Thoughts on an Ontology for Fault Management

Steven Jenkins
Systems and Software Division
Jet Propulsion Laboratory
California Institute of Technology
• We’d like to use the formalism of ontologies to represent knowledge in fields of interest to us:
  – Fault management in particular
  – Systems engineering in general
• We want these knowledge representation conventions to be stable and durable: independent of particular programs, projects, organizations, and software tools
• We want to customize or adapt our modeling and analysis tools to support our knowledge representation conventions
  – At least to translate to/from internal representations
  – Even better, to teach the tools to operate on our concepts and properties as extensions or specializations of their native counterparts
An ontology is more than a vocabulary or taxonomy.
In practical terms, it’s a grammar for a particular domain of discourse
- It sets rules for well-formed sentences

Some (simplified) well-formed sentences in Systems Engineering (about the MSL Rover Curiosity):
- Curiosity has type Component.
- Curiosity has mass 850 kg.
- Curiosity contains Science Payload.
- Rover Work Package supplies Curiosity.
- MSL Project authorizes Curiosity.

Some not-well-formed sentences:
- Curiosity supplies 850 kg.
- Rover Work Package contains Curiosity.
- Curiosity authorizes Science Payload.

An ontology is an agreement on usage, not a dictionary.
Why Ontologies? Why OWL?

- Some agreement on usage is necessary for effective information interchange
- Formal ontology standards, and OWL\(^1\) in particular, have large communities of practice with tools and training
- OWL includes serialization standards; defining an OWL ontology necessarily defines standard (XML-based) knowledge interchange formats
- OWL reasoners can find errors in ontology rules and facts
- OWL reasoners can draw inferences (entailments) from rules and facts
- OWL supports powerful query languages\(^2\) for application-specific reasoning, transformation, and reporting
- OWL/RDF\(^3\) databases\(^4\) have excellent scaling properties

\(^1\) OWL
\(^2\) for application-specific reasoning, transformation, and reporting
\(^3\) OWL/RDF
\(^4\) have excellent scaling properties
Some Simple Reasoning Examples

<table>
<thead>
<tr>
<th>Type</th>
<th>Given this input</th>
<th>A reasoner concludes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consistency</td>
<td>“has mass” is a functional property. Curiosity has mass 850 kg. Curiosity has mass 900 kg.</td>
<td>Inconsistent: at least two facts are mutually contradictory.</td>
</tr>
<tr>
<td>Satisfiability</td>
<td>Work Package and Organization are disjoint concepts. Every Project is both a Work Package and an Organization.</td>
<td>Unsatisfiable: no Project can exists that satisfies all rules.</td>
</tr>
<tr>
<td>Rules Entailment</td>
<td>Every Spacecraft is a Component. Every Orbiter is a Spacecraft.</td>
<td>Every Orbiter is a Component.</td>
</tr>
<tr>
<td>Facts Entailment</td>
<td>Every Spacecraft is a Component. MSL Rover (an individual, not a class) is a Spacecraft.</td>
<td>MSL Rover is a Component.</td>
</tr>
</tbody>
</table>

These examples are given in “equivalent” natural language, not OWL. The purpose is to show the kinds of problems for which reasoning is useful, not to demonstrate the mechanics.
This example illustrates an actual ontology hygiene check we apply.

Literally it says “If \( p_1 \) and \( p_2 \) are distinct properties such that \( p_1 \) is a subproperty of \( p_2 \), and \( p_1^{-1} \) and \( p_2^{-1} \) exist, then report whether \( p_1^{-1} \) is a subproperty of \( p_2^{-1} \)”

```sparql
SELECT DISTINCT ?p1 ?p2 ?inverse_ok
WHERE {
  { ?p1 owl:inverseOf ?p1_inverse } union { ?p1_inverse owl:inverseOf ?p1 }
  BIND (EXISTS { ?p1_inverse rdfs:subPropertyOf ?p2_inverse } AS ?inverse_ok)
  FILTER (?p1 != ?p2)
}
```

Most important features to note: it’s short and fast.

