Ontology-driven Applications Using Adaptive Ontologies

by Mike Bergman - Monday, November 23, 2009

http://www.mkbergman.com/847/ontology-driven-applications-using-adaptive-ontologies/

A Low-risk Path to the Open World, Semantic Enterprise

OK, you've been reading the literature and perhaps have attended a conference or two. You have heard a lot about semantic technologies, but have some real questions and concerns:

- How do we get started, especially with smaller proofs-of-concept?
- Do we need to abandon our past practices and systems in order to gain semantic advantages?
- To gain the advantages of interoperability, do we have to convert everything into RDF or OWL?
- Are semantic technologies limited to open or public data; how do we accommodate our proprietary information?

Such questions -- and more -- are not infrequent when organizations first contemplate making the transition to become a semantic enterprise.

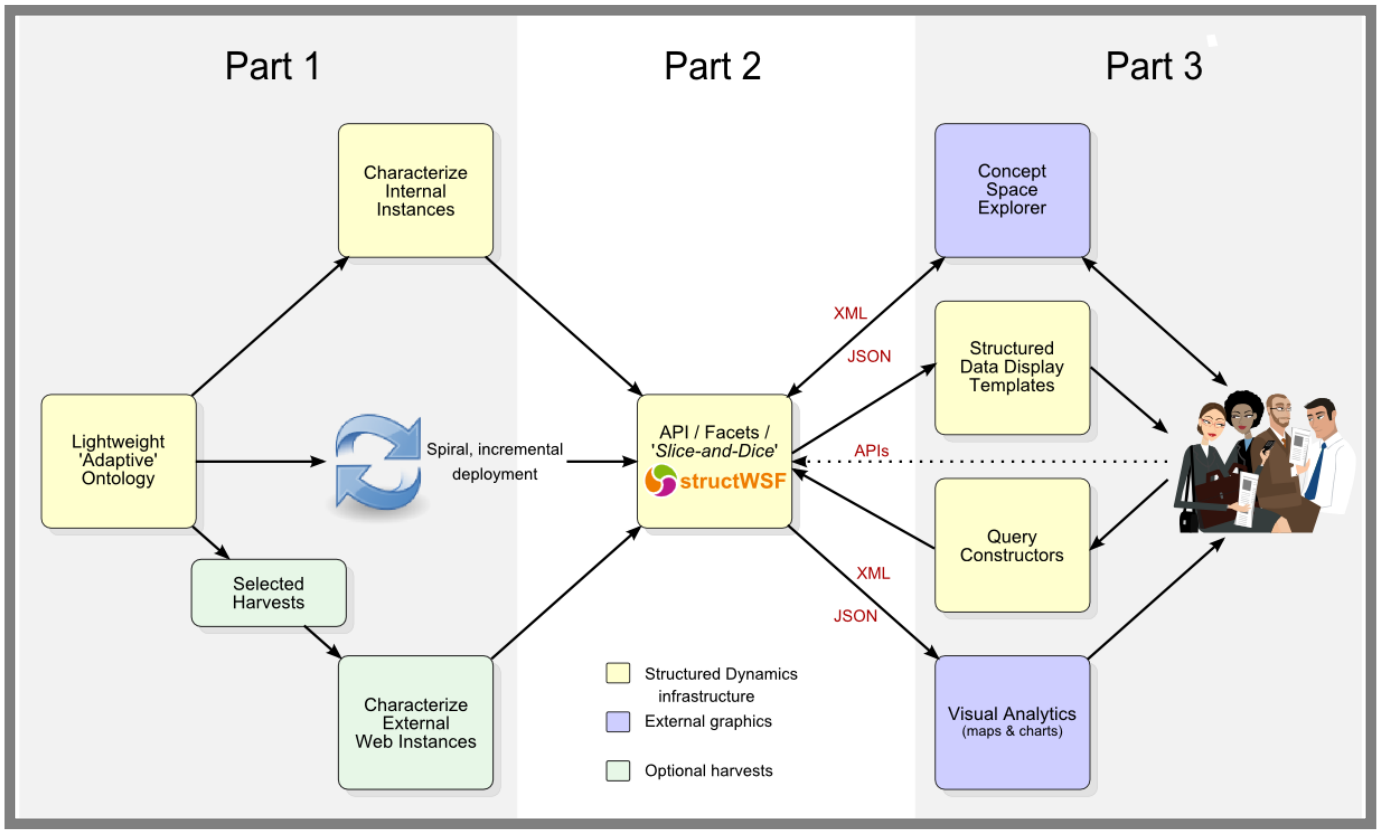
Overview

The diagram below shows a general workflow for migrating existing instance data into the semantic enterprise. The diagram is broken down into three parts. The first part is to characterize and stage existing data and information into the underlying structured data framework. This is what SD (that is, my firm, Structured Dynamics) does as data architects using our particular approach to adaptive ontologies. I'll touch on this again in a moment.

