SISO-GUIDE-010-2020

Guide for Command and Control Systems - Simulation Systems Interoperation

Version 1.0

25 April 2020

Prepared by:
Command and Control Systems - Simulation Systems Interoperation
Product Development Group

P.O. Box 781238
Orlando, FL 32878-1238, USA

SISO Inc. Board of Directors
P.O. Box 781238
Orlando, FL 32878-1238, USA

Revision History

Version	Section	Date (MM/DD/YYY)	Description
1.0	All	29 March 2020	Initial version from balloting

Participants

At the time this product was submitted to the Standards Activity Committee (SAC) for approval, the Command and Control Systems - Simulation Systems Interoperation Product Development Group had the following membership and was assigned the following SAC Technical Area Director:

Product Development Group

Dr. J. Mark Pullen (Co-Chair)
Kevin Galvin (Co-Chair)
Bruno Gautreau (Vice-Chair)
Dr. Douglas Reece (Vice-Chair)
Dr. Rob Wittman (Vice-Chair)

Dr. Curtis Blais (SAC Technical Area Director)

Jeff Abbott
Elaine Blount
Donald Brutzman
Nicholas Clark
Douglas Corner
Anthony Cramp
Xavier Cuneo
Thomas DeCarlo
Todd Decosta
Bradford Dillman
Uwe Dobrindt
Bruno Gautreau
Arno Gerretsen
Kevin Gupton
Paul Gustavson
Mark Hazen
Frank Hill
Elizabeth Hosand
Lionel Khimeche
Patrice Le Leydour
Kenneth LeSueur
Sean Litton
Sebastien Loze
Lance Marrou
Craig Marsden
Priscilla McAndrews
Mark McCall
Ole Martin Mevassvik
Michael Montgomery
Laurent Mounet
William Oates
Laurent Prignac
Nelson Reynolds
David Ronnfeldt
James Ruth
Roy Scrudder
Tommy Shook
John Shue
Robert Siegfried
Samuel Singapogu
Eric Whittington

The Product Development Group would like to especially acknowledge those individuals that significantly contributed to the preparation of this product as follows:

PDG Drafting Group

Dr. Samuel Singapogu (Lead Editor)
Thomas DeCarlo (Secretary)

Curtis Blais
Douglas Corner
Magdalena Dechand
Kevin Galvin
Bruno Gautreau
Lt Col Jens-Inge Hyndoy
Dr. Mark Pullen
Dr. Douglas Reece
James Ruth
Dr. Rob Wittman

The following individuals comprised the ballot group for this product.

Ballot Group

Deryck Arnold
Mohammad Ababneh
Curtis Blais
Davor Braut
Donald Brutzman
Bruce Clay
Douglas Corner
Anthony Cramp
Todd DeCosta
Saikou Diallo
Peter zu Drewer
Kevin Galvin
Bruno Gautreau
Jean-Louis Gougeat
Kevin Gupton
Elizabeth Hosang
Lionel Khimeche
Katherine Morse
Bharat Patel
J. Mark Pullen
Douglas Reece
Samuel Singapogu
Geir Sletten
Jeffrey Sugden

When the Standards Activity Committee approved this product on 04 June 2020, it had the following membership:

Standards Activity Committee

David Ronnfeldt (Chair)
Curtis Blais (Vice Chair)
Katherine Ruben (Secretary)

Grant Bailey
Brad Dillman
David Graham
Peggy Gravitz
John Hughes
Patrice Le Leydour
Sebastien Loze
Lance Marrou
Chris McGroarty
David ‘Fuzzy’ Wells

Executive Committee

Robert Lutz (Chair)
Jeff Abbott (Vice Chair)
Kenneth Konwin (Secretary)

Damon Curry
Paul Gustavson
Kurt Lessmann
Mark McCall
Lana McGlynn
Chris Metevier
Katherine Morse
David Ronnfeldt
Stefan Sandberg
Robert Siegfried

Introduction

Command and Control Systems to Simulation Systems Interoperation (C2SIM) is a standard for expressing and exchanging Command and Control (C2) information among C2 systems, simulation systems, and robotic and autonomous (RAS) systems in a coalition context.

This document is the guidance document for the C2SIM standard document. It has been assembled from various documents prepared by the C2SIM Drafting Team and used to support development and evaluative testing of C2SIM, in an extensive collaboration with North Atlantic Treaty Organization (NATO) Science and Technology Organization Modelling and Simulation Group Technical Activity 145. The various sections cover topics helpful to understand C2SIM's use of ontologies and also practical issues associated with deployment. Further documents describing development, testing and validation of C2SIM can be found in SISO Simulation Innovation Workshop proceedings from 2008 to present, in sessions dealing with Battle Management Language (BML) and interoperation of C2 with simulation systems and RAS.

Table of Contents

List of Figures

1 Brief overview of OWL

The Web Ontology Language (OWL) is a World Wide Web Consortium (W3C) Recommendation specifying a formal language for expressing classes of objects and relationships across those classes of objects. Such information is called an *ontology*, commonly defined as "a formal, explicit specification of a shared conceptualization" (Gruber 1993). The OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax can be found at http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/. The reader is also referred to the OWL 2 Web Ontology Language Document Overview (Second Edition), W3C Recommendation 11 December 2012, at http://www.w3.org/TR/2012/REC-owl2-overview-20121211/#Documentation_Roadmap and the OWL 2 Web Ontology Language Primer (Second Edition), W3C Recommendation 11 December 2012, at http://www.w3.org/TR/2012/REC-owl2-primer-20121211/. Some of the main features of OWL are described in this Appendix. Refer to the W3C documentation for a full description.

OWL is part of a number of W3C standards making up the Semantic Web stack, pictured in Figure 1.

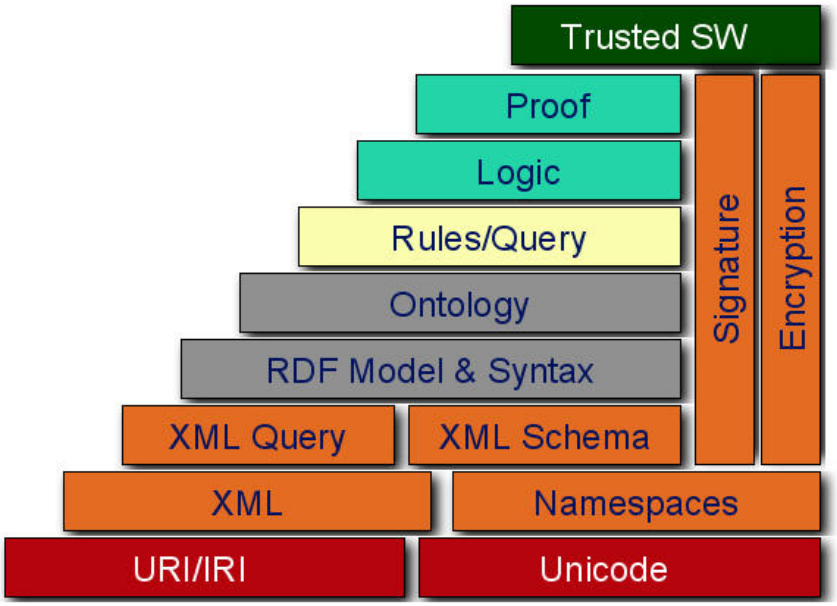

Figure 1: Semantic Web Stack (Bratt 2005)

Legend: RDF: Resource Definition Framework; SW: Semantic Web; URI/IRI: Uniform Resource Identifier / Internationalized Resource Identifier; XML: Extensible Markup Language

The Semantic Web is Tim Berners-Lee's vision of a trusted "web of data that can be processed directly or indirectly by machines" (Berners-Lee 1999). The Semantic Web stack provides a way to describe information on the World Wide Web with increasing levels of formalized semantics supporting information query, automated reasoning, and information integrity.

OWL provides a way to make assertions about various classes of objects and about individuals who may belong to those classes. In the case of C2SIM, the assertions describe objects, actions, or events occurring in the battlespace, or information relating to the structure and content of messages that will be exchanged across systems. The top-level class or concept in OWL is referred to as Thing (the most general concept). All other "things" in the domain of interest become specializations of Thing. For example, at the top level of the C2SIM ontology, we declare the C2SIMContent as a class, implicitly considered as a subclass of Thing. This assertion is expressed in OWL/Extensible Markup Language (XML) format as follows:

```
<owl:Class rdf:about="http://www.sisostds.org/ontologies/C2SIM#nceptoncept">
   <rdfs:comment>Top-level class for C2SIM concepts for information content and messaging.</rdfs:comment>
</owl:Class>
```

The URI (uniform resource identifier) in the *rdf:about* attribute of the *owl:Class* XML element precedes the name of the class (C2SIMContent) with the namespace for the C2SIM language, namely, http://www.sisostds.org/ontologies/C2SIM. As with common XML usage, namespaces allow an identical term to be used in multiple domains but be distinguished by its particular namespace modifier.

Moreover, the C2SIM ontology declares three subclasses of C2SIMContent; namely, C2SIMContent, MessageConcept, and InitializationConcept. The first two of these are expressed in OWL/XML format as follows:

```
<owl:Class rdf:about="http://www.sisostds.org/ontologies/C2SIM#C2SIMContent">
   <rdfs:subClassOf  rdf:resource="http://www.sisostds.org/ontologies/C2SIM#C2SIMContent"/>
   <rdfs:comment>Class of all concepts in the core C2SIM model.</rdfs:comment>
</owl:Class>

<owl:Class rdf:about="http://www.sisostds.org/ontologies/C2SIM#MessageConcept">
   <rdfs:subClassOf rdf:resource="http://www.sisostds.org/ontologies/C2SIM#C2SIMContent"/>
   <rdfs:comment>The subclasses of this class define the concepts needed to form message envelopes for
C2SIM.</rdfs:comment>
</owl:Class>
```

Notice these two *owl:Class* statements both contain the *rdfs:subClassOf* child element where the statements identify the parent class as the previously defined C2SIMContent class. Following set theory, classes can also be declared as intersections or unions of other classes, as we will see in later examples.

