A Reference Guide to Ontology Best Practices

by Mike Bergman - Monday, September 13, 2010

http://www.mkbergman.com/911/a-reference-guide-to-ontology-best-practices/

A Single-stop Assembly of Ontology Tips and Pointers

As we conclude this recent series on ontology tools and building [1], one item stands clear: the relative lack of guidance on how one actually builds and maintains these beasties. While there is much of a theoretic basis in the literature and on the Web, and much of methodologies and algorithms, there is surprisingly little on how one actually goes about creating an ontology.

An earlier posting pointed to the now classic *Ontology Development 101* article as a good starting point [2]. Another really excellent starting point is the Protégé 4 user manual [3]. Though it is obviously geared to the Protégé tool and its interface, it also is an instructive tutorial on general ontology (OWL) topics and constructs. I highly recommend printing it out and reading it in full.

If you do nothing else, you should download, print and study in full the Protégé 4 users manual [3].

Learning by Example

Another way to learn more about ontology construction is to inspect some existing ontologies. Though one may use a variety of specialty search engines and Google to find ontologies [4], there are actually three curated services that are more useful and which I recommend.

The best, by far, is the repository created by the University of Manchester for the now-completed TONES project [5]. TONES has access to some 200+ vetted ontologies, plus a search and filtering facility that helps much in finding specific OWL constructs. It is a bit difficult to filter by OWL 2-compliant only ontologies (except for OWL 2 EL), but other than that, the access and use of the repository is very helpful. Another useful aspect is that the system is driven by the OWL API, a central feature that we recommended in the prior tools landscape posting. From a learning standpoint this site is helpful because you can filter by vocabulary.

An older, but similar, repository is OntoSelect. It is difficult to gauge how current this site is, but it nonetheless provides useful and filtered access to OWL ontologies as well.

These sources provide access to complete ontologies. Another way to learn about ontology construction is from a bottom-up perspective. In this regard, the Ontology Design Patterns (ODP) wiki is the definitive source [6]. This is certainly a more advanced resource, since its premise begins from the standpoint of modeling issues and patterns to address them, but the site is also backed by an active community and curated by leading academics. Besides ontology building patterns, ODP also has a listing of exemplary ontologies (though without the structural search and selection features of the sources above). ODP is not likely the first place to turn to and does not give "big picture" guidance, but it also should be a bookmarked reference once you begin real ontology development.

It is useful to start with fully constructed ontologies to begin to appreciate the scope involved with them. But, of course, how one gets to a full ontology is the real purpose of this post. For that purpose, let's now turn our attention to general and then more specific best practices.

Sources of Best Practices

As noted above, there is a relative paucity of guidance or best practices regarding ontologies, their construction and their maintenance. However, that being said, there are some sources whereby guidance can be obtained.

To my knowledge, the most empirical listing of best practices comes from Simperl and Tempich [7]. In that 2006 paper they examined 34 ontology building efforts and commented on cost, effectiveness and methodology needs. It provides an organized listing of observed best practices, though much is also oriented to methodology. I think the items are still relevant, though they are now four to five years old. The paper also contains a good reference list.

Various collective ontology efforts also provide listings of principles or such, which also can be a source for general guidance. The OBO (The Open Biological and Biomedical Ontologies) effort, for example, provides a listing of principles to which its constituent ontologies should adhere [8]. As guidance to what it considers an exemplary ontology, the ODP effort also has a useful organized listing of criteria or guidance.

One common guidance is to re-use existing ontologies and vocabularies as much as possible. This is a major emphasis of the OBO effort [9]. The NeOn methodology also suggests guidelines for building individual ontologies by re-use and re-engineering of other domain ontologies or knowledge resources [10]. Brian Sletten (among a slate of emerging projects) has also pointed to the use of the Simple Knowledge Organization System (SKOS) as a staging vocabulary to represent concept schema like thesauri, taxonomies, controlled vocabularies, and subject headers [11].

The Protégé manual [3] is also a source of good tips, especially with regard to naming conventions and the use of the editor. Lastly, the major source for the best practices below comes from Structured Dynamics' own internal documentation, now permanently archived. We are pleased to now consolidate this information in one place and to make it public.

