COMBINING KNOWLEDGE GRAPHS AND ONTOLOGIES FOR DYNAMIC APPS

Michael K. Bergman[1], Coralville, Iowa USA

September 11, 2019

AI3:::Adaptive Information blog

In a recent article about knowledge graphs I noted that I tend to use the KG term interchangeably with the term 'ontology'. While this interchangeability is generally true when ontologies are used to model instance and class knowledge, in other words for knowledge representation (KR), it does overlook important cases when ontologies are themselves a specification for aspects such as access control, applications, or user interfaces. In these cases the ontology is less related to knowledge and more related to specifications or control. In such cases it is probably best to retain the distinction of an ontology from a knowledge graph (which I tend to think of as more oriented to content). I elaborate further on this distinction in this article.

What brought this distinction to mind was a recent post by Bob DuCharme on custom HTML forms to drive back-end SPARQL queries. The example Bob uses is getting a listing of cocktails from Wikidata given a specified ingredient. The example he provides uses Perl for a CGI (Common Gateway Interface) script. Bob has discussed generic SPARQL queries before; he features many useful Python examples in his excellent SPARQL book [1].

The basic idea is to provide values for variables entered via a Web page form to complete a patterned SPARQL query (SPARQL is the query language for RDF). The example Bob uses is to have the user enter a cocktail ingredient, which then returns all of the cocktails listed on Wikidata that contain that ingredient. The advantage of the idea is that users need know nothing about SPARQL or how to form a proper SPARQL query. By simply entering in missing information on a Web form or making other Web form choices (such as picking from a list or a radiobutton), all of the heavy lifting is done by the patterned SPARQL script in the background. Letting the Web forms provide the values for SPARQL variables is the key to the method.

We use this idea aggressively on, for example, our KBpedia Web site. By picking a search term from an auto-completed search listing [2] or picking a live link from that same page [3], we are able to re-use a fixed set of SPARQL query patterns to drive simple Web page templates. In our case, we use JavaScript to control the display and canvas and to invoke Clojure scripts that generate the SPARQL queries. (Over the years we have also used PHP and JavaScript directly to generate these queries. The point, as is made by DuCharme, is most any scripting language may be used for the backend.) You may inspect any of the sub-pages under the 'Knowledge Graph' section on the site by using 'View Page Source'. Sample Clojure code is also available for inspection to see how we have implemented the approach [4].

Ontology-driven Apps

This basic idea of patterned SPARQL queries forms the baseline for what we have been calling data-driven applications [5] for more than 10 years, when we first began experimenting with the approach and using it in early customer engagements. And our embrace of the idea is not the first. For example, in 1998, more than a

[1]Email: mike@mkbergman.com

decade before our own efforts, Guarino [6] was already talking of "ontology-driven" information systems and the use of ontologies for user interfaces. Still, ten years later, though Uschold was noting the same prospects, his survey of advances to that point showed little in actual development of ontologies "driving" applications [7].

It was roughly at that time that our own efforts began. One of our first realizations was that dynamic retrieval and presentation of data on a Web page only began the process. With the Web page as the medium of interaction, the idea of using interfaces to manage data became concrete. By organizing information into datasets and setting profiles for access and CRUD (*create – read – update – delete*) rights, an effective environment for data sharing and federation is established. We saw that we could abstract the complexity of the languages and specifications (SPARQL, RDF, and OWL) into the background, letting developers write the backend scripts, while letting the users and subject matter experts deal with updating, selecting and managing the content via Web front-ends.

Today, most approaches to semantic technologies and ontologies are still, unfortunately, rather static and fixed. Separate applications or IDEs are used to write and manage the ontologies. The ontologies are not generally subject to continuous upgrades and refinements, and end-users are the recipients, not the 'drivers' of the systems. But our early efforts showed how we could democratize this process, making updates and interactions dynamic.

With the embrace of CRUD, we also needed dynamic ways for changes made to the system — now codified and maintained in ontologies — to be reflected back to the user interfaces. We saw that a layer of specific Web services could both submit and query information to the ontology, and present those changes dynamically to scripts within the HTML user interfaces. (We also saw that access control to both data and applications needed to be imposed for enterprise uses, functions that can also be mediated by ontologies. Those topics are not discussed further here, but we have documented elsewhere [8]). Because the user interface was becoming the medium of interaction, it was also apparent that we needed to expand our use of labels in the ontologies. Thus, besides standard SKOS concepts like `altLabels` for node synonyms or `prefLabels` for preferred node labels, we also needed to accommodate labels for tooltips and labels that appear as titles or instructions on forms on user interfaces.