Greater specificity in the declarations occurs when classes are declared in terms of other classes and specific properties on those classes. OWL provides the means to declare two kinds of properties: (1) *object properties*, expressed as relations between members of a class and members of another class; and (2) *data properties*, expressed are relations between members of a class and specific literal values, such as strings or integers. The existence of such relationships, sometimes following certain restrictions, can be used in the declaration of a class, as in the declaration of the Action subclass of C2SIMContent below:

```
<owl:Class rdf:about="http://www.sisostds.org/ontologies/C2SIM#Action">
   <rdfs:subClassOf rdf:resource="http://www.sisostds.org/ontologies/C2SIM#C2SIMContent"/>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="http://www.sisostds.org/ontologies/C2SIM#hasPerformingEntity"/>
         <owl:minQualifiedCardinality rdf:datatype=
"http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:minQualifiedCardinality>
         <owl:onClass rdf:resource="http://www.sisostds.org/ontologies/C2SIM#Entity"/>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="http://www.sisostds.org/ontologies/C2SIM#hasActionTaskActivity Code"/>
         <owl:qualifiedCardinality rdf:datatype=
"http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:qualifiedCardinality>
         <owl:onDataRange rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="http://www.sisostds.org/ontologies/C2SIM#hasUUID"/>
         <owl:qualifiedCardinality rdf:datatype=
"http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:qualifiedCardinality>
         <owl:onDataRange rdf:resource="http://www.sisostds.org/ontologies/C2SIM#UUID"/>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:comment>An action that an ActorEntity can perform to change the state of the world.</rdfs:comment>
</owl:Class>
```

These statements declare the Action class to be a subclass of the intersection of several sets of “things”; namely:

- A subclass of things that are in C2SimContent (i.e., a subclass of C2SimContent):

```
<rdfs:subClassOf rdf:resource="http://www.sisostds.org/ontologies/C2SIM#C2SIMContent"/>
```

- *And* (intersection with), a subclass of things that have a *hasPerformingEntity* object property with at least one (*minCardinality* = 1) member of the Entity class:

```
<rdfs:subClassOf>
   <owl:Restriction>
      <owl:onProperty rdf:resource="http://www.sisostds.org/ontologies/C2SIM#hasPerformingEntity"/>
      <owl:minQualifiedCardinality rdf:datatype=
"http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:minQualifiedCardinality>
      <owl:onClass rdf:resource="http://www.sisostds.org/ontologies/C2SIM#Entity"/>
   </owl:Restriction>
</rdfs:subClassOf>
```

- *And*, a subclass of things that have a hasActionTaskActivityCode data property with exactly one (owl:qualifiedCardinality) string value (XML schema data type denoted string from the XML: schema namespace http://www.w3.org/2001/XMLSchema):

```
<rdfs:subClassOf>
   <owl:Restriction>
      <owl:onProperty rdf:resource="http://www.sisostds.org/ontologies/C2SIM#hasActionTaskActivity Code"/>
      <owl:qualifiedCardinality rdf:datatype=
"http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:qualifiedCardinality>
      <owl:onDataRange rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
   </owl:Restriction>
</rdfs:subClassOf>
```

- *And*, a subclass of things that have a hasUUID data property with exactly one member of the UUID class):

```
<rdfs:subClassOf>
   <owl:Restriction>
      <owl:onProperty rdf:resource="http://www.sisostds.org/ontologies/C2SIM#hasUUID"/>
      <owl:qualifiedCardinality rdf:datatype=
"http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:qualifiedCardinality>
      <owl:onDataRange rdf:resource="http://www.sisostds.org/ontologies/C2SIM#UUID"/>
   </owl:Restriction>
</rdfs:subClassOf>
```

These restrictions are somewhat easier to see in the declarations made in the Protégé graphical user interface developed by Stanford University as a tool for creating ontologies in OWL and RDF (http://protege.stanford.edu); as shown in Figure 2.

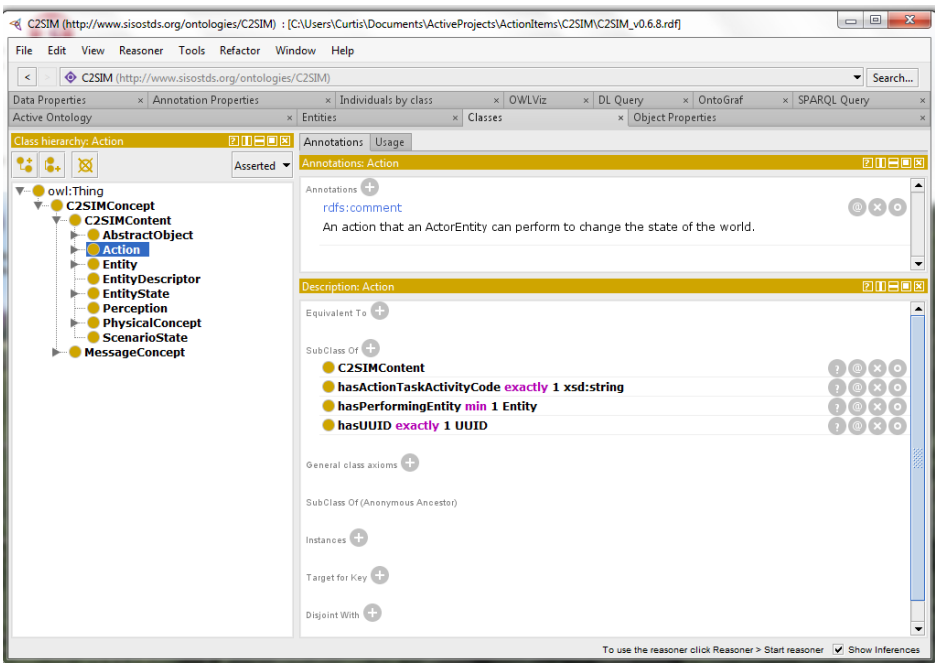

Figure 2: Declaration of the Action class in the Protégé ontology creation and editing tool

```
<owl:DatatypeProperty rdf:about="http://www.sisostds.org/ontologies/C2SIM#hasUUID">
   <rdfs:subPropertyOf rdf:resource="http://www.sisostds.org/ontologies/C2SIM#hasUniqueIdentifier"/>
</owl:DatatypeProperty>
```

Declaration of the data property hasUUID is shown below:

```
<owl:DatatypeProperty rdf:about="http://www.sisostds.org/ontologies/C2SIM#hasUniqueIdentifier"/>
```

Note this property is actually a sub property of hasUniqueIdentifier, which in turn is declared as follows:

As an example of declaration of an object property, consider the following declaration of the hasPerformingEntity object property used in the definition of the Action class:

```
<owl:ObjectProperty rdf:about="http://www.sisostds.org/ontologies/C2SIM#hasPerformingEntity"/>
```

The last two declarations simply declare the property to be of the respective property types. The properties can be defined with a number of other attributes, such as declaring them to be functional (single-valued), inverse functional, symmetric, asymmetric, transitive, reflexive, and irreflexive. Also, domains and ranges can be declared for properties, such that the properties are only defined on certain classes. For example, if the *hasPerformingEntity* object property is declared to have a domain of Entity (members of the Entity class), then if a *hasPerformingEntity* relation exists on some individual, a reasoner could infer that that individual must be a member of the Entity class.

Declarations of conditions in the definition of OWL classes can be far more complex than those we have shown here, but it is beyond the scope of this document to provide a complete overview of the language and the formal logics that can operate on OWL expressions (e.g., description logics). In addition to the W3C standards and Protégé documentation, there are numerous references and resources available to provide a deeper understanding of OWL to address the particular needs of any C2SIM user.

2 References

2.1 Other Documents

Berners-Lee, Tim. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. San Francisco: Harper Collins. 1999.

Bratt, Steve. “Developing the Foundational Standards for Web Services.” Presented to The Gartner Application Integration and Web Services Summit. Los Angeles. April 20, 2005. http://www.w3.org/2005/Talks/0420-sb-gartnerWS/slide1-0.html.

Gruber, Thomas R. “A Translation Approach to Portable Ontology Specifications.” *Knowledge Acquisition* 5(2):199-220. 1993.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F. (Eds.). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press: New York, N.Y. 2003.

2.2 Top Level Architecture

This architecture applies to all parts of C2SIM and shows how C2SIM interoperation has been achieved up to the time this standard was drafted. The C2SIM system-of-systems may be referred to as a "coalition" of systems. Its major components are:

1. C2 System(s) - networked software/hardware systems capable of composing orders, submitting those orders, accepting situational awareness reports, and conveying the contents of those reports externally
2. Server System(s) - networked platforms, multiple of which may cooperate in a distributed configuration, capable of receiving subscriptions from participating C2 and simulation systems, accepting orders and reports from those systems, storing the latest state of simulated objects, forwarding reports to subscribing systems, and responding to queries with latest state of the objects.
3. Simulation System(s) - networked software/hardware systems, capable of using physics and organizational principles to project likely outcome when objects in real or virtual worlds interact in response to direction from orders and emit report messages.
4. Transaction Log - automated, time-stamped record of all reports and orders accepted/transmitted by a server. Available for after-action inspection.
5. Replay - capability associated with a server that can read a transaction log and inject its contents, on a time-synchronized basis, back into a server. The result is expected to be that the participating C2 and simulation systems experience the original message flow associated with the log and therefore can reproduce the original results at their user interfaces.

