A New Landscape in Ontology Development Tools

by Mike Bergman - Tuesday, September 07, 2010

http://www.mkbergman.com/909/a-new-landscape-in-ontology-development-tools/

Shifting the Center of Gravity to the OWL API, Web Services

Previous installments in this series have listed existing ontology tools, overviewed development methodologies, and proposed a new approach to building lightweight, domain ontologies [1]. For the latter to be successful, a new generation in ontology development tools is needed. This post provides an explication of the landscape under which this new generation of tools is occurring. Ontologies supply the structure for relating information to other information in the semantic Web or the linked data realm. Because of this structural role, ontologies are pivotal to the coherence and interoperability of interconnected data. We are now concluding the first decade of ontology development tools, especially those geared to the semantic Web and its associated languages of RDFS and OWL. Last year we also saw the release of the major update to the OWL 2 language, with its shift to more expressiveness and a variety of profiles. The upcoming next generation of ontology tools now must also shift. The current imperative is to shift away from ontology engineering by a priesthood to pragmatic daily use and maintenance by domain practitioners. Market growth demands simpler, task-focused tools with intuitive interfaces. For this change to occur, the general tools architecture needs to shift its center of gravity from IDEs and comprehensive toolkits to APIs and Web services. Not surprisingly, this same shift is what has been occurring across all areas of software.

Methodology Reprise: The Nature of the Landscape

In the previous installment of this series, we presented a new methodological approach to ontology development, geared to lightweight, domain ontologies. One aspect of that design was to separate the operational workflow into two pathways:

- Instances, and their descriptive characteristics, and
- Conceptual relationships, or ontologies.

The ontology build methodology concentrated on the upper half of this diagram (blue, with yellow lead-ins and outcomes) with the various steps overviewed in that installment [2]:

