Ontology-Driven Apps Using Generic Applications

by Mike Bergman - Monday, March 07, 2011

http://www.mkbergman.com/948/ontology-driven-apps-using-generic-applications/

The Time and Technology is Here to Stand Software Engineering on its Head

As an information society we have become a software society. Software is everywhere, from our phones and our desktops, to our cars, homes and every location in between. The amount of software used worldwide is unknowable; we do not even have agreed measures to quantify its extent or value [1]. We suspect there are at least 1 billion lines of code that have accumulated over time [1,2]. On the order of $875 billion was spent worldwide on software in 2010, of which about half was for packaged software and licenses and the rest for programmer services, consulting and outsourcing [3]. In the U.S. alone, about 2 million people work as programmers or related [4].

It goes without saying that software is a very big deal.

No matter what the metrics, it is expensive to develop and maintain software. This is also true for open source, which has its own costs of ownership [5]. Designing software faster with fewer mistakes and more re-use and robustness have clearly been emphases in computer science and the discipline of programming from its inception.

This attention has caused a myriad of schools and practices to develop over time. Some of the earlier efforts included computer-aided software engineering (CASE) or Grady Booch's (already cited in [1]) object-oriented design (OOD). Fourth-generation languages (4GLs) and rapid application development

(RAD) were popular in the 1980s and 1990s. Most recently, agile software development or extreme programming have grabbed mindshare.

Altogether, there are dozens of software development philosophies, each with its passionate advocates. These express themselves through a variety of software development methodologies that might be characterized or clustered into the prototyping or waterfall or spiral camps.

In all instances, of course, the drivers and motivations are the same: faster development, more re-use, greater robustness, easier maintainability, and lower development costs and total costs of ownership.

The Ontology Perspective in this Mix

For at least the past decade, ontologies and semantic Web-related approaches have also been part of this mix. A good summary of these efforts comes from Michael Uschold in an invited address at FOIS 2008 [6]. In this review, he points to these advantages for ontology-based approaches to software engineering:

- Re-use -- abstract/general notions can be used to instantiate more concrete/specific notions, allowing more reuse
- Reduced development times -- producing software artifacts that are closer to how we think, combined with reuse and automation that enables applications to be developed more quickly
- Increased reliability -- formal constructs with automation reduces human error
- Decreased maintenance costs -- increased reliability and the use of automation to convert models to executable code reduces errors. A formal link between the models and the code makes software easier to comprehend and thus maintain.

These first four items are similar to the benefits argued for other software engineering methodologies, though with some unique twists due to the semantic basis. However, Uschold also goes on to suggest benefits for ontology-based approaches not claimed by other methodologies:

- Reduced conceptual gap -- application developers can interact with the tools in a way that is closer to their thinking
- Facilitate automation -- formal structures are amenable to automated reasoning, reducing the load on the human, and
- Agility/flexibility -- ontology-driven information systems are more flexible, because you can much more easily and reliably make changes in the model than in code.

In making these arguments, Uschold picks up on the "ontology-driven information systems" moniker first put forward by Nicola Guarino in 1998 [7]. The ideas around ODIS have had substantial impact on the semantic Web community, especially in the use of formal ontologies and modeling approaches. The FOIS series of conferences, and most recently the ODiSE series, have been spawned from these ideas. There is also, for example, a fairly rich and developed community working on the integration of UML via ontologies as the drivers or specifiers of software [8].

Yet, as Uschold is careful to point out, the idea of ODIS extends beyond software engineering to encompass all of information systems. My own categorization of how ontologies may contribute to information systems is:

1. Domain modeling -- this category includes the domain knowledge representations and reasoning and inference bases that are the traditional understanding of ontologies in the semantic space. The structural aspects are akin to a database schema definition; the unique aspects of ontologies reside in their logic foundations and graph structures, which offer more power in inferencing, reasoning and graph analysis than conventional approaches
2. Model-driven architectures (MDA) -- like UML, these are platform-independent specifications that provide the functional and dataflow definitions of "models" executed by the system. These are the natural progeny of earlier CASE approaches, for example. Such systems also potentially allow graphical or visual means for building or hooking together components as a substitute to direct coding
3. Program specifications and excecutables -- though fairly experimental at present, these approaches use the languages of RDF, OWL or direct use of logic languages to create the equivalent of executable software programs. A couple of experimental systems include Fhat and Neno, for example, point to possible future directions in this area [9]
4. Runtime or utility components -- proper construction of ontologies can be a source for labels and prompts within user interfaces and other runtime uses. Because of the ontology basis, these contributions may also be contextual [10]
5. Automated agents -- based on context, user choices and the governing ontologies, new instruction sets can be generated via what some term automated agents or "robots" to instruct subsequent steps in the software, including potentially analysis or validation. Mission Critical IT [11] is apparently the most advanced in this area; we discuss their ODASE approach more below
6. Bespoke drivers of generic applications -- through using and combining a number of the aspects above, in its totality this approach is a very different paradigm, as we describe below.

