Available Article

Author's final: This draft is prior to submission for publication, and the subsequent edits in the published version. If quoting or citing, please refer to the proper citation of the published version below to check accuracy and pagination.

Cite as: Bergman, M. K. Building Out the System. in *A Knowledge Representation Practionary: Guidelines Based on Charles Sanders Peirce* (ed. Bergman, M. K.) 273–294 (Springer International Publishing, 2018). doi:10.1007/978-3-319-98092-8_13

Official site: https://link.springer.com/book/10.1007/978-3-319-98092-8

Full-text: http://www.mkbergman.com/publications/akrp/chapter-13.pdf

Abstract: Most of the implementation effort is to conceptualize the structure of the new domain and to populate it with instances . In a proof-of-concept phase, the least-effort path is to leverage KBpedia or portions of it as is, make few changes to the knowledge graph, and populate and test local instance data. You may proceed to create the domain knowledge graph from pruning and additions to the base KBpedia structure, or from a more customized format.

13

BUILDING OUT THE SYSTEM

So, you have looked at the evidence and the prospects, and are now ready to contemplate moving ahead seriously with a knowledge management installation. You have some hoped-for target goals in various kinds of analysis or data interoperability or knowledge-based artificial intelligence. Where does one begin? What is the plan? How can one proceed with initial implementation and testing to keep risks manageable and to demonstrate tangible benefits?

These are the topics of this chapter.[1] We begin by looking at what is involved in tailoring a new installation for specific domain purposes. We identify the checklist of items that you should consider for domain use. We discuss how to conduct an inventory of information assets that we might apply to the instance, and where external sources and information can contribute. We pay particular attention to how to construct a phased implementation plan based on our own experience with successful client projects and lessons learned.

We next discuss the critical work tasks of any new domain installation: the creation of the domain knowledge graph and its population with relevant instance data. We look at the state-of-the-art in mapping methods and tools, and how we may apply those tools to these central tasks. We discuss methodologies and some of the publicly available databases — including those in KBpedia — that may be employed to help facilitate the new effort. We look at longer-term extensions to the base installation that we may contemplate as the effort proceeds from a proof-of-concept to a full-blown knowledge management installation for the enterprise. These factors contribute to how we can make practical choices to proceed given limited time and budget.

Besides the context of limited budgets, these efforts have high but uncertain expectations and a lack of trained creators and users of the system. We know that our efforts must meet the open-world nature of knowledge so that we can turn that fact to our advantage. It is just as defensible, and likely easier to implement and test, an incremental approach to our knowledge domain and data needs. With a pre-defined starting basis such as KBpedia, we can expand new portions with our domain scope and vocabulary in a piecemeal manner, tackling only the current new scope of the specific domain focus at hand, what we call ‘pay as you benefit.’

TAILORING FOR DOMAIN USES

Our prior discussions of '*domain*' make clear that the size and scope of what it means are flexible, from the minute and focused on to the broad and general, for any branch of inquiry. Further, in a purposeful, incremental plan, the domain coverage should also grow and expand. That is one of the beauties of the open-world nature of knowledge.

Scoping the current domain of interest is thus a central task for any existing plan. Most of the implementation effort is to conceptualize (in a *knowledge graph*) the structure of the new domain and to populate it with instances (data). We also find that as our domain scope grows, so does the justification and need for more general knowledge management functions and applications. These general KM tasks, as well as increased maintenance and testing that should accompany any more widely used apps, should be added to the roster of task considerations as incremental plans move forward.

A Ten-point Checklist for Domain Use

Each incremental expansion of the system, including the initial proof-of-concept, should consider, and incorporate as appropriate, specific points from a ten-point checklist:

1. Define potential scope, starting place; a starting place wants and needs champions; the scope is of much lesser importance;
2. Conduct inventory, interview stakeholders, evaluate assets;
3. Develop a phased plan; budget, schedule, and staffing; ID analysis and testing; define platform and ontologies scope and phasing;
4. Assemble assets (tools, data, structure, vocabularies);
5. Build and test domain ontology;
6. Build out platform;
7. Map and populate data;
8. Conduct and test target analysis;
9. Refine and use KM system; and
10. Document and proselytize results.

You should consult this basic checklist each increment of the plan. Some from this checklist may be active during any particular increment, others not. However, these are the general task areas from which to construct the current increment of the plan. You may also need to formalize certain areas over time, such as documentation now exposing and describing workflows, or including deployment requirements.

An Inventory of Assets

Without exception, you must inventory your potential information assets for the installation. This inventory need not be exhaustive to start, just relevant and applicable to the particular domain space that is your starting or expansion scope. Think about what is an information asset in this space, and how one finds and uses such information. Since knowledge workers know their information assets, an essential and integral requirement is to interview uses when assembling this inventory. It is with these same interviews that we identify and assemble the domain vocabulary. Discussions should also help identify early champions and possible project team members.

Recall that our system is capable of handling text and documents (*unstructured data*), mark-up documents and attribute-value pairs (*semi-structured data*), and *structured data* (database, spreadsheets, tables). Evaluate this possible content for consideration as part of the TBox knowledge graph or as new instance data (ABox). Different tests and checks apply to concepts and instances. Concept data comes from glossaries, tables of content, thesauri, sometimes bullet lists, or from more formal schema, such as hierarchies in spreadsheets or relational schema or ontologies. Text definitions, or links to encyclopedias, or links to specific Web pages, may be desirable content to add to the characterizations. You may uncover instance in infoboxes, spreadsheets, data tables, or text records with fixed fields. You should inspect the record form to identify the types to which the instances belong and their attributes. Favor complete structure, but gaps are OK given the open nature of knowledge. Text in the form of labels or *semset* entries that can accompany instance entries is desirable.

