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. Platforms and Knowledge Management. in *A Knowledge Representation Practionary: Guidelines Based on Charles Sanders Peirce* (ed. Bergman, M. K.) 251–272 (Springer International Publishing, 2018). doi:10.1007/978-3-319-98092-8_12

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

Full-text: http://www.mkbergman.com/publications/akrp/chapter-12.pdf

Abstract: One can create a proper enterprise knowledge management environment at acceptable cost using available open-source components and solid architectural design. Component services provide ontology and knowledge management functions in piecemeal functionality that we can integrate directly into existing workflows. These requirements mean that use and updates of the semantic technology portion, the organizing basis for the knowledge in the first place, must be part of daily routines and work tasking, subject to management and incentives.

12

PLATFORMS AND KNOWLEDGE MANAGEMENT

Having discussed terminology and components in previous chapters, now let us turn our attention in this *Part IV* to building an actual knowledge representation *system*. The major theme of the effort is to obtain maximum value from the work of converting and integrating data not only to achieve the aims of *data interoperability* and *knowledge-based artificial intelligence* (KBAI) but to leverage maximum benefits from *knowledge management* as well. We follow this chapter on platforms with two additional chapters in *Part IV* on how to build out and tailor a system for your own domain needs and on testing and best practices.

The material in these three chapters draws on our experience in building semantic technology platforms for a variety of clients and applications over the prior decade.[1] In various guises and tailorings, we have created standalone and Drupal-based platforms using PHP, and have created standalone systems using the Clojure language. Though we have released portions of these efforts as open source — Clojure components related to KBpedia, and PHP and Drupal frameworks for the Open Semantic Framework (OSF) — we are not prescriptive in this chapter or elsewhere in the book about how to build a KR/KM platform. Rather, we emphasize guidelines and lessons learned versus any specific design or language. Platforms will continue to emerge and evolve, and what we should seek from those platforms regarding design and architecture is of more guiding importance than any specific instantiation.

We begin this chapter by critically reviewing the work objectives of a platform. These functional understandings are related to the earlier *TBox* and *ABox* splits we discussed for *description logics* in *Chapter 8.* We also discuss the importance of content and general workflows. From this basis, we then proceed to look at platform considerations. As noted in *Chapter 4*, the platform should support three main opportunities in general *knowledge management*, *data interoperability* and *knowledge-based artificial intelligence* (KBAI). We also discuss access control and governance, and other enterprise considerations. The last section of this chapter deals with the overall *Web-oriented architecture*, emphasizing the importance of Web connectivity and the use of modular

Web services for scalability and flexibility. The entirety of these considerations helps us set the overall guidelines for the design and architecture for a responsive knowledge representation and management platform.

USES AND WORK SPLITS

To contemplate what a knowledge representation platform should look like, we first need to define what kinds of work we anticipate the platform to do. These work requirements are related to the purposes we have for the platform, as well as the existing state of tooling and applications available to support them. (*Chapters 15* and *16* offer additional use cases.) Workflows are also intimately tied to these questions.

The State of Tooling

I have been tracking and documenting the state of semantic technology, graphics visualization, and knowledge management tooling for nearly two decades. For many years I maintained *Sweet Tools*, a searchable and faceted compendium of semantic technologies that grew to a listing exceeding 1000 tools, the most comprehensive available.[2] In our platform work, we have used and integrated some of the leading tools available from this listing. We have also extended and created many of our tools and ontologies that we have contributed back to the community as open source.[3]

We now have much tooling and demo experience to draw upon since the seminal article on the semantic Web appeared in the *Scientific American* in 2001.[4] The primary sources for supporting the semantic Web are the European Union, mostly for academics, and the US government, mainly for intelligence and biomedical purposes to academics and businesses alike.

In the early years, ontology standards and languages were still in flux, and the tools basis was similarly immature. Frame logic, description logics, common logic and many others were competing at that time for primacy and visibility. Practitioners based most ontology tools at that time such as Protégé,[5] OntoEdit,[6] or OilEd[7] on F-logic or the predecessor to OWL, DAML+Oil. The emergence of OWL and then OWL 2 by the W3C helped solidify matters. The University of Manchester introduced the OWL API,[8] which now supports OWL 2.[9] Protégé, in version 5x, is now solely based on OWL 2 and has become a popular open source system, with many visualization and OWL-related plug-ins. A leading commercial editor is TopBraid Composer, which uses the Eclipse IDE platform and Jena API.[10] The OWL API is now a standard used by Protégé and leading reasoners (Pellet, HermiT, FaCT++, RacerPro). It supports a solid ontology management and annotation framework, and validators for various OWL 2 profiles (RL, EL, and QL).

RDF data management systems, or 'triple stores,' such as OpenLink's Virtuoso, Ontotext's GraphDB, and Franz's AllegroGraph, are now mature offerings. One may also apply modifications of existing data stores by Oracle, MarkLogic, and a variety of NoSQL databases to the design ideas presented herein. Developers presently have

multiple open source and commercial options to choose from, including cloud options such as Amazon's Neptune, for hosting RDF and OWL databases. The more comprehensive frameworks have opted to become ontology-engineering environments and to provide all capabilities in one box via plug-ins.

Java is the language of choice for about half of the semantic technologies, though existing toolsets use more than a score of different languages. Academic tools are often the most innovative, but the degree of completeness is often frustrating and most academic and grant-supported tools have limited or no support. Many, after a single experimental release, are abandoned or see no further development. Newer academic releases (often) are more strategically oriented and parts of broader programmatic emphases. Programs like AKSW from the University of Leipzig or the Freie Universität Berlin or Finland's Semantic Computing Research Group (SeCo), among many others, are exemplars of this trend. Promising projects and tools are now much more likely to be spun off as potential ventures, with accompanying better packaging, documentation and business models.

Full-text search is weak in RDF triple stores, and many leading approaches now match a text engine with the semantic portions. Some excellent components exist, but not yet packaged into single-stop solutions as RedHat did with Linux. The ontology tooling is especially difficult for standard knowledge workers to use, and the coupling of tools into current, actual workflows is lacking. Our experience is that most potential components are incompletely tested, and lack many basic expectations suitable for enterprise environments. Much scripting is necessary to glue together existing parts. However, some of the design guidance provided herein, especially about the use of canonical data forms, Web services, and suitable modular architectures, can help overcome many of these problems. It is possible to create a proper enterprise knowledge management environment at acceptable cost using available open source components and solid architectural design. The Apache Software Foundation is doing an especially good job of picking, incubating and supporting a diversity of open source tools useful to semantic technologies. These tools include Ant, Hadoop, HTTP server, Jackrabbit, Jena, Mahout, Marmotta, Maven, OpenNLP, Singa, Stanbol, SystemML, Tika, Tomcat, UIMA, ZooKeeper, and the Lucene and Solr search engines and Nutch crawler. Additional tooling that would make this task easier still includes:

- Vocabulary managers — we lack easy inspection and editing environments for concepts and predicates. Though standard editors allow direct ontology language edits (OWL or RDFS), these are not presently navigable or editable by non-ontologists. Intuitive browsing structures with more 'infobox'-like editing environments could be helpful here;
- Graph API — it would be wonderful to have a graph API (including analysis options) that could communicate with the OWL API. As a second option, it would be helpful to have a graph API that communicates well with RDF and ontologies;
- Large-graph visualizer — while I have earlier reviewed large-scale graph visualization software,[11] with Gephi and Cytoscape being my two preferred alterna-

tives, they are neither easy to set up nor use. I would like more easily to select layout options with quick zooms and scaling options;

- Graphical editor — some browsers or editors provide nice graph-based displays of ontologies and their properties and annotations. However, the better design we advocate here is to edit the ontology graph directly in its deployment environment; and
- Component services — we recommend piecing out ontology and knowledge management functions into individual components that we can integrate directly into existing workflows with minimal training.