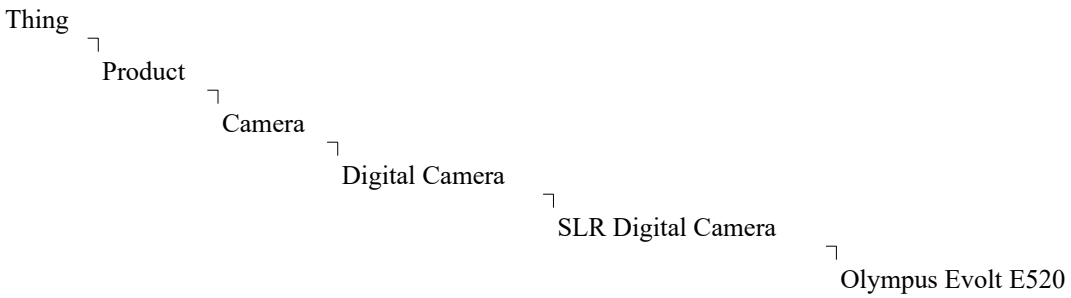
Once this rubicon of dynamic interfaces driven by ontologies is crossed, many new opportunities come to the fore. One opportunity, based on the idea of patterned information, is that different information in the ontology may lend itself to different display or visualization. For example, all location information may be displayed on a map as points, regions, or paths. Or, people and many objects may warrant displaying a picture if available. Or, numeric values over similar dimensions may lend themselves to charting. Or, ordered or unordered lists may warrant a listing display, or, when characterized by numeric values, by pie charts or other chart types.

These realizations led us to create a series of display and visualization components, the invoking of which may be triggered by the datatypes coming back in a results set initiated by a SPARQL query. The widget code for these display and visualization options may be served up by Web services in response to the characteristics in the results streams in a similar way we can serve up filtering, searching, browsing, import/export, or other functional widgets. In other words, the nature of the information in the ontology can inform what functions — including visualization — we can perform with a given results stream. (See, for example, any of the displays such as charts or maps for the Peg community indicator system built with our design for the United Way of Winnipeg.)

Another opportunity arises from the idea of a data record coming back in a results set. We see, for example, how the so-called 'infoboxes' in Wikipedia or on a Google search results page show us a suite of data attributes for a given entity. We see 'people' entities characterized by birth, death, parents, children, country of origin, occupation, and such. We see 'automobile' entities characterized by body type, brand, horsepower, year built, etc. These kinds of characterizations are patterned, too, and can begin to be organized into hierarchies and types.

Because of this patterned, structured nature of entity types, we can generalize our data display templates further. What if we detect our instance represents a camera but do not have a display template specific to

cameras? Well, the ontology and simple inferencing can tell us that cameras are a form of digital or optical products, which more generally are part of a product concept, which more generally is a form of a human-made artifact, or similar. However, if more specific templates occur in the inference path, they will be preferentially used. Here is a sample of such a path:



