Rationales for Typology Designs in Knowledge Bases

by Mike Bergman - Monday, June 06, 2016

http://www.mkbergman.com/1952/rationales-for-typology-designs-in-knowledge-bases/

Design is Aimed to Improve Computability

In the lead up to our most recent release of UMBEL, I began to describe our increasing reliance on the use of ***typologies***. In this article, I'd like to expand on our reasons for this design and the benefits we see.

'Typology' is not a common term within semantic technologies, though it is used extensively in such fields as archaeology, urban planning, theology, linguistics, sociology, statistics, psychology, anthropology and others. In the base semantic technology language of RDF, a 'type' is what is used to declare an instance of a given class. This is in keeping with our usage, where an instance is a member of a type.

Strictly speaking, 'typology' is the study of types. However, as used within the fields noted, a 'typology' is the result of the classification of things according to their characteristics. As stated by Merriam Webster, a 'typology' is "a system used for putting things into groups according to how they are similar." Though some have attempted to make learned distinctions between typologies and similar notions such as classifications or taxonomies [1], we think this idea of grouping by similarity is the best way to think of a typology.

In Structured Dynamics' usage as applied to UMBEL and elsewhere, we are less interested in the sense of 'typology' as comparisons across types and more interested in the classification of types that are closely related, what we have termed 'SuperTypes'. In this classification, each of our SuperTypes gets its own typology. The idea of a SuperType, in fact, is exactly equivalent to a typology, wherein the multiple entity types with similar essences and characteristics are related to one another via a natural classification. I speak elsewhere how we actually go about making these distinctions of natural kinds [2].

In this article, I want to stand back from how a typology is constructed to deal more about their use and benefits. Below I discuss the evolution of our typology design, the benefits that accrue from the 'typology' approach, and then conclude with some of the application areas to which this design is most useful. All of this discussion is in the context of our broader efforts in KBAI, or knowledge-based artificial intelligence.

Evolution of the Design

I wish we could claim superior intelligence or foresight in how our typology design came about, but it was truthfully an evolution of needing to deal with pragmatic concerns in our use of UMBEL over the past near-decade. The typology design has arisen from the intersection of: 1) our efforts with SuperTypes, and creating a computable structure that uses powerful disjoint assertions; 2) an appreciation of the importance of entity types as a focus of knowledge base terminology; and 3) our efforts to segregate entities from other key constructs of knowledge bases, including attributes, relations and annotations. Though these insights may have resulted from serendipity and practical use, they have brought a new understanding of how best to organize knowledge bases for artificial intelligence uses.

The Initial Segreation into SuperTypes

We first introduced SuperTypes into UMBEL in 2009 [3]. The initiative arose because we observed about 90% of the concepts in UMBEL were disjoint from one another. Disjoint assertions are computationally efficient and help better organize a knowledge graph. To maximize these benefits we did both top-down and bottom-up testing to derive our first groupings of SuperTypes into 29 mostly disjoint types, with four non-disjoint (or cross-cutting and shared) groups [3]. Besides computational efficiency and its potential for logical operations, we also observed that these SuperTypes could also aid our ability to drive display widgets (such as being able to display geolocational types on maps).

All entity classes within a given SuperType are thus organized under the SuperType (ST) itself as the root. The classes within that ST are then organized hierarchically, with children classes having a subClassOf relation to their parent. By the time of UMBEL's last release [4], this configuration had evolved into the following 31 largely disjoint SuperTypes, organized into 10 or so clusters or “dimensions”:

Constituents	
	Natural Phenomena
	Area or Region
	Location or Place
	Shapes
	Forms
Situations	
Time-related	
	Activities
	Events
	Times
Natural Matter	
	Atoms and Elements
	Natural Substances

	Chemistry
Organic Matter	
	Organic Chemistry
	Biochemical Processes
Living Things	
	Prokaryotes
	Protists & Fungus
	Plants
	Animals
	Diseases
Agents	
	Persons
	Organizations
	Geopolitical
Artifacts	
	Products
	Food or Drink
	Drugs
	Facilities
Information	
	Audio Info
	Visual Info
	Written Info
	Structured Info
Social	
	Finance & Economy
	Society