When we look at this list from the standpoint of conventional software or software engineering, we see that #1 shares overlaps with conventional database roles and #2, #3 and #4 with conventional programmer or software engineering responsibilities. The other portions, however, are quite unique to ontology-based approaches.

But Is Software Engineering Even the Right Focus?

For decades, issues related to how to develop apps better and faster have been proposed and argued about. We still have the same litany of challenges and issues from expense to re-use and brittleness. And, unfortunately, despite many methodologies *du jour*, we still see bottlenecks in the enterprise relating to such matters as:

Software is merely an intermediary artifact to accomplish some given tasks. Rather than "engineering" software, the focus should be on how to fulfill those tasks in an optimal manner -- and that demands a systems approach.

- data access
- queries
- data transformations
- data integration or federation

- reports
- other data presentations
- business analysis, and
- targeted, specialty functionality.

Promises such as self-service reporting touted at the inception of data warehousing two decades ago are still to be realized [12]. Enterprises still require the overhead and layers of IT to write SQL for us and prepare and fix reports. If we stand back a bit, perhaps we can come to see that the real opportunity resides in turning the whole paradigm of software engineering upside down.

Our objective should not be software *per se*. Software is merely an intermediary artifact to accomplish some given task. Rather than engineering software, the focus should be on how to fulfill those tasks in an optimal manner. How can we keep the idea of producing software from becoming this generation's new buggy whip example [13]?

For reasons we delve into a bit more below, it perhaps has required a confluence of some new semantic technologies and ontologies to create the opening for a shift in perspective. That shift is one from software as an objective in itself to one of software as merely a generic intermediary in an information task pipeline.

Though this shift may not apply (at least with current technologies) to transactional and process-based software, I submit it may be fundamental to the broad category of knowledge management. KM includes such applications as business intelligence, data warehousing, data integration and federation, enterprise information integration and management, competitive intelligence, knowledge representation, and so forth. These are the real areas where integration and reports and queries and analysis remain frustrating bottlenecks for knowledge workers. And, interestingly, these are also the same areas most amenable to embracing an open world (OWA) mindset [14].

If we stand back and take a systems perspective to the question of fulfilling functional KM tasks, we see that the questions are both broader and narrower than software engineering alone. They are broader because this systems perspective embraces architecture, data, structures and generic designs. The questions are narrower because software -- within this broader context -- can be now be generalized as artifacts providing the fulfillment of classes of functions.

ODapps: The Ontology-Driven Application Approach

Ontology-driven applications -- or *ODapps* for short -- based on *adaptive ontologies* are a topic we have been nibbling around and discussing for some time. In our oft-cited seven pillars of the semantic enterprise we devote two pillars specifically (#4 and #3, respectively) to these two components [15]. However, in keeping with the systems perspective relevant to a transition from software engineering to generic apps, we should also note that canonical data models (via RDF) and a Web-

oriented architecture are two additional pillars in the vision.

ODapps are modular, generic software applications designed to operate in accordance with the specifications contained in one or more ontologies. The relationships and structure of the information driving these applications are based on the standard functions and roles of ontologies (namely as domain ontologies as noted under #1 above), as supplemented by the UI and instruction sets and validations and rules (as noted under #4 and #5 above). The combination of these specifications as provided by both properly constructed domain ontologies and supplementary utility ontologies is what we collectively term *adaptive ontologies* [16].

ODapps fulfill specific generic tasks, consistent with their bespoke design (#6 above) to respond to adaptive ontologies. Examples of current ontology-driven apps include imports and exports in various formats, dataset creation and management, data record creation and management, reporting, browsing, searching, data visualization and manipulation (through libraries of what we call *semantic components*), user access rights and permissions, and similar. These applications provide their specific functionality in response to the specifications in the ontologies fed to them.

ODapps are designed more similarly to widgets or API-based frameworks than to the dedicated software of the past, though the dedicated functionality (*e.g.*, graphing, reporting, etc.) is obviously quite similar. The major change in these ontology-driven apps is to accommodate a relatively common abstraction layer that responds to the structure and conventions of the guiding ontologies. The major advantage is that single generic applications can supply shared functionality based on any properly constructed adaptive ontology.

In fact, the widget idea from Web 2.0 is a key precursor to the ODapps design. What we see in Web 2.0 are dedicated single-purpose widgets that perform a display operation (such as Google Maps) based on the properly structured data fed to them (structured geolocational information in the case of GMaps).

