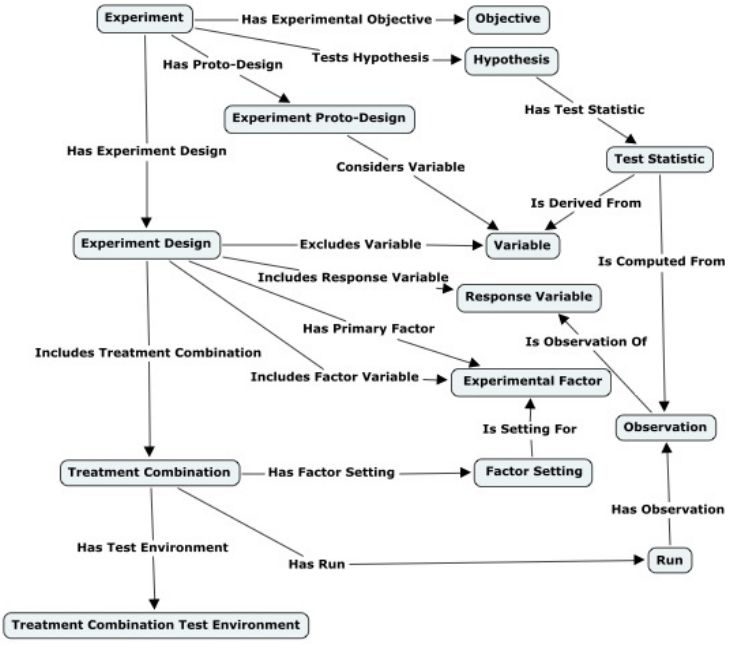

Fig. 1. DoEx Ontology

3 Test Environment

Each treatment combination defines a unique (up to isomorphism) test environment that is used for all runs associated with that treatment combination. This environment is represented in the CSTL ontology by the Treatment Combination Test Environment class. This class serves as a bridge between the statistical experiment design structure of the experiment and the system under test as embedded in the test. Individuals of the Treatment Combination Test Environment class are linked to two major individuals that further describe the test environment. These are members of the Network and Cyber Scientific Test Procedure ontology classes, each of which will be described in this section.

3.1 Network Topology Concepts

The *networktopology.owl* module provides extensions to the basic CSTL network concepts. The fundamental network concept is the Network Topology, which can describe either the logical topology or the physical topology of the network. The basic model for the network topology is a graph defined by a set of links and nodes. In most cases these links are symmetric but the network topology ontology does include terms to model asymmetric networks. Links are associated with nodes via the "Has Endpoint" property. Nodes may be directly asserted to be members of the network topology, but it is preferable to infer that the nodes belong to the topology using the property chain **Has Link** composed with **Has Endpoint** implies **Has Node**. Additional link and node

properties such as bandwidth, latency, addressing, and subnet masks are also defined in the network topology ontology. The current version of this ontology focuses on IPv4 terminology that is compatible with ns, but does provide for extensions to other addressing protocols.

The nodes and links provide the topological view of the network. Each node may also be associated with a device such as a computer host or a routing device. The network topology ontology provides object properties to make these associations.

A simple view of the network topology ontology is shown in Fig. 2.

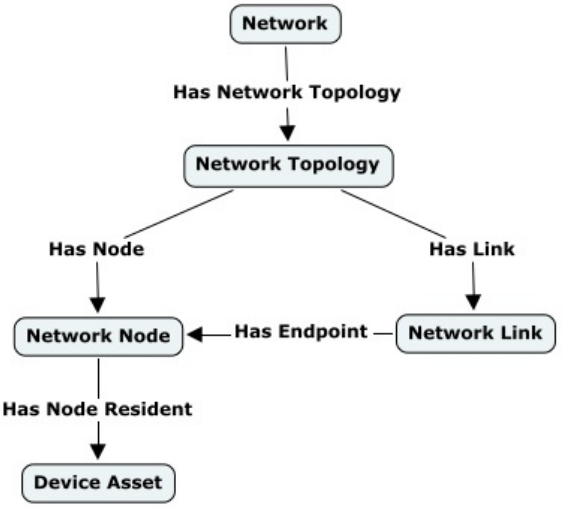

Fig. 2. Simple Network Topology

3.2 Test Procedures

The test environment is also linked to an individual from the Cyber Scientific Test Procedure class. This class serves as a collector for test instructions and event execution scripts. Test instructions can include written instructions for those aspects of the test that are not automated, or ground rules for human-in-the-loop experiments. Event execution scripts are described in Section 5.