Current SuperType Structure of UMBEL

We also used the basis in SuperTypes to begin cleaving UMBEL into modules, with geolocational types being the first to be separated. We initially began splitting into modules as a way to handle UMBEL's

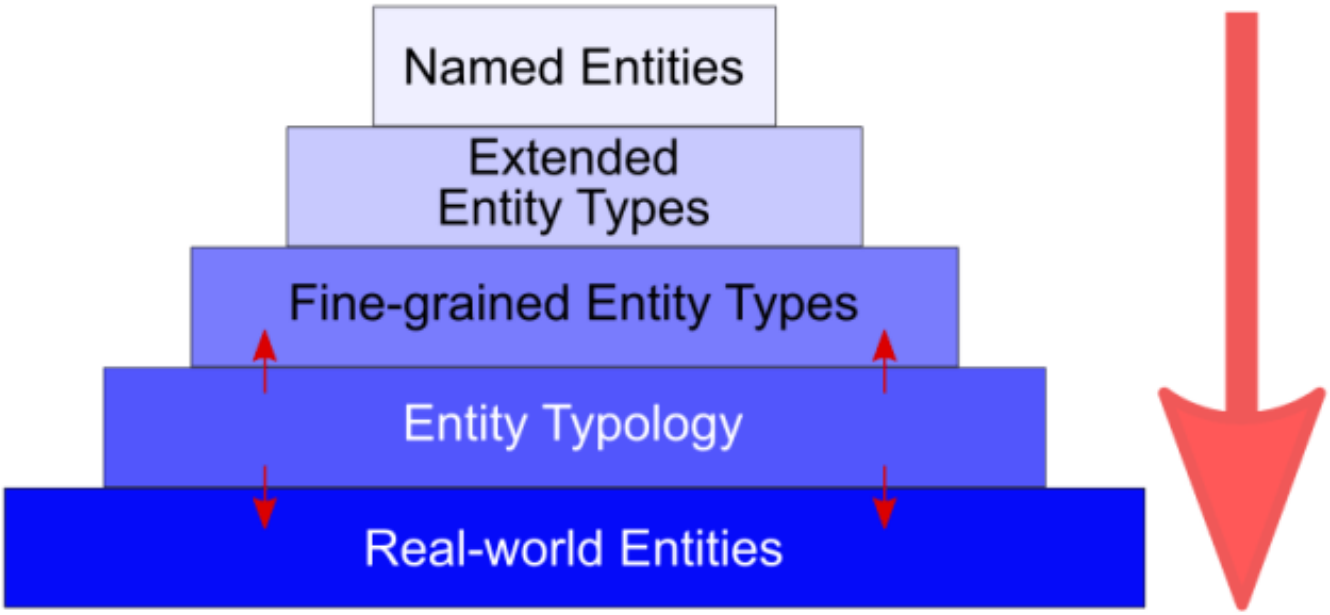
comparatively large size (~ 30K concepts). As we did so, however, we also observed that most of the SuperTypes could be also be segregated into modules. This architectural view and its implications were another reason leading to the eventual typology design.

A Broadening Appreciation for the Pervasiveness of Entity Types

The SuperType tagging and possible segregation of STs into individual modules led us to review other segregations and tags. Given that the SuperTypes were all geared to entities and entity types -- and further represented about 90% of all concepts in UMBEL -- we began to look at entities as a category with more care and attention. This analysis took us back to the beginnings of entity recognition and tagging in natural language processing. We saw the progression of understanding from named entities and just a few entity types, to the more recent efforts in so-called fine-grained entity recognition [5].

What was blatantly obvious, but which had been previously overlooked by us and other researchers investigating entity types, was that most knowledge graphs (or upper ontologies) were themselves made of largely entity types [5]. In retrospect, this should not be surprising. Most knowledge graphs deal with real things in the world, which, by definition, tend to be entities. Entities are the observable, often nameable, things in the world around us. And how we organize and refer to those entities -- that is, the entity types -- constitutes the bulk of the vocabulary for a knowledge graph.