In Structured Dynamics' early work with RDF-based applications by our predecessor company, Zitgist, we demonstrated how the basic Web 2.0 widget idea could be extended by "triggering" which kind of mashup widget got invoked by virtue of the data type(s) fed to it. The Query Builder presented contextual choices for how to build a SPARQL query via UI based on what prior dropdown list choices were made. The DataViewer displayed results with different widgets (maps, profiles, etc.) depending on which part of a query's results set was inspected (by responding to differences in data types). These two apps, in our opinion, remain some of the best developed in the semantic Web space, even though development on both ceased nearly four years ago.

This basic extension of data-driven applications -- as informed by a bit more structure -- naturally evolved into a full ontology-driven design. We discovered that -- with some minor best practice additions to conventional ontologies -- we could turn ontologies into powerhouses that informed applications through:

- An understanding of the kind of things under consideration, including their inference chains
- The types of data in results sets, and how that informs the nature of the widget(s) (maps, calendars, timelines, charts, tabular reports, images, stories, media, etc.) appropriate to display and manipulate that information, and
- UI and utility functions such as interface labels, mouseovers, auto-suggests, spelling suggestions,

synonym matches, etc.

Like the earlier Zitgist discoveries, basing the applications on only one or two canonical data models and serializations (RDF and a simple data exchange XML, which Fred Giasson calls structXML) provides the input uniformity to make a library of generic applications tractable. And, embedding the entire framework in a Web-oriented architecture means it can be distributed and deployed anywhere accessible by HTTP.

Booch has maintained for years that in software design abstraction is good, but not if too abstract [1]. ODapps are a balanced abstraction within the framework of canonical architectures, data models and data structures. This design thus limits software brittleness and maximizes software re-use. Moreover, it shifts the locus of effort from software development and maintenance to the creation and modification of knowledge structures. The KM emphasis can shift from programming and software to logic and terminology [16].

In the sub-sections below, we peel back some portions of this layered design to unveil how some of these major pieces interact.

Built Upon an Ontology- and Web-based Architecture

Again, to cite Booch, the most fundamental software design decision is architecture [1]. In the case of Structured Dynamics and its support for ODapps, its open semantic framework (OSF) is embedded in a Web-oriented architecture (WOA). The OSF itself is a layered design that proceeds from a kernel of existing assets (data and structures) and proceeds through conversion to Web service access, and then ontology organization and management via ODapps [17]. The major layers in the OSF stack are:

- Existing assets — any and all existing information and data assets, ranging from unstructured to structured. Preserving and leveraging those assets is a key premise
- scones / irON - the conversion layer, in part consisting of information extraction of subject concepts or named entities (scones) or the instance record Object Notation for conveying XML, JSON or spreadsheets (CSV) in RDF-ready form (via irON or RDFizers)
- structWSF - a platform-independent suite of more than 20 RESTful Web services, organized for managing structured data datasets; it provides the standard, common interface by which existing information assets get represented and presented to the outside world and to other layers in the OSF stack
- Ontologies — are the layer containing the structured assets “driving” the system; this includes the concepts and relationships of the domain at hand, and administrative ontologies that guide how the user interfaces or widgets in the system should behave
- conStruct - connecting modules to enable structWSF and sComponents to be hosted/embedded in Drupal, and
- sComponents - (mostly) Flex semantic components (widgets) for visualizing and manipulating structured data.