Jumping to the right-hand side of the diagram is the access and display part. It is here that developers or users can make selections from dropdown lists and so forth to define the "slices" of diced results sets they wish to display. The results of those interactions are structured data results sets that are pre-staged to "drive" various applications and displays [1,2]. These same capabilities can also be embedded into standard Web end user applications, such as content management systems.

The third and middle part of the diagram is the critical part, the pivot point. It is the interface layer between the structured data on the left and the display and presentation of that data on the right. As provided by SD, this abstraction layer is the **structWSF** Web services framework that "bridges" between the black box of what happens with RDF and semantic Web structured data characterizations on the left in order to feed, or "drive", useful services and functions on the right.

We call this general design and architecture "ontology-driven applications". The bulk of this posting explains each of these three parts in a bit more detail, organized from left-to-right by these Parts 1 to 3.



(click to expand)

Part 1: Structured Data Instances and Ontologies

Our approach relies on what we call "adaptive ontologies". These ontologies set the structural basis for all subsequent data display, analysis, inferencing, entailments, and the like. We call them "adaptive" because we embrace a set of unique best practices. These practices enable the ontologies to do the double-duty of first structuring data and then driving generic applications by properly informing user interfaces, dropdown lists, menus and the like.

This structuring results in faceting key important dimensions and attributes of available content. Structured data gets organized. Unstructured data (text) gets tagged via this structure and integrated with it.

As Structured Dynamics' general product schema makes clear (see the diagram at [3]), our approach leverages existing assets as much as possible. Often, this means leaving most existing data structures in place. These existing assets are staged and converted in two complementary manners that largely correspond to the conceptual *ABox* (instance) and *TBox* (concepts and schema) split central to description logics and pivotal to SD's methodology [4].

Whether transitioning small chunks or big chunks, this staging of existing data in Part 1 results in an RDF-

accessible characterization of the starting content. Instances and their attributes are represented via a common notation, generally based on **irON** (*instance record* and *Object Notation*) [5], that is an extensible notation and vocabulary for capturing the data characterizations, attributes and metadata of the candidate instance data ("records" in RDBMS parlance). These instances may either be internal or proprietary records, or instance data on the Web or in the public domain. By properly matching same or similar instances to one another, any source of instance characterization can be meaningfully combined.

This instance notation is extremely lightweight, and really is merely an RDF representation of data characterizations. In the characterizations to this point there is not yet any "world view" involved: we are simply describing instances and their attributes in a manner akin to key-value pairs. The process to this point is entirely descriptive.