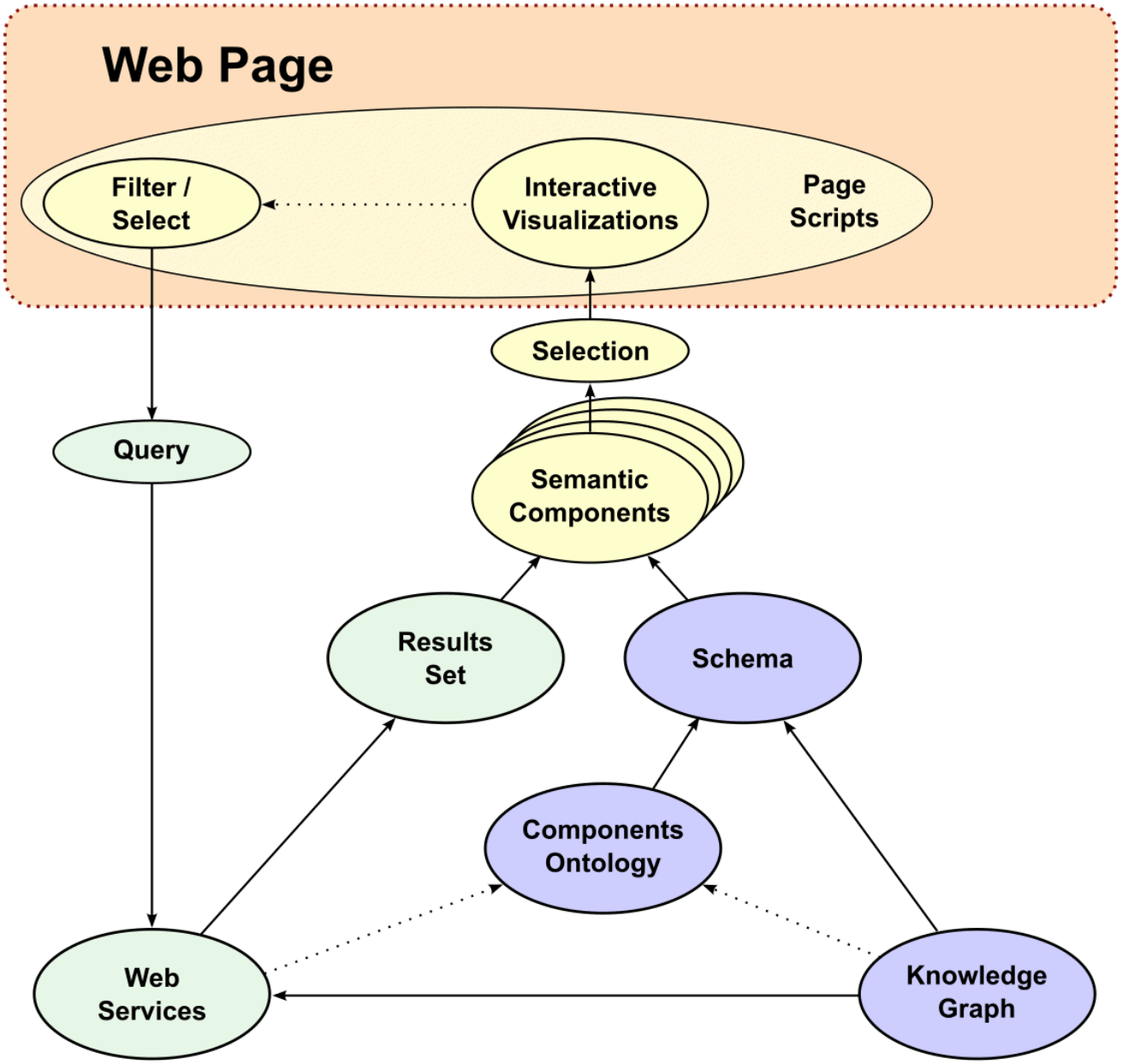
At the ultimate level of a particular model of Olympus camera, its display template might be exactly tailored to its specifications and attributes.

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. By tracing this inferencing chain from the specific to the more general we can 'fall back' until a somewhat OK display template is discovered, even in the absence of the better and more specific one. Then, if we find we are trying to display information on cameras frequently, we only need take one of the more general, parent templates and specifically modify it for the desired camera attributes. We also keep presentation separate from data so that the styling and presentation mode of these templates is also freely modifiable.

Coming to a Generalized Understanding

Within a couple of years of first working with this approach we came to have a more generalized understanding of what we call 'ODapps' [9]. We modularized the ontologies to separate the information (what is now called the 'knowledge graph') from the specifications of the semantic components. We also enhanced the label constructs in the KG to handle user interface labels and related. I have slightly updated the workflow we showed for this process back in 2011 (see next page).

The basic process begins when the user interacts with various semantic components embedded in the layout of the Web page. Once the user interacts with these various components, new queries are generated (most often as SPARQL queries) in the background to the various Web services endpoints, which are specific to either management or display functions. The first consequence of the query is to generate a results set of data from the knowledge graph. At the same time, the datatypes of the results inform a components ontology that produces a schema useful to the display widgets. This schema constitutes the formal instructions to the semantic components on the Web page.