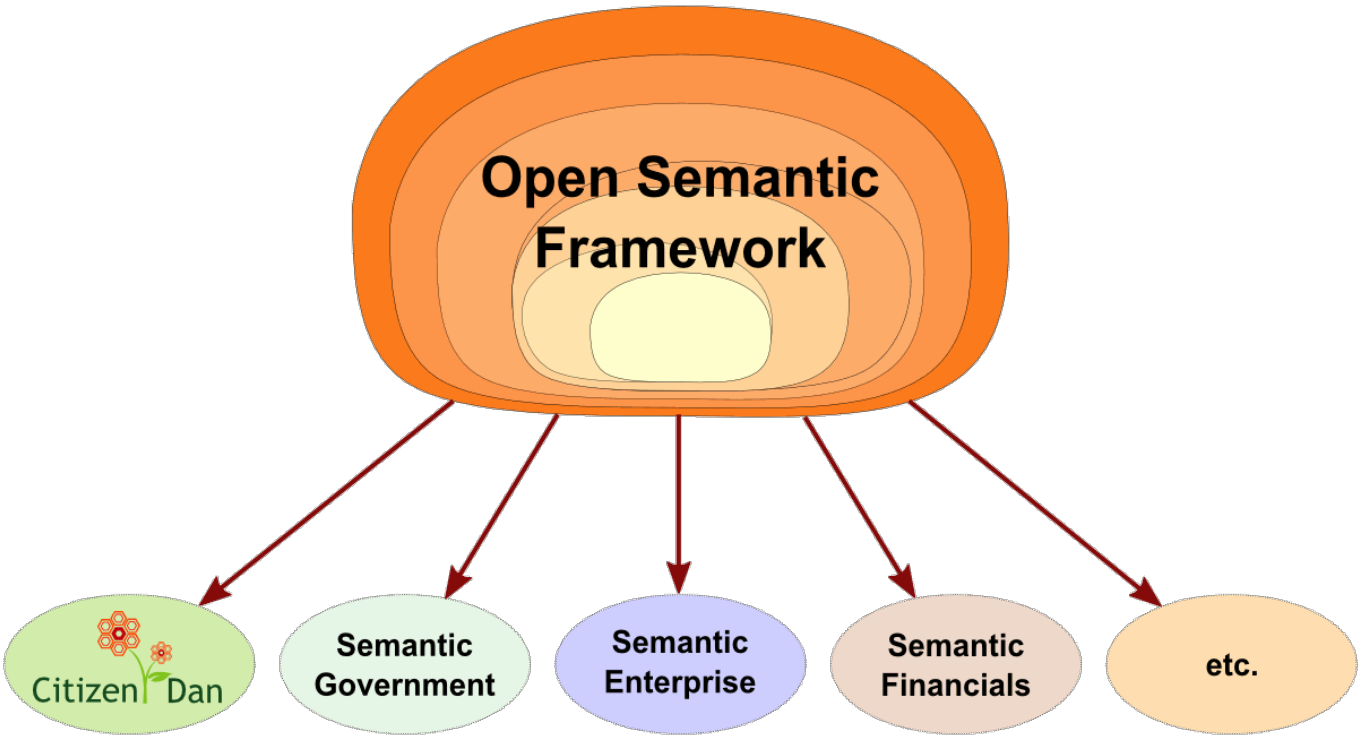
Not all of these layers or even their specifics is necessary for an ontology-driven app design [18]. However, the general foundations of generic apps, properly constructed adaptive ontologies, and canonical data models and structures should be preserved in order to operationalize ODapps in other settings.

OSF is the Basis for Domain-specific Instantiations

The power of this design is that by swapping out adaptive ontologies and relevant data, the entire OSF stack as is can be used to deploy multiple instantiations. Potential uses can be as varied as the domain coverage of the domain ontologies that drive this framework.

The OSF semantic framework is a completely open and generic one. The same set of tools and capabilities can be applied to any domain that needs to manage and understand information in its own domain. With the existing ODApps in hand, this includes from unstructured text or documents to conventional structured databases.

What changes from domain to domain are the data structures (the ontologies, schema and entity references) and their instance data (which can also be converted from existing to canonical forms). Here is an illustration of how this generic framework can be leveraged for different deployments. Note that Citizen Dan is a local government example of the OSF framework with relatively complete online demos:



(click for full size)

Structured Dynamics continues to wrinkle this basic design for different clients and different industries. As we round out the starting set of ODapps (see below), the major effort in adapting this generic design to different uses is to tailor the ontologies and "RDFize" existing data assets.

Lower Layers

Conversion of existing assets to RDF and canonical forms is not discussed further here. See the irON and scones documentation or the TechWiki for more information on these topics.

The structWSF Web Services Layer

The first suite of ODapps occurs at the structWSF Web services layer. structWSF provides a set of generic functions and endpoints to:

- Import or export datasets
- Create, update, delete (CRUD) or otherwise manage data records
- Search records with full-text and faceted search
- Browse or view existing records or record sets, based on simple to possible complex selection or filtering criteria, or
- Process results sets through workflows of various natures, involving specialized analysis, information extraction or other functions.

Here is a listing of current ODapp functions within structWSF (with links to details for each):

WSF management Web services	User-oriented Web services
• Auth: Validator • Auth: Lister • Auth Registrar: Access • Auth Registrar: WS • Ontology: Create • Dataset: Create • Dataset: Read • Dataset: Update • Dataset: Delete	• CRUD: Create • CRUD: Read • CRUD: Update • CRUD: Delete • Browse • Search • Scones • SPARQL • Import • Export

At this level the information access and processing is done largely on the basis of structured results sets. Other visualization and display ODapps are listed in the next subsection.