I do not advise beginning a KM project premised on paper conversion to digital. All first-iteration sources should be electronic, with the possible exception of subsumption hierarchies that you might obtain from paper listings or tables of content. If essential data only resides on paper, this kind of task should only be tackled in later increments after the basic system has justified itself. In the earliest phases of a project, avoid unusual formats or data that requires much wrangling or cleaning to stage for ingesting. Again, if essential, such sources can be tackled in later phases.

I do not advise beginning a KM project where security and access are a concern. I do recommend that proprietary and restricted access content be included in the initial inventory and interviewing steps. Early designs can anticipate possible security expansions, even though you may defer specifics and implementation. Since each implementation increment of the plan involves a new boundary for the domain, it is also appropriate that an updated inventory be conducted, perhaps putting on to the table sources that you chose to skip over in earlier phases.

These all constitute possible domain extensions. However, KBpedia and its 55,000 reference concepts and millions of organized instance data, is also available for free. Many of these also have links to text entries on Wikipedia or Wikidata, supplying that valuable content form. You may already find much, if not perhaps nearly all, of a starting skeleton for a given domain in the KBpedia structure.

Phased Implementation Tasks and Plan

Too many KM and business intelligence (BI) projects in the past have failed. The relational schema and its closed logic and brittleness have been one contributor to this record. Another reason for failure has been too-ambitious scope or expectations. By embracing open approaches to knowledge, we can also open up a development path that is phased and incremental. We can let the experience and results of prior phases justify new phases and expansions.

This philosophy fits well with a proof-of-concept approach, followed by staged and managed extensions. Repeating methods and continuing to refine tools as part of this phasing means we are climbing learning curves as more knowledge workers become exposed and facile in the use of the system. Expanding use and input helps provide continued knowledge and feedback into the plan and how we execute each incremental phase.

In a proof-of-concept phase, the least-effort path would be to leverage KBpedia or portions of it as is, make few changes to the knowledge graph, and populate and test local instance data. A next step may be to expand the knowledge graph with still more instances. As increments occur, consider more KM infrastructure for the system to accompany the expansion of domain scope.

Over time the plan should reflect its content and management pipeline. It is important to design the ability to swap in and out various options at multiple points from input to desired output. Then, because disparate sources and different formats must be accommodated, it is also important to use canonical syntaxes and standards for expressing the products and specifications at the various steps along that pipeline. The very notion of pipeline implies workflows, which are the actual drivers for how we design the pipeline. Evolve key workflow steps to include:

- Clean the input sources;
- Express the sources in a canonical form;[2]
- Identify and extract concepts;
- Map the structure to KB concepts;
- Identify and extract entities;
- Identify and extract relations;
- Type the entities, concepts, and relations;
- Extract attributes and values for identified entities;
- Add new import and export formats according to the needs of data interoperability and use of third-party analyses, machine learners, and tools;
- Test these against the existing KB;
- Update reference structures, including placement of the new assertions, as appropriate;
- Characterize and log to files;
- Commit to the KB, perhaps through formalized deployment steps; and
- Rinse and repeat.

Much information gets processed in these pipelines, and the underlying sources update frequently. Thus, the pipelines themselves should be designed for performance and based on solid code with appropriate workflow tagging and management.

Automation, within the demanding bounds of quality, is also an essential scalable condition. Functional programming languages align well with the data and schema in knowledge management functions. Ontologies, as structures, also fit well with functional languages. The ability to create DSLs (domain-specific languages) should continue to improve bringing the knowledge management function directly into the hands of its users, the knowledge workers. An essential design criterion is to have a methodology and workflow that explicitly accounts for interoperable and straightforward tools, following the scoping guidelines discussed in the previous chapter. You may need to include and justify specific tasking for any of these aspects in a given plan increment.

Over the timeframe of multiple increments for a phased project, consider clusters of work tasks to drawn upon for next increments:

Domain Knowledge Graph

You may start with KBpedia, though eventually, it is desirable to move toward a tailored domain knowledge graph. You may proceed to create the domain knowledge graph from prunings and additions to the base KBpedia structure, or from a more customized format such as the approach recommended in *Ontology Development 101*.[3] Some of your tasks in this area are to: determine the domain and scope of the ontology; incorporate domain terminology; consider reusing existing ontologies; enumerate important terms in the ontology; define the types and the class hierarchy, especially into typologies; and define the attributes of the types. After providing a preferred label, I encourage you to seek relevant alternative labels (for building the sem-sets).

The build methodology should re-use 'standard' ontologies as much as possible, to help promote interoperability. The 20 or so core and extended ontologies mapped to KBpedia are one starting point. To this base, you should add other commonly-used ontologies or those specific to your domain. You should make identification of these candidates an explicit part of the information inventory efforts. At a minimum, the ongoing working knowledge graph should conform to ontology building best practices (see *Chapter 14*) and complete enough such that it can be loaded and managed in an ontology editor or IDE. You can use this working structure with the OWL API for specialty tools and user maintenance functions.

Instance Data Population

Identifying, staging, transforming, incorporating, and vetting new instance data should be a continuous set of tasks for the installation. It is less risk to start with simple data formats populated with clean data. I suggest you cluster new, desired interfaces or translators with expansions into entirely new sources of instance data, such as from external sources or relational databases. Considerations like this can spread

needed development and tests over a complete project. A proper inventory of information assets will include file types and possible conversion tools for those types that may exist in the marketplace, preferably as open source. For example, a single conversion to the system's canonical format for a tool such as Tika can open up a thousand new data formats to the system. As a general guideline, it is much less effort and cost to investigate existing, available options, and then to adapt them to our data federation design, than it is to write converters from scratch.

For relational systems with large data stores, it may be justified to use third-party commercial tools for initial staging and conversion. We have had excellent experience with tools such as Safe Software's FME; many options exist for high-throughput situations or where updates are frequent.