(click for full size)

When this schema is combined with the results set data, the new instructions for the semantic components on the Web page are complete. 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 various semantic components, and determine 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, 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. In this way, the user interactions may act as a form of recorder for later automatic playback of the interaction choices.

A New Dynamic Paradigm for User Apps

ODapps are thus a balanced abstraction within the framework of canonical architectures, data models and data structures. Under this design, software developer time is focused on creating the patterned scripts that underlie the Web page layouts, developing the semantic component widgets, and writing the functional Web services. Users and subject matter experts can concentrate on doing analysis and keeping the ontologies and knowledge graph accurate and up-to-date. 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.

This new paradigm began with the simple observation that Bob DuCharme demonstrates that we can use SPARQL queries driven by users in a Web page form to get relevant information back to the user. We have taken this simple premise and have — over the past nearly ten years — expanded it to be a more generalized approach to ontology-driven apps, or ODapps. We have also continued to talk about how we may modularize our ontology architectures for a breadth of enterprise purposes [10].

Yet, while we have prototyped these capabilities and have demonstrated them within our own customer engagements, this general approach is by no means common.

Perhaps now, with the resurgent interest in knowledge graphs, we can finally see our way clear to a suite of semantic approaches that promise a revolution in software design practices and the democratization of information technologies. Through the ODapp approach, we believe that customers can see:

- 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
- Re-use — abstract/general notions can be used to instantiate more concrete/specific notions, allowing more reuse
- 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.

As I have noted before, these first four items are similar to the benefits that may accrue from other advanced software engineering methodologies, though with some unique twists due to the semantic basis. However, Uschold [7] 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 more easily and reliably make changes in the model than in code.

So, as practiced today, most uses of ontologies are for knowledge representation, and in that sense we may use the terms ‘knowledge graph’ and ‘ontologies’ more-or-less interchangeably. However, taken to its logical extent and embraced for driving software specifications, we see the term of ‘ontology’ as much more general and powerful. Like I have said before, the meaning of these terms is intimately related to their context of use.

Acknowledgements

This article was originally posted on the *AI3::Adaptive Information* Web site at http://www.mkbergman.com/2267/combining-knowledge-graphs-and-ontologies-for-dynamic-apps/. This version has been edited and reformatted slightly for PDF distribution. We thank Cognonto Corporation for making this content freely available.

---

[1] Bob DuCharme, *Learning SPARQL: Querying and Updating with SPARQL 1.1, Second Edition*, 2013, O’Reilly Media, 386 pp.

[2] From this URI, for example, http://kbpedia.org/knowledge-graph/reference-concept/?uri=Mammal, begin typing into the upper right search box and then picking one of the suggested auto-completion terms.

[3] For example, picking the ‘amniote’ link (http://kbpedia.org/knowledge-graph/reference-concept/?uri=Amniote) from the lower left Broader Concepts text box.

[4] To see an example of JS code calling the Clojure routines see http://kbpedia.org/entity/browse/js/browse-entities.js. Then, look for the Clojure call noted 'nb-entities'. You can see the actual Clojure routines under this same name in the sample http://www.mkbergman.com/wp-content/themes/ai3v2/files/2019Posts/named_entities.clj file. (This sample file contains other functions to clean up input strings, for example. Also note that most Clojure code used by the system is not available for inspection.)

[5] Our series on this topic began with the article, M.K. Bergman, "Concepts and an Introduction to the Occasional Series on 'Ontology Best Practices for Data-driven Applications,' *AI3:::Adaptive Information* blog, May 12, 2009, and continued with a more detailed discussion in M.K. Bergman, "Ontologies as the 'Engine' for Data-Driven Applications," *AI3:::Adaptive Information* blog, June 10, 2009. The later article introduced the ideas of data-driven displays and user interfaces based on ontologies specifically enhanced to include those specifications.

[6] Nicola Guarino, "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.

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

[8] M.K. Bergman, "structWSF: A Framework for Collaboration Networks," *AI3:::Adaptive Information* blog, July 7, 2009.

[9] M.K. Bergman, "Ontology-Driven Apps Using Generic Applications," *AI3:::Adaptive Information* blog, March 7, 2011.

[10] M.K. Bergman, "An Ontologies Architecture for Ontology-driven Apps ," *AI3:::Adaptive Information* blog, December 5, 2011.