Future Directions for Semantic Systems

John F. Sowa

Abstract. For over thirty years, the complexity of knowledge acquisition has been the greatest obstacle to widespread use of semantic systems. The task of translating information from a textbook to a computable semantic form requires the combined skills of a linguist, logician, computer scientist, and subject-matter expert. Any system that requires its users to have all those skills will have few, if any, users. The challenge is to design automated tools that can combine the contributions from multiple experts with different kinds of skills. This article surveys systems with different levels of semantics: lightweight, middleweight, and heavyweight. Linked data systems with lightweight semantics are easy to develop, but they can't interpret the data they link. The heavyweight systems of traditional AI can perform deep reasoning, but they place too many demands on the knowledge engineers. No one can predict what innovations will be discovered in the future, but commercially successful systems must satisfy two criteria: first, they must solve problems for which a large number of people need solutions; second, they must have automated and semi-automated methods for acquiring, analyzing, and organizing the required knowledge.

This is a slightly revised preprint of an article in *Intelligence-based Software Engineering*, edited by Andreas Tolk and Lakhmi C. Jain, Springer Verlag, Berlin, 2011, pp. 23-47.

1. The Knowledge Acquisition Bottleneck

Computers can process numbers, data structures, and even axioms in logic much faster than people can. But people take advantage of background knowledge that computers don't have. Hao Wang (1960), for example, wrote a program that proved all 378 theorems in propositional and first-order logic from the *Principia Mathematica*. On a slow vacuum-tube computer, Wang's program took an average of 1.1 seconds per theorem — far less time than Whitehead and Russell, the two brilliant logicians who wrote the book. But the theorems in the *Principia* require a negligible amount of built-in knowledge — just five axioms and a few rules of inference. The computer Wang used had only 144K bytes of RAM, but that was sufficient to store the rules and axioms and manipulate them faster than professional logicians.

During the 1970s and '80s, rule-based expert systems and programs for processing natural languages became quite sophisticated. But most applications required an enormous amount of background knowledge to produce useful results. Knowledge engineers and subject-matter experts (SMEs) had to encode that knowledge in formal logic or some informal rules, frames, or diagrams. The experts were usually highly paid professionals, such as physicians or geologists, and the knowledge engineers required long years of training in logic, ontology, conceptual analysis, systems design, and methods for interviewing the experts. For critical applications, the investment in knowledge acquisition produced significant results. For other applications, the cost of defining the knowledge might be justified, but the AI tools were not integrated with commercial software. Furthermore, most programmers did not know how to use AI languages and tools, and the cost of training people and adapting tools was too high for mainstream commercial applications.

During the 1990s, vast amounts of data on the World Wide Web provided raw data for statistical methods. Machine learning, data mining, and knowledge discovery found patterns more cheaply and often more accurately than rules written by experts. The more challenging goal of language understanding was largely abandoned in favor of statistical methods for information retrieval and information extraction. Although statistical methods are useful, they don't generate a semantic representation suitable for further reasoning or for explanations in ordinary language.

At the beginning of the 21st century, the Semantic Web adapted the AI technologies of the 1980s to the vast resources of the World Wide Web. But the mainstream commercial software, which had never been integrated with AI technology, was just as isolated from the Semantic Web. For most programmers and web masters, the languages and tools of the Semantic Web were unfamiliar, there was no migration path from conventional software to the new technology, and the task of knowledge acquisition was just as difficult as ever.

The complexity of knowledge acquisition increases with the complexity of the semantics, the amount of detail that must be specified, and the interdependencies among different aspects of the knowledge base. To relate different methods, this article uses a three-way distinction: *heavyweight* semantics is represented in a formal logic with detailed axioms that can support extended reasoning; *middleweight* semantics is based on formal or informal notations that support a modest amount of reasoning, but with less complexity than heavyweight semantics; *lightweight* semantics uses tags to classify information, to check simple constraints on types and connections, but not to perform extended reasoning.