The best practices herein are presented as single bullet points. Not all are required and some may be changed depending on your own preferences. In all cases, however, these best practices are offered from Structured Dynamics' perspective regarding the use and role of adaptive ontologies [12]. *To our knowledge, this perspective is a unique combination of objectives and practices, though many of the individual practices are recommended by others.*

General Best Practices

General best practices refer to how the ontology is scoped, designed and constructed. Note the governing perspective in this series has been on lightweight, domain ontologies.

Scope and Content

- Provide **balanced coverage** of the subject domain. The breadth and depth of the coverage in the ontology should be roughly equivalent across its scope
- **Reuse structure and vocabularies** as much as possible. This best practice refers to leveraging non-ontological content such as existing relational database schema, taxonomies, controlled vocabularies, MDM directories, industry specifications, and spreadsheets and informal lists. Practitioners within domains have been looking at the questions of relationships, structure, language and meaning for decades. Effort has already been expended to codify many of these understandings. Good practice therefore leverages these existing structural and vocabulary assets (of any nature), and relies on known design patterns
- Embed the domain coverage into a proper **context**. A major strength of ontologies is their potential ability to interoperate with other ontologies. Re-using existing and well-accepted vocabularies and including concepts in the subject ontology that aid such connections is good practice. The ontology should also have sufficient reference structure for guiding the assignment of what content “is about”
- Define clear **predicates** (also known as properties, relationships, attributes, edges or slots), including a precise definition. Then, when relating two things to one another, use care in actually assigning these properties. Initially, assignments should start with a logical taxonomic or categorization structure and expand from there into more nuanced predicates
- Ensure the relationships in the ontology are **coherent**. The essence of coherence is that it is a state of logical, consistent connections, a logical framework for integrating diverse elements in an intelligent way. So while context supplies a reference structure, coherence means that the structure makes sense. Is the hip bone connected to the thigh bone, or is the skeleton askew? Testing (see below) is a major aspect for meeting this best practice
- **Map to external ontologies** to increase the likelihood of sharing and interoperability. In Structured Dynamics' case, we also attempt to map at minimum to the UMBEL subject reference structure for this purpose [13]

- Rely upon a **set of core ontologies** for external re-use purposes; Structured Dynamics tends to rely on a set of primary and secondary standard ontologies [14]. The corollary to this best practice is don't link indiscriminantly.

Structure and Design

- Begin with a **lightweight, domain ontology** [15]. Ontologies built for the pragmatic purposes of setting context and aiding disparate data to interoperate tend to be lightweight with only a few predicates, such as `isAbout`, `narrowerThan` or `broaderThan`. But, if done properly, these lighter weight ontologies with more limited objectives can be surprisingly powerful in discovering connections and relationships. Moreover, they are a logical and doable intermediate step on the path to more demanding semantic analysis. Because we have this perspective, we also tend to rely heavily on the SKOS vocabulary for many of our ontology constructs [16]
- Try to structurally **split domain concepts from instance records**. Concepts represent the nodes within the structure of the ontology (also known as classes, subject concepts or the *TBox*). Instances represent the data that populates that structure (also known as named entities, individuals or the *ABox*) [17]. Trying to keep the ABox and TBox separate enables easier maintenance, better understandability of the ontology, and better scalability and incorporation of new data repositories
- Treat many concepts via **"punning" as both classes and instances** (that is, as either sets or members, depending on context). The "punning" technique enables "metamodeling," such as treating something via its IRI as a set of members (such as `Mammal` being a set of all mammals) or as an instance (such as `Endangered Mammal`) when it is the object of a different contextual assertion. Use of "metamodeling" is often helpful to describe the overall conceptual structure of a domain. See endnote [18] for more discussion on this topic
- Build ontologies **incrementally**. Because good ontologies embrace the open world approach [19], working toward these desired end states can also be incremental. Thus, in the face of common budget or deadline constraints, it is possible initially to scope domains as smaller or to provide less coverage in depth or to use a small set of predicates, all the while still achieving productive use of the ontology. Then, over time, the scope can be expanded incrementally. Much value can be realized by starting small, being simple, and emphasizing the pragmatic. It is OK to make those connections that are doable and defensible today, while delaying until later the full scope of semantic complexities associated with complete data alignment
- Build **modular** ontologies that split your domain and problem space into logical clusters. Good ontology design, especially for larger projects, warrants a degree of modularity. An architecture of multiple ontologies often works together to isolate different work tasks so as to aid better ontology management. Also, try to use a core set of **primitives** to build up more complex parts. This is a kind of reuse within the same ontology, as opposed to reusing external ontologies and patterns. The corollary to this is: the same concepts are not created independently multiple times in different places in the ontology. Adhering to both of these practices tends to make ontology development akin to object-oriented programming
- Assign **domains and ranges** to your properties. Domains apply to the subject (the left hand side of a triple); ranges to the object (the right hand side of the triple). Domains and ranges should not be understood as actual constraints, but as axioms to be used by reasoners. In general, domain for a property is the range for its inverse and the range for a property is the domain of its inverse. Use of domains and ranges will assist testing (see below) and help ensure the coherency of your