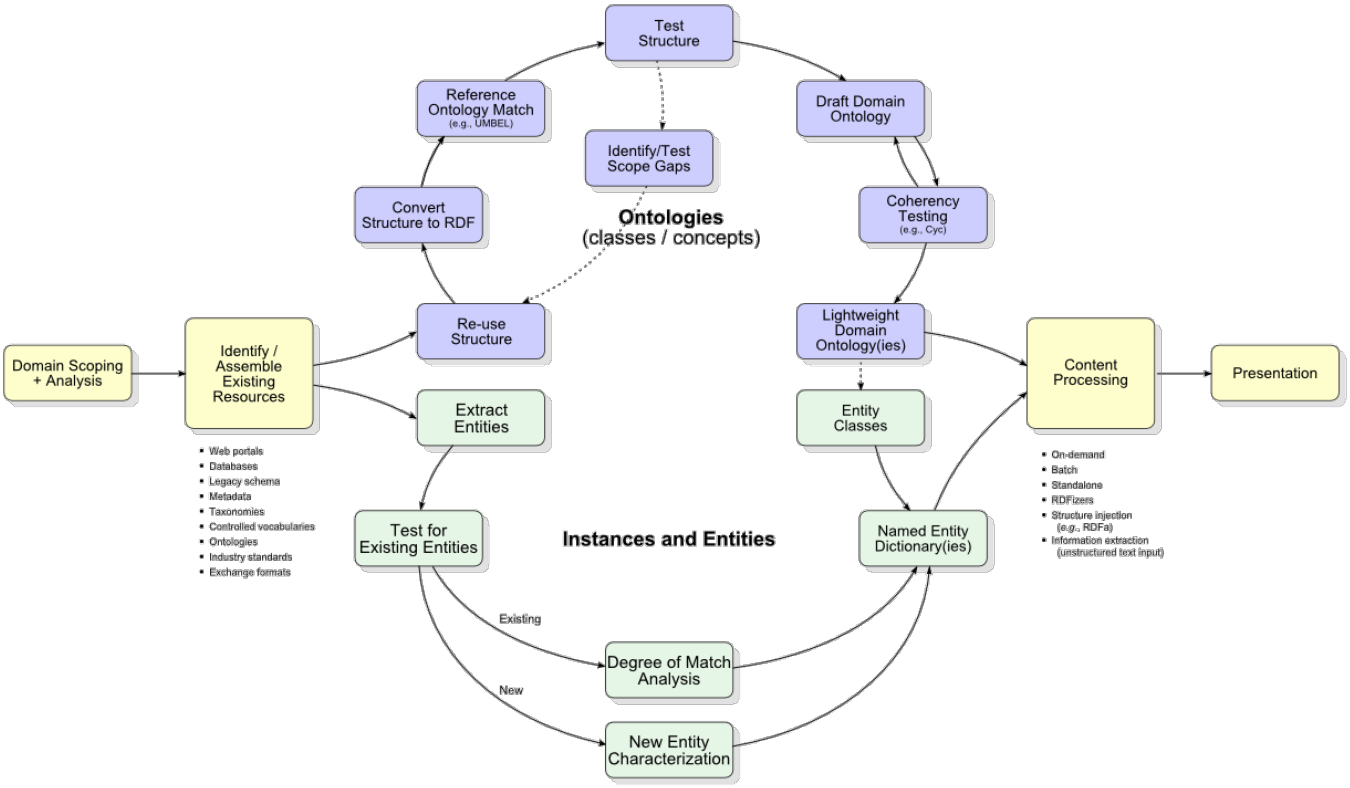
However, these instance characteristics do contain within them the semantics as to how to describe these attributes (your "glad" is my "happy"), as well as potentially a schematic or conceptual view of how these instances relate to one another and to the broader world. Instance characterizations provide the building blocks, that are then related and made semantically whole via a second "terminological" level.

These terminological, or conceptual, relationships (the *TBox* [4]), reside at a different level from simply decribing things. Rather, these schema -- what in this context are best known as *ontologies* -- provide a precise language and means for describing conceptual relationships. If these structural relationships are done well, they are coherent: the hip bone is connected to the thigh bone and not to the ear. Coherence is a matter of a consistent world view that "hangs together" when analyzed via powerful logical techniques available via description logics and other broader mechanisms of the semantic enterprise.

Thus, as we transition from the existing, the operational workflow splits the input data stream into two pathways:

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

A sequential flow of these steps and splits is provided by this diagram below that shows: 1) the conceptual structure of the concept ontology; as 2) matched with the instances and their descriptive attributes that populate that schema.



(click to expand)

A key point is that -- while a proper starting ontology is essential to our process and proofs-of-concept -- it can be grown and scaled incrementally. We leverage as much existing starting structure as possible and can readily bound the scope to meet budget and delivery imperatives.

The concepts and entities that occur within these structures help inform our fairly simple tagging system, scones [3]. (There are also benefits from "triangulating" between entity or instance identification and concept identification that helps inform disambiguation nearly for free; see further [6]). It is also possible to integrate these initial proof-of-concept approaches with third-party tools (*e.g.*, Calais, Expert System (Cogito), etc.) to improve unstructured content characterization.

These approaches are pretty straightforward for any organization wanting to test the idea of becoming a semantic enterprise. Real benefits -- such as concept retrievals overcoming the limitations of standard keyword search -- can be demonstrated from even small starting ontologies and structures. Given the inherent connectedness of the data, it is possible to expand the scope and usefulness of the information incrementally within fixed and manageable budgets.