Analysis and Content Processing

Each increment should target some form of analysis or content processing as its design objective. From the platform perspective, that means being able to select appropriate subsets from the knowledge base, process or transform them in some way, and then submit those results set to an external tool to conduct the designated work. Per the design philosophy, transformations or submittals of results sets should occur via an adequately scoped Web service. You should identify each new tool required for a given design objective, with integration part of the new tasking. Internal communications should also conform to the canonical data form. Some tasks may also require injecting analyzed results back into raw Web pages for display or visualization. Other tasks may need to expand Web pages to enable control and setting of tool parameters. You can also convert or export the information in various forms for direct use or incorporation into third-party systems.

You may drive visualization systems and specialized widgets using the results sets obtained from such queries or analysis, in which case you should include such in the task list. Our methodology also provides for administrative ontologies whose purpose is to relate structural understandings of the underlying data and data types with suitable end-use and visualization tools. You may therefore also need to consider tasks related to creating or modifying the administrative ontologies.

Use and Maintenance

The emerging knowledge system has practical uses including: search, querying, filtering, discovery, information federation, data interoperability, analysis, and reasoning. During use, you may discover many enhancements and improvements. Examples include improved definitions of concepts; expansions of synonyms, aliases and jargon (*semsets*) for concepts and instances; better, more intuitive preferred labels; better means to disambiguate between competing meanings; missing connections or excessive connections; and splitting or consolidating of the underlying structure. We want to see an evolution of tooling and incorporation into existing workflows such that we make these enhancements as encountered and without major work disrup-

tion. Today, practitioners most often do *not* pursue such maintenance enhancements because existing tools do not support such actions. Users and practitioners do not respond well to IDEs and tools geared to ontology engineering. A start small strategy, of course, lowers risk and is more affordable. However, for effectiveness, you must design an explicit strategy anticipating extension and expansion. Ontology growth thus occurs both from learning and discovery and from expanding the scope. Versioning, version control, and documentation (see below) therefore assume central importance as the system grows. Any of these items may form a nexus for work tasks in a given increment of the plan.

Testing and Mapping

As we generate new ontologies, we should test them for coherence against reasoning, inference, and other natural language processing tools. We also use gap testing to discover holes or missing links within the resulting ontology graph structure. Gap testing helps identify internal graph nodes needed to establish the integrity or connectivity of the concept graph. We may use coherence testing to find missing or incorrect axioms. Though used for different purposes, we may also use mapping and alignment tools to identify logical and other inconsistencies in definitions or labels within the graph structure. Mapping and alignment help establish the links that help promote ontology and information interoperability. We ask external knowledge bases to play crucial roles in testing and mapping. Depending on the phase, you may need to include such tasks for a given plan increment. Mapping is not always a part of a given increment. However, testing should be a part of all of your increments. Include unit tests for all new tools and converters or further target analyses.

Documentation

Ontologies give us as a way to capture the structure and relationships of a domain — which is also always changing and growing. We can use further use ontologies to document their development and versions. We need to apply better tools — such as vocabulary management and versioning — and better work processes to capture and record use of our ontologies. We can handle some of these aspects with utilities such as OWLdoc or wikis for standard knowledge capture and documentation. We have innovated many connectors to capture ontology knowledge bases on an ongoing basis. Still, these are rudimentary steps that we need to enforce with management commitment and oversight. Ongoing use and training demand that we adequately document the knowledge graphs, ontologies, tools, scripts, and instance record sources that support a given knowledge installation. Given the lack of tools or best practices in this area, you will need to commit to and monitor documentation.

MAPPING SCHEMA AND KNOWLEDGE BASES

Two critical work areas in tailoring your implementation are in building out the

schema (knowledge graph) and populating your installation with instance data. Various mapping methods and tools aid these two work areas. Given their importance, let's spend a bit of time discussing these work areas in more detail.

Mapping Methods and Tools

Mapping is the definition of a formal correspondence of objects in one knowledge source with objects in another knowledge source, with the latter most often being the reference knowledge graph. The correspondence takes the form of assigning a specific predicate linking an object in an external source to its subject in a reference source. Some mapping is straightforward; other mappings may be quite hard due to the vagaries of language and context. Mapping involves specific methods and algorithms to propose candidate matches, as well as tools or applications that embed these methods in user interfaces and workflows, often with the intent of supporting the broader mapping purpose. By making the reference knowledge graph the target, we only need to test the updated graph for coherency and consistency. The reference knowledge graph grows and changes shape and scope over time as new domain information is incorporated. Properly mapped external sources can become an integral part of the domain knowledge graph and participate in inferencing and other reasoning tasks.

Though some tout complete automation of mapping as desirable, there is no such thing, and even small assignment error rates can translate into noticeable errors in the knowledge base.[4] For this reason, we support what we call a 'semi-automatic' approach to mapping. The method involves using multiple methods to score potential matches, perhaps differentially weighted, and ultimately reviewed and vetted by human editors before acceptance into the system. The individual review steps are what make the approach 'semi-automated,' though to make that process efficient, it is also useful to automate away clear mismatches and other problems before the human review of candidates. By automating the process to reduce easily recognized non-candidates and score only candidates via the differing methods, we can reduce the number of uncertain candidates editors need to review. We can also apply this method for screening candidates for supervised machine learning. Efficiencies and learning curves should be fed back into the screening tools so that reviewers believe their input is valued and gets reflected in constantly improving tools, two unarguable objectives when mounting a knowledge management initiative.