Many systems use variations of these three kinds of semantics. Section 2 of this article uses the distinction to compare systems for natural language processing (NLP). Section 3 applies it to systems for reasoning and problem solving. Section 4 analyzes the Semantic Web technologies in these terms. Section 5 shows how the VivoMind Language Processor (VLP) uses all three kinds of semantics for language analysis and reasoning. The concluding Section 6 discusses the requirements for commercially successful systems and the ways of using AI technology to design and implement them.

2. Natural Language Processing

Documents that people write to communicate with other people are rarely as precise as logic. Yet people can read those documents and relate them to formal notations for science, mathematics, and computer programs. They can derive whatever information they need, reason about it, and apply it at an appropriate level of precision. That flexibility is essential for a system of knowledge acquisition — automated, semi-automated, or at least computer assisted. For the past half century, AI researchers and computational linguists have tried to achieve that goal.

Some of the most successful NLP systems use lightweight semantics. One of the first was the Georgetown Automatic Translator (GAT), for which research was terminated in 1963. Under the name Systran, it became the most widely used machine-translation system in the 20th century. A version is still available on the web under the name Babelfish. For each pair of languages to be translated, Systran uses a large dictionary of equivalent words and phrases. The computer processing consists of a limited amount of movement and adjustment to accommodate the syntactic differences between each language pair (Hutchins 1995). Constructing those dictionaries by hand requires many person-years of effort. With the large volumes of documents available on the web, statistical methods for detecting and aligning equivalent pairs have become more widely used. Although these techniques are useful for MT, they don't produce a semantic representation that can be used for reasoning. Hybrid systems that combine statistics with shallow parsing and templates are widely used for information extraction, but Hobbs and Riloff (2010) noted that such systems have reached a barrier of about 60% accuracy in recall and precision.

The most sophisticated NLP systems use heavyweight semantics based on some version of logic. Typical systems have two distinct levels: syntactic analysis to generate a parse tree and semantic interpretation to map the parse tree to a logical form. But after forty years of research, no system based on that approach can read one page of a high-school textbook and use the results to solve the problems as well as a B student. Even pioneers in logic-based methods have begun to doubt their adequacy. Kamp (2001), for example, admitted that "the basic concepts of linguistics — and especially those of semantics — have to be thought through anew" and "many more distinctions have to be drawn than are dreamt of in current semantic theory."

To understand the issues, consider the combination of syntax, semantics, and database structure necessary to analyze a question and answer it. As an example, the Transformational Question Answering system (Petrick 1981) analyzed English questions and used middleweight semantics about the subject matter to map English to and from logic. TQA also used heavyweight semantics to map logic to and from the SQL query language, which has the expressive power of first-order logic. The parser evolved from a research project that Petrick (1965) designed for his PhD dissertation under Chomsky's supervision. After joining IBM, Petrick collaborated with other researchers to develop TQA as an English front-end to a relational database.

To evaluate TQA's potential, IBM management wanted to test it on actual users. The nearby city of White Plains served as a test case. During the 1974 gasoline shortage, city officials had to search land-use records by hand to find the locations of all gas stations so that police could go there to direct traffic. Later, the records were stored on a computer, but somebody had to write a new program and print a new report for every question. Every follow-on question required another program. In 1978, the IBM researchers loaded the land-use files on a relational database at Yorktown, customized TQA to access the database, and connected it to a dedicated terminal in the city hall.

For a full year, the White Plains officials and land-use planners could type English questions to TQA and get immediate answers. Of 788 questions typed during the year, TQA answered 65% correctly and failed to parse 35%. For most parsing failures, the users rephrased the sentence in a way that TQA could answer. Occasionally, they called the IBM developers for help. Overall, the users loved it. They were unhappy when the trial period ended, and IBM unplugged the terminal (Damerau 1981). Following are some questions that TQA answered correctly:

```
What is the total area of the parcels in ward 6 block 72?
How many two family houses are there in the Oak Ridge
Residents Assn?
Where are the apartment dwellings which have more than
50 units
which are more than 6 stories high on Lake St?
```

The TQA test showed that subject-matter experts, who had no training in programming or database software, could effectively use an English front-end to conventional software. It also showed the kind of syntax and semantics that was needed to customize a language processor for each application. The syntax of the phrase *ward 6 block 72* is familiar to the SMEs, but it is rare in ordinary English. The TQA developers added grammar rules for many such phrases before the test period. During the test, they analyzed the questions that TQA failed to parse correctly and revised the grammar to accommodate them. The TQA users also learned to adjust their grammar to accommodate the parser.