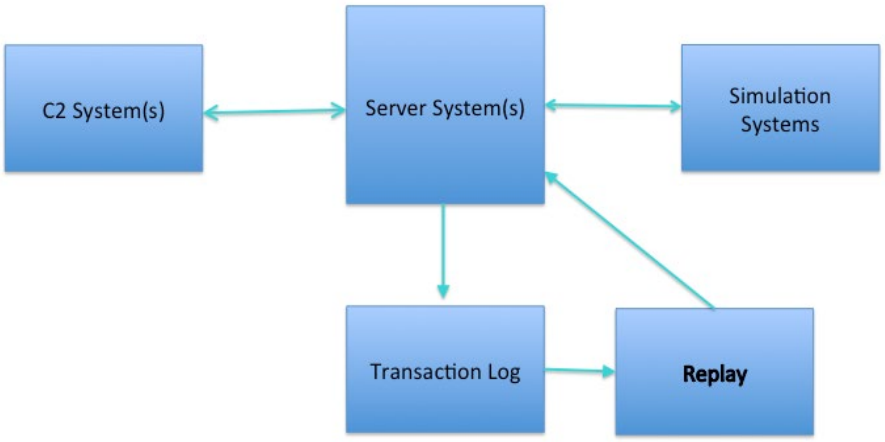

Figure 3: C2SIM Top-Level Architecture

(arrows show information flow as described above)

3 C2SIM Reference Implementation User Guide

3.1 Distributed Integration and Testing in NATO MSG-145

Integration and testing have been a significant challenge in developing C2SIM by multiple NATO national teams since its beginning as Battle Management Language (BML). In Modelling and Simulation Group (MSG)-048, an XML schema developed by one team was exchanged among the participating national teams. This formed the basis for quarterly integration and testing sessions where all teams assembled at a common location. Given the style differences among national teams involved, the integration process was both technical and social in nature and clearly was necessary in order to arrive at an integrated whole. However, it proved very expensive in terms of developer travel and the cost seemed likely to increase as the schemata evolved to become more complex during the standardization process. MSG-085 arrived at a style of Internet-based development and testing, consistent with the fact that the intended product was designed for distributed operation in a networked environment. The new style started as a VPN enclave, where any of the national teams could work with the same server to test and/or demonstrate C2SIM functionality.

3.2 C2SIM Sandbox Concept

In MSG-145, the integration and testing environment has expanded to become a full C2SIM capability, available over the Virtual Private Network (VPN) by remote desktop technology, allowing national teams to test and demonstrate any combination of C2 systems, simulation systems, and servers in the “C2SIM Sandbox.” The capability includes the open source Internet-based audio/video/whiteboard/chat conferencing system Jitsi, which greatly facilitates group communication.

The C2SIM Sandbox provides a continually available environment, available by VPN to national teams to test and demonstrate C2SIM. It supports SISO-STD-C2SIM-1.0.0 Core and SMX, including capabilities for orders, reports, and initializations. In addition to these functions, the C2SIM Sandbox is building experience toward a future “C2SIM as a Service” capability. This began by using the C2SIM Sandbox as the nucleus of a distributed testbed operated for MSG-145. Development and testing for Coalition Warrior Interoperability Exercise (CWIX) 2018 and 2019 deployment took good advantage of these capabilities in earlier forms.

3.3 Component Systems

Figure 4 shows the architecture of the C2SIM Sandbox, including the C2, simulation, server, scheduling, and collaboration/conferencing components. All of these are intended to be accessed remotely, via a Web browser. In addition to interacting with the sandbox components (individually or at multiple remote VPN sites), the systems in the Sandbox are able to interact over the VPN via C2SIM standards with other C2 systems, simulation systems, or server systems.

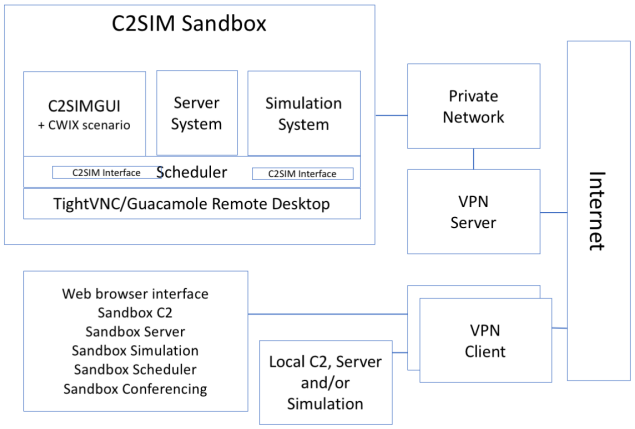

Figure 4: C2SIM Sandbox Architecture

The C2SIM Sandbox currently employs the open source C2SIM Graphical User Interface (C2SIMGUI) system as a surrogate for C2, the VT-MÄK commercial combat simulation VR-Forces, and an open source server, operating in a virtual computing environment. The server is the C2SIM Reference Implementation server developed for the SISO C2SIM standards effort; it features interoperation with Military Scenario Definition Language (MSDL), Coalition Battle Management Language (C-BML) and Integrated Battle Management Language 09 (IBML09) through schema translation. The Sandbox has been tested with external C2IS Norwegian Command and Control Information System (NORCCIS)/ Simulation-supported Wargaming for Analysis of Plans (SWAP) and simulations Joint Semi-Automated Forces (JSAF) and Korpsrahmen Simulationsmodell zur Offizierausbildung (KORA) in CWIX2018. Documentation for all of these systems is either posted on or linked on the webpage https://openc2sim.github.io.

An important capability employed in the Sandbox is virtual/remote desktop via Web browser, which recently has become available commercially in Apache Guacamole. This enables remote participants to work with any application incorporated in the Sandbox, using only the readily available open source OpenVPN client and any HTML5 compliant web browser. The Sandbox environment uses open source remote desktop gateway software and virtual network computing (VNC) to enable remote interaction with virtual machines hosted within a VMWare vSphere hypervisor as well as actual physical machines. The virtualized environment allows for flexibility in the applications and services made available for testing.

C2SIM Sandbox remote application interfaces are intended to be limited to the user Graphical User Interface (GUI) capability of each software system. In this mode the Sandbox is able to block inspection of the C2, simulation or server code so that privacy can be provided for software providers. For Windows systems, this is accomplished using a combination of group policy controls to operate the system in a kiosk mode. Instead of physical interaction, users work remotely through the remote desktop gateway accessed within the Sandbox environment's virtual private network. A similar capability is achieved on Linux systems using a window manager optimized for restricting application access. However, at present there are no software systems in the Sandbox requiring this privacy, so a Windows 10 desktop is displayed, limited to files made accessible to user “c2sim.”

A pre-packaged scenario from CWIX 2018 with instructions for operation is included with the Sandbox, to enable the C2SIM configuration to be exercised with only a minimal understanding of the software. The scenario is user-modifiable within a limited scope via the C2 system GUI, to allow users to run alternatives and observe results.

3.4 Scheduling C2SIM Sandbox Use

The Sandbox has a scheduling calendar. In keeping with the collaborative nature of the Sandbox, the scheduling mechanism does not have an enforcement mechanism; it is simply way for users to show when they want to reserve its use. Users are expected to cooperate by working in the Sandbox only at times they have reserved.

To reserve it is necessary to be connected to the Sandbox VPN (experiment-access) and then direct a Web browser to: http://10.2.10.30/reserve

One can reserve one-hour slots by entering first/last name, email, phone number, and purpose/note. A reservation can be made only from available timeslots. One can also view (read only) the calendar of reservations. The link is on the bottom of the reservation page.

When a reservation is made, your email is sent with a link with the info and a link to cancel the reservation or to delete all of your personal information from the system. The scheduling links only work when connected to the experiment VPN.

3.5 C2SIM Reference Implementation Server

The C2SIM Sandbox server is an open-source Java-based server, developed by the George Mason University (GMU) Command, Control, Communications, Computers, and Intelligence (C4I) and Cyber Center as a reference implementation for the C2SIM Logical Data Model Core plus Standard Military Extension plus its Land Operations extension. This is a further development based the earlier Web-based Inquiry Science Environment- Systems Biology Markup Language (WISE-SBML) translating server. It is capable of translating among XML documents based on different schemata if they are semantically equivalent and designed to be easy to reconfigure for new schemata. This capability provides a means of backward compatibility from C2SIM to first-generation BML standards MSDL and C-BML. Use of Java is intended to make the server more portable and understandable, though at the cost of lower performance than WISE-SBML; it nevertheless has shown transaction rates in excess of 200 reports per second. Like its predecessors, the server supports logging and replay; it also includes support for late joiners and checkpoint/restart.

3.6 C2SIM Sandbox Modes of Use

The C2SIM Sandbox is able to facilitate testing and demonstration in a variety of modes:

- C2SIM demonstrations
  - Schemata: IBML09, C-BML Light, C2SIM Core
  - C2SIM Core and SMX when available
  - CWIX 2019 scenario provided (others will be added if contributed)
- C2SIM testing
  - Test C2 with Sandbox Server and Simulation
  - Test Server with Sandbox C2 and Simulation
  - Test Simulation with Sandbox C2 and Server
  - Test C2-Simulation Coalitions with the Server
  - Distributed configurations of all sorts
- C2SIM validation with SISO
- C2SIM-based exercises (scope limited by server performance)
- In the future: C2SIM as a Service, based on the NATO C3 Taxonomy and Modeling and Simulation as a Service (MSaaS) Reference Architecture. The NATO Modelling & Simulation Centre of Excellence (MSCOE) intends to make the open source C2SIM Reference Implementation Server available in an MSaaS container.

3.7 Getting Started With The C2SIM Sandbox

1. Users need credentials for the Experiment-Access VPN where the Sandbox resides. Contact help@c4i.gmu.edu for this. If you need an assigned Internet Protocol (IP) address in the VPN address space (10.2.10/24), for example for a C2 or simulation system, be sure to state that in your request.
2. Much of the software and documentation for the Sandbox will be found on https://openc2sim.github.io.