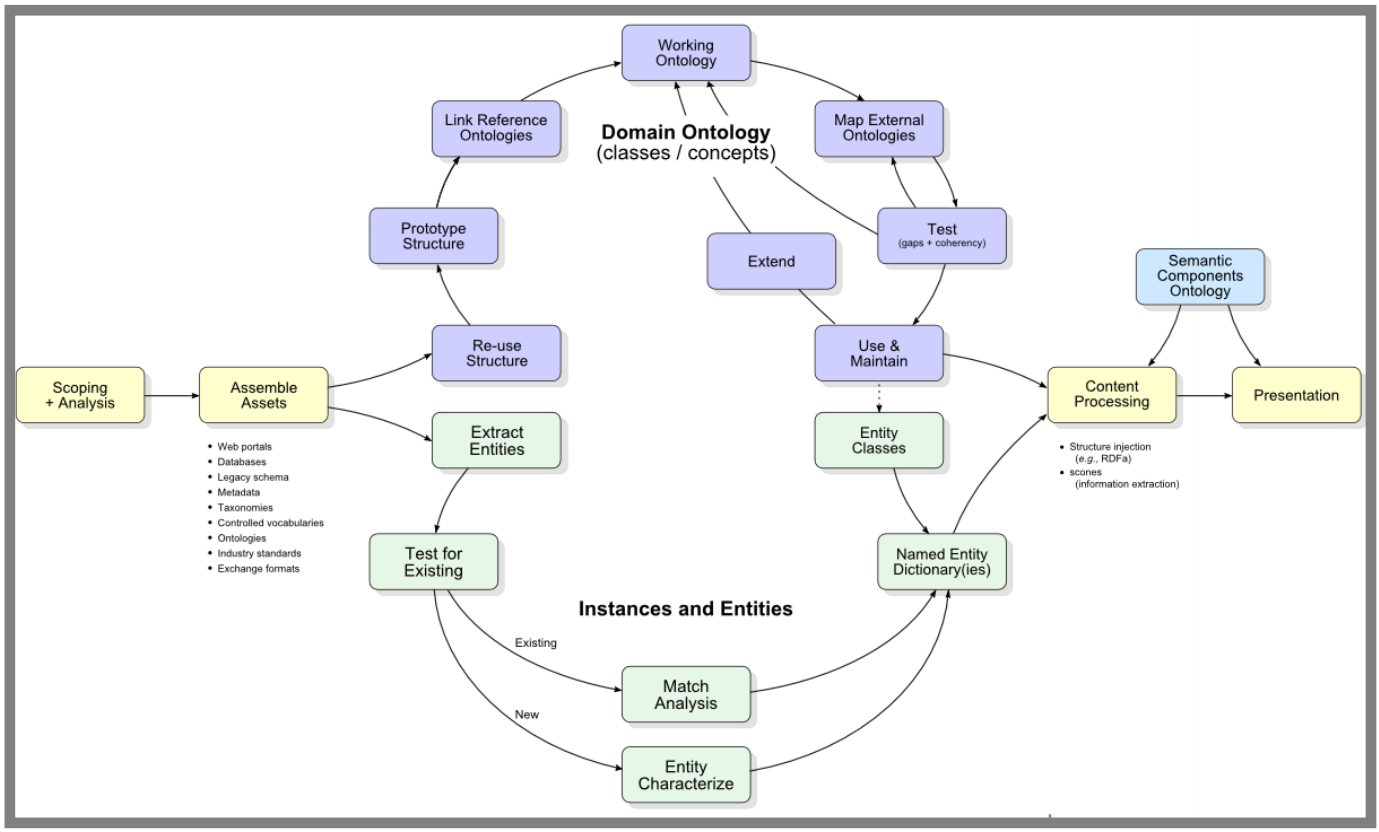

Figure 1. Flowchart of Ontology Development Methodology *(click to expand)*

The methodology captured in this diagram embraces many different emphases from current practice: re-use of existing structure and information assets; conscious split between instance data (ABox) and the conceptual structure (TBox) [3]; incremental design; coherency and other integrity testing; and explicit feedback for scope extension and growth. The methodology also embraces some complementary utility ontologies that also reflect the design of ***ontology-driven apps*** [4]. These are notable changes in emphasis. But they are not the most important one. The most important change is the tools landscape to implement this methodology. This landscape needs to shift to pragmatic daily use and maintenance by domain practitioners. That requires simpler and more task-oriented tools. And that change in tooling needs a still more fundamental shift in tools architecture and design.

A Legacy of Excellent First Generation Tools

In many places throughout this series I use the term "inadequate" to describe the current state of ontology development tools. This characterization is not a criticism of first-generation tools *per se*. Rather, it is a reflection of their inadequacy to fulfill the realities of the new tooling landscape argued in this series. The fact remains, as initial generation tools, that many of the existing tools are quite remarkable and will play central roles (mostly for the professional ontologist or developer) moving forward. At the risk of overlooking some important players, let's trace the (partial) legacy of some of the more pivotal tools in

today's environment. As early as a decade ago the ontology standards languages were still in flux and the tools basis was similarly immature. Frame logic, description logics, common logic and many others were competing at that time for primacy and visibility. Most ontology tools at that time such as Protégé [5], OntoEdit [6], or OilEd [7] were based on F-logic or the predecessor to OWL, DAML+Oil. But the OWL language was under development by the W3C and in anticipation of its formal release the tools environment was also evolving to meet it. Swoop [8], for example, was one of the first dedicated OWL browsers. A Protégé plug-in for OWL was also developed by Holger Knublauch [9]. In parallel, the OWL group at the University of Manchester also introduced the OWL API [10]. With the formal release of OWL 1.0 in 2004, ontology tools continued to migrate to the language. Protégé, up through the version 3x series, became a popular open source system with many visualization and OWL-related plug-ins. Knublauch joined TopQuadrant and brought his OWL experience to TopBraid Composer, which shifted to the Eclipse IDE platform and leveraged the Jena API [9,11]. In Europe, the NeON (Networked Ontologies) project started in 2006 and by 2008 had an Eclipse-based OWL platform using the OWL API with key language processing capabilities through GATE [12]. Most recently, Protégé and NeON in open source, and TopBraid Composer on the commercial side, have likely had the largest market share of the comprehensive ontology toolkits. So far, with the release of OWL 2 in late 2009, only Protégé in version 4 and the TwoUse Toolkit have yet fully embraced all aspects of the new specification, doing so by intimately linking with the new OWL API (version 3x has full OWL 2 support) [13]. However, most leading reasoners now support OWL 2 and products such as TopBraid Composer and Ontotext's OWLIM support OWL 2 RL as well [14]. The evolution of Protégé to version 4 (OWL 2) was led by the University of Manchester via its CO-ODE project [15], now ended, which has also been a source for most existing Protégé 4 plug-ins. (Because of the switch to OWL 2 and the OWL API most earlier plug-ins are incompatible with Protégé 4.) Manchester has also been a leading force in the development of OWL 2 and the alternative Manchester syntax. Though only recently stable because of the formalization of OWL 2, Protégé 4 and its linkage to the new OWL API provides for a very powerful combination. With Protégé, the system has a familiar ontology editing framework and a mechanism for plug-in migration and growth. With the OWL API, there is now a common API for leading reasoners (Pellet, HermiT, FaCT++, RacerPro, etc.), a solid ontology management and annotation framework, and validators for various OWL 2 profiles (RL, EL and QL). The system is widely embraced by the biology community, probably the most active scientific field in ontologies. However, plug-in support lags the diversity of prior versions of Protégé and there does not appear to be the energy and community standing behind it as in prior years.