TBox, ABox, and Work Splits

To better understand what kinds of functions we require and how they may relate to existing tools or applications, recall the discussion of *description logics* in *Chapter 8*. Description logics and their semantics traditionally split *concepts* and their relationships from the different treatment of *individuals* and their attributes and roles, expressed as fact assertions. The concept split is known as the *TBox* (for *terminological* knowledge, the basis for *T* in *TBox*) and represents the schema or taxonomy of the domain at hand, what we also call the *knowledge graph*. The TBox is the structural and extensional component of conceptual relationships. The second split of individuals is known as the *ABox* (for *assertions*, the basis for *A* in *ABox*) and describes the attributes of individuals, the roles between individuals, and other assertions about individuals regarding their class membership with the TBox concepts. The ABox is the repository for data records and can be a light layer over existing data stores. Both the TBox and ABox are consistent with set-theoretic principles.

TBox and ABox logic operations differ, and their purposes vary. TBox operations are based more on inferencing and tracing or verifying class memberships in the hierarchy (that is, the structural placement or relation of objects in the structure). ABox operations are more rule-based and govern fact checking, instance checking, consistency checking, and the like. ABox reasoning is often more complicated and at a larger scale than that for the TBox. However, even with these TBox and ABox splits, we can also see that some work done by a knowledge management system falls outside of the specific purview of instances and concepts:

TBox	*Possibly Separate Work Tasks*	*ABox*
• *Definitions* of the *concepts* and *properties* (relationships) of the controlled vocabulary • *Declarations* of *concept axioms* or *roles* • *Inferencing* of relationships, be they transitive, symmet-	• *Mappings* are the core of interoperability in that concepts, and attributes get matched across schema and datasets • *Transformations* are the means to bring disparate data into common grounds,	• *Membership assertions,* either as *concepts* or as *roles* • *Attributes assertions* • *Linkages assertions* that capture the above but also assert the external sources for these assign-

TBox	*Possibly Separate Work Tasks*	*ABox*
ric, functional or inverse to another property • *Equivalence testing* as to whether two classes or properties are equivalent to one another • *Subsumption*, which is checking whether one concept is more general than another • *Satisfiability*, which is the problem of checking whether a concept has been defined (is not an empty concept) • *Classification*, which places a new concept in the proper place in a taxonomic hierarchy of concepts • *Logical implication*, which is whether a generic relationship is a logical consequence of the declarations in the TBox • *Infer property assertions* implicit through the transitive property	the second leg of interoperability • *Entailments*, which are whether the stated condition implies other propositions • *Instance checking*, which verifies whether a given individual is an instance of (belongs to) a specified concept • *Knowledge base consistency*, which is to verify whether all concepts admit at least one individual • *Realization*, which is to find the most specific concept for an individual object • *Retrieval*, which is to find the individuals that are instances of a given concept • *Identity relations*, which is to determine the equivalence or relatedness of instances in different datasets • *Disambiguation*, which is resolving references to the proper instance • *Machine learning* based on entities and features in the knowledge base	ments • *Consistency checking* of instances • *Satisfiability checks*, which are meeting the conditions of instance membership

Table 12-1: Possible Work Activities in a Knowledge Management Platform

Searching across the entire database or conducting machine learning, as examples, are such functions that work against the whole knowledge structure, or which pose work requirements orthogonal to the TBox-ABox splits. *Table 12-1* summarizes how we may segregate these significant work areas against the TBox, the ABox, or possibly separate to them.

The TBox should be a coherent structural description of the domain, which expresses itself as a knowledge graph with meaningful and consistent connections across its concepts. Somewhat irrespective of the number of instances (the ABox) in the knowledge base, the TBox is relatively constant in size given the desired level of descriptive scope for the domain. (In other words, the logical model of the domain is mostly independent of the number of instances in the domain.) As its name suggests, the TBox is where we define terminology for the vocabulary of the domain, the predicates used, and the relationships of those concepts to one another via the predicates

Author Final

available. A key aspect of the TBox functionality is classification through subsumption hierarchies, from which we set much of the logic and inferencing capabilities of the structure. The TBox also requires checks during its building and maintenance to ensure that we have provided complete definitions (*satisfiability*) and consistency and logic tests to make sure our placements within the knowledge graph remain *consistent* and *coherent*.

The ABox of instances consists of the specific individual things in the KB that are relevant to the domain. Instances can be many or few, as in the millions within KBpedia, accounting for 90% or more of the total number of objects in the knowledge base. We characterize instances by various types of structured data, provided as attribute-value pairs, and which we describe with long or short texts and with multiple aliases and synonyms, and we relate to other instances via type or kind or other relations, possibly in multiple languages.

We can perhaps better illustrate this work split with *Figure 12-1* showing the interactions of all of these contributing parts:

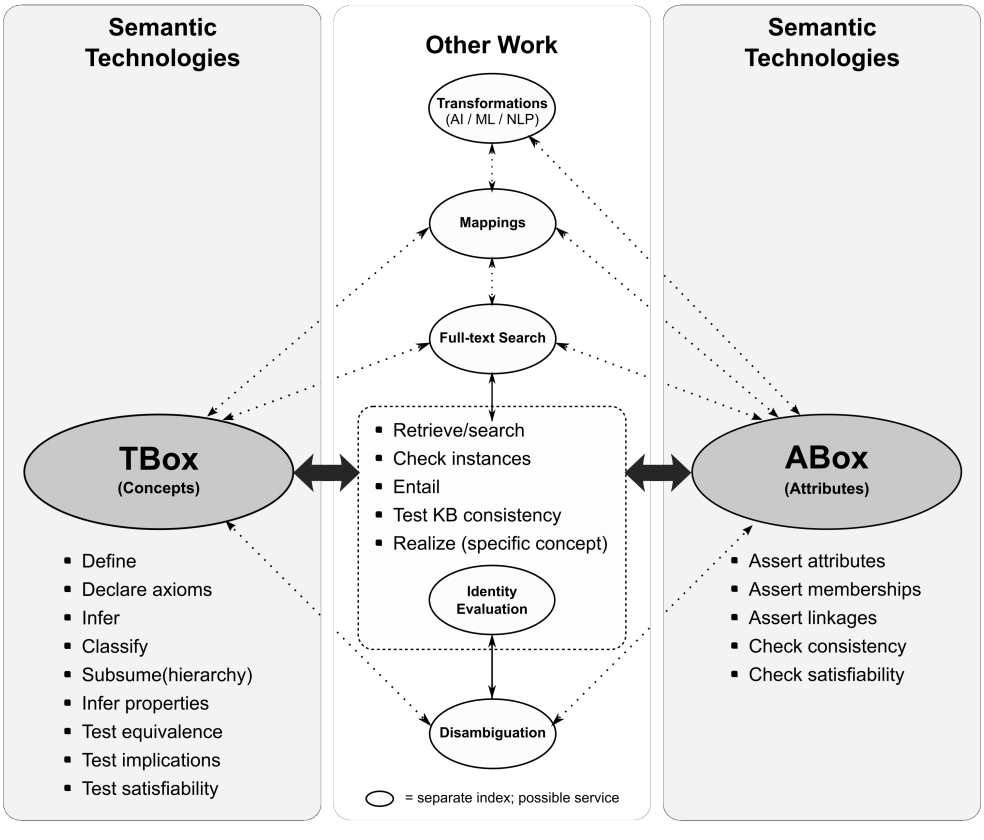

Figure 12-1: Possible Work Splits in a KM Platform

Whether a single database or the federation across many, we have data records (instances in the ABox) and a logical schema (*ontology* of concepts and relationships in the TBox) by which we try to relate this information. As *Table 12-1* and *Figure 12-1* show, the TBox is where the reasoning work occurs; the ABox is where assertions and data integrity occurs. This meaningful work broadly relates to the growth and maintenance of the knowledge base itself. For instance, all aspects of ontology editing relate to these components, as do logic, consistency, coherency and satisfiability checks. These portions are essential to the integrity of the knowledge structure via its editing and maintenance but represent very little of the desired work we want to extract from the knowledge structure. These work tasks are separate from the needs of the TBox and ABox themselves.

The middle column of *Table 12-1* and *Figure 12-1* list some of those work tasks that reside outside of the knowledge graph and knowledge base build and maintenance tasks. Some of these tasks may apply across the entire knowledge structure, such as search or retrieval. Other tasks are specialized ones that may involve subsets of the structure or dedicated extractions of one form or another.

What the *Figure 12-1* readily shows is that platforms with only semantic technologies lack the major work functions desired. It is this gap to bring in and facilitate dataset exchanges to external applications that most requires tailored scripting for specific installations (along with the need to create the domain knowledge graph and ingest data, of course). It is why standalone semantic technology platforms have not been, generally, commercially successful. Not shown in the figure is the further general weakness of semantic technology platforms; namely, they are hard to learn and use. We need more visual frameworks with well-segregated tasks, such as what we are beginning to see in such tools as the SKOS-based PoolParty.

Providers have increasingly embraced platforms that integrate conventional text search engines, such as Solr, for generalized retrieval, plus use in instance and consistency checks. However, critical areas such as mappings, transformations, and identity evaluation remain weak. *Mappings* refer to the suite of aids that suggest matching correspondences between objects in the domain knowledge base with external sources, with choices often manually vetted. *Transformation* is the ability to convert subsets of the knowledge graph to the dataset format required by various external applications. These include machine learning, AI, or specialized natural language processing (NLP) like parsing into parts of speech or transforming external sources into new records or updating the knowledge base. *Identity evaluation* means to contextualize a possible entity reference to its disambiguated actual subject. Maintaining identity relations and disambiguation as separate components also has the advantage of enabling us to swap out different methodologies or algorithms as better methods become available. We could apply a low-fidelity service, for example, for quick or free uses, while we reserve more rigorous methods for paid or batch mode analysis. We may deploy any of these mapping, transformation, or identification activities as a Web service, preferably using an internal canonical data transfer form, discussed further toward the end of this chapter.

Breaking our description logics design into the TBox and ABox, and then enumer-

ating the work tasks we wish to do against these structures, helps us to think through the modularity and architecture we want to see in our actual deployments. The practical aspects of our work tasks and where and how they should occur become clearer. We know that we can architect a framework that is amenable to swapping in and out different analysis methods, and that can be modular to use or not different work tasks and applications. Here are some general principles that should apply to most domain installations:

- We want to handle our concepts, and their definitions and relationships (TBox) separate from our instance data, and subject to rigorous testing, vetting, and updating since this is the controlling logical structure of our knowledge management system;
- The task of knowledge graph creation and maintenance should be the responsibility of knowledge workers and their management, not the IT department;
- We want to handle our instance data (ABox) separately and directly, using comparatively constant and readily understandable attribute-value pairs;
- We can re-use these instance records in varied and multiple worldviews in relation to different TBoxes or external applications; we can support these different perspectives without affecting instance data in the slightest;
- We should approach architectural decisions from the standpoint of the *work* to be done, leaving open unique analysis or tasks like disambiguation or full-text search as functions, which may be added or not at another time;
- Ontologies should be modular, scoped according to appropriate user groups, and kept as simple and easy to understand as possible; this is a significant rationale for the *typology* design discussed in *Chapter 10*. We should assert inter-ontology relationships via a rather simple upper ontology, such as what is provided by the KBpedia Knowledge Ontology;
- We may base mapping on suggestions from TBox (extensional) relationships or ABox (intensional) relationships, and is a particularly weak yet important part of tooling;
- We can treat logic and consistency testing as external applications, and conduct them on scheduled or on-demand via services using canonical formats;
- We should evaluate instances separately from concepts, which also via triangulation may aid such tasks as disambiguation or entity identification;
- We should include access control and governance (missing from *Figure 12-1*) in most enterprise settings or where we use proprietary or private data;
- We can often keep instance records *in situ*, especially useful when incorporating the massive amounts of data in existing relational databases;
- We may add to instance stores incrementally, via *in situ* or staged, following these same design principles; and, given the discussion in *Chapter 9*; and

- We should premise the entire system on continuous change given the nature of knowledge and its openness.

Content Workflows

Two of these critical work splits are thus to: 1) keep knowledge updated; and 2) directly involve knowledge workers and subject matter experts. These requirements go hand and hand since the source of new knowledge comes from these workers and their accumulated content in the first place. More simply put, to capture knowledge, the systems to do so must be in the hands of the knowledge workers themselves, and must integrate cleanly into their existing content workflows. It is inefficient not to leverage existing workflows. Users will likely ignore new knowledge graph maintenance and use tasks unless they are dead simple to implement. We best achieve adoption through an incremental series of non-threatening tasks.