ontology

- Assign **property restrictions**, but do so sparingly and judiciously [20]. Use of property restrictions will assist testing (see below) and help ensure the coherency of your ontology
- Use **disjoint classes** to separate classes from one another where the logic makes sense and dictates (if not explicitly stated, they are assumed to overlap)
- Write the ontology in a **machine-processable language** such as OWL or RDF Schema (among others), and
- Aggressively use **annotation properties** (see next) to promote the usefulness and human readability of the ontology.

Naming and Vocabulary Best Practices

- Name all **concepts as single nouns**. Use CamelCase **notation** for these classes (that is, class names should start with a capital letter and not contain any spaces, such as `MyNewConcept`)
- Name all **properties as verb senses** (so that triples may be actually read); e.g., `hasProperty`. Try to use **mixedCase notation** for naming these predicates (that is, begin with lower case but still capitalize thereafter and don't use spaces)
- Try to use common and **descriptive prefixes and suffixes** for related properties or classes (while they are just labels and their names have no inherent semantic meaning, it is still a useful way for humans to cluster and understand your vocabularies). For examples, properties about languages or tools might contain suffixes such as '`Language`' or '`Tool`' for all related properties
- Provide **inverse properties** where it makes sense, and adjust the verb senses in the predicates to accommodate. For example, `<Father> <hasChild> <Janie>` would be expressed inversely as `<Janie> <isChildOf> <Father>`
- Give all concepts and properties a **definition**. The matching and alignment of things is done on the basis of concepts (not simply labels) which means each concept must be defined [21]. Providing clear definitions (along with the coherency of its structure) gives an ontology its semantics. Remember not to confuse the label for a concept with its meaning. (This approach also aids multi-linguality). In its own ontologies, Structured Dynamics uses the property of `skos:definition`, though others such as `rdfs:comment` or `dc:description` are also commonly used
- Provide a **preferred label** annotation property that is used for human readable purposes and in user interfaces. For this purpose, Structured Dynamics uses the property of `skos:prefLabel`
- Include a class “**SemSet**”, which means a series of alternate labels and terms to describe the concept. These alternatives include true synonyms, but may also be more expansive and include jargon, slang, acronyms or alternative terms that usage suggests refers to the same concept. The `umbel:SemSet` construct enables a listing of individual members to be generated that provides the matching set for tagging and information extraction tasks. (As such, also include the `prefLabel` in the `SemSet` for proper lookup and tagging purposes.) The `SemSet` construct is similar to the "synsets" in Wordnet, but with a broader use understanding. This construct is an integral part of Structured Dynamics' approach to using ontologies for information extraction and tagging of unstructured text
- Try to assign l**ogical and short names to namespaces** used for your vocabularies, such as `foaf:XXX, umbel:XXX` or `skos:XXX`, with a maximimum of five letters preferred
- Enable **multi-lingual capabilities** in all definitions and labels. This is a rather complicated best

practice in its own right. For the time being, it means being attentive to the `xml:lang="en"` (for English, in this case) property for all annotation properties
- (If you disagree with these naming conventions, use your own, but in any event, be consistent!!).