4 Asset Description

A major portion of the CSTL ontology library is the collection of concepts that describe hardware and software assets. The asset description ontology included in CSTL is described at length in [2]. In this section we summarize the use of asset descriptions in CSTL.

4.1 Hardware and Software Assets

Repeatable experiment specification requires precise definition of the system under test and the testbed articles. For computers and other devices, a full description

includes identification of hardware components and installed software. Assets are described nominally (make and model) and according to their functions. Software asset function descriptions also specify installation and runtime dependencies and compatibility with other software and hardware.

4.2 Recipes, Images, and Asset Controllers

A goal of the NCR is automation of cyber experiments so that they can be performed quickly and exactly. This is important for efficiency in large-scale experiments involving large numbers of nodes and links, treatment and factor variables, or runs per treatment combination. Leveraging previous experience with Emulab [7], CSTL includes terms to describe the recipes used to build and configure host assets using disk images and asset controllers. Asset controllers are auxiliary software assets that can install and configure software on hosts. Disk images and partitions are prebuilt and copied to the experiment host. Host recipes coordinate and specify the building of host assets from images and asset controllers. The recipe optionally starts with a disk image loader which identifies the image that is to be copied onto the host and which partition is to be booted. The recipe then specifies an ordered list of recipe steps, each of which points to an asset controller and provides parameter value assignments for items such as license keys or installation directories. During testbed buildout, experiment control software builds the hosts according to the information contained in the host recipe.

Figure 3 illustrates the classes and object properties that describe the building of host assets from recipes via images and asset controllers.

Given a recipe, a reasoner can determine what software asset types are installed on each host in the experiment by following property chains using the object properties shown above. There are two property chains that provide this information, represented by paths from Host Asset to Software Asset or Software Asset Type respectively. Absent a reasoner, this information can also be computed by joining SPARQL queries over an RDF graph.

5 Test Execution

CSTL includes terms to describe the control structures and events involved during the performance of experiment runs. These include execution scripts, event coordination, and production of network traffic.

5.1 Event Execution Language

The Lockheed Martin implementation of NCR features a scripting language for the execution of test events. This Event Execution Language (EEL) is not directly ontology-based. Rather, CSTL encapsulates EEL scripts and can express script arguments. Each test procedure may refer to a set of EEL scripts for testbed buildout, validation, test execution, data archiving, post-test analysis, and testbed sanitization associated with it. Additionally, EEL scripts may be passed argument values in a manner analogous to the manner of providing parameters for asset controllers described in Section 4.2. So, while EEL is mostly opaque to CSTL, CSTL still provides ways to link to appropriate EEL content.

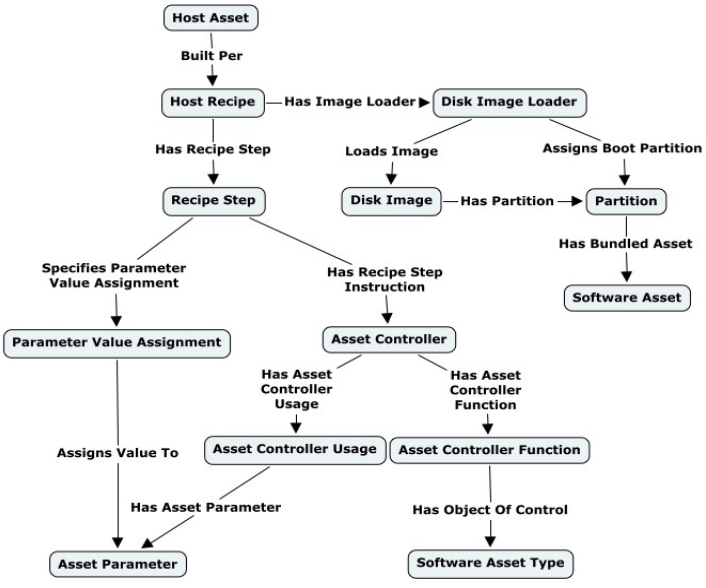

Fig. 3. Recipes, Images, and Asset Controllers

5.2 Test Execution Control Infrastructure

To ensure that test events are triggered automatically, in a synchronized fashion, the testbed needs to provide control infrastructure that interacts with the system under test but is not part of the system under test. The test designer does not usually need to specify the details for the control network at test specification. However, CSTL does need to be able to describe the control infrastructure for two reasons. First, elements of the system under test need to be identified as connection points for the control infrastructure. Second, after the test has been performed it is important to record all of the elements of the testbed and the system under test for provenance and experiment repeatability.