The following *Figure 12-1* sketches out broad steps and interactions that one might want to see in a content workflow:

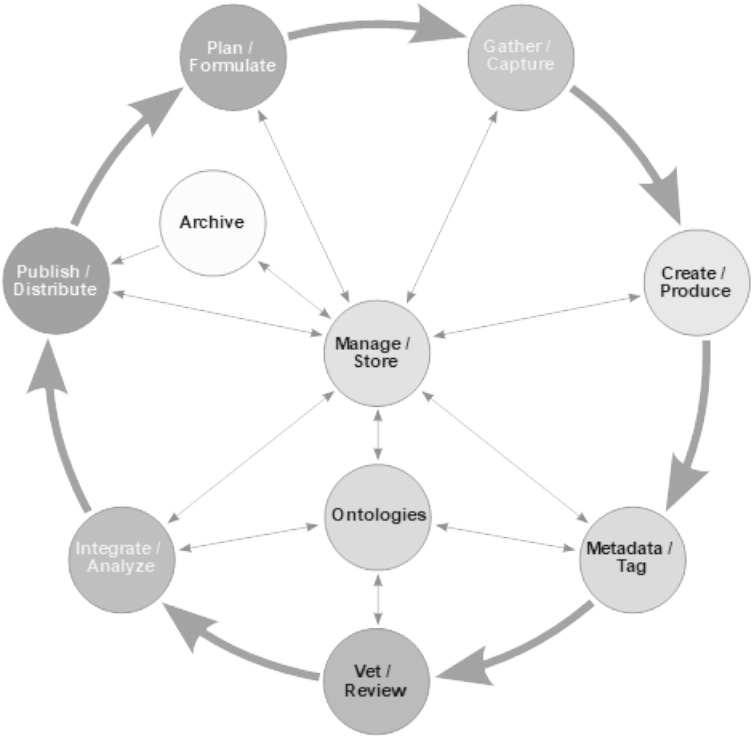

Figure 12-2: Content Workflows in a KM Platform

Respect for workflows is the first principle when setting boundaries around functional requirements. We express this respect in two different ways. The first is that

we cannot unduly disrupt existing workflows when introducing interoperability improvements. While workflows can be improved or streamlined, new tools and practices must fit with existing ways of doing tasks to see adoption. Users mostly resist jarring changes to existing work practices.

The second way is that we should explicitly model and codify the workflows of how we do tasks. This codification becomes the 'language' of our work and helps define the tooling points or points of interaction as we merge activities from multiple disciplines in our domain. These workflow understandings also help us identify useful points for APIs in our overall interoperability architecture. An excellent use of an administrative ontology is to codify and model the workflow and approval steps associated with informal and formal content workflows in the organization.

Some steps within *Figure 12-2* may not be active within an organization, such as tagging or assigning metadata. Cases like this probably need to identify tasks in the associated content creation and review where we can link metadata additions into current workflows. These kinds of incremental additions to existing workflows continue to suggest the wisdom of breaking apart the individual steps in ontology creation and maintenance to more atomic parts, such as flagging a new concept or adding to a *semset* label for an existing one. We may then slipstream these additional steps into separate ontology suggestions that authorized editors review and vet before final acceptance. These steps, of course, and how we refer to them, may vary across circumstances and organizations. Nonetheless, we may apply the general ideas of work steps, approval types, and users to any formal or informal workflow that presently exists.

These considerations provide the rationale for assigning metadata that characterizes our information objects and structure. We should base this metadata on controlled vocabularies and relationships in domain and administrative ontologies, as determined by their users (knowledge workers). The vocabularies and the tagging of information objects with them are a first principle for ensuring how we can find and transition states of information. These vocabularies need not be elaborate, but they should be constant and consistent across the entire content lifecycle. Backbone aspects of these vocabularies should capture the overall information workflow, as well as concrete steps for individual tasks. As a complement to such administrative ontologies, domain ontologies provide the context and meaning (semantics) for our information.

This common grounding of data model and semantics means we can connect our sources of information. The properties that define the relationships between things determine the structure of our knowledge graph. Seeking commonalities for how our information sources relate to one another helps provide a coherent graph for drawing inferences. How we describe our entities with attributes provides a second type of property. Attribute profiles are also a good signal for testing entity relatedness. Properties — either relations or attributes — give another filter to draw insight from available information.

If the above sounds like a dynamic and fluid environment, you would be right. Ultimately, knowledge is a challenge in a technology environment that is rapidly

changing. New facts, perspectives, devices, and circumstances are continually arising. For these very reasons a knowledge management framework must embrace the *open world assumption* (see *Chapter 9*), wherein we can grow and extend the underlying logic structure and its vocabulary and data at will.

Though perhaps not quite at the level of a first principle, I also think KM improvements should be easy to use, easy to share, and easy to learn. I imply tooling in this, but also it is important we be able to develop a language and framing for what constitutes our knowledge domain. We should pursue the question of interoperability to discover insights and gain efficiencies. The thing about interoperability is that it extends over all aspects of the information lifecycle, from capturing and creating information, to characterizing and vetting it, to analyzing it, or publishing or distributing it. Eventually, information and content already developed become input to new plans or requirements. These aspects extend across multiple individuals and departments and even organizations, with portions of the lifecycle governed (or not) by their own set of tools and practices.

Today, overall, we only embrace pieces of this cycle in most daily workflows. Editorial review and approvals, or database administration and management, or citation gathering or reference checking, or data cleaning, or ontology creation and management, or ETL activities, or hundreds of other specific tasks, sit astride this general backbone. Besides showing that interoperability is a systemic activity for any organization (or should be), we can also derive a couple of other insights from *Figure 12-2*. First, we can see that some form of canonical representation and management is central to interoperability. The form need not be a central storage system, but can be distributed using Web identifiers (IRIs) and protocols (HTTP). Second, we characterize and tag our information objects using ontologies, both from structural and administrative viewpoints, but also by domain and meaning. We can combine and analyze our information when we characterize it with a common semantics.

A third insight is that a global schema (from the standpoint of the enterprise) specific to workflows and our content is a key for linking and combining activities at any point within the cycle. A common vocabulary for stages and interoperability tasks, included as a best practice for our standard tagging efforts, provides the conventions for how batons can get passed between activities at any stage in this cycle. The challenge of making this insight operational is one more of practice and governance than of technology. It should be a purposeful activity in its own right, backed with appropriate management attention and incentives.

An enabling mindset for the knowledge workers involved is to pay explicit attention to content workflows and common vocabularies for those flows and the information objects they govern. This focus becomes the scaffolding for an administrative ontology and a basis for investigating tooling and automation in processing information. We can already put in place chains of tooling and workflows to achieve a degree of interoperability. We do not need to provide global answers or scope at the inception. We can start piecemeal, and expand as we benefit. The biggest gaps remain codification of workflows for the overall information lifecycle, and the application of taggers to provide the workflow and structure metadata at each stage in the cycle.

Again, these are not matters so much of technology or tooling, but willingness, and policy and information governance.