A collection of similar queries can form the basis of a continuous validation suite for ontology and model development.
A domain need not be crisply or formally defined in order to develop an ontology for it

- Instead, that can be a good reason to develop an ontology
- One of the most widely-known ontologies is Friend of a Friend, which is about human relationships

It’s helpful to think about what people say and what inferences we can draw from that:

- Blog posting X was written to Lorraine
- Lorraine is my friend
- I’m interesting in blog postings by friends
- Therefore, I’m interested in blog posting X
- Note: precise definition of friend not required

Careful thinking about inferences leads to improved semantics

Terms in an ontology are local: we’re not trying to define fault for the world, just for the FM community
• There does not appear to be a sharp distinction between things that are and are not *fault management*
  – That’s OK
• The challenge seems to be not to classify things as *fault management* or not, but instead to record what fault management authorities say within the scope of that authority
  – In that sense, fault management is whatever these authorities declare it be
  – That’s consistent with the idea that the scope of fault management for a given system is a design choice that reflects numerous factors:
    • criticality of mission objectives
    • epistemic and aleatory uncertainties
    • threats
    • programmatic resources
    • etc.
What is an Anomaly? Or perhaps more usefully, what would we say about an Anomaly?

- An Anomaly has a temporal extent
  - So we need to create an Epoch with start and end times
- An Anomaly declares that some Assertion is inconsistent with some Assessment during its Epoch
  - “I was expecting Lorraine at 5 but it’s 5:30 and I don’t see her.”
- So we need to create Assessment and Assertion
- We do not at this point need to define inconsistent
  - And are better off not trying to
  - Its meaning depends on the details of Assertion and Assessment
• **An Assertion is a predicate involving the world**
  - It need not be true, thought to be true, or desired
  - “It is raining.”
  - “Lorraine will be here at 5:00.”
  - “The World will end in 2012.”

• **Like an Anomaly, an Assertion has a temporal extent over which it is applicable**
  - So we should create an abstract concept called 
    DurativeElement (thing with an Epoch) and make Anomaly 
    and Assertion refinements of it

• **At this level of abstraction, we capture only the statement, not what it means in the real world**
  - We can create more refined concepts for restricted classes of predicates
    - “Mean temperature will increase at least 0.3 C by 2025.”
• An Assessment is an Assertion about the actual state of the world
  – “It is raining.”
  – “The estimated temperature is 32 C.”
  – “I don’t think Lorraine is here.”
• Because an Assessment is an Assertion, it also has an Epoch
• Again, at this level of abstraction, we don’t attempt to encode the semantics of the assertion
  – But we might for restricted specializations
We can refine Assertion to convey things we \textit{want} to be true

Probably need two levels of refinement

- Desire: something we merely want to be true
  - “I hope it doesn’t rain tomorrow.”
- Goal: something we will try to \textit{make} true
  - “I need to be at the bar by 5:00.”

Every Desire is an Assertion, every Goal is a Desire

Again, the ontology itself does not establish the precise criteria that distinguish Desire and Goal

- It provides the vocabulary to express the distinction
The general sense I get is that *fault* denotes something not as it *should be*

More formally, a Fault is an Anomaly for which the Assertion is a Desire

- “I was hoping it wouldn’t rain today but it’s pouring.”

The general sense I get is that *failure* denotes something we wanted *to achieve* but didn’t

More formally, a Failure is an Anomaly for which the Assertion is a Goal

- “I needed to be at the bar by 5:00 but I got there at 5:30.”

Every Failure is a Fault because every Goal is a Desire
• Is every inconsistency between an Assessment and a Desire a Failure? Is every inconsistency between an Assessment and a Goal a Failure?
• Yes, but only by convention
• Remember that we’re talking about Fault, Failure, and Goal in the local context of fault management
• An intent may be a Goal in the context of a general control system without being a fault management Goal
• The decision to make a control system Goal a fault management Goal is a design choice that places it within the scope of fault management
  – If that’s not what you mean, don’t say it
• Again, we don’t attempt to say precisely what being “in the scope of fault management” means
  – That will vary by project, application, etc.
• Everything we’ve discussed can be expressed simply in OWL
  – Classes: Epoch, Anomaly, Assertion, Assessment, DurativeElement, Desire, Goal, Fault, Failure
  – Class taxonomy: isa relationships among classes
  – Datatype properties: hasStartTime, hasEndTime
  – Object properties: hasEpoch, hasAssertion, hasAssessment
  – Range restrictions
    • range of hasAssertion restricted to Desire on domain Fault
    • range of hasAssertion restricted to Goal on domain Failure
• The ontology can be shown to be consistent and satisfiable
  – Not a big deal, but more important as it grows larger and more complex
Summary from Protégé\textsuperscript{5} OWL Editor
References

1. OWL 2 Web Ontology Language,
   http://www.w3.org/TR/owl-overview/
2. SPARQL 1.1 Query Language,
   http://www.w3.org/TR/2012/WD-sparql11-query-20120105/
3. RDF Primer,
   http://www.w3.org/TR/2004/REC-rdf-primer-20040210/
4. Sesame RDF Triple Store,
   http://www.openrdf.org/index.jsp
5. Protégé OWL Editor, http://protege.stanford.edu