The Lockheed Martin implementation of NCR includes deployment of a test event timing execution server (TET Server) and test management control/view application (TMVC) for test directors. CSTL provides descriptions for these as well as test event execution software added to the hosts in the system under test.

5.3 Artificial Network Traffic Generation

In order to run a cyber experiment that represents real network usage without requiring humans in the loop, it is necessary to generate network traffic artificially. As used by Lockheed Martin in NCR, artificial traffic generation is managed by a node in the control network, built according to a pre-defined recipe expressed in CSTL. Hosts in the system under test are also provided with additional instrumentation software to support the generation of web browsing or email traffic.

Similarly to TET and TMVC, the traffic generation control node and additional experiment host software are described in CSTL and included with the full description of the experiment after execution.

6 Data Analysis and Visualization

Collecting the appropriate data, analyzing it, and visualizing it are important aspects of experimentation. The nature of cyber experimentation is such that large-scale experiments with a myriad of potential data elements are possible. In order to keep up with the pace of the experiment and to ensure all of the desired data is collected, it is imperative that the test specification include descriptions for the sensors and data to be collected.

6.1 Sensors and Metrics

Many of the sensors that are used to collect experiment data are themselves software applications, resident on either the hosts in the system under test or on the control network. When software sensors are installed on experiment hosts there is a slight potential for interference with the experiment itself. For example, a CPU load-monitoring sensor contributes in a small way to the load on the CPU. Therefore, to facilitate scientific rigor and provenance, it is important to capture details about the sensors and their operation during the experiment. Additionally, the sensors need to be controlled by the test event subsystem using the EEL language. For this reason the CSTL representation for sensors includes descriptions of the EEL statements used to control the sensor. The sensor description also includes asset descriptions for the sensor assets as discussed in Section 4.1.

The sensors provide the data observations. Normally, the raw observations are only useful to the extent that they can be used to compute metrics describing the performance of the system under test. These metrics are computed using the data analysis subsystem described in Section 6.2. The test scientist selects the metrics. CSTL describes these metrics in terms of the sensors from which they receive input and other parameters.

6.2 Data Archiving and Analysis Subsystem

To support data archiving and analysis, the experiment control network includes a Data Analytics Supercomputer (DAS) developed by Lexis-Nexis [8]. As part of experiment definition the test scientist also needs to provide some details for DAS operation, including data archiving location and frequency and metric computation intervals. For provenance reasons this is also encoded in CSTL and saved with the experiment description.

7 Templates

There are two major cases where cyber experiment descriptions require the description of individuals that are identical modulo a small set of data property values or object property linkages. The first major case occurs most frequently during scalability testing, where a large number of identical hosts are added to the system under test. The second major case occurs with the addition of control network infrastructure to the testbed. To support these major cases, CSTL provides for the creation, application, and instantiation of templates. The idea behind a template is that it provides the instructions needed to create a set of individuals that can be linked into the Resource Description Framework (RDF) graph of ontology individuals that describes the test specification. In some cases, the graph of the template looks like the graph of the

individuals generated from it, with Internationalized Resource Identifiers (IRIs) assigned and data property values provided. The specification of a template includes object parameters, data parameters, invariant parameters, and multiplicity values.

The framework for templates that we developed for CSTL is not limited to use in cyber test specification. Thus, the same approach would work in other domains. A detailed exposition of our template approach will appear in a future paper. In the remainder of this section we will discuss the main ways templates are used in CSTL.