3. Minimum software to access the Sandbox is the OpenVPN client and Google Chrome or other Hypertext Markup Language 5 (HTML5) compliant browser. These are available at no cost on the Internet. Install both and use the information that comes with your VPN credentials to configure OpenVPN.
4. Open a VPN connection to Experiment-Access and use Google Chrome to access Uniform Resource Locator (URL) http://10.2.10.32:8080/guacamole/#/ and login to Guacamole with account:c2sim and password:sandbox.
5. Users can communicate with other teams working in the Sandbox VPN, via Jitsi. Use Google Chrome to access https://10.2.10.28/sandbox with an HTML5 compliant browser on your OpenVPN-connected system. Or you can use any other mutually agreeable Internet conferencing system, such as Zoom or WebEx.
6. Start VR-Forces on the Sandbox using the desktop link (a red ball). Chose “launch” on the first GUI and on the second “load recent scenario” or “load scenario from disk,” choosing CWIX. When the main VR-Forces GUI comes up, use the File Menu to load terrain “MAK Earth base (online)” and navigate to Bogaland (AKA west-central Sweden). If you are using a different scenario, delete all units and tactical graphics objects in the Objects List panel at left of the VR-Forces GUI. NOTE: If VR-Forces is already running, you can usually avoid all of the above by simply using the File dropdown to “close scenario” and “load recent scenario.”
7. Start the c2simVRF interface using the desktop link. Information and open source code for this software is available at OpenBML/C2SIM-VRForces-v2.2.pdf. It will connect to VR-Forces, subscribe to the C2SIM Server in the Sandbox, and listen for C2SIM Orders.
8. Start the C2SIMGUI using the desktop link. This will let you push C2SIM Order to the Server and observe reports being made by VR-Forces. See OpenBML/C2SIMGUI_User_Guide_v2.9.1.pdf.
9. Users will need to follow this procedure on the C2SIMGUI to start system operation:
    a) Use the File dropdown on C2SIMGUI to send the server STOP, RESET and INITIALIZE.
    b) Use the File dropdown to Open and Push Initialization (ObjectInitialization message) into the server. Red and Blue test initializations are available to use.
    c) Use the File dropdown to send the server SHARE and START. The objects you initialize will appear on VR-Forces main GUI; if you don’t see them try zooming out. They also appear on the C2SIMGUI.
    d) Now you are ready to Open and Push C2SIM orders. Red and Blue test orders are available, or you can create your own. The C2SIMGUI will let you edit an existing order or create a new one, under the File menu.
    e) You can also enter orders and receive reports on your own C2IS system, if it has a C2SIM Interface. The server is at 10.2.10.30 in the VPN. See the open source for VR-Forces interface on OpenBML for examples of C++ connection, and the open source for C2SIMGUI on OpenBML for examples for Java connection, using the ClientLib software from OpenBML in both cases.
    f) You can also receive orders and generate reports on your own simulation, if it has a C2SIM Interface. The same ClientLib applies there.
    g) The C2SIMGUI will record C2SIM XML messages from the server’s Streaming Text Oriented Message Protocol (STOMP) output and replay those messages as input to the server so as to repeat a period of Sandbox operation. Use top-row buttons “Record STOMP” and “Play Recording.”
    h) You can visually monitor the STOMP traffic sent by the server by clicking on the runSTOMP link on the Sandbox desktop. This shows all order and reports distributed by the server. It does not monitor Representational State Protocol (REST) inputs; however successful XML inputs generally are reflected in the STOMP output.

i) You can check the operational status of the server by accessing http://10.2.10.30:8080/BMLServer/status. (The current message provides somewhat limited status; we welcome suggestions for improving it.)

j) Please send suggestions for improvement and trouble reports to help@c4i.gmu.edu.

10. Files can be saved on the Sandbox Windows system; however, they will be deleted nightly.
11. When finished with a Sandbox session, simply shutdown the browser that was connected to http://10.2.10.32:8080/guacamole/#/.

Important note: None of the above requires sandbox login/logout or shutdown. Do not use the windows shutdown or logout controls. These will disable the sandbox until an operator can reset it. Please close the browser that is connected to Guacomole.

4 Procedure for Merging C2SIM Ontologies in Protégé

The C2SIM standard uses a formal ontology language (the Web Ontology Language, OWL) to specify information needed for interchange across C2 and Simulation systems. Protégé, an open-source ontology editing tool developed and maintained by Stanford University, is the preferred tool for working with the C2SIM ontology. The standard specifies a core ontology and a methodology for creating extensions to the core. For example, in the initial specification of C2SIM, a Standard Military Extension (SMX) ontology and a Land Operations Extension (LOX) ontology are defined to extend the specification into concepts related to those domains. The SMX builds on the Core, while the LOX builds on both the Core and the SMX. In the ontology files, these relationships across ontologies are identified using the OWL *imports* statement. The following statement appears in the SMX ontology file:

```
<owl:imports rdf:resource="http://www.sisostds.org/ontologies/C2SIM"/>
```

Similarly, the following statement appears in the LOX ontology file:

```
<owl:imports rdf:resource="http://www.sisostds.org/ontologies/smx"/>
```

These statements identify the XML namespace associated with each ontology. The Protégé tool maintains a catalog associating namespaces to the physical file locations. In normal use, the rdf:resource attributes will refer to a web-based location for the respective ontology file (or, can identify a locally-stored file using the URI structure file:///drivename:/local_file_path for local development/testing).

The process of transforming the C2SIM into an XML schema document for implementing the information exchange capabilities of C2SIM requires that the core ontology plus any extensions be merged into a single ontology in the Protégé tool, and then saved out into the rdf/xml format for processing by the C2SIM ontology-to-schema Extensible Stylesheet Language Transformations (XSLT) document. The following steps show how to use Protégé to merge multiple C2SIM ontology files into a single ontology file.

1. *Start the Protégé tool*. Double-click on the Protégé icon or executable file, depending on how you installed Protégé on your system.
2. *Open the ontology files in Protégé*: In this example, we use the ontology files for the core, SMX, and LOX as described above. Because the LOX imports the SMX which imports the core, we only need to perform this action on the LOX file. In Figure 5 below, we select a version of the C2SIM_LOX ontology to open in Protégé.

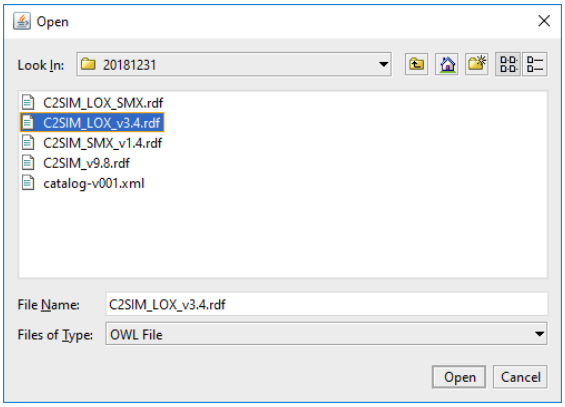

Figure 5: Starting Protégé

After opening the file, Protégé shows summary information about all three ontologies, identifying the imported ontologies in the lower part of the window, as shown in Figure 6 below.

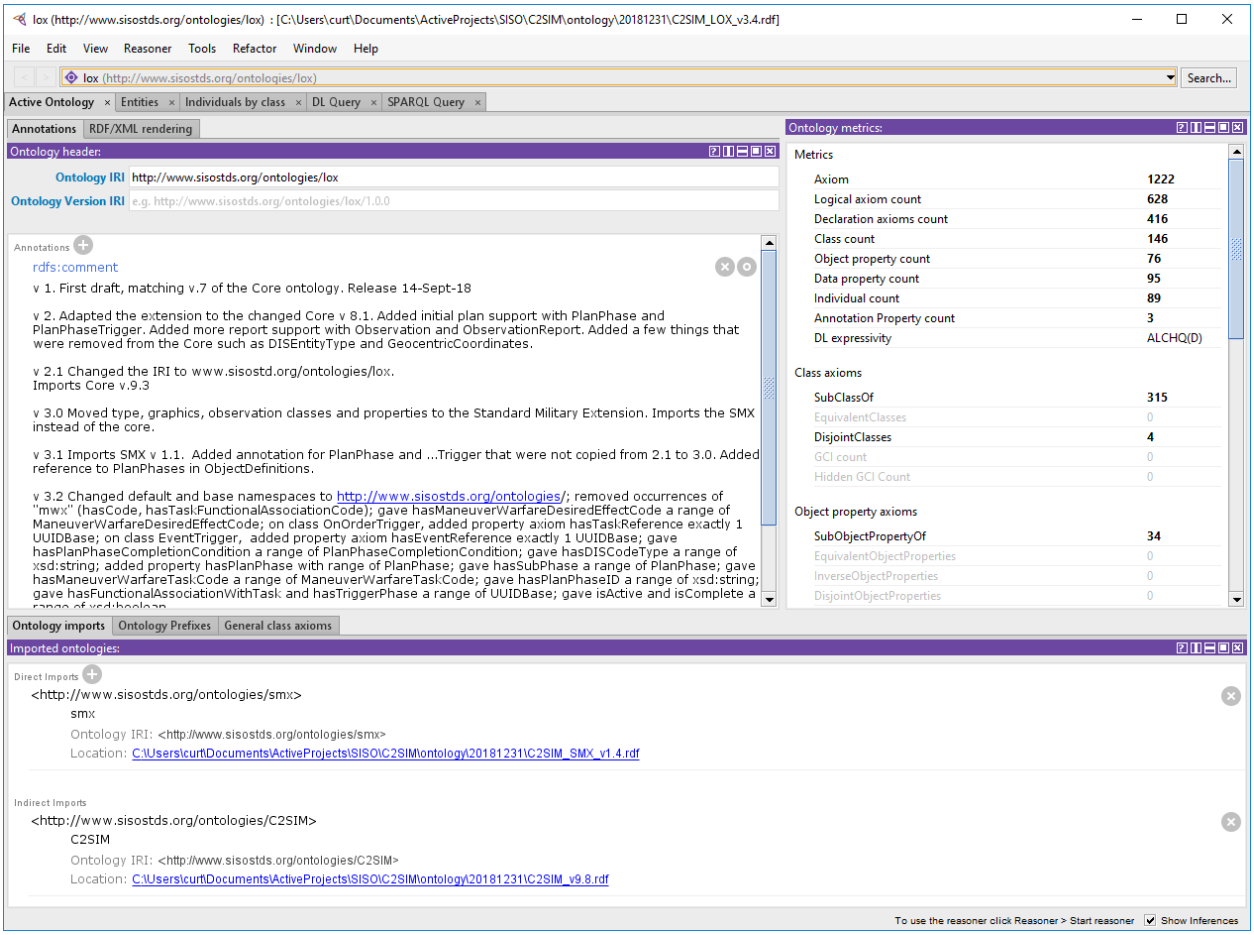

Figure 6: Protégé showing ontologies to merge

3. *Merge the ontologies*: In the Protégé menu bar, select “Refactor / Merge ontologies…,” as shown in Figure 7 below.