The mapping methods are varied and tend to reflect the same broad clusters of semantic heterogeneities as provided by *Table 5-1*. We may use various ways to classify these mapping types, but the central options tend to focus on schema, labels/lexical, labels/definitions/semantics, instances, relations, machine learning, or mediated, by using external KBs or thesauri or WordNet. The most straightforward approaches look only at labels and propose various kinds of string matches. Better ones look at attributes, external relations, subsumption hierarchies, and the semantics of labels and concepts. Some of the tools provide multiple methods, and the user may combine or not multiples of them with user-assigned weights.

The heyday of tools developed in the areas of *ontology alignment*, or *mapping*, or *matching*, was in the 2000 to 2005 timeframe. Still, the sophistication and usability of these tools have continued to improve, even if the pace of new offerings has slowed. A major driver for these advances has been the annual OAEI (Ontology Alignment Evaluation Initiative) conference, which has provided a competitive contest and established evaluation test sets and criteria on a yearly basis since 2004. My recent survey specific to ontology mapping identified 30 different existing mapping tools, many embracing multiple methods, and most open source.[5]

Building Out the Schema

If you ask most knowledgeable enterprise IT executives what they understand ontologies to mean and how to use them, you would likely hear that ontologies are expensive, complicated and challenging to build. Reactions such as these (and not trying to set up strawmen) result from the relative lack of guidance on how one builds and maintains these beasties. The use of ontology design patterns is one helpful approach.[6] Such patterns help indicate best design practice for particular use cases and relationship patterns. However, while such patterns should be part of a general methodology, they do not themselves constitute a methodology.

The focus here is on *domain ontologies*, which are descriptions of particular subject or domain areas. The last known census of ontologies in 2007 indicated there were more than 10,000 then in existence, though today's count is likely in excess of 40,000.[7] Because of the scope and coverage of these general and domain representations, and the value of combining them for specific purposes, *ontology alignment* has been a topic of practical need and academic research. According to Corcho *et al.*[8] "a domain ontology can be extracted from special purpose encyclopedias, dictionaries, nomenclatures, taxonomies, handbooks, special scientific languages (say, chemical formulas), specialized KBs, and from experts." Another way of stating this is to say that a domain ontology — adequately constructed — should also be a faithful representation of the language and relationships for those who interact with that domain.

Overview of Approaches

There is a spectrum of approaches for how to conduct these mappings. At the simplest and least accurate end of the spectrum is string matching methods, sometimes supplemented by regular expression processing and heuristic rules. An intermediate set of methods uses concepts already defined in a knowledge base as a way to 'learn' representations of those concepts; while many techniques exist, two common ones are explicit semantic analysis and word embedding. Most of these intermediate methods require some form of supervised machine learning or other ML techniques. At the more state-of-the-art end of the spectrum are graph embeddings or deep learning, which also capture context and conceptual relationships as codified in the graph.

Aside from the string match approaches, all of the intermediate and state-of-the-art methods use machine learning. Depending on the method, these machine learn-

ers require developing either training sets or corpora as a reference basis for tuning the learners. These references should be manually scoped, as in the case of training corpora for unsupervised learning, or manually scored into true and false positives and negatives (labeled results) for training sets for supervised learning. All of these techniques are useful, but you should, in any case, supplement them with logic tests and scripts to test coherence and consistency issues that may arise. You should test the coherency of the target knowledge graph after any new mappings.

Practitioners of ontology development have been documenting approaches since at least Jones *et al.* in 1998.[9] That early study outlined common steps and noted typical stages to produce first an informal description of the ontology and then its formal embodiment in an ontology language. The existence of these two descriptions is an important characteristic of many ontologies, with the informal description often carrying through to the formal description. Corcho *et al.* did the next major survey in 2003.[8] This built on the earlier Jones survey and added more recent methods. The survey also characterized the methods by tools and tool readiness. More recently the work of Simperl and her colleagues has focused on empirical results of ontology costing and related topics. This series has been the richest source of methodology insight in recent years.[10] [11] [12] Though not a survey of methods, one of the more attainable descriptions of ontology building is Noy and McGuinness' well-known *Ontology Development 101.*[3]

Another way to learn more about ontology construction is to inspect some existing ontologies. Though one may use a variety of specialty search engines and Google to find ontologies,[13] some current repositories also deserve inspection. Examples include the University of Manchester, VIVO, TONES, the Protégé ontology library, the Linked Open Vocabularies (LOV), the NanJing Vocabulary Repository, the Online Ontology Set Picker (OOSP), and the OBO (biomedical) Foundry. An older, but similar, repository is OntoSelect. Another way to learn about ontology construction is from a bottom-up perspective. In this regard, the Ontology Design Patterns (ODP) wiki is a source of building patterns and exemplary ontologies. ODP is not likely the first place to turn to and does not give 'big picture' guidance, but it also should be a bookmarked reference once you begin real ontology development.

For the last twenty years, there have been many methods put forward for how to develop ontologies. Though new methodology developments have diminished somewhat in recent years, our reviews suggest this is the current state of ontology development methodologies:

- Very few uniquely different methods exist, and those that do are relatively older in nature;
- The methods tend to either cluster into incremental, iterative ones or those more oriented to comprehensive approaches;
- There is a general logical sharing of steps across most methodologies from assessment to deployment and testing and refinement;
- Actual specifics and flowcharts are quite limited; except the UML-based sys-

tems, most appear not to meet enterprise standards;

- Discussion of supporting toolsets is often lacking, and most of the examples, if even provided, are based solely on a single or governing tool. Tool integration and interoperability is almost non-existent in narratives; and
- Development methodologies are not as active an area of recent research.

While there is by no means unanimity in the community, we can see some consensus from these prior reviews.[14] We have taken these consensus items and added to them some points from our experience, resulting in these eight general guidelines for what you should consider in a domain ontology:

- Be lightweight and modular;
- Use reference structures;
- Re-use existing structure;
- Build incrementally;
- Use simple predicates;
- Test for logic and consistency;
- Map to external ontologies; and
- Map reciprocally.