7.1 Templates for Cloning

One common scenario in cyber testing is the connection of multiple hosts to a switch or router in a star network topology as part of the system under test. The hosts may be identical except for host name and IP address assignment. That is, they may all be built from the same recipe or disk image. The experiment may vary the number of hosts for purposes of scalability testing. The template for this network would feature:

- Object parameters for the hosts, network nodes, and network links.
- Invariant (singleton) object parameters for the host recipe, the switch and its network node.
- Data parameters for switch, host, node, and link characteristics.
- A multiplicity parameter for the number of hosts desired, expressed as a data property value on the host node object parameter individual.

The template is illustrated in Fig. 4. An instantiation of the template is shown in Fig. 5.

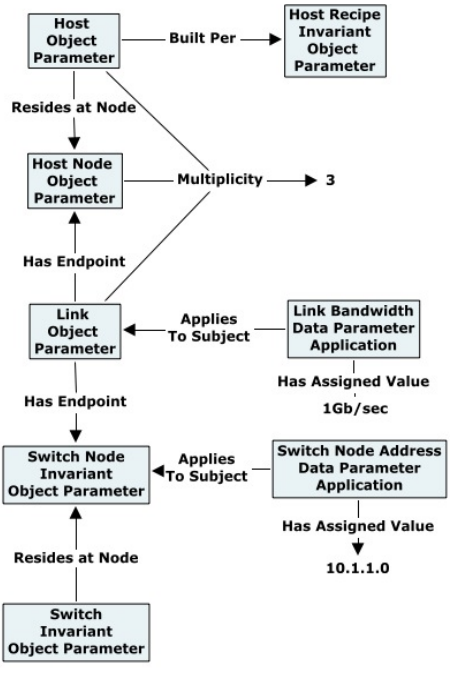

Fig. 4. Example Star Topology Template

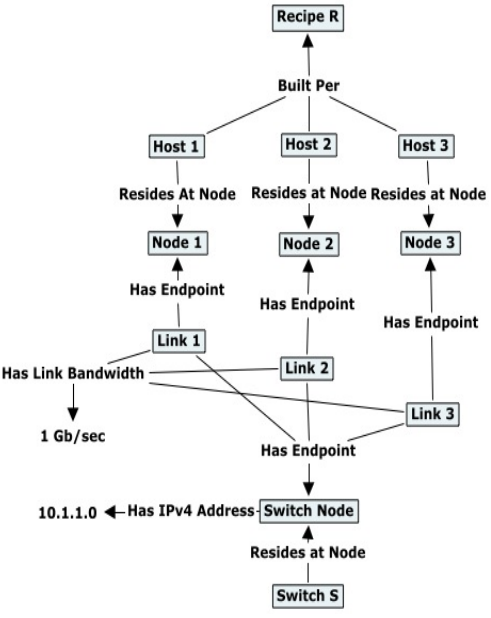

Fig. 5. Star Topology Template Instantiated

The instantiation of the template produced 12 new individuals, 11 object property assertions linking them, and four data property assertions. The resulting network has three host nodes connected to a single switch node. Different values for the multiplicity parameter would have produced different numbers of hosts, host nodes, and links. Tying this back to the design of experiment considerations discussed in Section 2.3, one can envision an experiment design with two factors: the number of hosts in the network and the software recipe used to build each of the hosts. For example, one of the treatment combinations might use seven hosts and recipe "B". Another might use five hosts and recipe "A". The setting values would be used as parameter assignments by the template to quickly define the system under test for a given run.

7.2 Templates for Test Infrastructure

The second major motivating case for using templates is the use of predefined test support assets in the experiment. For example, artificial traffic generation requires the installation of client software on hosts in the system under test. In Fig. 6 we see an example of such a template. Traffic generation requires software added to the experiment hosts to control Thunderbird for email and Internet Explorer for web browsing. The template is applied to the system under test by passing IRIs for the hosts and their recipes as arguments. The recipe extension individual is created and linked to the recipe steps and the original recipe for the host via appropriate object properties.

The Lockheed Martin specific portion of the CSTL ontology library includes similar templates for traffic generation, test execution, test management view, and sensors.