We can see this progression of understanding moving from named entities and fine-grained entity types, all the way through to an entity typology -- UMBEL's SuperTypes -- that then becomes the tie-in point for individual entities (the ABox):



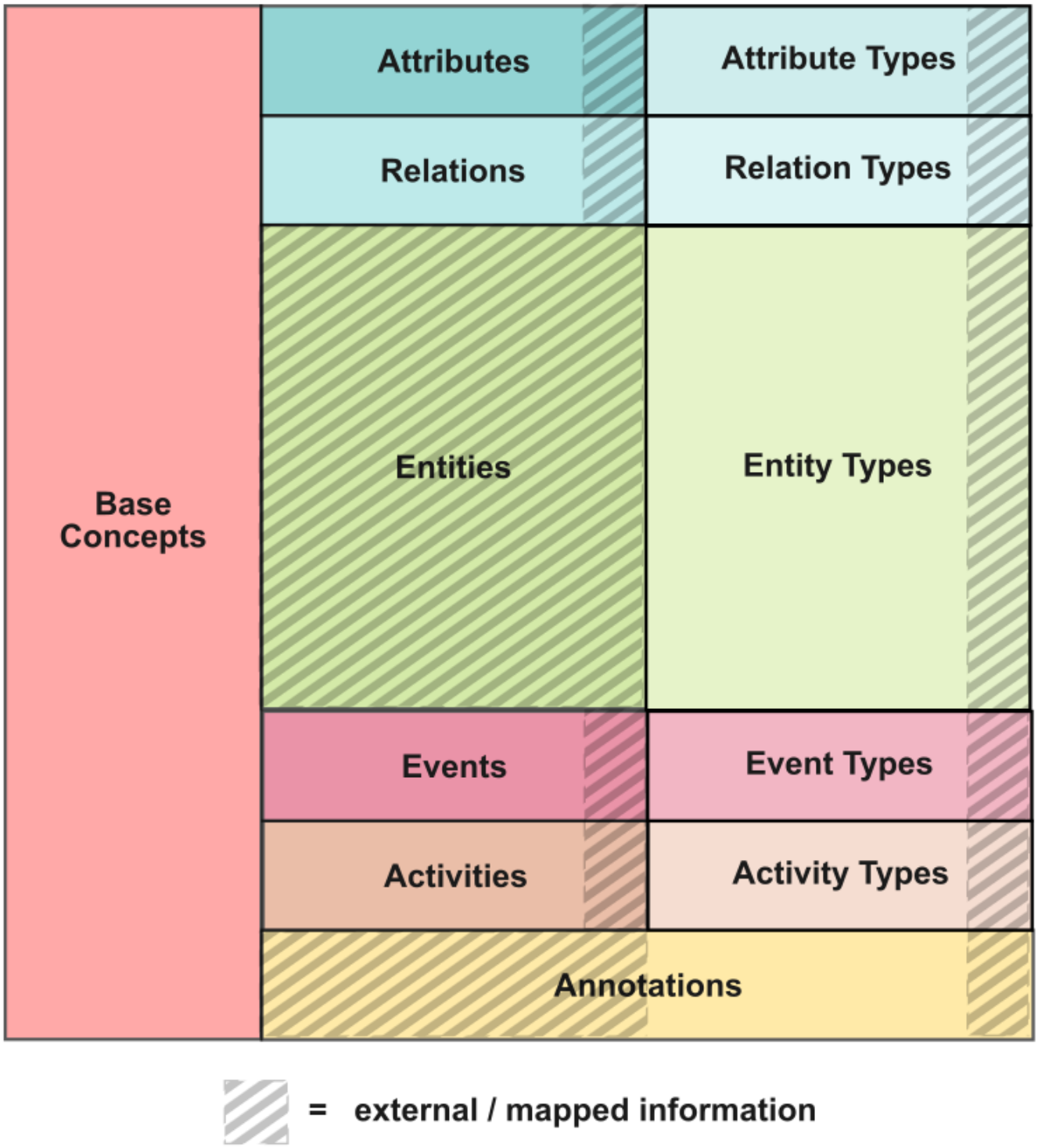
Evolving Sophistication of Entity Types

The key transition is moving from the idea of discrete numbers of entity types to a system and design that

supports continuous interoperability through an "accordion-like" typology structure.