Part 2: structWSF: A Web-oriented Services API and Framework

A pivotal part of SD's infrastructure software is **structWSF** [7], our platform-independent Web services middleware. **structWSF** is an abstraction layer that provides the APIs, search endpoints, and specific Web services for accessing, querying or getting results sets from the underlying structured data and ontologies.

structWSF has a standard set of access and retrieval services including browse, full-text search, CRUD, direct record retrievals, and the like. It is embedded within an access and permissions service that acts at the level of registered datasets. Then, based on the requested protocol, **structWSF** returns the filtered results set. These results sets can be delivered as XML, JSON, or any of the other formats already available [7]. They can readily and dynamically populate HTML pages and forms in any deployment framework. For specific purposes, these results sets can also be returned as pre-staged, properly formatted results streams for driving specific applications.

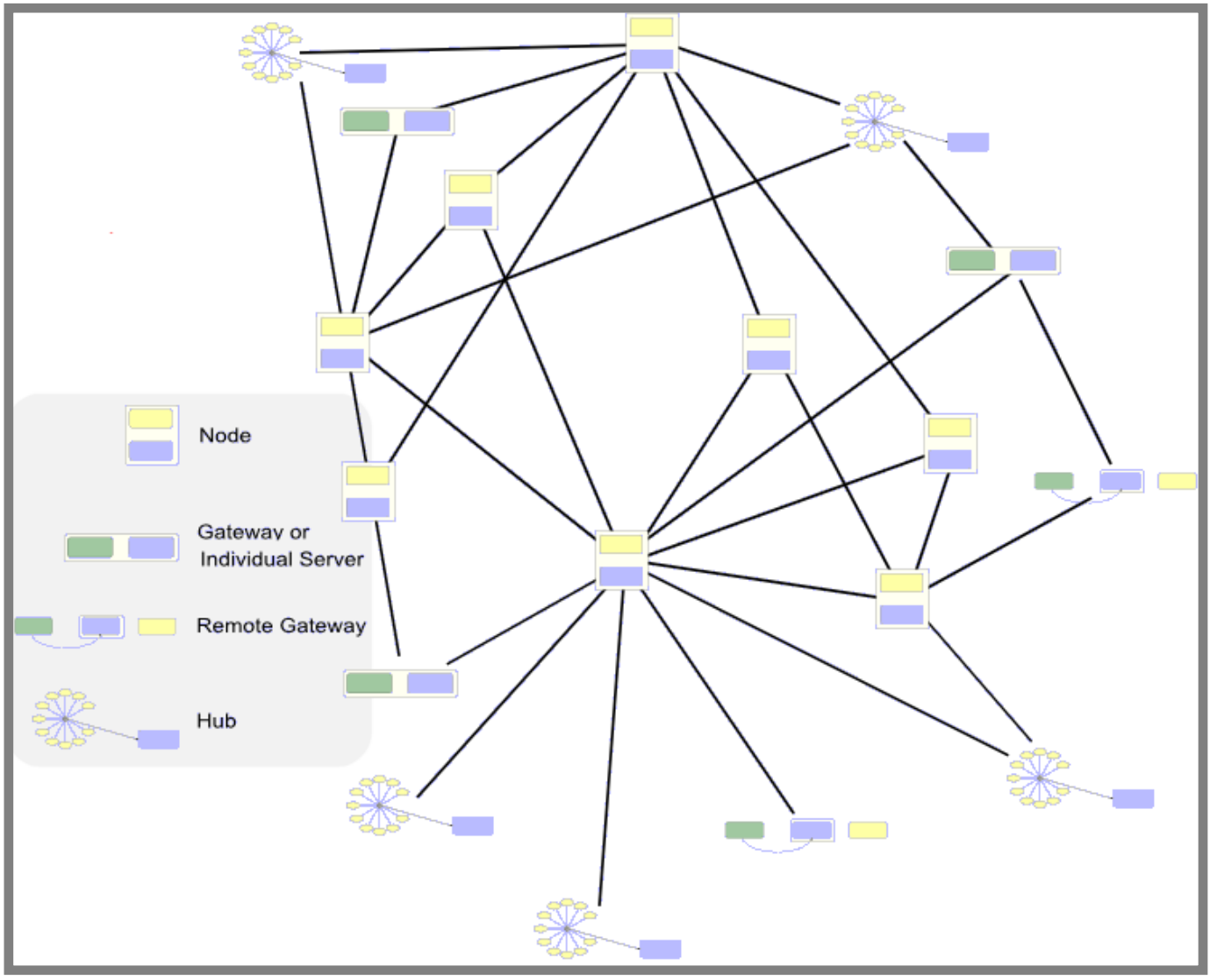
As an API, the **structWSF** Web services can be interacted with and driven via standard HTTP requests. Alternatively, these requests can come from simple to complicated Web apps that create the API queries based on user interface choices such as selections from dropdown lists or clicking on various listed options. An interactive demo of this approach is shown by SD's **conStruct** application [8], though even simpler Web pages or forms may drive the query interface.