A Normative Tools Landscape

These leading frameworks and toolkits have opted to be "ontology engineering" environments. Via plug-ins and complicated interfaces (tabs or Eclipse-style panes) the intent has apparently been to provide "all capabilities in one box." The tools have been IDE-centric. Unfortunately, one must be a combination of ontologist, developer, programmer and IDE expert in order use the tools effectively. And, as incremental capabilities get added to the systems, these also inherit the same complexity and style of the host environment. It is simply not possible to make complex environments and conventions simple. Curiously, the existence or use of APIs have also not been adequately leveraged. The usefulness of an API means that subsets of information can be extracted and worked on in very clear and simple ways. This information can then be roundtripped without loss. An API allows a tailored subset abstraction of the underlying data model. In contrast, IDEs, such as Protégé or Eclipse, when they play a similar role, force

all interfaces to share their built-in complexity. With these thoughts in mind, then, we set out to architect a tools suite and work flow that could truly take advantage of a central API. We further wanted to isolate the pieces into distributable Web services in keeping with our standard structWSF Web services framework design. This approach also allows us to split out simpler, focused tools that domain users and practitioners can use. And, we can do all of this while also enabling the existing professional toolsets and IDEs to also interoperate in the environment. The resulting tools landscape is shown in the diagram below. This diagram takes the same methodology flow from Figure 1 (blue and yellow boxes) and stretches them out in a more linear fashion. Then, we embed the various tools (brown) and APIs (orange) in relation to that methodology:

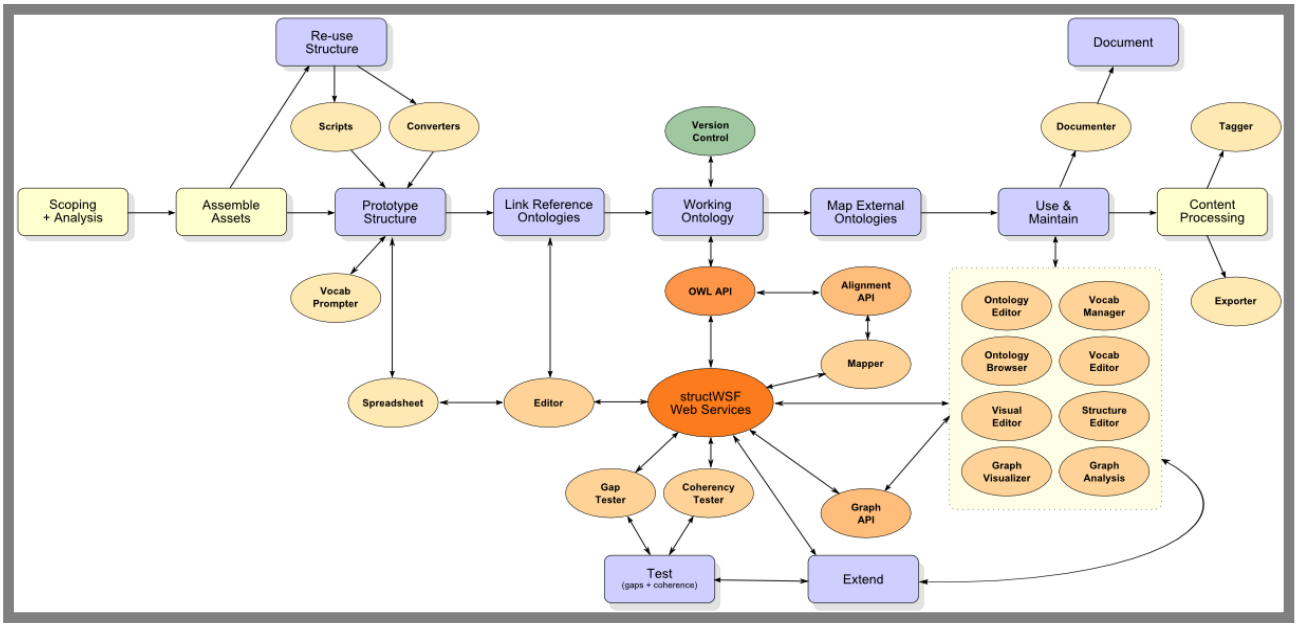

Figure 2. The Normative Ontology Tools Landscape *(click to expand)*

This diagram is worth expanding to full size and studying in some detail. Aspects of this diagram that deserve more discussion are presented in the sections below.

OWL API as Center of Gravity

As noted in the preceding methodology installment, the working ontology is the central object being managed and extended for a given deployment. Because that ontology will evolve and grow over time, it is important the complete ontology specification itself be managed by some form of version control system (green) [16]. This is the one independent tool in the landscape. Access to and from the working ontology is mediated by the OWL API [13]. The API allows all or portions of the ontology specification to be manipulated separately, with a variety of serializations. Changes made to the ontology can also be tested for validity. Most leading reasoners can interact directly with the API. Protégé 4 also interacts directly with the API, as can various rules engines [17]. Additionally, other existing APIs, notably the Alignment API with its own mapping tools and links to other tools such as S-Match can interact with the OWL API. It is reasonable to expect more APIs to emerge over time that also interoperate [18]. The OWL API is the best current choice because of its native capabilities and because Jena does not yet

support OWL 2 [11]. However, because of the basic design with structWSF (see next), it is also possible to swap out with different APIs at a later time should developments warrant. In short, having the API play the central management role in the system means that any and all tools can be designed to interact effectively with the working ontology(ies) without any loss in information due to roundtripping.

Web Services (structWSF) as Canonical Access Layer

The same rationale that governed our development of structWSF [19] applies here: to abstract basic services and functionality through a platform-independent Web services layer. This Web services layer has canonical (standard) ways to interact with other services and is generally RESTful in design to support distributed deployments. The design conforms to proper separation of view from logic and structure. Moreover, because of the design, changes can be made on either side of the layer in terms of user interface or functionality. Use of the structWSF layer also means that tools and functionality can be distributed anywhere on the Web. Specialized server-side functions can be supported as well as dedicated specialty hardware. Text indexing or disambiguation services can fit within this design. The ultimate value of piggybacking on the structWSF framework is that all other extant services also become available. Thus, a wealth of converters, data managers, and semantic components (or display widgets) can be invoked depending on the needs of the specific tool.

Simpler, Task-specific Tools

The objective, of course, of this design is to promote more and simpler tools useful to domain users. Some of these are shown under the Use & Maintain box in the diagram above; others are listed by category in the table below. The RESTful interface and parameter calls of the structWSF layer further simplify the ontology management and annotation abstractions arising from the OWL API. The number of simple tools available to users under this design is virtually limitless. These tools are also fast to develop and test.

Combining These New Thrusts and Moving Forward

This landscape is not yet a full reality. It is a vision of adaptive and simpler tools, working with a common API, and accessible via platform-independent Web services. It also preserves many of the existing tools and IDEs familiar to present ontology engineers. However, pieces of this landscape ***do*** presently exist and more are on the way. The next section briefly overviews some of the major application areas where these tools might contribute.