I expand further on these points in the next sections.

Some Design Guidelines

Effective ontology development is as much as anything a matter of mindset. This mindset is grounded in leveraging what already exists, 'paying as one benefits' (see below) through an incremental approach, and starting simple and adding complexity as we gain understanding and experience. Inherently this approach requires domain users to drive ongoing development with appropriate tools to support that emphasis. Ontologists and ontology engineering are important backstops, but not in the lead design or development roles. The net result of this mindset is to develop pragmatic ontologies that are understood — and used by — actual domain practitioners. Let's look more closely at the individual design guidelines just listed to see what goes into this mindset.

1. Be Lightweight and Modular

Begin with a *lightweight, domain ontology,*[15] which is hierarchical or classificatory in nature. Ontologies built for the pragmatic purpose of interoperating different contexts and data should start lightweight with only a few predicates, such as `subClassOf`, `isAbout`, `narrowerThan` or `broaderThan`. If done properly, these lighter weight ontologies with more limited objectives can be surprisingly robust in discovering connections and relationships. Moreover, they are a logical and doable intermediate step on the path to more demanding semantic analysis. Because we have this perspective, we also tend to rely heavily on the SKOS vocabulary for many of our ontol-

ogy constructs[16] and use *typologies* in our overall design

Provide *balanced coverage* of the subject domain. The breadth and depth of the coverage in the ontology should be roughly equivalent across its scope. Build *modular* ontologies that split your domain and problem space into logical clusters. Try to *split domain concepts from instance records structurally*. Concepts represent the nodes within the structure of the ontology (also known as classes, types, or the *TBox*). Instances represent the data that populates that structure (also known as entities, individuals, or the *ABox*). Use *disjoint classes* to separate classes from one another where the logic makes sense, and let dissimilarities guide the bounding of types in the first place. An architecture of multiple ontologies often works together to isolate different work tasks to aid better ontology management. Also, try to use a core set of *primitives* to build up more complex parts. This approach is a kind of reuse within the same ontology, as opposed to reusing external ontologies and patterns. The corollary to this is that the same concepts should not be created independently multiple times in different places. Adhering to these practices is akin to object-oriented programming.

Try to think of your knowledge graph as also providing context, by explicitly considering what the best way is to describe what your content 'is about.' A good gauge for whether the context is adequate is whether one has sufficient concept definitions to disambiguate common concepts in the domain. As we add relationships and the complexities of the world get further captured, ontologies migrate from the lightweight to a more 'heavyweight' end of the spectrum.

2. Use Reference Structures

One benefit is that reference structures of any kind provide a focus, by definition, of common or canonical referents. This commonality leads to better defined, better understood and more widely used referents. Common referents become a kind of common vocabulary for the space, upon which other vocabularies and datasets can depend. A common language, of sorts, can begin to emerge. Reference structures also provide a grounding, a spoke-and-hub design, that leads to an efficient basis for external vocabularies and datasets to refer to one another. Of course, any direct mapping can provide a means to relate this information, but such pairwise mappings are not scalable nor efficient. In a spoke-and-hub design, the number of mappings required goes down significantly with the number of datasets or items requiring mapping. The spoke-and-hub design,[17] for example, is at the heart of such disciplines as master data management. Another benefit of common reference structures is that they provide a common target for the development of tools and best practices. These kinds of 'network effects' lead to still further tooling and practices.

3. Re-use Existing Structure

Reuse structure and vocabularies as much as possible. Fundamental to the whole concept of coherence is the fact that domain experts and practitioners have been looking at the questions of relationships, structure, language, and meaning for decades. Massive time and effort have already been expended to codify some of these

understandings in various ways and at multiple levels of completeness and scope. A short list of these potential sources demonstrates the treasure trove of structure and vocabularies available to any enterprise for re-use: Web portals; databases; relational database schema; industry specifications and standards; spreadsheets; informal lists; legacy schema; metadata; taxonomies; controlled vocabularies; ontologies; master data (MDM) directories and catalogs; exchange formats, etc. Metadata and available structure may have value no matter where or how it exists, and a fundamental aspect of the build methodology is to bring such candidate structure into a standard tools environment for inspection and testing. It is wasteful to ignore prior investments that have been used to characterize or organize information assets.

We closely relate this guidance to our earlier advocacy that you should accompany each incremental phase of development with an update to the information inventory. The most productive methodologies for modern ontology building are those that re-use and reconcile prior investments in structural knowledge, not ignore them. These existing assets take the form of already proven external ontologies and internal and industry structures and vocabularies. Besides assembling and reviewing current sources, those selected for re-use must be migrated and converted to a proper ontological form (OWL in our case). Others have demonstrated some of these techniques for prior patterns and schema.[18] [15] In other instances, you may employ various converters or scripts to conduct the migration. Many tools and options exist at this stage, even though as a formal step, practitioners often neglect this conversion.

4. Build Incrementally

Build ontologies *incrementally*. Much value can be realized by starting small, being simple, and emphasizing the pragmatic. It is OK to make those connections that are doable and defensible today while delaying until later the full scope of semantic complexities associated with complete data alignment. An open world approach provides the logical basis for incremental growth and adoption of ontologies. You need to repeat the process of modifying a working ontology, testing it, maintaining it, and then revising and extending it over multiple increments. In this manner, the deployment proceeds and gets refined as learning occurs. Importantly, too, this approach also means that complexity, sophistication, and scope only grow consistent with demonstrable benefits. Thus, in the face of typical budget or deadline constraints, you may initially scope domains smaller or provide less coverage in depth or use a smaller set of predicates, all the while still achieving productive use of the ontology.