PLATFORM CONSIDERATIONS

Semantic technologies have not yet reached the point of fulfilling their prophecy nor of being sufficiently buzz-worthy to fuel their demand.* Enterprise customers are intrigued with the idea of semantic solutions but remain skeptical. Better search is often the crucial leverage point in the sale. Enterprises do not seem interested in linked data alone (if at all), though some like the idea of possibly contributing linked data back to others. On the other hand, all enterprises competing in the current environment understand that knowledge, and their use and management of it, is perhaps the pivotal factor in their relevance and survival.

I have had the good fortune to work with some cutting-edge, reference enterprise deployments of semantic technologies. These efforts in enterprise-scale systems have been eye-opening. We have opened one eye for how semantic technologies need to integrate and adapt to existing enterprise practices and deployments. We have opened the other eye to see how semantic technologies should be presented and sold to internal enterprise stakeholders.

We have a working example in the Open Semantic Framework that shows the way for how a few common representations and conventions can work to distribute both schema and information (data) across a potentially distributed network. Further, by not stopping at the water's edge of data interoperability, we can also embrace further, structural characterizations of our content. Adding this wrinkle enables us to support a variety of venues for content consumption simultaneously and efficiently, as well as to broaden our leverage of the knowledge asset through cheaper, more streamlined machine learning and artificial intelligence. What I set out in the next section are the multiple purposes and the ontology-driven aspects of a general knowledge representation and management platform to support enterprise aims.

Supporting Multiple Purposes

Our avowed purposes in data interoperability and KBAI, supported by general KM (knowledge management) uses, sets the overall application scope for our platform. At the same time, we understand that particular uses of the platform will vary by domain, desired application emphases, and the actual instance data. We further assume that initial demands and scope may warp and grow as we experience platform results, and external demands dictate. All of these considerations demand a platform design that is open, modular, and extensible, capable of supporting multiple purposes (and, thus, cost justifications). We need to put forward reasonable projected benefits that greatly exceed development costs, and then to continue to justify such assertions to sustain a healthy, dynamic knowledge management system. Specific do-

* See *Chapters 15* and *16*.

main applications are surely the instrumental justification for an initial installation, but an adaptive KM platform should also meet the two core requirements of search and knowledge management.

Search

Enterprises, familiar with structured query language (SQL), have understood for quite some time that queries and search are more than text searches to search engines. Semantic technologies have their structured query approach, SPARQL. State-of-the-art semantic search has found a way to combine these various underlying retrieval engines with the descriptive power of the graph and semantic technologies to provide a universal search mechanism across all types of information stores. The simplest way to understand semantic search is to de-construct the basic RDF triple down to its fundamentals. This first observation is that the RDF data model can represent anything, that is, an object or idea. Moreover, we can represent that object in virtually any way that any viewer would care to describe it, in any language. In semantic search, we may derive facets from not only what types of things exist in the search space, but also what kinds of attributes or relations connect them. Gratifyingly, this all comes for free. Unlike conventional faceting, no one needs to decide what are the important 'dimensions' or any such. With semantic search, the very basis of describing the domain at hand creates an organization of all things in the space.

In semantic search, every property represents a different pathway, and every node is an entry point. SPARQL enables us to pose queries, including with variables, which can navigate and slice-and-dice the information space into usable results subsets at will. We do not need to state all of the relationships and types of things in our information space; we can infer them from the assertions already made. We can use these broad understandings of our content to do better targeting, tagging, highlighting or relating concepts to one another. The fact that semantic search is a foundation for semantic publishing is noteworthy.

We first adopted Solr (and then Lucene) because traditional text search of RDF triple stores was not sufficiently performant and made it difficult to retrieve logical (user) labels in place of the IRIs used in semantic technologies. In our design, the triple store is the data orchestrator. The RDF data model and its triple store are used to populate the Solr schema index. The structural specifications (schema) in the triple store guide the development of facets and dynamic fields within Solr. These fields and facets in Solr give us the ability to gain Solr advantages such as aggregates, autocompletion, filtering, spell checkers and the like. We also can capture the full text if the item is a document, enabling us to combine standard text search with the structural aspects orchestrated from the RDF. On the RDF side, we can also leverage the schema of the underlying ontologies to do inferencing (via forward chaining). We have been able to (more-or-less) seamlessly embrace geo-locational based search, time-based search, the use of multiple search profiles, and switchable ranking and scoring approaches based on context (using Solr's powerful extended disMax *edismax*

parser).[12] This combination gives us an optimal search platform to do full-text search, aggregates, and filtering.

Knowledge Management

Our earlier *Figure 12-1* showed the two bracketing left- and right-work areas in semantic technologies. These are the very same knowledge graph (TBox) and instance data (ABox) areas that form the knowledge base that our KM system must manage. Here are some of the tasks we need to manage: 1) insert and update concepts in the upper ontology; 2) update and manage attributes and track specific entities as new sources of data are entered into the system; 3) establish coherent linkages and relations between things; 4) ensure these updates and changes are done wholly and consistently, while satisfying the logic already in place; 5) update how we name and refer to things as we encounter them; 6) understand and tag our content workflows such that we can determine provenance and authority and track our content; and 7) do these tasks using knowledge workers, who already have current tasks and activities.

These actions should be continuous, and established procedures with annotations and logging should govern them. The entire premise of a knowledge management system is to keep current and up-to-date. This need for currency means that use and updates of the semantic technologies portion, which is the organizing basis for the knowledge in the first place, must be part of daily routines and work tasking, subject to management and incentives. Responsive, tailored tooling linked to current workflows is the technical requirement. Management procedures and training need to complement the technology to ensure the human factors are also in place.

An Ontologies-based Design

We have seen that an upper ontology governs the overall knowledge graph, with typologies and domain ontologies tailoring the scope and providing instance coverage. We have also seen, in the case of the content lifecycle, where we can capture content workflows and approvals into metadata that tracks content across the system and provides provenance information using an administrative ontology. The platform should also provide a standard set of access and retrieval services including browse, full-text search, CRUD, direct record retrievals, and the like. We may embed these within an access and permissions service, also governed by an administrative ontology, that acts at the level of registered datasets (see next section). We should also design our queries and requests to the platform to include a parameter for getting results sets in particular formats such as XML or JSON or RDF (various flavors), or others of domain importance. Administrative ontologies can also guide how HTML pages and forms are dynamically populated, often contextually, based on standard SPARQL queries. For specific purposes, we can also return these results sets as prestaged, properly formatted results streams (usually in the form of SPARQL queries) for driving particular applications. We only need to add a basic converter to the plat-

form's Web services stack to 'drive' a new application in a specific format.

As explained in the concluding section of this chapter, we recommend packaging these platform capabilities as Web services that we can interact with and drive via standard HTTP requests using standard *application programming interfaces* (APIs). Alternatively, we can issue these requests from simple to comprehensive Web apps that create the API queries based on user interface choices such as selections from dropdown lists or clicking on various listed options. The platform thus acts as a single, uniform Web interface to all of the capabilities of the structured data system organized by the adaptive ontologies. Further, we may ingest virtually any data structure and convert it via an import service made part of the underlying canonical structure. Lastly, the dataset nature of the framework, and its neutrality to underlying data stores or content management systems, also makes the platform an excellent framework for one or many nodes to share information and collaborate across the Web.