The General Applicability of 'Typing' to All Aspects of Knowledge Bases

The "type-orientation" of a typology was also attractive because it offers a construct that can be applied to all other (non-entity) parts of the knowledge base. Actions can be typed; attributes can be typed; events can be typed; and relations can be typed. A mindset around natural kinds and types helps define the *speculative grammar* of KBAI (a topic of a next article). We can thus represent these overall structural components in a knowledge base as:



Typology View of a Knowledge Base

The shading is in reference to that which is external to the scope of UMBEL.

The intersection of these three factors -- SuperTypes, an "accordion" design for continuous entity types,

and overall typing of the knowledge base -- set the basis for how to formalize an overall typology design.

Formalizing the Typology Design

We have first applied this basis to typologies for entities, based on the SuperTypes. Each SuperType becomes its own typology with natural boundaries and a hierarchical structure. No instances are allowed in the typology; only types.

Initial construction of the typology first gathers the relevant types (concepts) and automatically evaluates those concepts for orphans (unconnected concepts) and fragments (connected portions missing intermediary parental concepts). For the initial analysis, there are likely multiple roots, multiple fragments, and multiple orphans. We want to get to a point where there is a single root and all concepts in the typology are connected. Source knowledge bases are queried for the missing concepts and evaluated again in a recursive manner. Candidate placements are then written to CSV files and evaluated with various utilities, including crucially manual inspection and vetting. (Because the system bootstraps what is already known and structured in the system, it is important to build the structure with coherent concepts and relations.)

Once the overall candidate structure is completed, it is then analyzed against prior assignments in the knowledge base. ST disjoint analysis, coherent inferencing, and logical placement tests again prompt the creation of CSV files that may be viewed and evaluated with various utilities, but, again, ultimately manually vetted.

The objective of the build process is a fully connected typology that passes all coherency, consistency, completeness and logic tests. If errors are subsequently discovered, the build process must be run again with possible updates to the processing scripts. Upon acceptance, each new type added to a typology should pass a completeness threshold, including a definition, synonyms, guideline annotations, and connections. The completed typology may be written out in both RDF and CSV formats. (The current UMBEL and its typologies are available here.)

Integral to the design must be build, testing and maintenance routines, scripts, and documentation. Knowledge bases are inherently open world [6], which means that the entities and their relationships and characteristics are constantly growing and changing due to new knowledge underlying the domain at hand. Such continuous processing and keeping track of the tests, learnings and workflow steps place a real premium on literate programming, about which Fred Giasson, SD's CTO, is now writing [7].

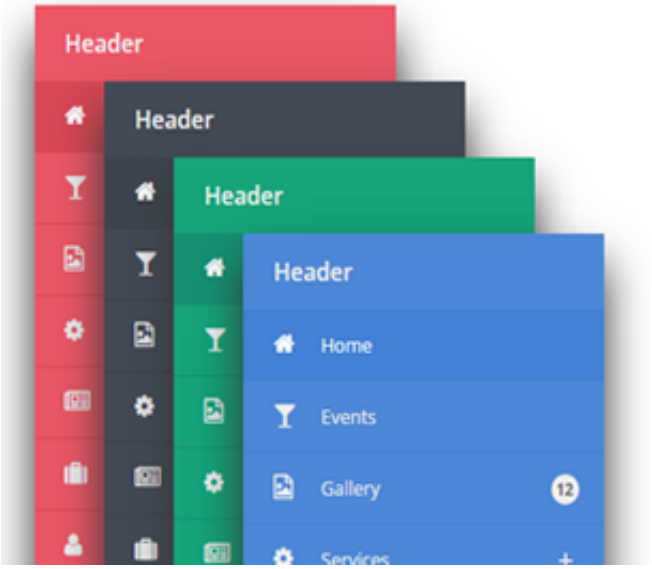
Because of the very focused nature of each typology (as set by its root), each typology can be easily incorporated or excised from a broader structure. Each typology is rather simple in scope and simple in structure, given its hierarchical nature. Each typology is readily maintained and built and tested. Typologies pose relatively small ontological commitments.

