

Reflection in Ontology

David Dodds open-meta.com

Reflection? What is that?

Reflection, in the context of this poster, is a tool used by programs which require the ability to examine or modify the runtime behavior of applications running in the machine.

This poster briefly explains what reflection accomplishes in programming languages (without the usual microfiche of programming code embedded in the poster).

It also briefly explains what applying these concepts to ontologies means.

Computer ontologies are basically taxonomies, and are much less sophisticated than the ontologies discussed by philosophers.

Computer ontologies are not intended to provide or bring to the computer all the capabilities explicitly and implicitly 'in' ontologies of / for humans, such as those ontologies developed by philosophers.

[If you are interested in reading some material about these latter ontologies have a look at 'Towards an Ontology of Experience - methodological reflections' or generally use Google to look for 'ontology'.]

Computer ontologies are formalisms to represent certain kinds of data, having certain kinds of data structures.

They are less sophisticated in the kinds of data, such as the universe of data types that the data may take, and the data structures are less sophisticated / complex than those appearing in the ontologies prepared by philosophers and which relate to humans / human cognitive levels.

Generally, contemporary computer ontologies are graph structures stated as hierarchies of nodes and links, such as classes and subclasses.

One ontology language used to describe ontologies is OWL, the Web Ontology Language. It is a member of the RDF family, which is a system of representation often appearing as triples.

[See <http://www.w3.org/TR/owl-features/> for the details of OWL and <http://www.w3.org/RDF/> for the details of RDF.]

Computer ontologies often appear as lexical constructs using XML as a structure defining representation.

[See <http://www.owl-ontologies.com/> unnamed.owl, Driver.owl, Person.owl, Human.owl, etc for examples of OWL "code".]

Such ontologies are capable of providing the features / capabilities of 'a data model that represents a set of concepts within a domain and the relationships between those concepts.

It is used to reason about the objects within that domain.' (wiki)

but amongst other things ontologies have limited ability to specify mathematical / mathematically related concepts in the formalism.

One way of overcoming that is to have a joint ontology which specializes in bringing numerical concepts into an ontological setting.

NASA JPL has done just that with their geospatial SWEET (Semantic Web for Earth and Environmental Terminology) system of (15) ontologies, among them being a numerical ontology.

[If you are interested in reading details about the JPL NASA 'SWEET system of ontologies' look at <http://sweet.jpl.nasa.gov/ontology/>.]

Additionally to the ontological content of the ontology data set one may make use of semantic processing programs such as SWRL <http://www.w3.org/Submission/SWRL/> and SPARQL Query Language for RDF <http://www.w3.org/TR/rdf-sparql-query/>

SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. and SWRL: A Semantic Web Rule Language Combining OWL and RuleML, (SWRL) based on a combination of the OWL DL and OWL Lite sublanguages.

Still yet additionally one may use topic map technology in a plug-board connector paradigm to functionally link XML elements constituting ontology components.

One topic map structure I have used consists of a graph (structure) of nodes, each having a property list and a set of operators. (The latter may be used in a kind of 'if-needed daemon' manner as in some LISP systems.)

XML IDs may be used like 'jacks' or connectors, to which the 'plugs' of the graph-edges which connect the topic map nodes together, connect to.

Topic maps may be said to (virtually) hover above the XML based data sets, such as OWL based ontologies, and provide "references into" or (virtual) connectors to the XML based data sets.

The nodes (with their property lists and operators) may then, as though a patch-board connected collection of processors, bring extra (externally provided) functionality to that formally supported via the XML structure (such as ontologies) with which it is associated.

Using this topic map reference into XML IDs in the ontological collection and the use of the techniques for ontological alignment and merging, such as discussed by Noy et al at Stanford University, it is then possible to have a collection of ontologies which function collectively (as a single (multi-domain!) knowledge-base).

Multiple formal ontologies are thereby functionally 'fused' (collectivised) thus providing to a computer some of the cognitive benefits which we saw earlier for humans in seamless omni-domain situations.

Yet remaining is the problem of typical ontologies being 'passive' in the manner that the content of books and of databases is 'passive'.

Database systems typically consist of large amounts of dead or passive data just as it is in books, and the database engine provides some ability to perform some operations upon that data.

While a large technical library (of books) HAS knowledge the books themselves are inert, passive and do not read themselves or in other ways process the knowledge they contain.

In this respect most ontologies are implemented in a manner similar to a book, they are themselves inert or passive.

Programs like SPARQL and SWRL etc can provide some aspects of 'active' content, just as the engine in a database system gives the user the effect of active data there. Triggers, in databases, can perform some of the functions of 'if-needed daemon' but are less sophisticated.

If one adds reflection to an ontology, one provides the means for a program to examine or "introspect" upon that ontology, and manipulate internal properties of the ontology. This is truly what meta-programming is about, whether Java Reflection or ontology reflection.

Scanning or parsing the XML structure of the ontology, perhaps using SAX or DOM, XSLT etc permits the examining program to obtain certain kinds of information about the ontology:

- what sorts of predicates / relationships (between the classes / nodes) are there,
- what is the nature of semantic 'restrictions' found there,
- the names of nodes / concepts there,
- the other namespaces involved in the ontology (such as the identities of all other participating ontologies,
- and any syntactical schema processors used).
- XML ID 'connector' points can be identified for topic map "reference into'.

When multiple ontologies for a single domain, such as 3 different space ontologies are there the required information needed to perform merging and alignment can be determined,

and any required operators in the topic map, (edge(s)) needed to transform the content between an ontology point and the topic map node receiver.

This is useful in the case where there is a not-one-to-one transformation of material from the ontology to the topic map or using the topic map as a functional transition device between one ontology and a receiving (but different) ontology.

The topic map also provides a means of implementing such processes as temporal accommodation as in

the following: An ontology contains an `*:id="John Smith"`.

'John' has the temporal attribute value of 'student' at any time prior to 'has-PHD', and any time afterward he has the temporal attribute value of 'professor'.

He is always a PERSON, both before and after the temporal 'point' or 'event' 'has-PHD'.

Ordinarily, ontologies have difficulty representing temporally defined values and the topic map provides a means of dealing with situations, such as time sensitive content, which they ordinarily are challenged to handle. [eg 'Engineering a complex ontology with time', Jorge Santos, Steffen Staab.]

Such explicit temporal enumeration and other external (eg tmap) operator processing allows the ontology to (partly) deal with the 'frame problem'. (such as: [NOT] your front door paint colour changing because you aren't looking at it, or the doorknob relocating as the result of you sneezing at work.) Implementation of 'reminding'.