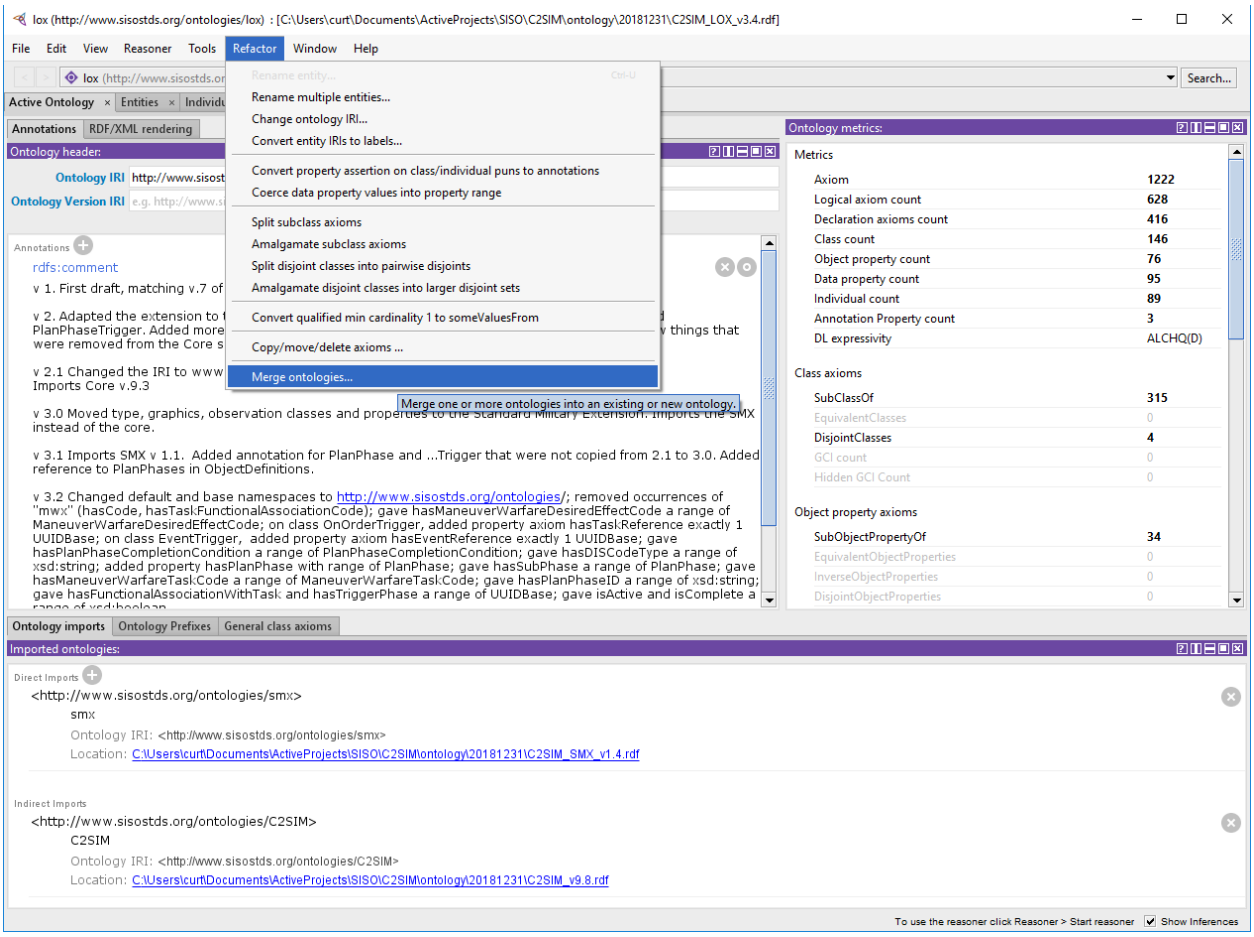

Figure 7: Protégé merging ontologies

This brings up a dialog window to select the ontologies to merge, shown in Figure 8. For this example, highlight all three (e.g., click on C2SIM, move your cursor to SMX, and shift-click to select all three ontologies) and then select “Continue.”

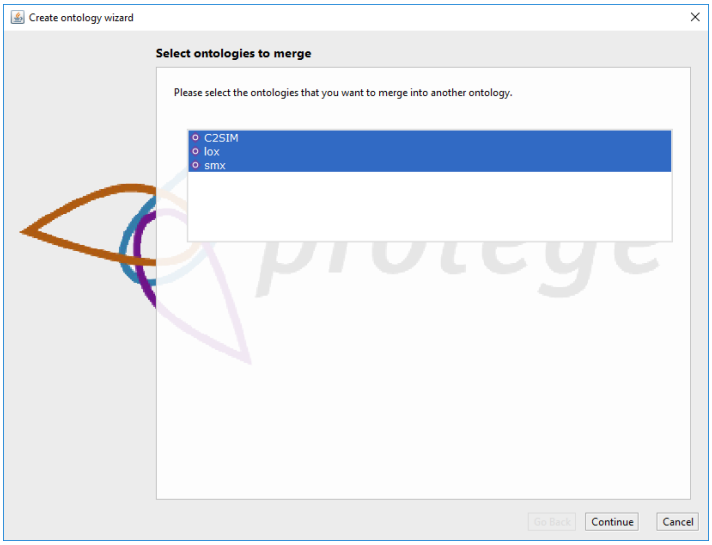

Figure 8: Selecting ontologies in Protégé

Protégé brings up another dialog window, shown in Figure 9, asking if you want to merge the ontologies into an existing ontology or into a new ontology. Select “Merge into new ontology” as shown in Figure 9 below, and then select “Continue.”

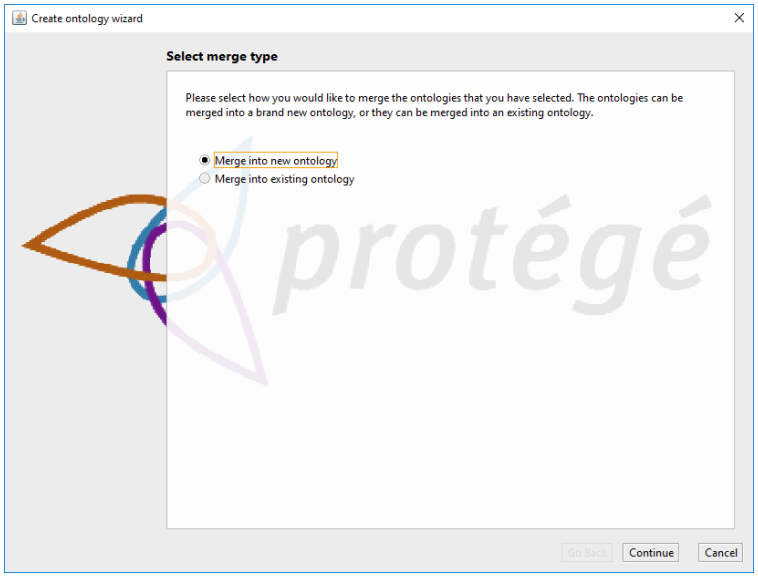

Figure 9: Selecting Merge into new ontology in Protégé

On the next screen, enter the ontology IRI (Internationalized Resource Identifier) you want for the combined ontology. This namespace must be different from the ones used in the source ontology files (but has no bearing on the targetNamespace assigned in the generated XML schema document). For example, Figure 10 below shows the C2SIM namespace “http://www.sisostds.org/ontologies/C2SIM_SMX_LOX” as the IRI for the merged ontology. Then select “Continue.”

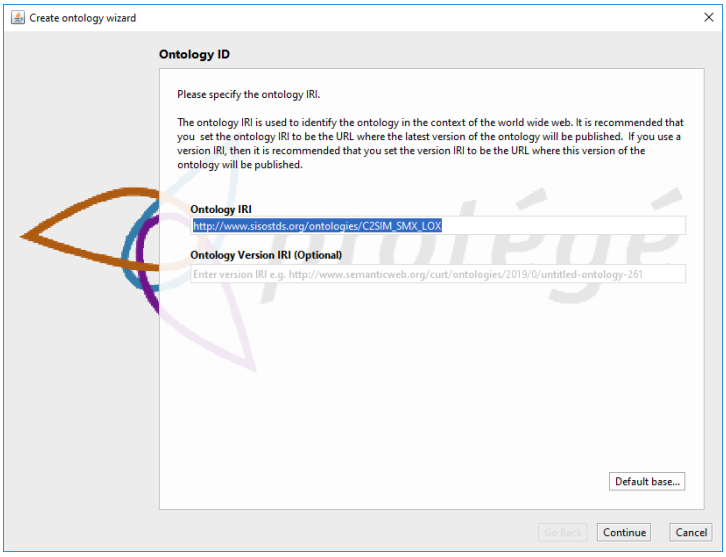

Figure 10: C2SIM namespace in Protégé

On the next screen, enter the path to the folder where you want to store the combined ontology. Enter the file name for the merged ontology as well. For example, Figure 11 below shows a local path to a file named C2SIM_SMX_LOX.rdf. Then select “Continue.”