Benefits of the Design

The simple bounding and structure of the typology design makes each typology understandable merely by inspecting its structure. But the typologies can also be read into programs such as Protégé in order to

inspect or check complete specifications and relationships.

Because each typology is (designed to be) coherent and consistent, new concepts or structures may be related to any part of its hierarchical design. This gives these typologies an "accordion-like" design, similar to the multiple levels and aggregation made possible by an accordion menu:



An 'Accordion' Design Accommodates Different Granularities

The combination of logical coherence with a flexible, accordion structure gives typologies a unique set of design benefits. Some have been mentioned before, but to recap they are:

Computable

Each type has a basis -- ranging from attributes and characteristics to hierarchical placement and relationship to other types -- that can inform computability and logic tests, potentially including neighbor concepts. Ensuring that type placements are accurate and meet these tests means that the now-placed types and their attributes may be used to test the placement and logic of subsequent candidates. The candidates need not be only internal typology types, but may also be used against external sources for classification, tagging or mapping.

Because the essential attributes or characteristics across typologies in an entire domain can differ broadly — such as entities *v* attributes, living *v* inanimate things, natural things *v* man-made things, ideas *v* physical objects, etc. — it is possible to make disjointedness assertions between entire groupings of natural classes. Disjoint assertions, combined with logical organization and inference, provide a typology design that lends itself to reasoning and tractability.

The internal process to create these typologies also has the beneficial effect of testing placements in the knowledge graph and identifying gaps in the structure as informed by fragments and orphans. This

computability of the structure is its foundational benefit, since it determines the accuracy of the typology itself and drives all other uses and services.

Pluggable and Modular

Since each typology has a single root, it is readily plugged into or removed from the broader structure. This means the scale and scope of the overall system may be easily adjusted, and the existing structure may be used as a source for extensions (see next). Unlike more interconnected knowledge graphs (which can have many network linkages), typologies are organized strictly along these lines of shared attributes, which is both simpler and also provides an orthogonal means for investigating type-class membership.

Interoperable

The idea of nested, hierarchical types organized into broad branches of different typologies also provides a very flexible design for interoperating with a diversity of world views and degrees of specificity. A typology design, logically organized and placed into a consistent grounding of attributes, can readily interoperate with these different world views. So far, with UMBEL, this interoperable basis is limited to concepts and things, since only the entity typologies have been initially completed. But, once done, the typologies for attributes and relations will extend this basis to include full data interoperability of attribute:value pairs.

Extensible

A typology design for organizing entities can thus be visualized as a kind of accordion or squeezebox, expandable when detail requires, or collapsed to more coarse-grained when relating to broader views. The organization of entity types also has a different structure than the more graph-like organization of higher-level conceptual schema, or knowledge graphs. In the cases of broad knowledge bases, such as UMBEL or Wikipedia, where 70 percent or more of the overall schema is related to entity types, more attention can now be devoted to aspects of concepts or relations.

Each class within the typology can become a tie-in point for external information, providing a collapsible or expandable scaffolding (the ‘accordion’ design). Via inferencing, multiple external sources may be related to the same typology, even though at different levels of specificity. Further, very detailed class structures can also be accommodated in this design for domain-specific purposes. Moreover, because of the single tie-in point for each typology at its root, it is also possible to swap out entire typology structures at once, should design needs require this flexibility.

Testable and Maintainable

The only sane way to tackle knowledge bases at these structural levels is to seek consistent design patterns that are easier to test, maintain and update. Open world systems must embrace repeatable and largely automated workflow processes, plus a commitment to timely updates, to deal with the constant, underlying change in knowledge.