5. Use Simple Predicates

Define unambiguous *predicates* (also known as properties, relationships, attributes, edges or slots), including a precise definition. Then, when relating two things to one another, use care in actually assigning these properties. Initially, assignments should start with a logical taxonomic or categorization structure and expand from there into more nuanced predicates. Though not involved in any reason-

ing, aggressively use *annotation properties* to promote the usefulness and human readability of the ontology, as well as to provide text support for the better characterization of entities and concepts.

Assign *domains* and *ranges* to your properties. Domains apply to the subject (the left-hand side of a triple), ranges to the object (the right-hand side of the triple). You should not view domains and ranges as real constraints, but as axioms used by reasoners. In general, the domain for a property is the range for its inverse and the range for a property is the domain of its inverse. (You can envision this by understanding that domain applies to the subject, while range applies to the object. If you invert these roles, domain and range switch.) Use of domains and ranges will assist testing and help ensure the coherency of your ontology. Assign *property restrictions*, but do so sparingly and judiciously. Use of property restrictions will also support testing and provides possibly new features to machine learners.

6. Test for Logic and Consistency

We must always test our knowledge graphs for logic, consistency, completeness, and coherence. Test each increment; no official or public release should be made that does not pass all tests. As we learn, we should continue to add to the comprehensiveness of our tests. We test logic as we build with inference engines and reasoners. We look for completeness and consistency regarding standard ontology errors, such as what the tool OOPS! helps identify,[19] and follow our best practices for completeness and the use of *semsets*.

The essence of *coherence* is that it is a state of logical, consistent connections, a logical framework for intelligently integrating diverse elements. So while context supplies a reference structure, coherence means that the structure makes sense. Is the hip bone connected to the thigh bone, or is the skeleton askew? Coherence means that we draw the right connections (edges or predicates) between the right object nodes (or content) in the graph. Relating content coherently itself demands a coherent framework. At the upper reference layer, this begins with KBpedia, which begins as a coherent structure. If KBpedia continues as the basis for the modified domain ontology, and if incremental changes are tested for logic and consistency as they occur, then you should be able to continue to evolve the domain knowledge graph coherently. Absent starting reference structures, it is tough to create a cohesive starting knowledge graph, since any new assertion may not have been encountered in a related form before.

7. Map to External Ontologies

Mapping to external ontologies increases the likelihood of sharing and interoperability, but importantly from an ontology building perspective, also helps to identify gaps or errors in the reference knowledge graph. Mapping helps expose the importance of 'punning,' since depending on use or context, we may want to treat a given concept as either a class or instance. Given our domain and our interoperability goals, we likely want to rely on a *set of core ontologies* for external re-use purposes. For

interoperability purposes, we also want to write our ontologies in *machine-processable languages* such as OWL or RDF Schema.

Building Out the Instances (Knowledge Bases)

The conceptual and logical demands for adding instances are different in scope and kind than that for the conceptual knowledge graph. When adding instances, ensure the quality of the input data with reliable provenance; you may be required to justify your sources. An attributes ontology, embedded as one of the backbones in KBpedia, is a useful starting place to map data attributes and characteristics. We grow and mature the reference structure for this using similar considerations as to what we followed for the overall knowledge graph, including logic and consistency tests (though they will be of a different character, more akin to data validation). When adding instances, it is essential you relate all entities to a type and pay attention to other aspects of the instance's data record that may be useful to include as disambiguation cues.

In building out and then using instance data, we can see a cycle of ten or so broad guidelines. Note that I refer to the input instance source as a knowledge base, though, of course, any instance data repository may be a source. A relational data store, for example, would follow these guidelines, but also would need to go through some form of relational to RDF converter. Other types of data stores may impose similar wrinkles.

Here are the ten guidelines for building out instances:

1. Update Changing Knowledge

We need to ensure that the input knowledge bases to the overall domain knowledge structure are current and accurate. Don't start with dated material! Depending on the nature of the KM system, there may be multiple input KBs involved, each demanding updates. Besides capturing the changes in the base information itself, many of the steps below may also be required to process this changing input knowledge correctly.

2. Process the Input KBs

Process the input KBs to be machine-readable. We also desire processing to expose features for machine learners and to do other clean up of the input sources, such as removal of administrative categories and articles, cleaning up category structures, consolidating similar or duplicative inputs into canonical forms, and the like. This step is highly contextual, and may require multiple steps or scripting.

3. Install, Run and Update the System

The KBs themselves reside on their host databases or triple stores. Each of the processing steps may have functional code or scripts associated with it. All general

management systems should be installed, kept current, and secured. The management of system infrastructure sometimes requires a staff of its own, let alone install, deploy, monitoring and update systems. It is here that we may need to add specific source converters to the system.

4. Test and Vet Placements

New entities and types added to the knowledge base should be placed into the overall knowledge graph and tested for logical placement and connections. Though we should manually verify final placements, the sheer number of concepts in the system places a premium on semi-automatic tests and placements. Placement metrics are also valuable to help screen candidates. This task area requires similar tools and user interfaces, plus incorporation into existing workflows, as is required for concept placements into the governing knowledge graph.

5. Test and Vet Mappings

If we add new types or concepts to the governing knowledge graph, then these should be tested and mapped with appropriate mapping predicates to external or supporting KBs. Any new mappings to the base KB should be re-investigated and confirmed.

6. Test and Vet Assertions

Testing does not end with placements and mappings. Attributes and values often characterize concepts; sometimes we may give them internal assignments as SuperTypes; and, we must test all new assertions against what already exists in the KB. Though the tests may individually be straightforward, thousands may require testing, and cross-consistency is vital if one is adding large instance stores. Each of these assertions is subject to unit tests.