Individual Tools within the Landscape

If one inspects the earlier listing of 185 ontology tools it is clear that there is a diversity of tools both in terms of scope and function across the entire ontology development stack. It is also clear that nearly all of those 185 tools listed do not communicate with one another. That is a tremendous waste. Via shared APIs and some degree of consistent design it should be possible to migrate these capabilities into a more-or-less interoperating whole. We have thus tried to categorize some important tool types and exemplar tools from that listing to show the potential that exists. (Please note that the **Example Tools** are links to the tools and categories from the earlier 185 tools listing.) This correlation of types and example tools is not

meant to be exhaustive nor a recommendation of specific tools. But, this tabulation is illustrative of the potential that exists to both simplify and extend tool support across the entire ontology development workflow:

Tool Type	Comments	Example Tools
OWL API	OWL API is a Java interface and implementation for the W3C Web Ontology Language (OWL), used to represent Semantic Web ontologies. The API provides links to inferencers, managers, annotators, and validators for the OWL2 profiles of RL, QL, EL	OWL API
Web Services Layer	This layer provides a common access layer and set of protocols for almost all tools. It depends critically on linkage and communication with the OWL API	structWSF
Ontology Editor (IDE)	There are a variety of options in this area. Generally, more complete environments (that is, IDEs) based on OWL and with links to the OWL API are preferred. Less complete editor options are listed under other categories. Note that only Protégé 4 incorporates the OWL API	NeOn toolkit, Protégé, TopBraid Composer
Scripts	In all pragmatic cases the migration of existing structure and vocabulary assets to an ontology framework requires some form of scripting. These may be off the shelf resources, but more often are specific to the use case at hand. Typical scripting languages include the standard ones (Perl, Python, PHP, Ruby, XSLT, etc.) and often involve some form of	variety; specific to use case

	parsing or regex	
Converters	Converters are more-or-less pre-packaged scripts for migrating one serialization or data format to another one. As the scripts above continue to be developed, this roster of off-she-shelf starting points can increase. Today, there are perhaps close to 200 converters useful to ontology purposes	irON, ReDeFer, SKOS2GenTax; also see RDFizers
Vocabulary Prompter	Domain ontologies are ultimately about meaning, and for that purpose there is much need for definitions, synonyms, hyponyms, and related language assets. Vocabulary prompters take input documents or structures and help identify additional vocabulary useful for characterizing semantic meaning	see the TechWiki's vocab prompting tools; ROC
Spreadsheet	Spreadsheets can be important initial development environments for users without explicit ontology engineering backgrounds. The biggest issue with spreadsheets is that what is specified in them is more general or simplistic compared to what is contained in an actual ontology. Attempts to have spreadsheets capture all of this sophistication are often less than satisfactory. One way to effective "round trip" with spreadsheets (and many related simple tools) is to adhere to an OWL API	Anzo, RDF123, irON (commON), Excel, Open Office
Editor	Ontology editing spans from	see the TechWiki's

(general)	simple structures useful to non-ontologists to those (like the IDEs or toolkits) that capture all aspects of the ontology. Further, some of these editors are strictly textual or (literally) editors; others span or attempt to enable visual editing. Visual editing (see below) can ultimately extend to the ontology graph itself	ontology editing tools
Alignment API	The Alignment API is an API and implementation for expressing and sharing ontology alignments. The correspondences between entities (e.g., classes, objects, properties) in ontologies is called an alignment. The API provides a format for expressing alignments in a uniform way. The goal of this format is to be able to share on the web the available alignments. The format is expressed in RDF	Alignment API
Mapper	A variety of tools, algorithms and techniques are available for matching or mapping concepts between two different ontologies. In general, no single method has shown itself individually superior. The better approaches use voting methods based on multiple comparisons	see the TechWiki's ontology mapping tools
Ontology Browser	Ontology browsers enable the navigation or exploration of the ontology -- generally in visual form -- but without allowing explicit editing of the structure	Relation Browser, Ontology Browser, OwlSight, FlexViz