The Semantics Components Layer

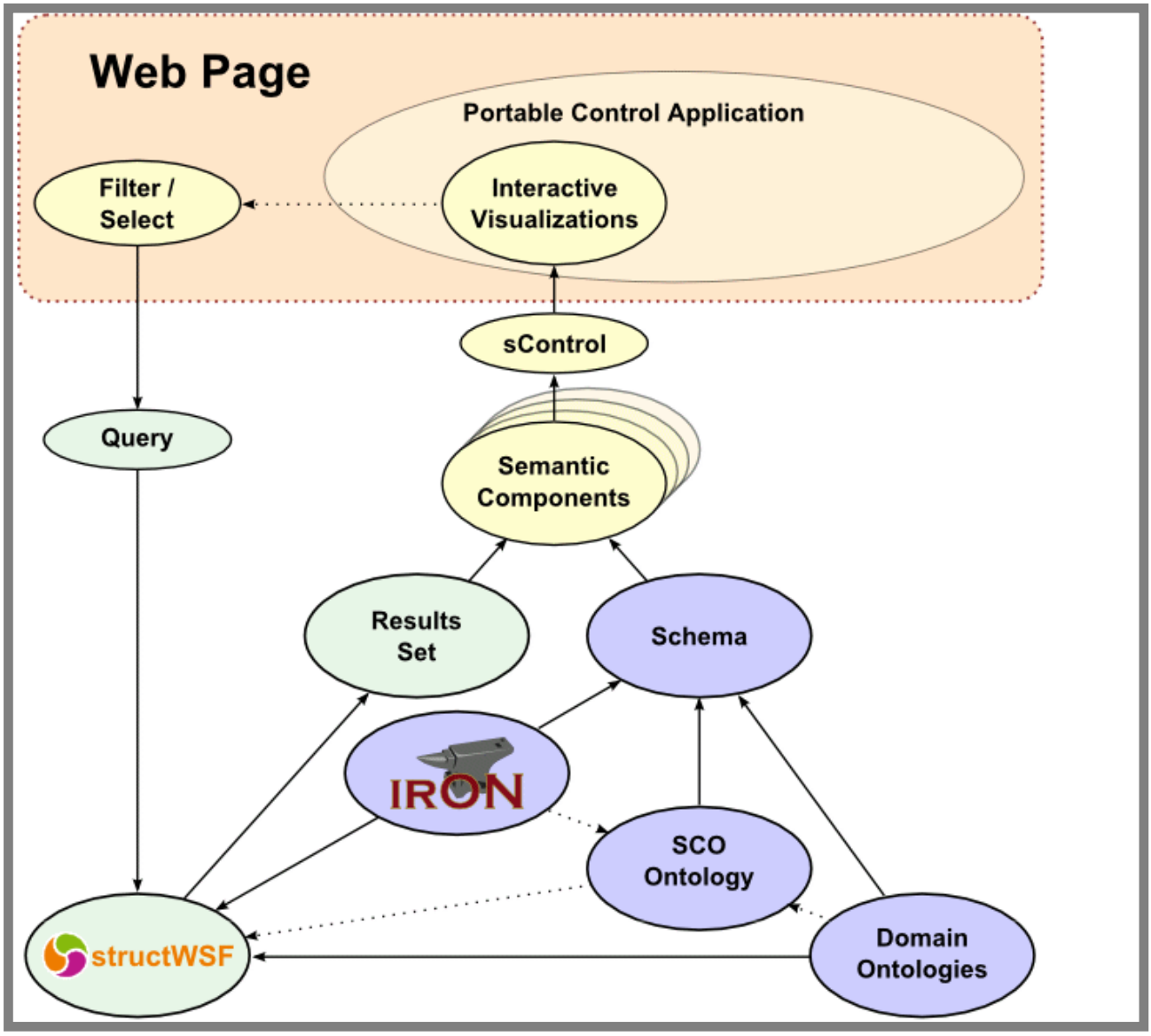
The visualization and data display and manipulation ODapps are provided via the semantic components layer. Structured Dynamics's sComponents are Flex-based widgets that conform to a standard, generic design. Other developers using the OSF framework are developing JavaScript versions [19]. Here is the current library (with links to details for each):

New Components	Components Extending Flex
• Portable Control Application • sBarChart • SCO Ontology • sControl • sDashboard • sGenericBox • sLinearChart • sMap • sPieChart	• sHBox • sImage • sText

• sRelationBrowser • sStory • scones: Story Tagging • sWebMap (in development)	

These components can be used in combination with any of the structWSF ODapps, meaning the filtering, searching, browsing, import/export, etc., may be combined as an input or output option with the above.

The next animated figure shows how the basic interaction flow works with these components:



(click for full size)

Using the ODapp structure it is possible to either “drive” queries and results sets selections via direct HTTP request via endpoints (not shown) or via simple dropdown selections on HTML forms or Flex widgets (shown). This design enables the entire system to be driven via simple selections or interactions

without the need for any programming or technical expertise.

As the diagram shows, these various sComponents get embedded in a layout canvas for the Web page. By interacting with the various components, new queries are generated (most often as SPARQL queries) to the various structWSF Web services endpoints. The result of these requests is to generate a structured results set, which includes various types and attributes.