7. Ensure Completeness

Our standard practice calls to accompany each new concept in the KB with a definition, complete characterization and connections, and synonyms or semsets to aid in natural language tasks. If updates are periodic or scheduled, as opposed to one-time batch incorporation, then we recommend writing scripts for the appropriate tests. Any activity that we can reasonably anticipate to occur three times or more deserves scripting attention.

8. Test and Vet Coherence

As we build and extend the broader structure, we apply system tests to ensure the overall graph remains coherent. We address and correct outliers, orphans, and fragments when encountered. We do some of this testing via component typologies, and some we do using various network and graph analyses. You should flag possible

problems and document or present them for manual inspection. Like other manual vetting requirements, confidence scoring and ranking of issues and candidates helps speed up this screening process.

9. Generate Training Sets

A key objective of populating our knowledge system with instance data is to enable the rapid creation of positive and negative training sets for machine learning. We need to generate candidates; they should be scored and tested; and, we need to vet their final acceptance. Once vetted, we may need to express the training sets in different formats or structures (such as finite state transducers, one of the techniques we often use) for them to perform well in actual analysis or use. Since machine learners may require many iterations to refine input parameters, your scripting attention is certainly required here.

10. Test and Vet Learners

We can then apply machine learners to the various features and training sets produced by the system. Each learning application involves the testing of one or more learners; the varying of input feature or training sets; and the testing of various processing thresholds and parameters (including possibly white and blacklists). This set of requirements is one of the most intensive on this listing, and requires you to document test results, alternatives tested, and other observations useful to a cost-effective application.

Rinse and Repeat

Each of these ten steps is not a static event. Instead, given the constant change inherent in knowledge sources, including the ongoing addition of new instances, we must repeat the entire workflow on a periodic basis. The inexorable pull is to automate more steps and generate more documentation to reduce the tension between updating effort and current accuracy. A lack of automation leads to outdated systems because of the effort and delays in updates. The imperative for automation, then, is a function of the change frequency in the input KBs or the use of learners.

'PAY AS YOU BENEFIT'

As best as I can tell, Alon Halevy was the first to use the phrase 'pay as you go' in 2006 to describe the incremental aspect of the open world approach applied to the semantic Web.[20] Others had earlier applied the 'pay as you go' phrase to data management and storage; it had also been used to describe phone calling plans. Unfortunately, the 'pay as you go' phrase has (and still is) largely confined to incremental, open world approaches involving the semantic Web. Nonetheless, I like the phrase, and I think it evokes the right mindset. In fact, I think with linked data and many other aspects of the current semantic Web we see such approaches come to fruition.

Inch-by-inch, brick-by-brick, we see useful data on the Web getting exposed and interlinked. 'Pay as you go' is incremental, and that is good.

Still, I think we can express this idea better. The idea of 'pay as you benefit' more directly ties the question of project funding and project staging to project benefits. It ties directly into the open nature of knowledge and dovetails nicely with the repeated recommendations to implement your knowledge management initiatives incrementally. The idea of 'pay as you benefit' is purposeful, and may be planned and implemented on standard enterprise cost-benefit principles.* What the 'pay as you benefit' idea means is you can start small and be incomplete. You can target any domain or department or scope that is most useful and illustrative for your organization. You can deploy your first stand-ups as proofs-of-concept or sandboxes. Moreover, you can build on each prior step with each subsequent one. Of course, you must communicate with stakeholders to get this message out and to overcome the glazed eyes that might accompany the terminology of knowledge management and ontologies. 'Pay as you benefit' is a guiding pragmatic principle for how you can build out your domain knowledge management system. So, how does one move ahead with a 'pay as you benefit' strategy?

Placing the First Stake

The first step is always the hardest on a new journey. We can minimize risk by planning an incremental roll-out and scoping and bounding our first step carefully, but it is still important the first step be successful to move the journey forward. I have discussed elsewhere the wisdom of designing the first step for success, and to limit unneeded or risky development. Leveraging existing KBpedia assets as supplemented by your domain instance data is one way to bound this risk.

The players in the first step of a KM initiative should be those with a need and who are supportive. It is perhaps essential that the initial team include champions, who are smart and willing to learn. We need to spread the seeds of knowledge management on fertile ground, which also has some visibility to other portions of the organization. We almost assuredly bake in failure when we attempt such initiatives too broadly or without local support. Because of the shortcomings of past 'solutions' such as BI or data warehousing, we also see a decline and a reluctance for IT to embrace new and transforming approaches. These considerations argue strongly for embedding first stakes in a KM project within a department or group directly involved in knowledge work or management. KM projects are almost always of some threat to IT departments as they presently understand their role. As a general rule, do not attempt to start KM projects there, and expect resistance and naysaying from some in IT.

* Including, of course, explicit attempts to model intangible benefits realistically.

Incremental Build Outs Follow Benefits

We make much of 'incremental' or 'agile' deployments within enterprises, but the nature of the traditional data system (and its closed world assumption) can act to undermine these laudable steps. The inherent nature of an open world approach, matched with methodologies and best practices, can work wonderfully with KM-related projects. We have seen how we can incrementally stage our phases, moving into more complicated and enterprise-visible areas over time.

Learn to Quantify and Document Benefits

The grounding of a KM system in the information that knowledge workers have, how they presently conduct their work, and what they need to improve it, provides the same bases for documenting benefits from a new initiative. You should document current practices to capture and model workflows, and you should record time and effort associated with ongoing work tasks. These are the required metrics to show whether KM initiatives are improving productivity or not and, if so, by how much. (Of course, you need to measure and document benefits as well.) These kinds of considerations should be central in the design of a KM initiative because, without you collecting and monitoring such data, it will be impossible to project the documented savings and improvements needed to justify ongoing commitments. James Hendler once stated that "a little semantics goes a long way."[21] That truth — and it is true — when combined with incremental deployment firmly tied to demonstrable results, promises a different way to do business.