'Ontology-driven apps' through this platform design thus provide two profound benefits. First, once we write the templates, we can drive the entire system via simple Web form selections or interactions without the need for any programming or technical expertise. Second, we can power entirely new applications through the addition of new, minor output converters. These potentials arise from the native power of the design basis for ontology-driven apps. Conceptually, the design is simplicity itself. Operationally, the system is extremely flexible and robust. Strategically, it means our development and specification efforts may now move from coding and programmers to the subject matter users who define ontologies and depend on them.

Enterprise Considerations

Security is an additional enterprise requirement that warrants particular attention. Whether profit or non-profit, all enterprises are unique, with potential proprietary information both internally and externally (with the public or possible competitors). Though individual consumers also have requirements for privacy and confidentiality, these information flows are strictly between the individual and outside entities. In an enterprise, access may occur and be among many internal individuals and all of their external contacts. *Access control* is the protection of resources against unauthorized access. It is a process by which use of resources is regulated according to a security policy and is permitted by only authorized system entities according to that policy.[13]

We may provide access control, like many other enterprise considerations, through a third-party application, by an administrative ontology linked to other features tagged in the knowledge store, or both. As one example, we have provided access control in some installations of the Open Semantic Framework using a three-dimensional matrix of datasets, users/groups, and CRUD rights to tools/endpoints. A dataset refers to a named grouping of records, best designed as similar in record types and intended access rights (though technically a dataset is any named grouping of records). We need to first grant access for given user/group to a particular

Web service, and specify whether that user has CRUD (*create – read– update – delete*) rights in whole or part to interact with specified datasets within the knowledge base. It is in the nexus of user type, a tool (API), and dataset that we may establish access control for the semantic system.

In an enterprise context, a given individual (user) may have different access rights depending on circumstance. A worker in a department may be able to see and do different things for departmental information than for enterprise information. A manager may be able to view budget information that is not readable by support personnel. A visitor to a different Web site or portal may see different information than visitors to other Web sites. Supervisors might be able to see and modify salary data for individual employees that is not viewable by others. The user role or persona thus becomes the access identifier to the system. As system managers, we define what information and what tools users might use for the datasets for which they have access.

The combination of datasets * tools * roles can lead to many access permutations. With, say, 20 tools with five different roles and just ten different datasets, we already have about 1,000 permutations. As portals and dataset numbers grow, this combinatorial explosion gets even worse. Of course, not all combinations of datasets, tools, and roles make sense. In fact, only a relatively few number of patterns covers 95% or more of all likely access options. Because access rights are highly patterned, these theoretical combinations can, in fact, be boiled down to a small number of practical templates — which we call profiles — to which we may assign a newly registered dataset or user. (Of course, the enterprise could also tweak any of the standard profiles to meet any of the combinatorial options for a specific, unusual individual, such as for a tax auditor.)

Another enterprise consideration relates to training. Inter-team communications must be grounded in shared vocabulary and concepts. Even then, it is still necessary to continuously describe and explicate the benefits due to semantic approaches over conventional ones. Because of its general foundational nature, semantic approaches are often hidden or at the core of the information solution. It is not always self-evident what the advantages of semantic approaches are because their results can be mimicked via conventional approaches (though at a higher cost with greater brittleness). Semantic concepts are not (generally) intuitive to content editors, information architects, project managers or fellow developers or project vendors. It is imperative to engage in continuous training and knowledge transfer during a semantic deployment. Unlike just a few years back, we no longer see resistance to open source solutions. In fact, for early semantic adopters, open source is a positive feature. However, open source in a complicated enterprise environment comes with challenges. Support is often weak and integrating the pieces becomes one of the project responsibilities and risks. Open APIs and Web service endpoints still can lead to integration challenges. Encoding mismatches or how error messages get generated or treated, as two examples, point to some of the challenges in creating an integrated enterprise environment from multiple open source pieces.

Enterprise funding is still another concern. Enterprise IT budgets have come un-

der pressure. The justification for many projects resides in being able to offset annual licensing and maintenance fees, which can impose delivery constraints based on renewal dates. Existing enterprise IT budgets have also been made more incremental, with milestone achievements often required for moving forward. These trends are putting a premium on agile development and the need for enterprise-scale deployment and testing tools. Repeatable build processes and scripts are an essential component now for complex stack deployments.

Many of the issues that emerge in enterprise deployments are ancillary to or independent of specific knowledge components. Logging, testing, security, access, service buses and deployment builds are an umbrella over entire deployments. In these regards, too, we must adhere to enterprise build practices and standards. The frequency of repeating builds and testing means we need to create scripts for these steps and improve deployment documentation and practices. In these regards, knowledge and semantic technologies are no different from other components in the broader, enterprise-wide stack.

Another reality of semantic technologies in the enterprise is that few champions and advocates exist within many organizations. We must find means to communicate to semantic newbies and to enlist the aid of champions in carrying the message forward within the organization. In multi-vendor deployments, we should seek single points of contact able to communicate with their colleagues. In turn, the consumers of knowledge applications – namely subject matter experts, employees, partners, and stakeholders – now become the active contributors to the graphs themselves, focusing on reconciling terminology and ensuring adequate entity and concept coverage. Graph-driven applications mean that those closest to the knowledge problems will also be those directly augmenting the graphs. These changes act to democratize the knowledge function and lower overall IT costs and risks.

A WEB-ORIENTED ARCHITECTURE

Web-oriented architecture, or *WOA*, is a subset of the service-oriented architectural (SOA) style, wherein we package discrete functions into modular and shareable elements ('services') that we make available in a distributed and loosely coupled manner. WOA uses the representational state transfer (REST) style, geared to the HTTP hypertext transfer model. Roy Fielding defined the REST architectural style in his 2000 doctoral thesis.[14] Fielding is also one of the principal authors of the Hypertext Transfer Protocol (HTTP) specification. We couch WOA guidelines within the framework of a generalized *architectural style,* and while not limited to the Web, are a foundation for it.

Nick Gall, a Gartner analyst, was one of the first to coin the WOA moniker. Gall describes WOA as based on the architecture of the Web as aq "globally linked, decentralized, and [with] uniform intermediary processing of application state via self-describing messages." REST provides principles for how resources are defined and used and addressed with simple interfaces without additional messaging layers such as SOAP or RPC. REST and WOA stand in contrast to earlier Web service styles known by

the WS-* acronym (such as WSDL). WOA has proven highly scalable and robust for decentralized users since all messages and interactions are self-contained (convey 'state'). It is not surprising that the largest existing knowledge networks on the globe — such as Google, Wikipedia, Amazon, and Facebook — are Web-based. These pioneers have demonstrated the wisdom of WOA for cost-effective scalability and universal access.