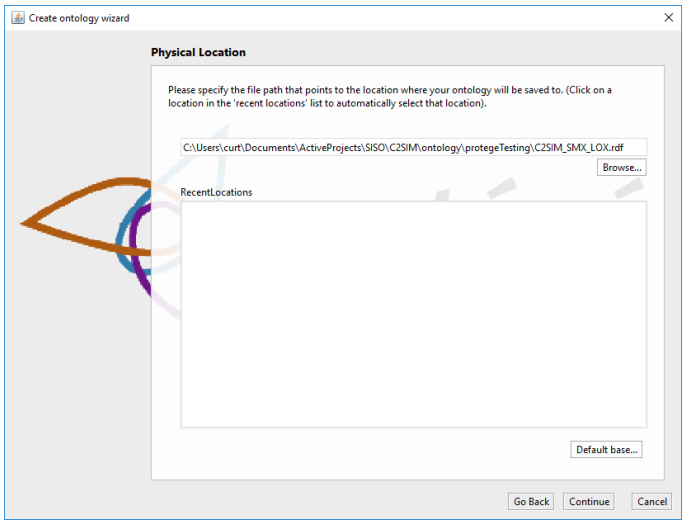

Figure 11: Entering target path in Protégé

Finally, select the format for the merged ontology. The preferred format (indeed, the *required* format for the XSLT to generate an XML schema document from the merged ontology) is RDF/XML syntax, as shown in Figure 12 below. Then select "Finish."

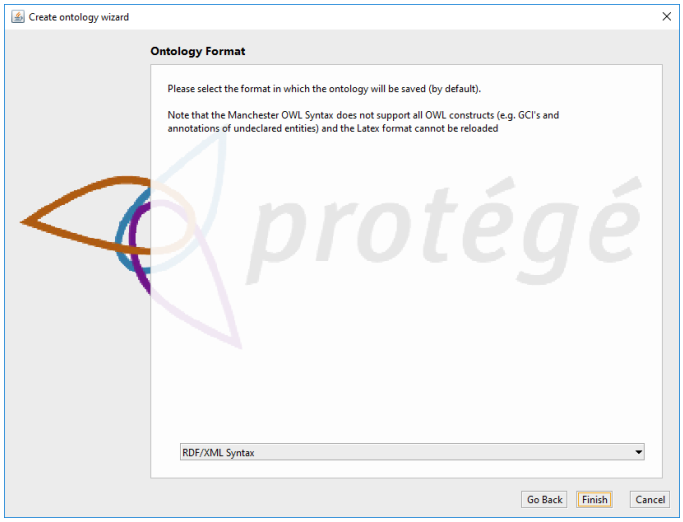

Figure 12: Selecting RDF/XML output in Protégé

4. *Save the merged ontology with a specific name in the location desired*: While this may seem like an unnecessary action based on the prior steps, it is useful to save the merged ontology explicitly to a specific named file in a desired location. This step uses the normal "File / Save as…" selection from the Protégé menu bar as shown in Figure 13 below.

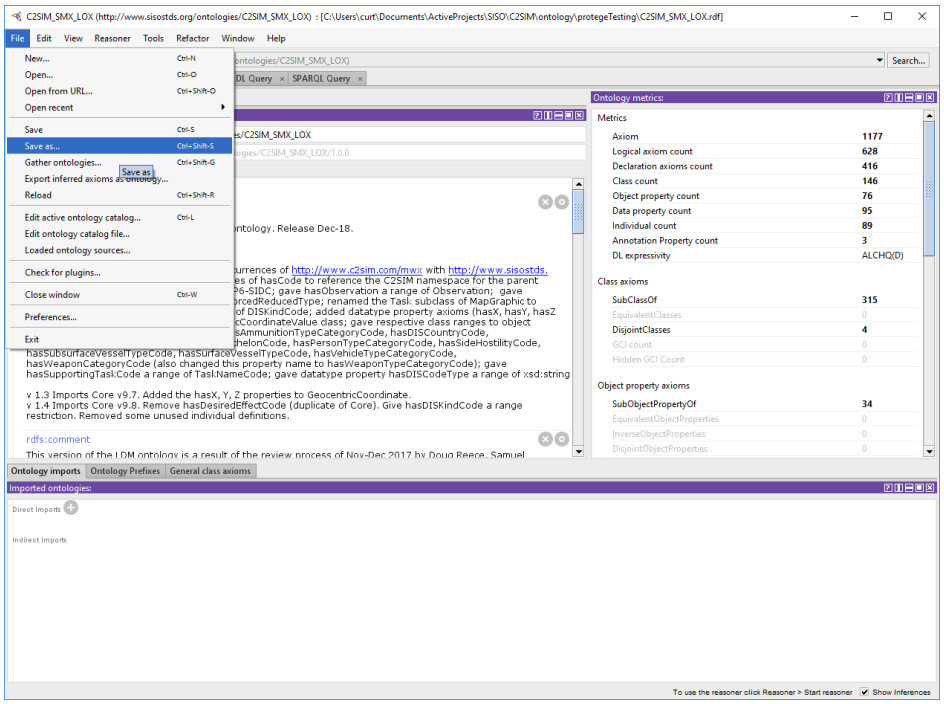

Figure 13: Saving the merged ontology in Protégé

This again brings up the dialog to select the desired output format. Select “RDF/XML Syntax” and select “OK” as shown in Figure 14.

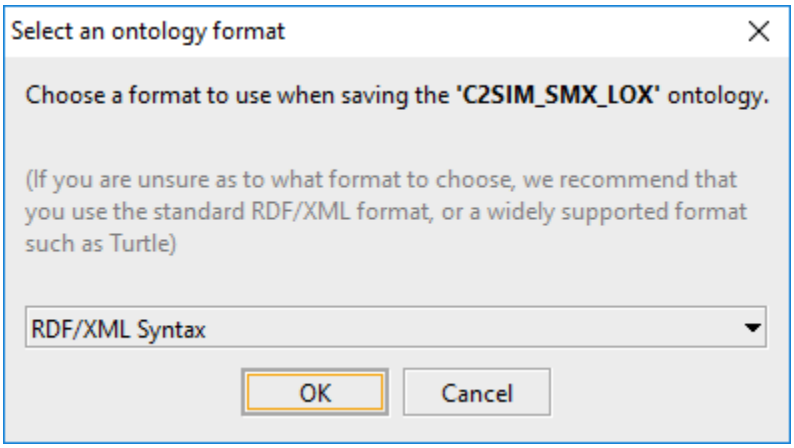

Figure 14: Selecting output format in Protégé

Browse to the desired location and provide a name for the merged ontology. Also, select “All Files” (in the “Files of Type” drop-down menu) and use the .rdf extension in the file name when saving in the RDF/XML syntax as selected above. Select “Save” to complete this procedure.

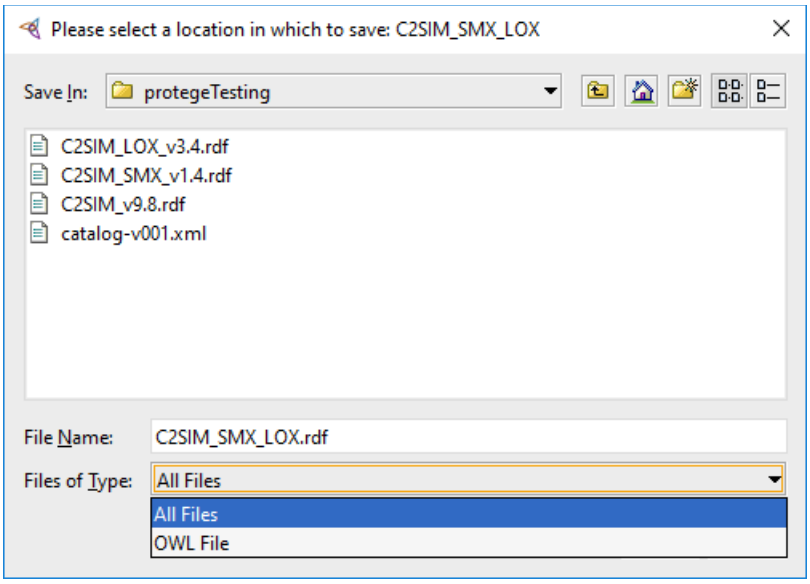

Figure 15: Providing an output file name in Protégé

Your named file now contains the assertions from all the selected ontologies and can be used in the transformation process to generate an XML schema file for use in C2SIM implementations.

5 Procedure to extract class descriptions from C2SIM ontology

Open the C2SIM ontology in Protégé and open the SPARQL query tab (shown in Figure 16 below):

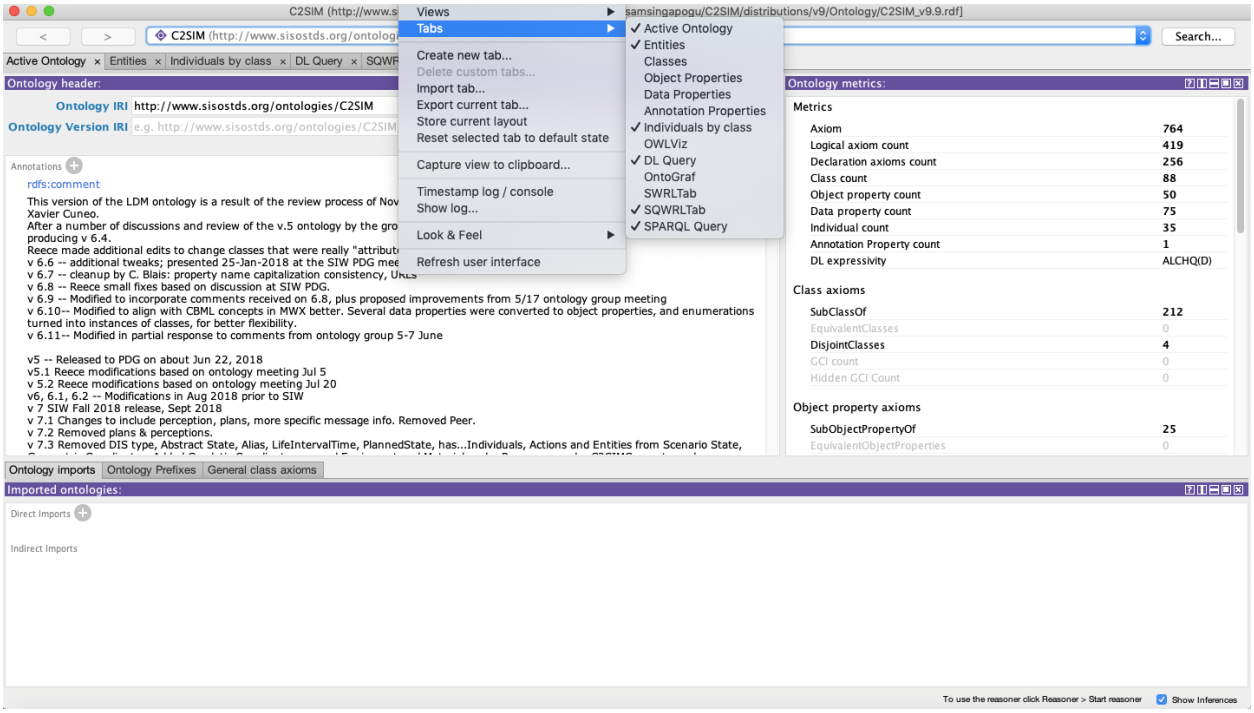

Figure 16: SPARQL query tab in Protégé

Run the following SPARQL query in the SPARQL tab and click on the “Execute” button which is at the bottom of the SPARQL tab (screenshot below)