An internal ontology that embodies the desired behavior and display options (SCO, the Semantic Component Ontology) is matched with these types and attributes to generate the formal instructions to the sComponents. When combined with the results set data, and attribute information in the irON ontology, plus the domain understanding in the domain ontology, a synthetic schema is constructed that instructs what the interface may do next. Here is an example schema:

```
<geoname_wikipediaArticle>
   <prefLabel>wikipedia article</prefLabel>
   <description>Wikipedia article web page</description>
   <allowedType>Neighborhood</allowedType>
   <allowedType>City</allowedType>
   <allowedType>Province</allowedType>
   <allowedType>Country</allowedType>
   <allowedValue>
       <primitive>String</primitive>
   </allowedValue>
</geoname_wikipediaArticle>

<muni_population_year_2006>
   <prefLabel>population in 2006</prefLabel>
   <description>Population of a geographical region in 2006</description>
   <allowedType>Neighborhood</allowedType>
   <allowedType>City</allowedType>
   <allowedType>Province</allowedType>
   <allowedType>Country</allowedType>
   <allowedValue>
       <primitive>Integer</primitive>
   </allowedValue>
   <maxValues>1</maxValues>
   <orderingValue>2006</orderingValue>
   <comparableWith>muni_population_year_1971</comparableWith>
   <comparableWith>muni_population_year_1976</comparableWith>
   <comparableWith>muni_population_year_1981</comparableWith>
   <comparableWith>muni_population_year_1986</comparableWith>
   <comparableWith>muni_population_year_1991</comparableWith>
   <comparableWith>muni_population_year_1996</comparableWith>
   <comparableWith>muni_population_year_2001</comparableWith>
   <displayControl>sBarChart</displayControl>
   <displayControl>sLinearChart</displayControl>
</muni_population_year_2006>
```

(click for full size)

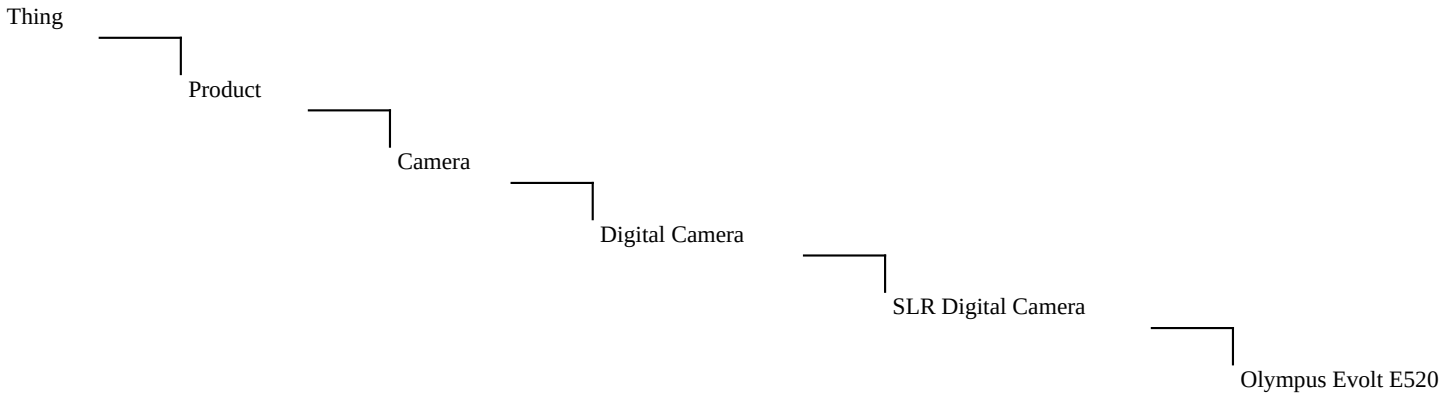
These instructions are then presented to the sControl component, which determines which widgets (individual components, with multiples possible depending on the inputs) need to be invoked and displayed on the layout canvas.

As new user interactions occur with the resulting displays and components, the iteration cycle is generated anew, again starting a new cycle of queries and results sets. Importantly, as these pathways and associated display components get created, they can be named and made persistent for later re-use or within dashboard invocations.

Self-service Reporting

Since self-service reporting has been such a disappointment [12], it is worth noting another aspect from this ODapp design. Every "thing" that can be presented in the interface can have a specific display template associated with it. Absent another definition, for example, any given "thing" will default to its parental type (which, ultimate, is "Thing", the generic template display for anything without a definition; this generally defaults to a presentation of all attributes for the object).

However, if more specific templates occur in the inference path, they will be preferentially used. Here is a sample of such a path:



At the ultimate level of a particular model of Olympus camera, its display template might be exactly tailored to its specifications and attributes.

This design is meant to provide placeholders for any "thing" in any domain, while also providing the latitude to tailor and customize to every "thing" in the domain.

It is critical that generic apps through an ODapp approach also provide the underpinnings for self-service reporting. The ultimate metric is whether consumers of information can create the reports they need without any support or intervention by IT.