Vocabulary Manager	Vocabulary managers provide a central facility for viewing, selecting, accessing and managing all aspects of the vocabulary in an ontology (that is, to the level of all classes and properties). This tool category is poorly represented at present. Ultimately, vocabulary managers should also be one (if not the main) access point to vocabulary editing	PoolParty, TermWiki, UMBEL Web service
Vocabulary Editor	Vocabulary editors provide (generally simple) interfaces for the editing and updating of vocabulary terms, classes and properties in an ontology	Neologism, TemaTres, ThManager, Vocab Editor
Structure Editor	A structure editor is a specific form of an ontology editor, geared to the subsumption (taxonomic) organization of a largely hierarchical structure. Editors of this form tend to use tree controls or spreadsheets with indented organization to show parent and child relationships	PoolParty, irON (commON)
Graph Analysis	Ontologies form graph structures, which are amenable to many specific network and graph analysis algorithms, included relatedness, shortest path, grouped structures, communities and the like	SNAP, igraph, Network Workbench, NetworkX, Ontology Metrics
Graph API	Graph visualization with associated tools is best enabled by working from a common API. This allows for expansion and re-use of other capabilities. Preferably, this graph API would also have direct interaction with the	under investigation

	OWL API, but none exist at the moment	
Graph Visualizer	Graph visualizers enable the ontology to be rendered in graph form and presentation, often with multiple layout options. The systems also enable export to PDF or graphics formats for display or printing. The better tools in this category can handle large graphs, can have their displays easily configured, and are performant	see the TechWiki's ontology visualization tools
Visual Editor	An ontology visual editor enables the direct manipulation of the graph in a visual mode. This capability includes adding and moving nodes, changing linkages between nodes, and other ontology specification. Very few tools exist in this category at present	COE, TwoUse Toolkit
Coherence Tester	Testing for coherence involves whether the ontology structure is properly constructed and has logical interconnections. The testing either involves inference and logic testing (including entailments) based on the structure as provided; comparisons with already vetted logical structures and knowledge bases (*e.g.*, Cyc, Wikipedia); or both	Cyc, OWLim, FactForge
Gap Tester	Related to coherence testing, gap testing is the identification of key missing pieces or intermediary nodes in the ontology graph. This tends to happen when external specification of the ontology	requires use of a reference external ontology; see above

	is made without reference to connecting information	
Documenter	Ontology documentation is not limited to the technical specifications of the structure, but also includes best practices, how-to and use guides, and the like. Automated generation of structure documentation is also highly desirable	TechWiki, SpecGen, OWLDoc
Tagger	Once constructed, ontologies (and their accompanying named entity dictionaries) can be very powerful resources for aiding tagging and information extraction utilities. Like vocabulary prompting, there is a broad spectrum of potential tools and uses in the tagging category	GATE (OBIE); many other options
Exporter	Exports need to range from full-blown OWL representations to the simpler export of data and constructs. Multiple serialization options and the ability to support the input requirements of third-party tools is also important	OWL Syntax Converter, OWL Verbalizer; many various options

The beauty of this approach is that most of the tools listed are open source and potentially amenable to the minor modifications necessary to conform with this proposed landscape.

Key Gaps in the Landscape

Contrasting the normative tools landscape above with the existing listing of ontology tools points out some key gaps or areas deserving more development attention. Some of these are:

- Vocabulary managers -- easy inspection and editing environments for concepts and predicates are lacking. Though standard editors allow direct ontology language edits (OWL or RDFS), these are not presently navigable or editable by non-ontologists. Intuitive browsing structures with more "infobox"-like editing environments could be helpful here
- Graph API -- it would be wonderful to have a graph API (including analysis options) that could

communicate with the OWL API. Failing that, it would be helpful to have a graph API that communicated well with RDF and ontology structures; extant options are few
- Large-graph visualizer -- while we have earlier reviewed large-scale graph visualization software, the alternatives are neither easy to set up nor use. Being able to readily select layout options with quick zooms and scaling options are important
- Graphical editor -- some browsers or editors (*e.g*, FlexViz) provide nice graph-based displays of ontologies and their properties and annotations. However, there appear to be few environments where the ontology graph can be directly edited or visually used for design or expansion.