Queries and requests to **structWSF** may also include a parameter for results sets to be returned in particular formats. SD's **irON** protocol [5] supports requests or results in CSV, XML or JSON, in addition to other flavors including multiple serializations of RDF.

In this manner, only a simple converter need be added to the **structWSF** Web services stack in order to "drive" a new application with a particularly formatted results set stream.

structWSF thus acts as a single, uniform Web interface to all of the "black box" nuances of the structured data system organized by the adaptive ontologies. Further, virtually any data structure may be ingested and converted from external sources via an import service and made part of the underlying canonical structure, making the framework perfect for data federation [9]. Lastly, the dataset nature of the framework, and its neutrality to underlying data stores or content management systems, also makes **structWSF** an excellent framework for one or many nodes to share information and collaborate across the Web [10].

The following diagram shows how a diverse, Web-based network, involving a diversity of Web portals and data gateways and hubs, can work via the **structWSF** framework to establish a complete collaboration network. Via datasets and differential access rights and permissions, virtually any combination of potential interactions can be supported:



(click to expand)

These potentials are really fundamentally new, and we ourselves are still trying to find the language and analogies to best explain them. **structWSF** was initially designed as a platform-independent layer between the structured data representation of existing assets and the ontology-driven applications that interact with them. We are now finding that deployment in a broader Web-based context provides additional exciting prospects for integrating various regional offices or to enable direct collaboration with customers, partners or suppliers.

Part 3: Ontology-driven Applications

The basic design of **structWSF** is to provide a middleware layer that fulfills one or more of these broad user interaction modes:

- 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 a workflow of some nature, involving specialized analysis, applications, or visualization.

SD has developed generic applications in these areas (with many more possible), the operations of which are guided by the instructions and nature of the underlying data that feeds them. We have proven it is possible to adopt data characterization practices within those ontologies so as to stage or "drive" such generic applications.

In the case of a standard structured data display (say, a simple table like a Wikipedia infobox, for example), 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 some specialized application (say, Adobe Flash), the information output of the results set may need to contain certain formats and attributes.

SD's "ontology-driven apps", then, are really informed structured results sets that are outputted in a form suitable to various intended applications. This output form can include a variety of serializations, formats or metadata. This flexibility of output that is tailored to and responsive to particular generic applications is what makes our ontologies "adaptive".

Expressed in this manner, "ontology-driven apps" seem neither remarkably profound nor clever. They are simply attentive to their intended uses.

Using this structure, then, it is possible to either "drive" queries and results sets selections via direct HTTP request or via simple dropdown selections on HTML forms (that is, from *right* to *left* as shown on the first diagram). 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 (that is, from *left* to *right* on the first diagram).

"Ontology-driven apps" through SD's 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. If, say, Adobe graphics applications need to change tomorrow for Microsoft Silverlight, that switch is easy and can be made transparent to the designer.

The Complete Picture: Embrace the Open World

The ability to develop these systems incrementally and the ability to integrate with external, public data is fundamentally dependent on the open world assumption. The open world assumption is a different logic premise than what many enterprises are used to; relational database systems, for example, embrace the alternate closed world premise.

Open world does not necessarily mean open data and it does not mean open source. Open world is merely a way to think about the information we have and how we act on it. An open world assumption accepts that we never have all necessary information and lacking that information does not itself lead to any conclusions.

Some enterprise circumstances -- say a complete enumeration of customers or products or even controlled engineering or design environments -- may warrant a closed world approach. In those circumstances, the domain of inquiry is well bounded and we can get relatively complete information about it. Engineering an oil drilling platform or launching the Space Shuttle in fact demands that.

But, in most real world circumstances, there is much we don't know and we interact in complex and external environments. Open world is the proper logic premise for these circumstances. These circumstances also happen to be the very basis in which most most knowledge workers and analysts reside.