Documentation Best Practices

- Like good software programs, a **properly constructed and commented ontology** is the first requirement of best practice documentation
- The entire **ontology vocabulary** should be documented via a dedicated system that allows finding, selecting and editing of any and all ontology terms and their properties
- The **methodologies should be documented** for ontology construction and maintenance, including naming, selection, completeness and other criteria. Documents such as this one and others in this series provide examples of important supplementary documentation regarding methodology and practice
- Provide a **complete** TechWiki**-like documentation** system for use cases, best practices, evaluation and testing metrics, tools installation and use, and all aspects of the ontology lifecycle should be provided and supported [22]
- Develop a **complete graph of the ontology** and make it available via graph visualization tools to aid understanding of the ontology in its complete aspect [23], and
- Other ample **diagrams and flowcharts** should also be prepared and made available for knowledge workers' use. UML diagrams, for example, might be included here, but general workflows and concept relationships should be explicated in any case through visual means. Such diagrams are much easier to understand and follow than the actual ontology specification.

Organizational and Collaborative Best Practices

- **Collaboration** is an implementation best practice [24]
- **Re-use** of already agreed-up structures and vocabularies respects prior investments and needs to be emphasized
- Improved **processes for consensus making**, including tools support, must be found to enable work groups to identify and decide upon terminology, definitions, alternative labels (`SemSets`), and relations between concepts. These processes need not be at the formal ontology level, but at the level of the concept graph underlying the ontology [24].

Testing Best Practices

- Test **new concepts**, aided by proper domain, range and property restrictions; by invoking reasoners such that inconsistencies can be determined [25]
- Test **new properties**, aided by invoking reasoners, which will identify inconsistencies [25]
- Test via **external class assignments**, by linking to classes in external ontologies, which acts to 'explode the domain' [26]
- Use **external knowledge bases and ontologies**, such as Cyc or UMBEL [27], to conduct coherency testing for the basic structure and relationships in the ontology
- Evolve the ontology specification to include **necessary and sufficient conditions** [25] aid more complete reasoner testing for consistency and coherence.

Best Practices for Adaptive Ontologies

In the case of ontology-driven applications using ***adaptive ontologies*** [28], there are also additional instructions contained in the system (via administrative ontologies) that tell the system which types of widgets need to be invoked for different data types and attributes. This is different from the standard conceptual schema, but is nonetheless essential to how such applications are designed.

- Use the **structWSF** middleware layer [29] as the abstract access point to:
    - To create, update, delete or otherwise manage data records
    - To browse or view existing records or record sets, based on simple to possible complex selection or filtering criteria, or
    - To take one of these results sets and progress it through various workflows involving specialized analysis, applications, or visualization.
- Supplement the domain ontology with a **semantic component ontology** for the purposes of guiding data widget display and visualization [30], and
- Supplement the domain ontology with the irON (*instance record Object Notation*) for dataset exchange and interoperability [31].

The administrative ontologies supporting these applications are managed differently than the standard domain ontologies that are the focus of most of the best practices above. Nonetheless, some of the domain ontology best practices work in tandem with them, the combination of which are called ***adaptive ontologies***.

---

[1] This posting is part of a current series on ontology development and tools, now permanently archived and updated on the OpenStructs TechWiki. The series began with An Executive Intro to Ontologies, then continued with an update of the prior Ontology Tools listing, which now contains 185 tools. It progressed to a survey of ontology development methodologies. That led to a presentation of a new, Lightweight, Domain Ontologies Development Methodology. That piece was then expanded to address A New Landscape in Ontology Development Tools. This portion completes the series.

[2] Natalya F. Noy and Deborah L. McGuinness, 2001. "Ontology Development 101: A Guide to Creating Your First Ontology," Stanford University *Knowledge Systems Laboratory Technical Report KSL-01-05*, March 2001. See http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html.

[3] Matthew Horridge et al., 2009. *A Practical Guide to Building OWL Ontologies Using Protégé 4 and CO-ODE Tools*, manual prepared by the *University of Manchester*, March 13, 2009. 108 pp. See http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_2.pdf.