The test version of TQA also generated a rudimentary *echo* that showed how each question was parsed. Unfortunately, some echos used syntax that the parser failed to recognize when the users typed them back. Mueckstein (1983) later designed Q-TRANS to generate an echo that TQA could always parse. Following is a question processed by TQA:

```
What parcels in the R5 zone on Stevens St. have greater than
5000 sq. ft.?
```

TQA translated that question to the following SQL:

```
SELECT UNIQUE A.JACCN, B.PARAREA
FROM ZONEF A, PARCFL B
WHERE A.JACCN = B.JACCN
AND B.STN = 'STEVENS ST'
AND B.PARAREA > 5000
AND A.ZONE = R5;
```

Q-TRANS translated that SQL to the following echo:

```
Find the account numbers and parcel areas for lots that have
the street name STEVENS ST, a parcel area of greater than
5000 sq. ft., and zoning code R5.
```

These examples show the kind of customization required by any processor that maps natural language queries to and from a computer system: first, an ontology of the entities, relations, and constraints in the subject matter; second, a lexicon that maps words and phrases to and from the ontology; third, specialized syntax for patterns that are rare in ordinary language; and finally, mappings of the ontology to computer formats and interfaces. To simplify the task, Damerau (1988) designed a tool to enable "database administrators to generate robust English interfaces to particular databases without help from linguistic experts." IBM management, however, decided that it was too complex for most customers and the potential market was too small to be profitable. Therefore, they canceled the TQA project.

TQA was one of many NLP systems that demonstrated usefulness for some applications, but were not commercially successful. Systems with lightweight semantics, such as Systran, have been more successful. Some of the most successful are search engines that index documents by the words they contain without using any explicit semantics. Google improved the search with statistical methods for deriving some implicit semantics from the patterns of cross references. In general, systems based on lightweight semantics depend on the readers to use their background knowledge to fill in the gaps, but no human army could process the huge volumes of data on the web. For some applications, statistical methods can filter out much of the irrelevant data, but even a thousand-to-one reduction in petabytes still leaves terabytes. NLP systems with heavyweight semantics are necessary to interpret the details.

3. Reasoning and Problem Solving

Since the 1950s, research in AI explored a wide range of techniques from neural networks to formal logic. But the classical AI paradigm combines some knowledge representation language with some formal or informal methods of reasoning. Two classical systems of radically different sizes illustrate the problems and the range of possible solutions: the very large Cyc system, which shows the power of a general-purpose, heavyweight semantics; and a simpler system designed for online sales, which shows the ease of use of middleweight semantics combined with semi-automated methods for knowledge acquisition.

The expert systems of the 1980s showed that the level of expertise increased as more rules and facts were added. Some AI experts estimated that a human level of intelligence could be achieved with less than a million concepts encoded in some computable form. Lenat and Feigenbaum (1987) summarized the arguments:

- Lenat estimated that encyclopedic coverage of the common knowledge of typical high-school graduates would require 30,000 articles with about 30 concepts per article. That justified the Cyc Project, whose name comes from the stressed syllable of *encyclopedia*.

- The Japanese Electronic Dictionary Research Project (EDR) estimated that the knowledge of an educated speaker of several languages would require about 200K concepts represented in each language.
- Marvin Minsky noted that less than 200,000 hours elapses between birth and age 21. If each person adds four new concepts per hour, the total would be less than a million.

For the Cyc Project, they concluded that a knowledge base "of under a million frames" could be constructed in one decade with $50 million and less than two person-centuries of work.

The original version of Cyc was an informal system of frames with heuristic procedures for processing them (Lenat & Guha 1990). But as the knowledge base grew, the dangers of contradictions, spurious inferences, and incompatibilities became critical. As a result, the frames had to be more highly structured, and the procedures became more systematic and tightly controlled. Eventually, the CycL language and its inference engines were rewritten as a superset of first-order logic with extensions to support defaults, modality, metalanguage, and higher-order logic. The most significant innovation was a context mechanism for partitioning the knowledge base into a basic core and an open-ended collection of independently developed *microtheories* (Guha 1991).