Listing of Broad Application Areas

Some of the more evident application areas for this design -- and in keeping with current client and development activities for Structured Dynamics -- are the following:

- ***Domain extension*** -- the existing typologies and their structure provide a ready basis for adding domain details and extensions;
- ***Tagging*** -- there are many varieties of tagging approaches that may be driven from these structures, including, with the logical relationships and inferencing, ontology-based information tagging;
- ***Classification*** -- the richness of the typology structures means that any type across all typologies may be a possible classification assignment when evaluating external content, if the overall system embracing the typologies is itself coherent;
- ***Interoperating datasets*** -- the design is based on interoperating concepts and datasets, and provides more semantic and inferential means for establishing MDM systems;
- ***Machine learning (ML) training*** -- the real driver behind this design is lowering the costs for supervised machine learning via more automated and cost-effective means of establishing positive and negative training sets. Further, the system's feature richness (see next) lends itself to unsupervised ML techniques as well; and
- ***Rich feature set*** -- a design geared from the get-go to emphasize and expose meaningful knowledge base features [8] perhaps opens up many new fruitful avenues for machine learning and other AI. More expressed structure may help in the interpretability of latent feature layers in deep learning. In any case, more and coherent structure with testability can only be goodness for KBAI going forward.

One Building Block Among Many

The progressions and learning from the above were motivated by the benefits that could be seen with each structural change. Over nearly a decade, as we tried new things, structured more things, we discovered more things and adapted our UMBEL design accordingly. The benefits we see from this learning are not just additive to benefits that might be obtained by other means, but they are systemic. The ability to make knowledge bases computable -- while simultaneously increasing the features space for training machine learners -- at much lower cost should be a keystone enabler at this particular point in AI's development. Lowering the costs of creating vetted training sets is one way to improve this process.

Systems that can improve systems always have more leverage than individual innovations. The typology design outlined above is the result of the classification of things according to their shared attributes and essences. The idea is that the world is divided into real, discontinuous and immutable ‘kinds’. Expressed another way, in statistics, typology is a composite measure that involves the classification of observations in terms of their attributes on multiple variables. In the context of a global KB such as Wikipedia, about 25,000 entity types are sufficient to provide a home for the millions of individual articles in the system.

As our next article will discuss, Charles Sanders Peirce’s consistent belief that the real world can be logically conceived and organized provides guidance for how we can continue to structure our knowledge bases into computable form. We now have a coherent base for treating types and natural classes as an essential component to that thinking. These insights are but one part of the KB innovations suggested by Peirce's work.

---

[1] See, for example, Alberto Marradi, 1990. "Classification, Typology, Taxonomy", *Quality & Quantity* 24, no. 2 (1990): 129-157.

[2] M.K. Bergman, 2015. "'Natural Classes' in the Knowledge Web," *AI3:::Adaptive Information* blog, July 13, 2015.

[3] M.K. Bergman, 2009. "'SuperTypes' and Logical Segmentation of Instances," *AI3:::Adaptive Information* blog, September 2, 2009.

[4] umbel.org, "New, Major Upgrade of UMBEL Released," *UMBEL press release*, May 10, 2016 (for UMBEL v. 1.50 release)

[5] M.K. Bergman, 2016. "How Fine Grained Can Entity Types Get?," *AI3:::Adaptive Information* blog, March 8, 2016.

[6] M.K. Bergman, 2012. "The Open World Assumption: Elephant in the Room," *AI3:::Adaptive Information* blog, December 21, 2009.

[7] See http://fgiasson.com/blog/index.php/category/programming/literate-programming/; the last update was, Frédérick Giasson, 2016. "Creating and Running Unit Tests Directly in Source Files with Org-mode", *fgiasson.com/blog*, May 30, 2016.

[8] M.K. Bergman, 2015. "A (Partial) Taxonomy of Machine Learning Features," *AI3:::Adaptive Information* blog, November 23, 2015.