[4] Specialty search engines for ontologies include Swoogle, FalconS, Watson, Sindice and SWSE. In addition, one can use a general search engine such as Google with a search query such as `<topic> owl:equivalentClass filetype:owl`. Note the filetype might also include RDF or a variant such as N3, and other language-specific constructs of interest can also be substituted for the `owl:equivalentClass`.

[5] The TONES Ontology Repository is primarily designed to be a central location for ontologies that might be of use to tools developers for testing purposes. It has a nice browse facility, as well as filtering by OWL vocabulary. The system contains about 220 ontologies and is powered by the OWL API.

[6] OntologyDesignPatterns.org is a semantic Web portal dedicated to ontology design patterns (ODPs). The portal was started under the NeOn project, which still partly supports its development.

[7] Elena Paslaru Bontas Simperl and Christoph Tempich, 2006. "Ontology Engineering: A Reality Check," in *Proceedings of the 5th International Conference on Ontologies, Databases, and Applications of Semantics ODBASE2006*, 2006. See http://ontocom.ag-nbi.de/docs/odbase2006.pdf .

[8] See http://obofoundry.org/wiki/index.php/OBO_Foundry_Principles.

[9] Barry Smith et al., 2007. "The OBO Foundry: Coordinated Evolution of Ontologies to Support Biomedical Data Integration," in *Nature Biotechnology* **25**: 1251 - 1255, published online 7 November 2007; see http://www.nature.com/nbt/journal/v25/n11/pdf/nbt1346.pdf.

[10] See the NeOn networked ontologies project; see http://www.neon-project.org/. The four-year project began in 2006 and its first open source toolkit was released by the end of 2007. OWL features were added in 2008-09. NeON has since completed, though its toolkit and plug-ins can still be downloaded as open source.

[11] Brian Sletten, 2008. "Applying SKOS Concept Schemes," on the *DevX Web site*, July 22, 2008; see http://www.devx.com/semantic/Article/38629.

[12] M. K. Bergman, 2009. "Confronting Misconceptions with Adaptive Ontologies," *AI3:::Adaptive Information* blog, Aug. 17, 2009.

[13] UMBEL (*Upper Mapping and Binding Exchange Layer*) is an ontology of about 20,000 subject concepts that acts as a reference structure for inter-relating disparate datasets. It is also a general vocabulary of classes and predicates designed for the creation of domain-specific ontologies.

[14] Core ontologies are Dublin Core, DC Terms, Event, FOAF, GeoNames, SKOS, Timeline, and UMBEL. The various criteria that are considered in nominating an existing ontology to "core" status is that it should be general; highly used; universal; broad committee or community support; well done and documented; and easily understood. Though less universal, there are also a number of secondary ontologies, namely BIBO, DOAP, and SIOC.

[15] See Fausto Giunchiglia, Maurizio Marchese and Ilya Zaihrayeu, 2006. "Encoding Classifications into Lightweight Ontologies," see http://www.science.unitn.it/~marchese/pdf/encoding%20classifications%20into%20lightweight%20ontologies_JoDS8.pdf. Also, M. K. Bergman, 2010. "A New Methodology for Buidling Lightweight, Domain Ontologies," *AI3:::Adaptive Information* blog, Sept. 1, 2010.

[16] Alistair Miles and Sean Bechhofer, eds., 2009. *SKOS Simple Knowledge Organization System Reference*, W3C Recommendation, 18 August 2009. See http://www.w3.org/TR/skos-reference/. Some of the common SKOS predicates used in our ontologies include `skos:definition, skos:prefLabel, skos:altLabel, skos:broaderTransitive, skos:narrowerTransitive`.

[17] The ***TBox*** portion, or classes (concepts), is the basis of the ontologies. The ontologies establish the structure used for governing the conceptual relationships for that domain and in reference to external (Web) ontologies. The ***ABox*** portion, or instances (named entities), represents the specific, individual things that are the members of those classes. Named entities are the notable objects, persons, places, events, organizations and things of the world. Each named entity is related to one or more classes (concepts) to which it is a member. Named entities do not set the structure of the domain, but populate that structure. The ABox and TBox play different roles in the use and

organization of the information and structure. These distinctions have their grounding in description logics.