Adaptive Analysis

The Mission Critical IT reference provided earlier [11] helps point to the potentials of this paradigm in a different way. Mission Critical also shows user interfaces contextually chosen based on prior selections. But they extend that advantage with context-specific analysis and validation through the SWRL rules-

base semantic language. This is an exciting extension of the base paradigm that confirms the applicability of this approach to business intelligence and general enterprise analytics.

Standing Software Engineering on its Head

All of this points to a very exciting era for enterprise and consumer apps moving into the future. We perhaps should no longer talk about "killer apps"; we can shift our focus to the information we have at hand and how we want to structure and analyze it.

Using ontologies to write or specify code or to compete as an alternative to conventional software engineering approaches seems too much like more of the same. The systems basis in which such methodologies such as MDA reside have not fixed the enterprise software challenges of decades-long standing. Rather, a shift to generic applications driven by adaptive ontologies -- ODapps -- looks to shift the locus from software and programming to data and knowledge structures.

This democratization of IT means that everything in the knowledge management realm can become "self service." We can create our own analyses; develop our own reports; and package and disseminate what we and our colleagues need, when they need it. Through ontology-driven apps and adaptive ontologies, we can turn prior decades of software engineering practices on their head.

What Structured Dynamics and a handful of other vendors are showing is by no means yet complete. Our roster of ODapp widgets and templates still needs much filling out. The toolsets available for creating, maintaining, mapping and extending the ontologies underlying these systems are still woefully inadequate [20]. These are important development needs for the near term.

And, of course, none of this means the end of software development either. Process and transactions systems still likely reside outside of this new, emerging paradigm. Creating great and solid generic ODapps still requires software. Further, ODapps and their potential are completely silent on how we create that software and with what languages or methodologies. The era of software engineering is hardly at an end.

What is exceptionally powerful about the prospects in ontology-driven apps is to speed time to understanding and place information manipulation directly in the hands of the knowledge worker. This is a vision of information access and control that has been frustrated for decades. Perhaps, with ontologies and these semantic technologies, that vision is now near at hand.

---

[1] This estimate is from Grady Booch, 2005. "The Complexity of Programming Models," see http://www.cs.nott.ac.uk/~nem/complexity.pdf. He comments on the weakness of software lines of code as a meaningful measure. At the time in 2005, he estimated perhaps 800 billion lines of code has accumulated, which given growth and vagaries of such guesstimates I have updated to the 1 billion number noted.

[2] For a wildly different estimate, that has been criticized somewhat, see Blackduck Software, 2009. "Estimating the Development Cost of Open Source Software," at http://www.blackducksoftware.com/development-cost-of-open-source. According to Blackduck's research there are over 200,000 OSS projects on the Internet representing more than 4.9 billion lines of available code from 4,000 sites that the company monitors. Blackduck estimates that reproducing this OSS would cost $387 billion for "typical" SLOC estimating bases. While Blackduck is likely in the best place of any organization to track open source given their business model, others have criticized the estimates

because only a portion (fewer than 10%, consistent with my own research) of open source projects are active, and many active projects also share significant code bases. Nonetheless, there is still a huge disparity between the 1 billion SLOC estimate in [1] and this estimate of 5 billion for open source alone. This disparity is an indicator of the measurement challenges.

[3] See IMAP, 2010. *Computing & Internet Software Global Report — 2010*, 40 pp, see http://imap.com/imap/media/resources/HighTechReport_WEB_89B4E29C01817.pdf. The relative splits they show for software packages and licenses, IT consulting or outsourcing are 48%, 29% and 23%, respectively, of the total shown. Note however, that Gartner estimates are as high as 2x these amounts, again showing the uncertainty of measuring software; see, for example, http://www.gartner.com/it/page.jsp?id=1209913.

[4] For this and related measures, see Business Software Alliance, 2009. *Software Industry Facts and Figures*, see http://www.bsa.org/country/Public%20Policy/~/media/Files/Policy/Security/General/sw_factsfigures.ashx.

[5] Simply conduct a Web search on "'open source" "cost of ownership"' to see the many studies in this area. Depending on advocacy, estimates may be as high as proprietary software to a lower, but still substantial percentage. In no cases are open source understood to be fully "free" once maintenance, upgrades, modifications, and site adaptations are considered.

[6] Michael Uschold, 2008. "Ontology-Driven Information Systems: Past, Present and Future," in *Proceedings of the Fifth International Conference on Formal Ontology in Information Systems (FOIS 2008)*, Carola Eschenbach and Michael Grüninger, eds., IOS Press, Amsterdam, Netherlands, pp 3-20; see http://mba.eci.ufmg.br/downloads/recol/FormalOntologyinInformationSystems2008.pdf.

[7] Nicola Guarino, 1998. "Formal Ontology and Information Systems," in *Proceedings of FOIS'98*, Trento, Italy, June 6-8, 1998. Amsterdam, IOS Press, pp. 3-15; see http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.1776&rep=rep1&type=pdf.