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
Select ?className ?classLabel
{
  ?className rdf:type owl:Class.
  OPTIONAL {?className rdfs:label ?classLabel.}
}
```

Figure 17 shows the result:

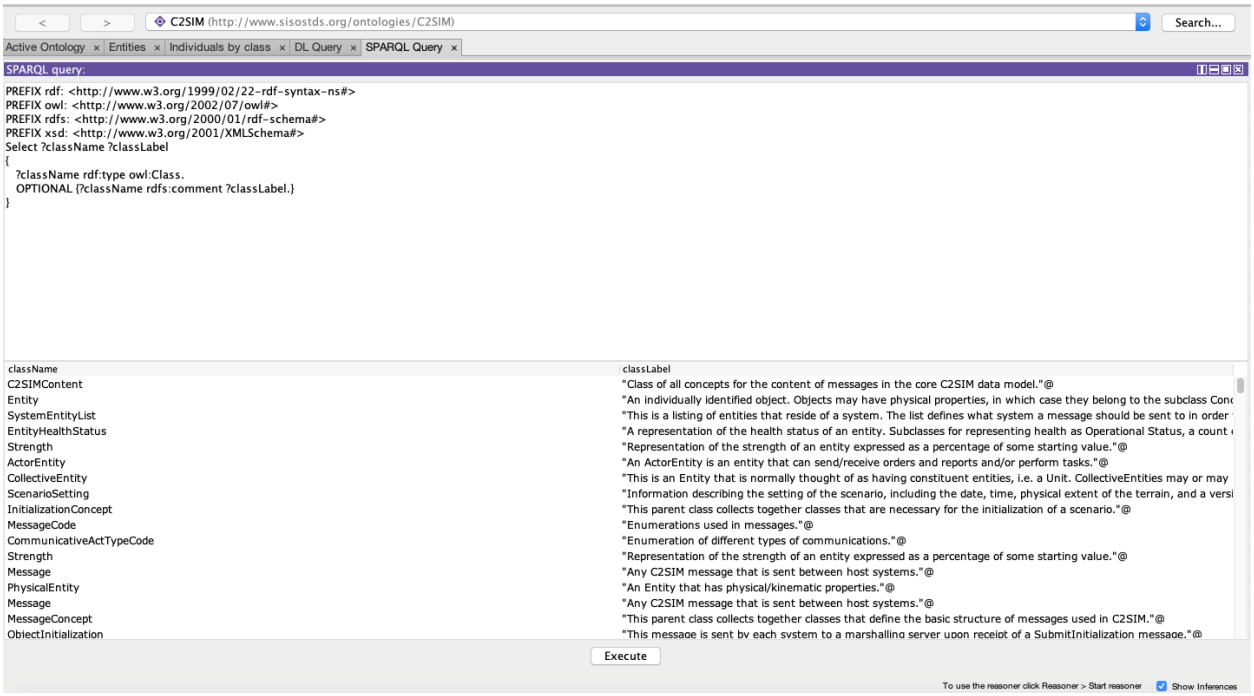

Figure 17: Extracting class description in Protégé

6 Message Processing

This processing approach is encouraged for use by all C2SIM actors. Note that it imposes no ordering on the messages so that initialization messages could be received at any point in the operation. In cases where a client is unable to handle particular messages in arbitrary order, the client is free to reject messages it is unable to process.

Actors are encouraged to implement the Acknowledgement processes shown in Figure 18, using the AcknowledgeTypeCode and TaskStatusCode provided in the C2SIM Core Ontology.

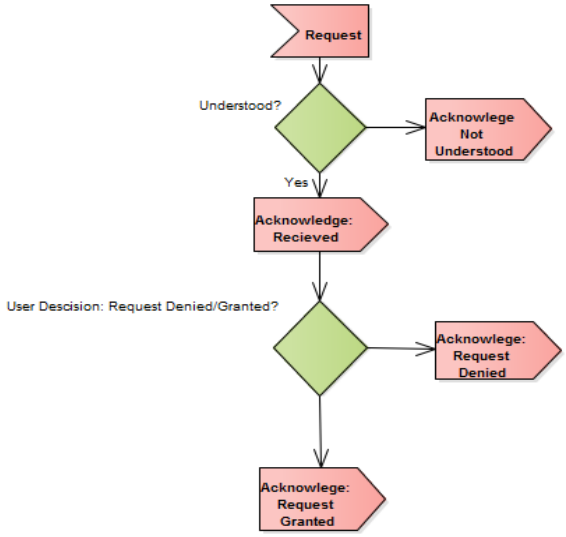

Figure 18: Message flow that is triggered by order or request

When an actor receives a report, it should react according to the flow diagram shown in Figure 19.

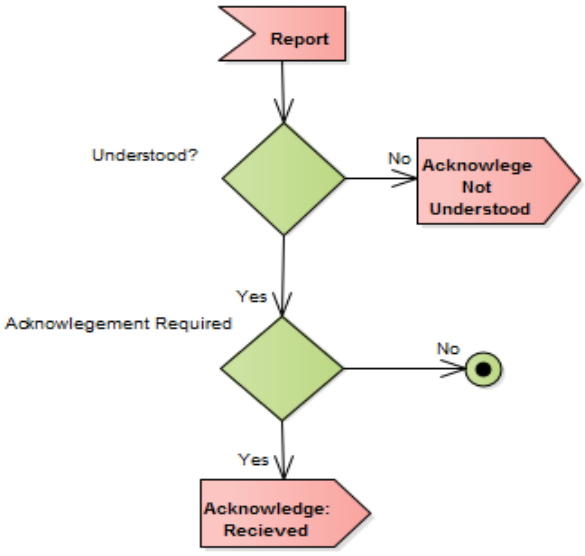

Figure 19: Message flow that is triggered by a report

7 Extending the C2SIM Ontology

The C2SIM standard describes the Core ontology and a Standard Military Extension (SMX). The SMX *extends* the core ontology by adding new subclasses to the existing class hierarchy, adding new data types, data properties, and object properties, and creating new individuals, as needed to describe standard military concepts not found in the Core ontology. The new individuals act as additional "values" for the Core classes, much as enumerations delineate specific values for simple string types in an XML schema document. Consider the Core ontology to be the foundational concepts to which are added standard military concepts (the SMX) to create a combined ontology for use in expressing information interchange in that domain. On the one hand, we can think of SMX as an extension to the Core, since it contains new concepts, properties, and individuals (collectively referred to as "ontology entities") not found in the Core. On the other hand, because the SMX imports the Core, we can also think of the SMX as an ontology that combines the Core ontology entities and the standard military ontology entities.

A companion standard describes the Land Operations Extension (LOX) that extends the combined Core and SMX ontologies by taking the same actions in adding ontology entities, but describing concepts and values pertaining to land operations. In this case, the SMX serves as the foundational ontology for the LOX. However, recognize again that the SMX ontology file contains an owl:imports declaration to bring in the Core ontology as well. The LOX is an extension of the combined concepts contained in the Core ontology and the SMX ontology.

The extension is accomplished by providing an owl:imports statement in the new ontology file to load in the foundational ontology. Section 4 in this Guide described the process of creating a merged ontology from multiple ontologies. While in some cases it may be possible to create a "chain" of imported ontologies, as done in the case of the LOX, where the LOX ontology file imports the SMX ontology which imports the Core ontology, this might not always be possible. When the chain of imports is not feasible for a new extension, a better approach would be to create an intermediate merged ontology consisting of the concepts, and then importing that merged ontology as the foundation for the new extension.

In each case, we assign a new namespace to the concepts being introduced in an extension. The Core ontology namespace is "https://www.sisostds.org/ontologies/c2sim." The SMX ontology introduces a new namespace, "https://www.sisostds.org/ontologies/smx." The LOX ontology introduces its own namespace, "https://www.sisostds.org/ontologies/lox." By convention, when merging ontologies, we introduce a namespace reflecting the combination of the component ontologies, such as in the example earlier in this Guide where we assigned the merged ontology the namespace "http://www.sisostds.org/ontologies/C2SIM_SMX_LOX." Note, though, that the ontology entities in the merged ontology retain their original namespace prefixes so it is always possible to determine the source ontology for all of the ontology entities in the merged ontology.

This is the essence of the process for extending the C2SIM standard:

1. Create a new ontology file (e.g., using Protégé or some other tool).
2. Declare a new namespace for this extension ontology.
3. Import one or more C2SIM ontology files that the extension is building upon.
4. Declare new ontology entities (subclasses, properties, datatypes, individuals).
5. Modify declarations of any existing ontology entities (e.g., adding a property axiom to a subclass).

If employing a tool or software that has a description logics reasoner, use the reasoner liberally throughout the development process. This will help make you aware of implications of your changes with respect to the class hierarchy (such as a new class actually, and perhaps unexpectedly, being a subclass of an existing class) and possible introduction of logical inconsistencies (such as a defined class not being able to have any members according to the axioms declared and inferred in the ontology). As a best practice, whenever possible communicate extensions to the C2SIM community so there can be technical interchange on ways to improve the extensions or to deal with implications to the other parts of the C2SIM standard.

8 Employing a Reasoner in Developing Extensions to C2SIM

The C2SIM standard uses OWL as its specification language in order to obtain the benefits of a logical formulation. OWL is based on the Description Logic (DL), a branch of first-order logic with desirable computational characteristics. For a definitive treatment of the topic, see (Baader, et al. 2003).

Reasoning under DL can accomplish a number of objectives, for example:

- Based on the class definition axioms, determine if a class is a subclass or superclass of another class (i.e., class subsumption).