[18] In the domain ontologies that are the focus here, we often want to treat our concepts as both classes and instances of a class. This is known as "metamodeling" or "metaclassing" and is enabled by "punning" in OWL 2. For example, here a case cited on the OWL 2 wiki entry on "punning":

People sometimes want to have metaclasses. Imagine you want to model information about the animal kingdom. Hence, you introduce a class a:Eagle, and then you introduce instances of a:Eagle such as a:Harry.

(1) a:Eagle rdf:type owl:Class
(2) a:Harry rdf:type a:Eagle

Assume now that you want to say that "eagles are an endangered species". You could do this by treating a:Eagle as an instance of a metaconcept a:Species, and then stating additionally that a:Eagle is an instance of a:EndangeredSpecies. Hence, you would like to say this:

(3) a:Eagle rdf:type a:Species
(4) a:Eagle rdf:type a:EndangeredSpecies.

This example comes from Boris Motik, 2005. "On the Properties of Metamodeling in OWL," paper presented at *ISWC 2005*, Galway, Ireland, 2005.

"Punning" was introduced in OWL 2 and enables the same IRI to be used as a name for both a class and an individual. However, the direct model-theoretic semantics of OWL 2 DL accommodates this by understanding the class `Father` and the individual `Father` as two different views on the same IRI, *i.e.*, they are interpreted semantically as if they were distinct. The technique listed in the main body triggers this treatment in an OWL 2-compliant editor. See further Pascal Hitzler *et al.*, eds., 2009. *OWL 2 Web Ontology Language Primer*, a W3C Recommendation, 27 October 2009; see http://www.w3.org/TR/owl2-primer/.

[19] There is a role and place for closed world assumption (CWA) ontologies, though Structured Dynamics does not engage in them.

CWA is the traditional perspective of relational database systems within enterprises. The premise of CWA is that which is not known to be true is presumed to be false; or, any statement not known to be true is false. Another way of saying this is that everything is prohibited until it is permitted. CWA works well in bounded systems such as known product listings or known customer rosters, and is one reason why it is favored for transaction-oriented systems where completeness and performance are essential. In an ontology sense, CWA works best for bounded engineering environments such as aeronautics or petroleum engineering. Closed world ontologies also tend to be much more complicated with many nuanced predicates, and can be quite expensive to build.

The open world assumption (OWA), on the other hand, is premised that the lack of a given assertion or fact being available does not imply whether that possible assertion is true or false: it simply is not known. In other words, lack of knowledge does not imply falsity, and everything is permitted until it is prohibited. As a result, open world works better in knowledge environments with the incorporation of external information such as business intelligence, data warehousing, data integration and federation, and knowledge management.

See further, M. K. Bergman, 2009. "The Open World Assumption: Elephant in the Room," *AI3:::Adaptive Information blog*, Dec. 21, 2009.

[20] See [3] for a good description of property restrictions in Section 4 and Appendix A.

[21] As another commentary on the importance of definitions, see http://ontologyblog.blogspot.com/2010/09/physician-decries-lack-of-definitions.html.

[22] The technical wiki (TechWiki) is the central repository for all documentation related to OpenStructs projects. TechWiki is the location for users and interested parties to learn about these projects and their applications, and for developers to author and write about their use and best practices. Both the TechWiki's content and its software and organizatonal structure may be downloaded for free for setting up similar local technical documentation.

[23] See M. K. Bergman, 2008. "Large-scale RDF Graph Visualization Tools," *AI3:::Adaptive Information* blog, Jan. 28, 2008; and M. K. Bergman, 2008. "Cytoscape: Hands-down Winner for Large-scale Graph Visualization," *AI3:::Adaptive Information* blog, Jan. 28, 2008.

[24] The central role of ontologies is to describe a "worldview" and in specific organizations this means a shared understanding of the concepts, relations and terminology to describe the participants' common domain. In turn, these shared understandings establish the semantics for how to effect communication and understanding within the population of domain users. All of this means that finding ways to identify and agree upon shared vocabularies and understandings is central to the task of modeling (creating an ontology) for the domain.