[8] See Phil Tetlow *et al.*, eds., 2006. *Ontology Driven Architectures and Potential Uses of the Semantic Web in Software Engineering*, a W3C Editor's Draft on Best Practices, February 11, 2006; see http://www.w3.org/2001/sw/BestPractices/SE/ODA/. UML class diagrams have close resemblance to certain ontology structures. This effort was part of a formal collaboration between W3C and the Object Management Group (OMG), which resulted among other things in the production of the Ontology Definition Metamodel (ODM). In the OMG's model-driven architecture (MDA) initiative, models are used not only for design and maintenance purposes, but as a basis for generating executable artifacts for downstream use. The MDA approach grew out of much of the standards work conducted in the 1990s in the Unified Modeling Language (UML).

[9] Neno is a semantic network programming language and Fhat is a virtual machine that works off of it. These two projects have been largely abandoned. A related project is Ripple, a relational, stack-based dataflow language by Joshua Shinavier, which is episodically updated.

[10] Holger Knublauch of TopQuadrant has made the point that ontologies can also have runtime uses as well: "In contrast to conventional Model-Driven Architecture known from object-oriented systems, semantic applications use their data models not only at design time, but also as runtime components. The rich declarative semantics of ontological data models can be exploited to drive user interfaces and to control an application's behavior." See H. Knublauch, 2007. "From Ontology Design to Deployment: Semantic Application Development with TopBraid," presented at the *2007 Semantic Technology Conference*, San Jose, CA; see http://www.semantic-conference.com/2007/sessions/l5.html.

[11] Mission Critical IT describes its ODASE platform (Ontology Driven Architecture for Software Engineering) as a

set of tools to facilitate the creation of working applications from a semantic business model (an ontology), using the open standards OWL, SWRL and RDF. The ODASE code generators (a.k.a "robots") generate an API based on the business terminology defined by the OWL+SWRL+RDF business model, which the ODASE platform then uses to execute the rules and reasoning as contextual choices are made by the user. Among other links, the company has an impressive online demo that shows a consumer telecommunications purchase example; there is also a video explaining the rules basis of the ODASE framework.

[12] See Wayne W. Eckerson, 2007. "The Myth of Self-Service Business Intelligence," in *TDWI Online*, October 18, 2007; see http://tdwi.org/articles/2007/10/18/the-myth-of-selfservice-bi.aspx.

[13] The buggy whip industry as a major economic entity ceased to exist with the introduction of the automobile, and is cited in economics and marketing as an example of an industry ceasing to exist because its market niche, and the need for its product, disappears. Not recognizing what industry or business purpose is being served is an oft-cited cause for obsolescence. Thus, software engineering is a practice that serves the creation of software, which itself is only a means to a functional end.

[14] See M. K. Bergman, 2009. "The Open World Assumption: Elephant in the Room," *AI3:::Adaptive Information* blog, December 21, 2009. The open world assumption (OWA) generally asserts 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. Another way to say it is that everything is permitted until it is prohibited. OWA lends itself to incremental and incomplete approaches to various modeling problems.

[15] See M.K. Bergman, 2010. "Seven Pillars of the Open Semantic Enterprise", *AI3:::Adaptive Information* blog, January 12, 2010.

[16] See M.K. Bergman, 2009. "Ontologies as the 'Engine' for Data-Driven Applications", *AI3:::Adaptive Information* blog, June 10, 2009, for the first presentation of these topics, but the specific term *adaptive ontology* was not yet used. That term was first introduced in "Confronting Misconceptions with Adaptive Ontologies" (August 17, 2009). The dedicated treatment of these topics and their interplay was provided in M.K. Bergman, 2009. "Ontology-driven Applications Using Adaptive Ontologies", *AI3:::Adaptive Information* blog, November 23, 2009. The relation of these topics to enterprise software was first presented in M.K. Bergman, 2009. "Fresh Perspectives on the Semantic Enterprise", *AI3:::Adaptive Information* blog, September 28, 2009.

[17] Some 250 pp of complete technical documentation for these projects is provided on the Structured Dynamics' open source OpenStructs TechWiki.

[18] For more discussion of semantic components, see F. Giasson, 2010. "Semantic Components," in his blog, July 5, 2010. For more discussion of the layered OSF design, see M.K. Bergman, 2010. "Domain-specific Instantiations based on the Open Semantic Framework", *AI3:::Adaptive Information* blog, June 17, 2010.

[19] To find these groups and follow the open source OSF developments, see xxx. So long as the basic design comports with the foundations herein, sComponents may be developed in any rich Internet application (RIA) environment.

[20] Ontology development, management and mapping is the emerging imperative in the semantic technology space. For some thoughts on how Structured Dynamics is approaching this question, see a Normative Landscape of Ontology Tools on the TechWiki.

PDF generated by *AI3:::Adaptive Information* blog