Chapter Notes

1. Some material in this chapter was drawn from the author's prior articles at the *AI3:::Adaptive Information* blog: "Open SEAS: A Framework to Transition to a Semantic Enterprise" (Mar 2010); "'Pay as You Benefit': A New Enterprise IT Strategy" (Jul 2010); "A Brief Survey of Ontology Development Methodologies" (Aug 2010); "A New Methodology for Building Lightweight, Domain Ontologies" (Sep 2010); "Research Shows Natural Fit between Wikipedia and Semantic Web" (Oct 2008); "Shaping Wikipedia into a Computable Knowledge Base" (Mar 2015); "Reciprocal Mapping of Knowledge Graphs" (Feb 2017).
2. Galárraga, L., Heitz, G., Murphy, K., and Suchanek, F. M., "Canonicalizing Open Knowledge Bases," ACM Press, 2014, pp. 1679–1688.
3. Noy, N. F., and McGuinness, D. L., *Ontology Development 101: A Guide to Creating Your First Ontology*, Stanford University Knowledge Systems Laboratory, 2001.
4. As *Chapter 14* explains, an F1 score of 95% is still based on an annotator agreement basis of perhaps 70-80%, which means an actual F1 score error rate of, say, 65%. With 10 million assertions, this translates into as many as 3.5 million being in error. Were actual F1 scores at 95%, that still means 500,000 errors.
5. Bergman, M. K., "30 Active Ontology Alignment Tools," *AI3:::Adaptive Information* Available: http://www.mkbergman.com/?p.
6. OntologyDesignPatterns.org (http://ontologydesignpatterns.org/wiki/Main_Page) is a semantic Web portal dedicated to ontology design patterns (ODPs). The portal was started under the NeOn project in 2009.
7. A simple Web search of https://www.google.com/search?q=filetype:owl (OWL is the Web Ontology Language, one of the major ontology formats) shows nearly 39,000 results. Still, multiple ontology languages

are available, such as RDF, RDFS, and others (though use of any of these languages does not necessarily imply the artifact is a vocabulary or ontology).

8. Corcho, O., Fernandez, M., and Gomez-Perez, A., "Methodologies, Tools and Languages for Building Ontologies: Where is the Meeting Point?," *Data & Knowledge Engineering 46*, 2003.
9. Jones, D. M., Bench-Caponand, T. J. M., and Visser, P. R. S., "Methodologies for Ontology Development," *Proceedings of the IT and KNOWS Conference of the 15th FIP World Computer Congress*, 1998.
10. Simperl, E. P. B., and Tempich, C., "Ontology Engineering: A Reality Check," *On the Move to Meaningful Internet Systems*, Springer, 2006, pp. 836–854.
11. Simperl, E., Tempich, C., and Vrandečić, D., "A Methodology for Ontology Learning," *Frontiers in Artificial Intelligence and Applications 167 from the Proceedings of the 2008 Conference on Ontology Learning and Population: Bridging the Gap between Text and Knowledge*, 2008, pp. 225–249.
12. Simperl, E. P. B., Tempich, C., and Sure, Y., "ONTOCOM: A Cost Estimation Model for Ontology Engineering," *The Semantic Web - ISWC 2006*, I. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L.M. Aroyo, eds., Berlin, Heidelberg: Springer, 2006, pp. 625–639.
13. Specialty search engines for ontologies include Swoogle, FalconS, Watson, Sindice, and SWSE. In addition, one can use a general search engine such as Google with a search query such as <topic> owl:equivalentClass filetype:owl. Note the filetype might also include RDF or a variant such as N3, we can substitute other language-specific constructs of interest for owl:equivalentClass.
14. Simperl, E., Mochol, M., and Burger, T., "Achieving Maturity: the State of Practice in Ontology Engineering in 2009," *International Journal of Computer Science and Applications*, vol. 7, 2010, pp. 45–65.
15. Giunchiglia, F., Marchese, M., and Zaihrayeu, I., "Encoding Classifications into Lightweight Ontologies," *Proceedings of the 3rd European Semantic Web Conference (ESWC*, 2006.
16. *SKOS Simple Knowledge Organization System Reference: W3C Recommendation*, World Wide Web Consortium, 2009.
17. The main advantage of a grounding reference is that it allows a spoke-and-hub design for data mapping, which is tremendously more efficient than pairwise mappings. In a spoke-and-hub design, where the reference ontology is the common node at the hub, only n – 1 routes are necessary to connect all sources, meaning that it scales linearly with the number of sources and attributes. Without a grounding reference, these same mapping capabilities would require routes in a pairwise (point-to-point) approach, which also scales poorly as a quadratic function. A system of ten datasets would require n(n-1)/2 composite mappings in the reference grounding case, but 45 in a pairwise approach. Of course, datasets themselves contain tens to thousands of attributes, compounding the map scaling problem further.
18. van Assem, M., Malaisé, V., Miles, A., and Schreiber, G., "A Method to Convert Thesauri to SKOS," *The Semantic Web: Research and Applications*, Y. Sure and J. Domingue, eds., Berlin, Heidelberg: Springer, 2006, pp. 95–109.
19. Poveda Villalón, M., "Ontology Evaluation: A Pitfall-Based Approach to Ontology Diagnosis," Ph.D., Universidad Politécnica de Madrid, ETSI_Informatica, 2016.
20. Halevy, A., Franklin, M., and Maier, and D., "Principles of Dataspace Systems (PODS)," *Proceedings of ACM Symposium on Principles of Database Systems*, 2006, pp. 1–9.
21. James Hendler, "a little semantics goes a long way." See http://www.cs.rpi.edu/~hendler/LittleSemanticsWeb.html.