After the first 25 years, Cyc grew far beyond its original goals: 100 million dollars had been invested in 10 person-centuries of work to define 600,000 concepts by 5 million axioms organized in 6,000 microtheories. Cyc can also access relational databases and the Semantic Web to supplement its own knowledge base. For some kinds of reasoning, Cyc is faster and more thorough than most humans. Yet Cyc is not as flexible as a child, and it can't read, write, or speak as well as a child. It has not yet achieved the goals of the "sweeping three-stage research programme" outlined by Lenat and Feigenbaum in 1987:

1. "Slowly hand-code a large, broad knowledge base."
2. "When enough knowledge is present, it will be faster to acquire more through reading, assimilating data bases, etc."
3. "To go beyond the frontier of human knowledge, the system will have to rely on learning by discovery, carrying out research and development projects to expand its KB."

The first goal has been achieved. The second goal was far more difficult than expected. Cyc cannot yet read a textbook and map the knowledge to CycL, and it can only access external databases whose metadata or ontology has been mapped to CycL concepts. The third goal is still a dream.

Even though Cyc did not achieve all the original goals, it remains the world's largest body of knowledge represented in logic and suitable for detailed deduction. For any given problem, Cyc automatically selects the required axioms and an inference method that is suitable for that problem. The Cyc tools can also be used as a development platform for defining axioms that can drive other inference engines. As an example, Peterson et al. (1998) designed a knowledge compiler to translate a subset of axioms from CycL to more restricted logics that drive a deductive database: Horn-clause rules for the inference engine, and database constraints stated in SQL WHERE-clauses. For a sample problem, they extracted 5532 axioms (about 1% of the five million axioms in the Cyc knowledge base). Of those axioms, 84% could be translated directly to Horn-clause rules for performing inferences. The remaining 16%, which required full first-order logic, were translated to update constraints in SQL to ensure that the database is always consistent with the axioms.

For the first dozen years, the Cyc Project focused on research, but the academic research was not easy to commercialize. Later, they gradually increased the time devoted to applications. As a result, Cyc earned more money from applications in the years 2008 to 2010 than in the previous 24 years. Some of the fastest growing applications are in medical informatics. At the Cleveland Clinic, about 1700 axioms from the general Cyc ontology are used to understand and respond to a typical query. The applications show considerable promise, but most application programmers find it difficult to adapt their software and databases to the Cyc knowledge base. Although Cyc is primarily a reasoning system, it also supports an English interface, which requires customization similar to TQA.

In contrast with Cyc, which has been in continuous development for over 25 years, smaller AI systems can be implemented much faster. As an example, Tesco, a large UK retailer, sells a variety of goods, ranging from groceries to electronic equipment. For their online branch, Tesco.com, they wanted a flexible system that employees could update dynamically. One software vendor designed a system based on RDF and OWL, but Tesco employees could not modify it. Calling an OWL expert for every update would be too slow, and hiring one for every store would cost too much. They needed a simpler system that current employees could modify without lengthy and costly training.

As an alternative, Gerard Ellis, an employee of the vendor, designed and implemented a prototype of a more flexible system in just a few weeks. Tesco liked it, and the complete system was delivered to them in a few months (Sarraf & Ellis 2006). Unlike the heavyweight semantics of Cyc, which requires professional knowledge engineers to update and modify, the Tesco system had middleweight semantics that could be updated by Tesco employees who had no training in AI, logic, or ontology. Automated tools could also check that the knowledge base is consistent and help Tesco employees correct any errors. The reason why Ellis could implement the new system so quickly is that he had spent a dozen years in developing a toolkit of AI software and related technologies (Ellis et al. 1994). To replace the system that used RDF+OWL, he put together the following components:

- Conceptual graphs (CGs) as the internal knowledge representation with basic tools for storing, retrieving, and manipulating CGs. Communication with other components was based on the Conceptual Graph Interchange Format (CGIF).
- A version of controlled English (CE) as the notation for subject-matter experts (SMEs) with tools to map CE to and from CGIF.
- Ripple-down rules (RDR) as the technology for learning, reasoning, and maintaining the knowledge base with a mapping to and from CGIF.