We recommend a WOA architecture for knowledge management and representation purposes. Like the Internet itself, WOA has the advantage of being scalable and distributed, all (mostly) based on open standards. RESTful application programming interfaces (APIs) extend interoperability to outside systems and provide flexibility for swapping in new features or functionality as new components or developments arise. Under this design, all components and engines ('services') become in effect 'black boxes,' with information exchange via standard vocabularies and formats using APIs as the interface for interoperability.

Web-orientation and Standards

Two main reasons, plus a host of others, justify basing our KM architecture on the Web. The first main reason is a crowning achievement of the semantic Web, which is the simple use of uniform resource identifiers (URIs, now internationalized to IRIs) to identify data. Further, if the resource identifier can resolve to a representation of that data, it now becomes an integral part of the HTTP access protocol of the Web while providing a unique identifier for the data. The HTTP protocol is the second main reason, through which we gain access to a global, distributed network. These innovations provide the basis for distributed data at global scale, all accessible via Web devices such as browsers and smartphones that are now a ubiquitous part of our daily lives. The combination of RDF with Web identifiers also means that we may expose any information from a given knowledge repository and make it available to others as linked data. This approach makes the Web a universal database.

We often think of HTTP as a communications protocol, but it is much more.[15] It represents the operating system of the Web as well as the embodiment of a design philosophy and architecture. Within its specification lies the secret of the Web's success. REST and WOA quite possibly require nothing more to understand than the HTTP specification. HTTP provides the distinctions of GET and POST and persistent IRIs and the need to maintain stateless sessions with an idempotent design. HTTP also provides for content and serialization negotiation, and error and status messages for HTTP requests. HTTP also includes: language, character set, encoding, serialization and mime type enforced by header information and conformance with content negotiation; common and consistent terminology to aid understanding of the universal interface; a resulting component and design philosophy that is inherently scalable and interoperable; and a seamless consistency between data and services. CRUD is readily applicable to HTTP.

Besides these reasons, WOA is consistent with the many open Web standards we use in KBpedia and our platform designs. See further *Chapter 9*.

A Modular Web Services Design

I have emphasized two themes throughout this chapter. The first theme is to scope and bound functionality related to design needs. The second theme is to integrate these functions within current content workflows. We express these themes using individual RESTful Web services in our design, as exposed and accessed through their *application programming interfaces* (*APIs*). We have already seen how the WOA approach enables us to use the HTTP protocol for accessing RESTful Web services. The specific scoping and design of the functional modules provide the complementary part of the overall design. Since the resulting APIs are independent of any particular operating environment, we can reduce implementation costs for multi-platform user agents and promote the development of multi-platform services.

We determine the modularity of the services through analysis of the work tasks (see *Figure 12-1*). Where appropriate, we embed these modules into other current applications or workflows (*Figure 12-2*). Enterprise considerations such as security, access control, or workflow management enter in at this point to help complete the roster of desired services. These definitions help provide the boundary responsibilities of each Web service and what types of API instructions they may need. Platform-wide requirements, such as access control, must inform some of these needs.

We tend to follow a few guidelines in designing our Web services. We emphasize 1) use of a canonical, internal data representation format; 2) unit testing for all services; 2) attentiveness to error numbering and conformity of error messages, some of which we discover during testing; 3) similar granularity and order for specifying parameters across the APIs; 4) provision of online demo examples; 5) standard import and export formats; and 6) dual access to the API via SPARQL and programmatically. We tend to use a 'triples' or N3 RDF format for our internal canonical representation, which has a standard specification. (We also allow multiple import or export formats beyond the internal canonical form.) The provision for dual access to the APIs gives us the standard query basis of SPARQL, plus faster programmatic calls when using internal network transfers.

The size of payloads in both query results and as results set objects can be a challenge for RESTful Web services. Long HTTP queries with many parameter requests and large results sets can be a problem to handle, especially in the security layer. In some cases, we may need to look at ways to minimize and package (consolidate) parameter options to make endpoint requests more efficient. Encoding mismatches are a further challenge. It is best, for example, to adhere to a standard UTF-8 encoding via all semantic component interfaces. Consistent encoding requires attention and coordination on both sides of the interface and in tool use, especially the use of spreadsheets or CSV files.

The more fundamental challenge, however, is one of mindset. Effective interfaces require effective communications of the participating vendors across the boundary. The terminology, concepts, logic and open-world approach to knowledge management and semantic technologies are not easily communicated nor immediately understood by traditional vendors. We must continuously work on communications to

overcome past practices and embrace the flexibilities provided by semantic technologies.

REST Web services[16] and linked data are naturally compatible approaches. Linked data is a set of best practices for publishing and deploying data on the Web using the RDF data model. The data objects are named using Web uniform resource identifiers (IRIs), emphasize data interconnections, and adhere to REST principles. We also see the ideas of RESTful Web services morph into ones with more limited and targeted functionality. These microservices have a broad swath of definitions. Some of the narrower ones, including in their ideas of choreographing and aggregating multiple small services, bear a close resemblance to the particular flavor of Web services that we recommend.

An Interoperability Architecture

Figure 12-3 presents our generic architecture for this WOA design. The three tiers of the system are content acquisition, the repository, and content consumption:

