USING SEMANTIC TECHNOLOGY TO VALIDATE SMART GRID MODELS

STEVE RAY (STEVE.RAY@SV.CMU.EDU)
Carnegie Mellon University – Silicon Valley

RALPH HODGSON (RHODGSON@TOPQUADRANT.COM)
GOKHAN SOYDAN (GSOYDAN@TOPQUADRANT.COM)
TOPQUADRANT
Big Challenge: How to reconcile vocabularies, concepts and relations among all the smart grid standards?

• Currently 238* standards under review for the SGIP Catalog of Standards
  – Overlapping, different, sometimes contradictory vocabularies and definitions

• We need a way to manage
  – Differences
  – Constraints on usage
  – Relationships between vocabularies

* 82 from NIST Framework 1.0, 27 from PAPs, 129 from other sources
Our Approach

- Using the ASHRAE / NEMA Facility Smart Grid Information Model (FSGIM)
  - Validating the concepts within the model
  - Uncovering inconsistencies, misalignments
  - Think of this demonstration as a “precompiler run” of the model, like LINT
First Step: Converting the UML Model to OWL

- Fully automatic process
  - Export XMI from Enterprise Architect
  - Interpretation into OWL using (SPARQL rules)
Structure of the SPC201P FSGIM

Project Browser

- Model
  - Model Components
    - Energy Manager Component
    - Experimental Load Component
    - Generator Component
    - Load Component
    - Meter Component
    - Other
  - Example Use Cases (Informative)
  - Model Elements from External Sources
    - EI_Classes
    - EMIX-PR03
    - WSCPIM_D3_20110318
    - IEC61850
    - IEC_CIM_NAESP_EUI_MODEL_20101111
  - Supplemental Information
Import Process

- Imported the 4 core FSGIM components
  - Energy Manager
  - Generator
  - Load
  - Meter
- ...plus the Interface Information Elements
- ...then imported all referenced external classes from other standards
  - EI Classes, EMIX, WSCPIIM, WXXM, IEC 61850, NAESB EUI
UML to OWL SPARQLMotion script

Converts XMI to RDF triples. SXML mapping reads the XML tree and generates instances against OWL class definitions for UML constructs.

Converts UML-based RDF triples to OWL classes, properties and instances.

Iterates over packages selected by the user and generates an ontology for each: Copies relevant triples from the conversion; constructs import triples to identify dependent ontologies that were generated from other packages or coming from external sources.
UML to OWL SPIN rule for UML aggregation

# STEP UML-SR-802 Construct an object property for an aggregation

CONSTRUCT {
  ?propertyURI a owl:ObjectProperty .
  ?propertyURI ocmof:hasCMOFbasis ?this .
  _:b0 a owl:Restriction .
  _:b0 owl:onProperty ?propertyURI .
  _:b0 ocmof:hasCMOFbasis ?this .
  _:b0 owl:allValuesFrom ?rangeClass .
  ?domainClass rdfs:subClassOf _:b0 .
}

WHERE {
  ?this ocmof:name ?name .
  ?this ocmof:aggregation ?aggregation .
  FILTER (?aggregation != "none") .
  ?this ocmof:refersTo ?rangeElement .
  ?rangeClass ocmof:hasCMOFbasis ?rangeElement .
  ?this ocmof:hasAssociation ?association .
  ?association ocmof:hasMemberEnd ?memberEnd .
  ?ownedAttribute a ocmof:OwnedAttribute .
  ?typeElement a ocmof:Type .
  ?domainClassElement xmi:id ?idref .
  ?domainClass ocmof:hasCMOFbasis ?domainClassElement .
  BIND (fn:concat("has", o2o:normalizeCamelCaseString(?name), "Ref") AS ?propertyName) .
  BIND (xmi.common:makeUML-URI-Thru-PackagedElement-Children(?this, ?propertyName) AS ?propertyURI) .
}
OK, Now what?

- OWL representation supports:
  - Arbitrary SPARQL queries for interrogation, concept creation, and automated reasoning
Modifying External Standards

Checking for external classes pointing to FSGIM classes

Updates to the external standards will cause these pointers to disappear. Using them will cause failures.
Declaring Primitive Datatypes

Finding all declared primitive datatypes in the model that inherit from definitions from OMG or W3C (i.e. UML or xsd definitions)

These look good
Declaring Primitive Datatypes (continued)

Finding all declared primitive datatypes in the model that **DO NOT** inherit from definitions in OMG or W3C (i.e. UML or xsd definitions).