Sometimes this perception of shared views is too strictly interpreted as needing to have one and only one understanding of concepts and language. Far from it. One of the strengths of ontologies and language modeling within them is that multiple terms for the same concept or slight differences in understandings about nearly similar concepts can be accommodated. It is perfectly OK to have differences in terminology and concept understandings so long as those differences are also captured and explicated within the ontology. The recommendations herein that all concepts and terminology be defined, that `SemSets` be used to capture alternative ways to name concepts, and that concepts often be treated as both classes and instances are some of the best practices that reflect this approach.

So, while consensus building and collaboration methods are at the heart of effective ontology building, those methods need not also strive for a imposition of language and concepts by fiat. In fact, trying to do so undercuts the ability of the collaborative process to lead to greater shared understandings.

[25] See [3] for a good description of various testing and consistency checks in Sections 4.9 to 4.14.

[26] See Frédérick Giasson, 2008. "Exploding the Domain," from his blog, April 20, 2008. 'Exploding the domain' means what happens when internal ontology concepts are linked to related ones on the external Web, which helps to bring in more information and context about the concept. It is also a way to test the coherence of the original concept.

[27] Already vetted knowledge bases can be a good reference testbed for testing the coherence of concepts in a new

domain ontology. If the domain ontology describes concepts quite differently than standard practice (Wikipedia, Cyc and UMBEL are good for testing this), or if relationships between concepts are greatly at variance (Cyc and UMBEL are good for this), then there are likely coherency problems. In other domains other reference knowledge bases, more specific to the domain, can be used in similar ways.

[28] Structured Dynamics' *ontology-driven apps* are generic applications, the operations of which are guided by the instructions and nature of the underlying data that feeds them. For example, in the case of a standard structured data display (say, a simple table like a Wikipedia infobox), such generic design includes templates tailored to various instance types (say, locational information presenting on a map versus people information warranting a image and vital statistics). Alternatively, in the generic design for a data visualization application using Adobe Flash, the information output of the results set contains certain formats and attributes, keyed by an administrative ontology linked by data type to a domain ontology's results sets.

These *ontology-driven apps*, then, are informed structured results sets that are output in a form suitable to various intended applications. This output form can include a variety of serializations, formats or metadata. This flexibility of output is tailored to and responsive to particular generic applications; it is what makes our ontologies "adaptive". Using this structure, it is possible to either "drive" queries and results sets selections via direct HTTP request or via simple dropdown selections on HTML forms. Similarly, it is possible with a single parameter change to drive either a visualization app or a structured table template from the equivalent query request. *Ontology-driven apps* through this ontology and architecture design thus provide two profound benefits. First, the entire system can be driven via simple selections or interactions without the need for any programming or technical expertise. And, second, simple additions of new and minor output converters can work to power entirely new applications available to the system.

[29] The structWSF Web services framework is generally RESTful middleware that provides a bridge between existing content and structure and content management systems and available indexing engines and RDF data stores. structWSF is a platform-independent means for distributed collaboration via an innovative dataset access paradigm. It has about twenty embedded Web services. See http://openstructs.org/structwsf.

[30] A semantic component is a Flex component that takes record descriptions and schema as input, and then outputs some (possibly interactive) visualizations of that record. Depending on the logic described in the input schema and the input record descriptions, the semantic component will behave differently to optimize its presentation to the users. About a dozen semantic components are available from the Semantic Component (Flex) Library. The Semantic Component Ontology is the governing structure for these schema.

[31] irON (*instance record and Object Notation*) is a abstract notation and associated vocabulary for specifying RDF triples and schema in non-RDF forms. Its purpose is to allow users and tools in non-RDF formats to stage interoperable datasets using RDF. The notation supports writing RDF and schema in JSON (irJSON), XML (irXML) and comma-delimited (CSV) formats (commON). The notation specification includes guidance for creating instance records (including in bulk), linkages to existing ontologies and schema, and schema definitions. Profiles and examples and code parsers and converters are also provided for the irXML, irJSON and commON serializations.