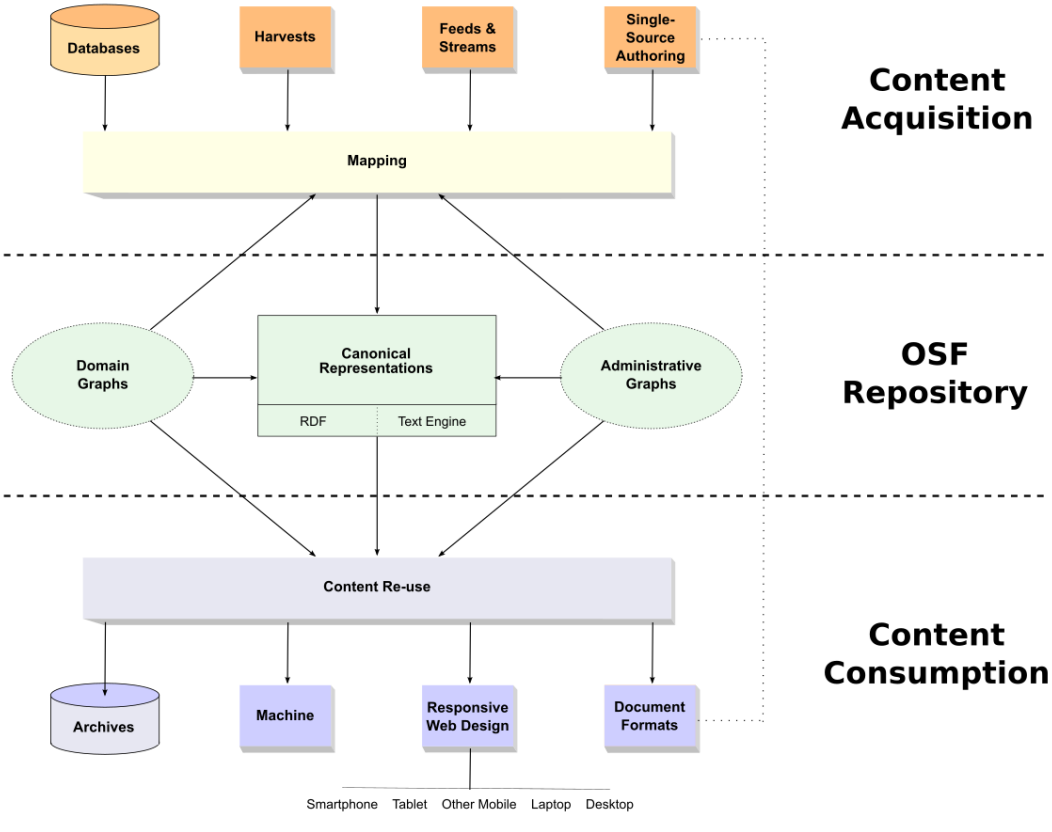

Figure 12-3: An Interoperability Architecture

We have tended to abstract our WOA services into simple and compound ones (which are combinations of the simple). All Web services have uniform interfaces and conventions and share the error codes and standard functions of HTTP. We further extend the WOA definition and scope to include linked data, which is also RESTful. Thus, our WOA also sits atop an RDF (Resource Description Framework) database ('triple store') and full-text search engine.

The content acquisition tier is where all information comes into the system. For new sources, this involves mapping the concepts and other conformities to the existing knowledge graph. Already mapped sources and concepts require fewer integrity checks when we add instances or updates. Because we are using semantic technologies, we are agnostic as to the content source and can handle most any content. The content ingestion step is where we employ the limited number of canonical forms and use RDF as our data transfer model (see *Chapter 9*).

The repository tier is where the knowledge graph, knowledge base, triple store, OWL API, and full-text search engine reside. Most all knowledge management (KM) functions reside in this tier. All ontologies and their management reside at this tier. The full-text search engine and triple stores are mostly agnostic third-party systems. While some differences in open source search engines and triple stores exist, we may plug most into the design. We have used Jena and Virtuoso as triple stores in the past, as well as the Lucene and Solr search engines. Many other options exist.

Many of the specialized work functions shown in the middle sections of *Table 12-1* and *Figure 12-1* reside in the bottom (as shown in Figure *12-3*) content consumption tier. Within this tier, we may move some content to an archive data store, or we may transform subsets for machine learning purposes or to re-purpose existing content. Some of the transformations at this tier are merely transfer conventions with an external application. In addition to such tailored forms and their dedicated Web services, we also make available the general output in a variety of standard formats. Note that the content re-use and mapping layers, as well as the repository, use the internal canonical data representation.

Chapter Notes

1. Some material in this chapter was drawn from the author's prior articles at the *AI3:::Adaptive Information* blog: "' and Now You Know the REST of the Story . . .'" (Feb 2007); "WOA: A New Enterprise Partner for Linked Data" (Oct 2008); "WOA! So RESTful it is UMBELievable!" (Oct 2008); "A General Web-oriented Architecture (WOA) for Structured Data" (May 2009); "The Fundamental Importance of Keeping an ABox and TBox Split" (May 2009); "Ontology-driven Applications Using Adaptive Ontologies" (Nov 2009); "The Open World Assumption: Elephant in the Room" (Dec 2009); "I Have Yet to Metadata I Didn't Like" (Aug 2010); "An Ontologies Architecture for Ontology-driven Apps" (Dec 2011); "Architecting Semantic Technologies for the Enterprise" (Jan 2013); "Enterprise-scale Semantic Systems" (Jan 2013); "Semantic Technology Access Control Using Datasets" (Feb 2013); "Five Fundamental Distinctions of Enterprise Software" (Jan 2014); "Logical Implications of Interoperability" (Jun 2015); "Creating a Platform for Knowledge-based Machine Intelligence" (Sep 2015); "A Foundational Mindset: Firstness, Secondness, Thirdness" (Mar 2016); "Why I Study CS Peirce" (Aug 2017).
2. *Sweet Tools* is still online with its searchable tool listing at http://www.mkbergman.com/sweet-tools/ (for statistics on the latest release, see http://www.mkbergman.com/991/the-state-of-tooling-for-semantic-

technologies/). However, it has not been updated since the beginning of 2012 and is substantially out of date. There is no alternative survey to my knowledge.

3. Prior tools that we have released under various open source licenses include BibJSON, Citizen Dan, conStruct, irON, Open Semantic Framework, OSF for Drupal, scones information tagger, structWSF, and structXML. Ontologies and vocabularies that we have released under various open use licenses include BIBO (Bibliographic Ontology), the MUNI Ontology, the Music Ontology, and UMBEL.
4. Berners-Lee, T., Lassila, O., and Hendler, J., "The Semantic Web," *Scientific American Magazine*, 2001.
5. Noy, N. F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R. W., and Musen, M. A., "Creating Semantic Web Contents with Protege-2000," *IEEE Intelligent Systems*, vol. 16, 2001, pp. 60–71.
6. Sure, Y., Erdmann, M., Angele, J., Staab, S., Studer, R., and Wenke, D., "OntoEdit: Collaborative Ontology Development for the Semantic Web," *The Semantic Web—ISWC 2002*, 2002, pp. 221–235.
7. Bechhofer, S., Horrocks, I., Goble, C., and Stevens, R., "OilEd: a Reasonable Ontology Editor for the Semantic Web," *Working Notes of the 2001 International Description Logics Workshop (DL-2001)*, pp. 1–9.
8. Bechhofer, S., Volz, R., and Lord, P., "Cooking the Semantic Web with the OWL API," *International Semantic Web Conference*, Springer, 2003, pp. 659–675.
9. Horridge, M., and Bechhofer, S., "The OWL API: A Java API for OWL Ontologies," *Semantic Web*, vol. 2, 2011, pp. 11–21.
10. Jena is fundamentally an RDF API. Jena's ontology support is limited to ontology formalisms built on top of RDF. Specifically this means RDFS, the varieties of OWL, and the now-obsolete DAML+OIL.See http://jena.apache.org/.
11. Bergman, M. K., "Large-scale RDF Graph Visualization Tools," *AI3:::Adaptive Information*, Jan. 2008.
12. Similar capabilities may be implemented with Lucene or ElasticSearch.
13. See further RFC 2828,"Internet Security Glossary," May 2000, *The Internet Society*, provided by the Internet Engineering Task Force (IETF) (see http://www.ietf.org/rfc/rfc2828.txt).
14. Fielding, R. T., "Architectural Styles and the Design of Network-Based Software Architectures," Ph.D., University of California, Irvine, 2000.
15. The current specification is RFC 2616 (June 1999), which defines HTTP/1.1; see http://tools.ietf.org/html/rfc2616. For those wanting an easier printed copy, a good source in PDF is http://www.faqs.org/ftp/rfc/rfc2616.pdf.
16. Richardson, L., and Ruby, S., *RESTful Web Services*, Sebastopol, CA: O'Reilly Media, Inc., 2008.

Author's Final