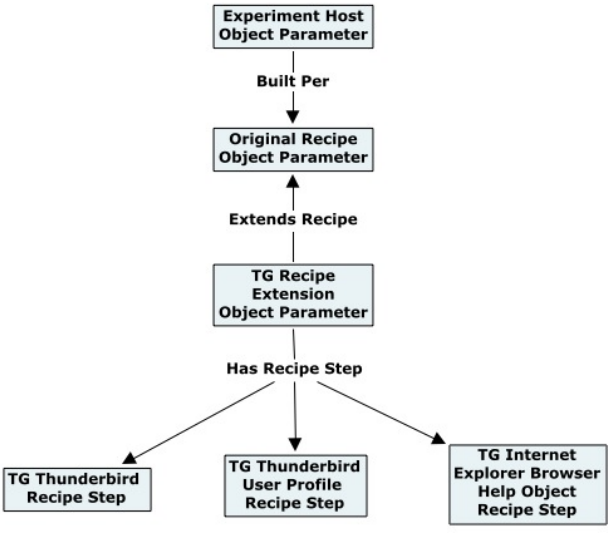

Fig. 6. Traffic Generation Support Template

8 Sparse Descriptions

In cyber testing it is not desirable to require a test scientist to provide exact descriptions of all aspects of the elements of the system under test. For example, the test scientist may not care about the exact clock speed of host CPUs above some minimum threshold. Furthermore, range scheduling is more flexible when reasonable substitute assets can be used in experiments. For this reason CSTL has adopted the notion of sparse description of test assets. A sparse description defines a class expression on the set of available asset types. It may be associated with a query on the inventory of assets at the range to determine a set of acceptable assets for use in the experiment. This allows the test scientist to focus on the characteristics of greatest importance while being flexible in other characteristics. It also allows the range facility operator to apply additional business rules in scheduling range assets while meeting the needs of the test scientists.

As we studied the situations where sparse descriptions were desired we realized that there are two fundamental kinds of sparseness. The first kind we refer to as *categorical sparseness*. The simplest form of categorical sparseness is a named ontology class. (For example, "x86 Host Asset Type".) More complicated categorical sparseness might include existential restrictions on object properties defined on a base class. (Example: "Software providing the NTP Client Function"). The second kind of sparseness we refer to as *range-constrained sparseness*. This kind of sparseness focuses on values for data properties and whether they fall in a particular range. An example might be "Processor with clock speed no less than 2.0 GHz".

Sparse descriptions correspond naturally with queries against the knowledge base to list the individuals that fit the sparse descriptions. For example, in the SPARQL [9] query language, data property value range constraints correspond to FILTER statements in queries.

Please note that while the idea of a sparse description was motivated by some of the challenges of cyber test design and planning, the idea can be generalized to other domains.

9 Transforms

While CSTL can be used to represent most of the details of a cyber experiment, there are some components of the NCR that require different input formats. Also, there are elements of the experiment lifecycle that require code to perform transformations on the test specification. For this reason the Lockheed Martin team has developed a set of transformations that operate on CSTL to produce either modified CSTL or other output formats. These transformations are performed by custom Java code using the Jena OWL Microreasoner [10]. Some of the CSTL transforms are:

- **NS Translator:** This transform takes a CSTL network description in OWL and produces a network description in ns format, suitable for use by Emulab.
- **Treatment Combination Expansion Transform:** This transform creates treatment combination test environments from a basic test environment and the collection of factor settings from the experiment design. The input and output are both expressed in CSTL.
- **Recipe Infrastructure Transform:** This transform takes host recipes as inputs and extends them to add several pieces of software needed for test instrumentation. The input and output of the transform are expressed in CSTL.
- **Template Instantiator:** As the name suggests, this transform takes a template specification in CSTL along with parameter value assignments and produces the resulting individuals in a similar manner to that discussed in Sections 7.1 and 7.2. The output format is CSTL.

10 Query Library

In addition to the CSTL ontology library, we have developed a library of knowledge base queries in the SPARQL language. These queries are used by components of NCR to retrieve germane information. For example, tools to automatically build the system under test in the testbed perform queries against the test specification to determine which disk images and installers to use. The query library contains many queries that can be used by any tool that can execute SPARQL queries. An additional use of the query library is in regression testing. A collection of sample test specifications serves as an adjunct extension to the main CSTL ontology library. Whenever changes are made to CSTL we can run the query library and compare it with a baseline set of results to determine the impact of the ontology changes. We can examine this impact from the standpoint of APIs for the NCR software components that use the query. We also have a set of sanity check queries that allow us to look for missing or correct information that might result from an ontology change.