Finally, it does appear that the effort and focus behind Protégé seems to be slowing somewhat. The future has clearly shifted to OWL 2 with Protégé 4. Yet, besides the admirable CO-ODE project (now ended), tools and plug-in support seems to have slowed. Many of the admirable plug-ins for Protégé 3x do not appear to be under active development as upgrades to Protégé 4. While Protégé's future (and similar IDEs) seems assured, its prominence possibly will (and should) be replaced by a simpler kit of tools useful to users and practitioners.

Funding and Pending Project Priorities

For the past few months we at Structured Dynamics have seen ontology design and management as the pending technical priorities within the semantic technology space. Now that the market no longer looks at "ontology" as a four-letter word, it is imperative to simplify the development and use of ontologies. The first generation of tools leading up to this point have been helpful to ***understand*** the semantic space; changes are now necessary to ***expand*** it. In our first generation we have begun to understand the types and nature of needed tools. But our focus on IDEs and comprehensive toolsets belies a developer's or technologist's perspective. We need to now shift focus and look at tool needs from the standpoint of users and actual use of ontologies. Many players and many toolmakers and innovators will need to contribute to build this market for semantic technologies and approaches. Fortunately, replacing an IDE focus with one based around APIs and Web services should be a fairly smooth and natural transition. If we truly desire to be market makers, we need to stand back and place ourselves into the shoes of the domain practitioners, the subject matter experts. We need to shield actual users from all of the silly technical details and complexity. And, then, let's focus -- task-by-task -- on discrete items of management and use of ontologies. Growth of the semantic technology space depends on expanding our practitioner base. For its part, Structured Dynamics is presently seeking new projects and sponsors with a commitment to these aims. Like our prior development of structWSF and semantic components, we will be looking to make simpler ontology tools a priority in the coming months. Please let me know if you want to partner with us toward this commitment.

---

[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 update of the prior Ontology Tools listing, which now contains 185 tools. It continued with a survey of ontology development methodologies. The last part presented a Lightweight, Domain Ontologies Development Methodology. This part is archived on the TechWiki as the Normative Landscape of Ontology Tools. The last installment in the series is planned to cover ontology best practices .

[2] The original version, now slightly modified, was first published in M. K. Bergman, 2009. "Ontology-driven Applications Using Adaptive Ontologies," *AI3:::Adaptive Information* blog, Nov. 23, 2009.

[3] 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.

[4] See M.K. Bergman, 2009. "Ontology-driven Applications Using Adaptive Ontologies," *AI3:::Adaptive Information* blog, November 23, 2009.

[5] Natalya F. Noy, Michael Sintek, Stefan Decker, Monica Crubézy, Ray W. Fergerson and Mark A. Musen, 2001. "Creating Semantic Web Contents with Protégé-2000," *IEEE Intelligent Systems*, vol. 16, no. 2, pp. 60-71, Mar/Apr. 2001. See http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.24.7177&rep=rep1&type=pdf.

[6] York Sure, Michael Erdmann, Juergen Angele, Steffen Staab, Rudi Studer and Dirk Wenke, 2002. "OntoEdit: Collaborative Ontology Development for the Semantic Web," in *Proceedings of the International Semantic Web Conference (ISWC)* (2002). See http://www.aifb.uni-karlsruhe.de/WBS/Publ/2002/2002_iswc_ontoedit.pdf.

[7] Sean Bechhofer, Ian Horrocks, Carole Goble and Robert Stevens, 2001. "OilEd: a Reasonable Ontology Editor for the Semantic Web," in *Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence*.

[8] Aditya Kalyanpur and James Hendler, 2004. "Swoop: Design and Architecture of a Web Ontology Browser/Editor," Scholarly Paper for Master's Degree in Computer Science, University of Maryland, Fall 2004. See http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.87.1779&rep=rep1&type=pdf.

[9] Holger Knublauch was formerly the designer and developer of Protégé-OWL, the leading open-source ontology editor. TopBraid Composer leverages the experiences gained with Protégé and other tools into a professional ontology editor and knowledge-base framework. Composer is based on the Eclipse platform and uses Jena as its underlying API. See further http://www.topquadrant.com/composer/tbc-protege.html.