<table>
<thead>
<tr>
<th>[cls]</th>
<th>role</th>
<th>range</th>
<th>rootDatatypeOfRange</th>
</tr>
</thead>
<tbody>
<tr>
<td>ei_classes:EiEvent</td>
<td>ei_classes:hasResponseValue</td>
<td>ea_java_types_package:emixRealPowerType</td>
<td>ea_java_types_package:emixRealPowerType</td>
</tr>
<tr>
<td>ei_classes:EiEvent</td>
<td>ei_classes:hasModificationDateTime</td>
<td>ea_java_types_package:TimeStampType</td>
<td>ea_java_types_package:TimeStampType</td>
</tr>
<tr>
<td>ei_classes:EiEvent</td>
<td>ei_classes:hasLocation</td>
<td>ea_java_types_package:emixEmixInterfaceType</td>
<td>ea_java_types_package:emixEmixInterfaceType</td>
</tr>
<tr>
<td>ei_classes:EiEvent</td>
<td>ei_classes:hasEnergyBaselineValue</td>
<td>ea_java_types_package:emixRealPowerType</td>
<td>ea_java_types_package:emixRealPowerType</td>
</tr>
<tr>
<td>ei_classes:EiEvent</td>
<td>ei_classes:hasOperatingDay</td>
<td>ea_java_types_package:TimeStampType</td>
<td>ea_java_types_package:TimeStampType</td>
</tr>
<tr>
<td>ei_classes:EiEvent</td>
<td>ei_classes:hasModificationReason</td>
<td>ea_java_types_package:string</td>
<td>ea_java_types_package:string</td>
</tr>
<tr>
<td>ei_classes:EiEvent</td>
<td>ei_classes:hasVtnComment</td>
<td>ea_java_types_package:string</td>
<td>ea_java_types_package:string</td>
</tr>
</tbody>
</table>

Are these really primitive datatypes?

Why a local definition of “string”?
Locally defined datatypes involving "time"

<table>
<thead>
<tr>
<th>cls</th>
<th>role</th>
<th>range</th>
<th>rootDataTypeOfRange</th>
</tr>
</thead>
<tbody>
<tr>
<td>ei_classes:EEvent</td>
<td>ei_classes has ModificationDateTime</td>
<td>ea_java_types_package:TimeStampType</td>
<td>ea_java_types_package:TimeStampType</td>
</tr>
<tr>
<td>ei_classes:EEvent</td>
<td>ei_classes has Energy Baseline Timestamp</td>
<td>ea_java_types_package:TimeStampType</td>
<td>ea_java_types_package:TimeStampType</td>
</tr>
<tr>
<td>ei_classes:EProgram</td>
<td>ei_classes has Created DateTime</td>
<td>ea_java_types_package:TimeStampType</td>
<td>ea_java_types_package:TimeStampType</td>
</tr>
<tr>
<td>energy_manager_component:DemandThreshold</td>
<td>energy_manager_component has Peak Timestamp</td>
<td>ea_java_types_package:wscalTimeStampType</td>
<td>ea_java_types_package:wscalTimeStampType</td>
</tr>
<tr>
<td>load_component:CurtailableLoad</td>
<td>load_component has Minimum Curtail Time</td>
<td>ea_java_types_package:Duration</td>
<td>ea_java_types_package:Duration</td>
</tr>
<tr>
<td>load_component:CurtailableLoad</td>
<td>load_component has Maximum Curtail Time</td>
<td>ea_java_types_package:Duration</td>
<td>ea_java_types_package:Duration</td>
</tr>
<tr>
<td>load_component:CurtailableLoad</td>
<td>load_component has Minimum Release Time</td>
<td>ea_java_types_package:Duration</td>
<td>ea_java_types_package:Duration</td>
</tr>
<tr>
<td>load_component:CurtailableLoad</td>
<td>load_component has Curtailment State Time Remaining</td>
<td>ea_java_types_package:Integer</td>
<td>ea_java_types_package:Integer</td>
</tr>
<tr>
<td>p01_weather:EIS_Weather</td>
<td>p01_weather has Sunrise Time</td>
<td>ea_java_types_package:Time</td>
<td>ea_java_types_package:Time</td>
</tr>
<tr>
<td>p01_weather:EIS_Weather</td>
<td>p01_weather has Start Date Time</td>
<td>ea_java_types_package:AbsoluteDateTime</td>
<td>ea_java_types_package:AbsoluteDateTime</td>
</tr>
<tr>
<td>p01_weather:EIS_Weather</td>
<td>p01_weather has End Date Time</td>
<td>ea_java_types_package:AbsoluteDateTime</td>
<td>ea_java_types_package:AbsoluteDateTime</td>
</tr>
<tr>
<td>p01_weather:EIS_Weather</td>
<td>p01_weather has Sunset Time</td>
<td>ea_java_types_package:Time</td>
<td>ea_java_types_package:Time</td>
</tr>
<tr>
<td>p61850-cdc:SCA</td>
<td>p61850-cdc has Time</td>
<td>ea_none_types_package:dateTime</td>
<td>ea_none_types_package:dateTime</td>
</tr>
</tbody>
</table>

Opportunity for consolidation of datatypes
Different definitions of “load”?

- Energy Manager component invokes Emix datatype
- Load component uses stand-alone definition
Many more such queries are possible

Demonstration