11 Challenges

The development of the CSTL ontology library presented a number of challenges. These are categorized and discussed below.

11.1 Reasoner Support and OWL 2 Features

The CSTL ontology library is fairly complex. We wished to leverage OWL 2 features such as property chains to express natural domain axioms. We mentioned some of these earlier in the paper. Unfortunately, the available reasoners that could compute property chains were unable to reason on the whole of the CSTL ontology library in a satisfactory time. Therefore we had to suspend the use of property chains and require tools to make assertions of statements that would have been derivable using property chains. We are hopeful that as reasoners get more capable we will be able to return to using these advanced features in the future.

11.2 Second-Order Reasoning and Punning

There were a few cases where the design of CSTL had the potential to require second order reasoning. One of these areas, involving assets, asset types, and classes of asset types, is discussed in [2]. The second case arose in the development of templates, specifically in the application of data parameters. The description of how a template data parameter is to be applied needs to identify the appropriate data property in the ontology that provides the predicate for the assertion (subject, predicate, assigned data value). We did not want to use an object property that took the data property as a value, as this would cause difficulty in reasoning. To work around this, we use the IRI for the data property as a data value, which provides enough information to the template instantiator.

12 Summary

This paper provided an overview of the Cyber Scientific Test Language (CSTL) ontology library as used by the Lockheed Martin team for the National Cyber Range (NCR). CSTL can be used to describe experiment design, the computer systems and networks under test, test instrumentation, test planning and administration details, and data analysis. Test specification and reports expressed in CSTL facilitate knowledge management and provenance of results. CSTL includes approaches for the use of templates and sparse descriptions that are broadly applicable outside of the cyber domain.

The Lockheed Martin implementation of NCR includes custom code for transforming CSTL, with output to CSTL or other languages such as ns. An extensive query library is used as an API for NCR components and for regression testing. The implementation of a knowledge base for NCR needed to overcome or work around some challenges that limited our ability to use the full expressivity of OWL 2. We hope to overcome this in the future.

Acknowledgment. The United States Defense Advanced Research Projects Agency (DARPA) funded this research and development activity under contract HR0011-09-C-0042. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. This is in accordance with DoDI 5230.29, January 8, 2009. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

References

1. OWL 2 Web Ontology Language Document Overview (2009), http://www.w3.org/TR/owl2-overview/ (last accessed on August 19, 2011)
2. Nodine, M., Grimshaw, R., Haglich, P., Wilder, S., Lyles, B.: Computational Asset Description for Cyber Experiment Support using OWL. Submitted to International Conference on Semantic Computing 2011 (2011)
3. The Network Simulator – ns-2 (2010), http://www.isi.edu/nsnam/ns/ (last accessed on August 19, 2011)
4. Halbleib, L.L., Crowder, S.V.: Teaching Design of Experiments Using a Job Aid and Minitab, http://www.minitab.com/en-US/uploadedFiles/Shared_Resources/Documents/Articles/Crowder.pdf (last accessed on August 19, 2011)
5. Eskridge, T., Hayes, P., Hoffman, R., Warren, M.: Formalizing the Informal: A Confluence of Concept Mapping and the Semantic Web. In: Cañas, A.J., Novak, J.D. (eds.) Concept Maps: Theory, Methodology, Technology. Proceedings of the Second International Conference on Concept Mapping, vol. 1. Universidad de Costa Rica, San Jose (2006)
6. NIST/SEMATECH e-Handbook of Statistical Methods (2010), http://www.itl.nist.gov/div898/handbook/pri/section1/pri1.htm (last accessed on August 19, 2011)
7. Emulab.Net – Emulab – Network Emulation Testbed Home (2011), http://www.emulab.net/ (last accessed on August 19, 2011)
8. Data Analytics Supercomputer – LexisNexis (2011), http://www.lexisnexis.com/government/solutions/data-analytics/supercomputer.aspx (last accessed on August 19, 2011)
9. W3C. SPARQL Query Language for RDF (2008), http://www.w3.org/TR/rdf-sparql-query/ (last accessed on August 19, 2011)
10. Reynolds, D.: Jena 2 Inference Support (2010), http://jena.sourceforge.net/inference/#owl (last accessed on August 19, 2011)