[10] Sean Bechhofer, Phillip Lord and Raphael Volz, 2003. "Cooking the Semantic Web with the OWL API," in *Proceedings of the 2nd International Semantic Web Conference, ISWC*, Sanibel Island, Florida, October 2003. See http://homepages.cs.ncl.ac.uk/phillip.lord/download/publications/cooking03.pdf.

[11] Jena is fundamentally an RDF API. Jena's ontology support is limited to ontology formalisms built on top of RDF. Specifically this means RDFS, the varieties of OWL, and the now-obsolete DAML+OIL. At the time of writing, no decision has yet been made about when Jena will support the new OWL 2 features. See http://jena.sourceforge.net/ontology/.

[12] The NeON Toolkitis built on OntoStudio. It is based on Eclipse with support for the OWL API. A series of its key plug-ins utilize various aspects of GATE (General Architecture for Text Engineering). 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.

[13] OWL API is a Java interface and implementation for the W3C Web Ontology Language (OWL), used to represent Semantic Web ontologies. The API provides links to inferencers, managers, annotators, and validators for the OWL2 profiles of RL, QL, EL. Two recent papers describing the updated API are: Matthew Horridge and Sean Bechhofer, 2009. "The OWL API: A Java API for Working with OWL 2 Ontologies," presented at *OWLED 2009*,

6th OWL Experienced and Directions Workshop, Chantilly, Virginia, October 2009. See http://www.webont.org/owled/2009/papers/owled2009_submission_29.pdf; and, Matthew Horridge and Sean Bechhofer, 2010. "The OWL API: A Java API for OWL Ontologies," paper submitted to the *Semantic Web Journal*; see http://www.semantic-web-journal.net/sites/default/files/swj107.pdf.

[14] These links show the status of TopBraid Composer and Ontotext's OWLIM with regard to OWL 2 RL. A newer effort, based on Eclipse, with broader OWL API support is the MOST Project's TwoUse Toolkit. In all likelihood, the number of other tools with OWL 2 support is larger than our informal survey has found. Importantly, Jena still has not upgraded to OWL 2, but its open source site suggests it may.

[15] The CO-ODE project aimed to build authoring tools and infrastructure to make ontology engineering easier. It specifically supported the development and use of OWL-DL ontologies, by being heavily involved in the creation of infrastructure and plugins for the Protégé platform and OWL 2 support for the OWL API.

[16] For a great discussion on the unique aspects of version control in ontologies, see T. Redmond, M. Smith, N. Drummond and T. Tudorache, 2008. "Managing Change: An Ontology Version Control System," paper presented at *OWLED 2008*, Karslruhe, Germany. See http://bmir.stanford.edu/file_asset/index.php/1435/BMIR-2008-1366.pdf.

[17] Birte Glimm, Matthew Horridge, Bijan Parsia and Peter F. Patel-Schneider, 2009. "A Syntax for Rules in OWL 2," presented at the *Sixth OWLED Workshop*, 23-24 October 2009. See http://www.webont.org/owled/2009/papers/owled2009_submission_16.pdf.

[18] The Alignment API is one of the more venerable ones in this environment. A couple of other examples include a SKOS API (Simon Jupp, Sean Bechhofer and Robert Stevens, 2009. " A Flexible API and Editor for SKOS," presented at the 6th Annual European Semantic Web Conference (ESWC2009); see http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-401/iswc2008pd_submission_88.pdf) and the Ontology Common API Tasks (Tomasz Adamusiak, K Joeri van der Velde, Niran Abeygunawardena, Despoina Antonakaki, Helen Parkinson and Morris A. Swertz, 2010. "OntoCAT — A Simpler Way to Access Ontology Resources," presented at *ISMB2010*, 10 July 2010. See http://www.iscb.org/uploaded/css/58/17254.pdf. OntoCAT is an open source package developed to simplify the task of querying heterogeneous ontology resources. It supports NCBO BioPortal and EBI Ontology Lookup Service (OLS), as well as local OWL and OBO files).

[19] 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.