Open world frameworks provide some incredibly important benefits if the circumstances of their use apply:

- Domains can be analyzed and inspected incrementally
- Schema can be incomplete and developed and refined incrementally
- The data and the structures within these open world frameworks can be used and expressed in a piecemeal or incomplete manner
- We can readily combine data with partial characterizations with other data having complete characterizations
- Systems built with open world frameworks are flexible and robust; as new information or structure is gained, it can be incorporated without negating the information already resident, and
- Open world systems can readily bridge or embrace closed world subsystems.

One might argue, as we believe, that the biggest impediment to the semantic enterprise is the mind shift necessary to start thinking about and accepting the open world premise. Again, this perspective is not applicable to all problems and domains. But, where it is, much can be left in place and leveraged with semantic technologies, so long as the enterprise begins to look at these existing assets through a different open-world lens.

Summary

So, let's return to the rhetorical questions that began this posting.

It should now be clear that it is possible to start small in testing the transition to a semantic enterprise. These efforts can be done incrementally and with focus on early, high-value applications and domains.

Further, we need not abandon past practices. There is much that can be done to leverage existing assets. Indeed, those prior investments are often the requisite starting basis to inform semantic initiatives. However, in leveraging those assets, it is important that the enterprise begin to embrace and understand the open world assumption.

We also see that RDF and OWL, while important behind the scenes as a canonical data model and languages for organizing this information, need not be exposed as such to most users. Most instance data can be expressed as is with the data languages of choice such as XML, JSON or whatever.

We also see these technologies are neutral to the question of open or public sources. The techniques can

equivalently be applied to internal, closed, proprietary data and structures. Moreover, the technologies can themselves be used as a basis for bringing external information into the enterprise.

Without a doubt, some of the early years in describing semantic technologies were burdened with some unfortunate bad information and lack of sophistication. Today's semantic Web is nimble, agile, and ready to be deployed immediately at low cost and risk. So, jump on in! We think you'll find the water to be just fine.

This post is Part V of an occasional **AI3** series on ontology best practices.

---

[1] These selections and requests need not occur *only* via user interfaces or HTML forms, but also programmatically via API or direct Web services calls.

[2] There are two main classes of visualizations possible with our systems: 1) navigations or explorers of the concept space, which is a particularly open challenge for large, graph-based knowledge bases (see, for example, our Subject Concept Explorer using the UMBEL Financial Account concept, and click on the bubbles); or 2) conventional data visualizations or graphics or mappings of instance data. Both are shown as workflow boxes on the first diagram above.

[3] See http://structureddynamics.com/products.html for a general descriptive illustration of Structured Dynamics' product stack. There is also a longer slideshow, from which this diagram is drawn as slide #37.

[4] We use the reference to ABox and TBox in accordance with our working definition for description logics:

"Description logics and their semantics traditionally split *concepts* and their relationships from the different treatment of *instances* and their attributes and roles, expressed as fact assertions. The concept split is known as the TBox (for *terminological* knowledge, the basis for *T* in *TBox*) and represents the schema or taxonomy of the domain at hand. The TBox is the structural and intensional component of conceptual relationships. The second split of instances is known as the ABox (for *assertions*, the basis for *A* in *ABox*) and describes the attributes of instances (and individuals), the roles between instances, and other assertions about instances regarding their class membership with the TBox concepts."

[5] For the specification and a use case of **irON** using the CSV (**commON**) serialization, see http://openstructs.org/iron.

[6] Via this approach we now can assess concept matches in addition to entity matches. This means we can triangulate between the two assessments to aid disambiguation. Because of these logical segmentations, we also have multiple “clusters” (that is, either the *concept*, *type*, *superType* or *dimension*) upon which to do our disambiguation evaluations, either between concepts and entities or within the various concept clusters. We can do so via either multiple semantic vectors (for statistical-based methods) or multiple features (for machine learning methods). In other words, because of logical segmentation, we have increased the informational power of our concept graph. See further http://www.mkbergman.com/759/supertypes-and-logical-segmentation-of-instances/.

[7] See http://openstructs.org/structwsf/architecture; also, the available export formats are shown at http://constructscs.com/documentation/instructions/export.

[8] There is an online demo of **conStruct** using the Sweet Tools database of semantic Web and -related tools at http://constructscs.com/conStruct/browse/; for background on this use case, see http://www.mkbergman.com/845/a-most-un-common-way-to-author-datasets/.

[9] See, for example, http://www.mkbergman.com/496/structwsf-a-framework-for-data-mixing/.

[10] See, for example, http://www.mkbergman.com/497/structwsf-a-framework-for-collaboration-networks/.