- Based on the class definition axioms and the assertion of certain properties on an individual, determine if the individual is a member of that class.
- Based on the class definition axioms, determine if any class cannot have any members (i.e., the class is empty), considered to be a logical inconsistency in the ontology.

As the ontology becomes larger and more complex, it is difficult, if not impossible, for humans to determine if the combination of axioms are logically consistent or if we have introduced an impossibility, such as a class that cannot have any members. At this point in the standardization of C2SIM, the ontology declares "necessary" conditions for something to be considered a member of a class. With necessary conditions alone, we cannot say that "if something fulfills these conditions then it MUST be a member of this class." That is, determining that any random individual that satisfies the conditions to be in the class MUST be a member of that class. The reasoner can only automatically classify an individual (determine class membership) under what are sometime called *defined* classes; that is, classes with at least one set of necessary AND sufficient conditions. As the C2SIM ontology matures and use of the representation grows, the community will move toward this higher level of formalization and precision in order to take advantage of the benefits of such reasoning.

The reasoner is also able to compute automatically the class hierarchy. Again, this is a major benefit when constructing very large ontologies where the ability to compute subclass-superclass relationships become important. Otherwise, it is difficult for developers to keep the ontology in a maintainable and logically correct state while minimizing human error. A best practice in this case is to use one superclass in manually constructing the hierarchy and exploit the power of the reasoner to compute and maintain multiple inheritance and class subsumption.

For these reasons, it is best practice for developers to use a reasoner as they develop changes to the standard ontology or extensions to the standard. As changes are made, the developer should execute the reasoner to determine if any issues have arisen in accordance with the discussion above. Eventually, when developers begin to use OWL directly in their software systems to represent C2SIM information and messages as individuals in the ontology, the reasoner can determine if the individuals satisfy all axioms to be identified as members of certain classes. For example, if a developer creates an individual representing a C2SIM message but does not include all of the information required to make it a valid message, the reasoner will not classify that individual as a member of the desired message subclass, making it apparent to the developer that some information is missing or invalid in accordance with the definition of that subclass. In other words, this is a message validation test: the message semantics are correct if the reasoner can classify the message as being in the intended subclass. Here is where much greater capabilities become possible, such as various reasoning and querying on the content of the scenario and on the information interchange during and after execution of the C2SIM coalition. This is an unexplored area for use of the OWL representation of C2SIM that creates opportunities for future development.

The Protégé ontology development tool has a number of built-in reasoners, such as HermiT and Pellet. These are found in the top-level menu item "Reasoner," as shown in Figure 20 below.

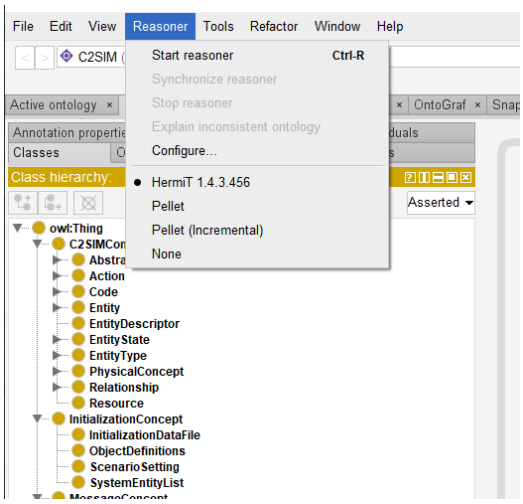

Figure 20: Top-level menu item "Reasoner"

In this case, the HermiT reasoner is selected. By choosing the "Start reasoner" menu item, that reasoner will process the asserted axioms in this ontology (in this case, the core C2SIM ontology) and will identify any inferred axioms (logical consequences of the asserted axioms in the ontology) that could indicate a problem in the axiom or could be information that should be added to the ontology. As another example, Protégé provides a plug-in called DLQuery that can be used to explore the structure of the ontology and its inferences for better understanding. The following screenshot shows provides a simple example of the use of DL Query.

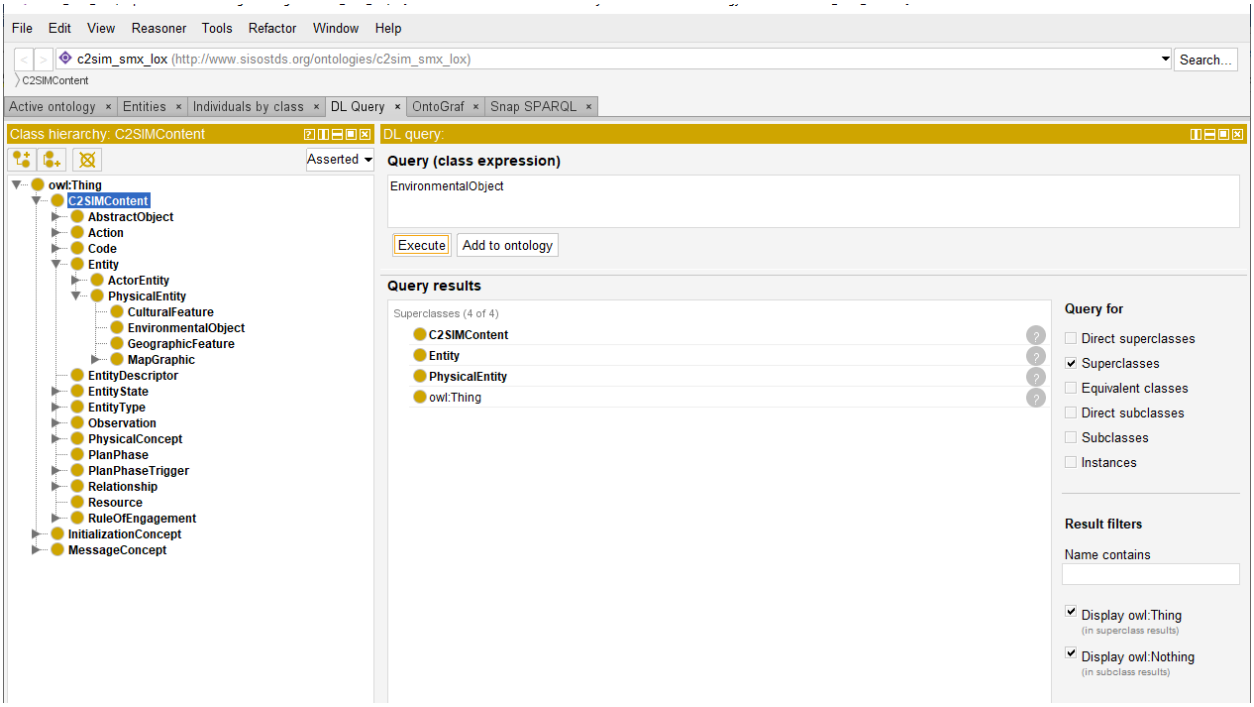

Figure 21: Use of DLQuery

Here, a class expression, simply the name of one of the subclasses in the merged Core+SMX+LOX ontology, has been entered into the pane titled “Query (class expression).” Note in the right-hand side column, the “Query for” selection is set to “Superclasses.” The query, then, is asking, “What are all the superclasses of the subclass EnvironmentalObject?” The “Query results” pane identifies all of the classes that are in the superclass chain for subclass EnvironmentalObject (of course, the owl:Thing class is a superclass of all classes in the ontology). This result had to be *inferred* by the reasoner – for example, there is no explicit assertion in the ontology that EnvironmentalObject is a subclass of the Entity class. The only explicit assertion in the ontology relating to the superclass-subclass relationships for EnvironmentalObject is the assertion that EnvironmentalObject is a subclass of PhysicalEntity. If we click on the “?” to the right side of the list of query results, Protégé reports the assertions that were used to derive the inference that Entity is a superclass of EnvironmentalObject, as shown in the following screenshot.

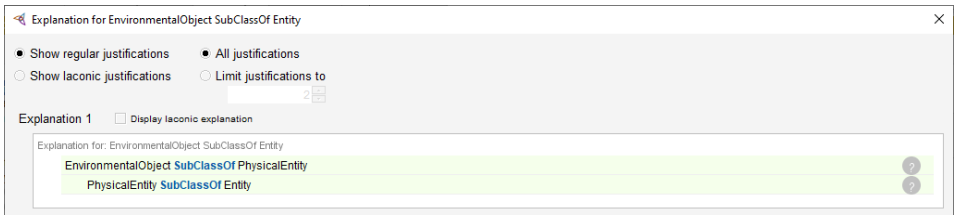

Figure 22: Protégé response

As expected, the inference was computed because EnvironmentalObject is declared as a subclass of PhysicalEntity, which is declared as a subclass of Entity. In this simple example the query results are rather obvious to a human examining the class hierarchy on the left side of the Protégé GUI, but that is because in this example the full hierarchy is readily viewable. If the ontology was much larger, with a much deeper class substructure, this may not be the case. A tool like this helps the ontology developer better understand the structure of the ontology and could make inferences that were not obvious to the human.