The SMEs were Tesco employees, who used controlled English to edit the rules, get an explanation of how a conclusion was derived, and correct any errors by typing the conclusion that should have been derived. This application was designed for selling groceries and later adapted for the electrical and wine departments.

Ripple-down rules are derived from a decision tree that is compiled to a nest of if-then-else statements (Quinlan 1993; Compton et al. 2006). The raw data for deriving a decision tree is a set of *cases*, each of which is described by one or more conditions and one or more conclusions. Each link of the tree is labeled with one condition, and each leaf (end point) shows one or more conclusions implied by the conjunction of all the conditions leading to that leaf. To derive a complete and consistent tree, the algorithms detect possible conflicts, show the conflicting cases, and request additional information to resolve the conflicts. For major updates, the algorithms can derive a new tree from the raw data, but for minor editing, they can make local changes to the tree. For the Tesco application, SMEs describe the cases by CE statements, and the system generates the rules. Following are some rules derived for the grocery application:

- **`If a television product description contains "28-inch screen", add a screen_size attribute_inches with a value of 28.`**

- **If a recipe ingredient contains butter, suggest "Gold Butter" as an ingredient to add to the basket.**
- **If a customer buys 2 boxes of biscuits, the customer gets one free.**
- **If the basket value is over £100, delivery is free.**
- **If the customer is a family with children, suggest "Buy one family sized pizza and get one free".**

The RDR rule format has proved to be convenient for SMEs from a wide range of backgrounds, especially medical informatics. Compton et al. (2006) describe an application developed by pathologists who used RDR tools to derive a knowledge base of 16,000 rules from a set of 6 million cases. But RDR is just one of a large class of tools for *case-based reasoning*, which overlap methods of machine learning. Some of them, like RDR, draw sharp distinctions that can be expressed in a subset of logic. Others use statistics, clustering algorithms, neural networks, and fuzzy logic for learning and reasoning from cases without sharp boundaries. Still others use analogies, which can derive sharp or fuzzy distinctions under varying conditions.

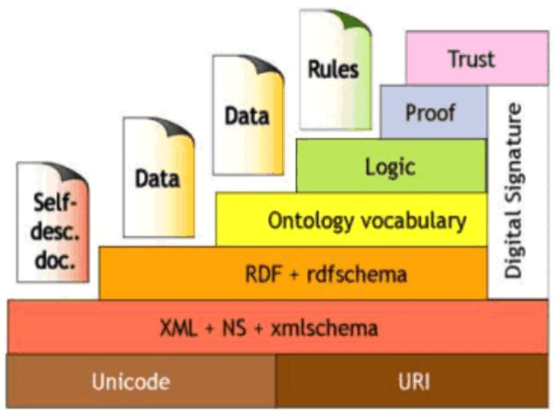
In summary, a large ontology such as Cyc does not, by itself, lead to successful commercial applications. A great deal of work on customization and knowledge acquisition is necessary to adapt Cyc to more conventional software. The Tesco.com application shows how systems with middleweight semantics can often simplify the task of knowledge acquisition. But the people who develop systems that SMEs find easy to use require advanced education and a toolkit of sophisticated software. With appropriate tools and methodologies, a convenient front-end could make any system easier to use. A challenging research goal is to develop an integrated knowledge acquisition system that could support both AI and conventional software.

4. Semantic Web

The Semantic Web was inspired by Tim Berners-Lee, but it was designed by a committee. It evolved from a keynote speech at the First World Wide Web Conference (Berners-Lee 1994):

> Adding semantics to the Web involves two things: allowing documents which have information in machine readable forms, and allowing links to be created with relationship values.

At this level of detail, nobody could object. But the speech didn't describe the machine readable formats, the kinds of relationship values, the logical operations on those values, or any influence from the 40 years of research on semantics in artificial intelligence, computational linguistics, and software engineering. The W3 Consortium, which met for the first time at that conference, took charge of the design. Although every branch and nearly every aspect of computer science was represented by one or more members of the W3C, a design by committee is a compromise of good people pulling in different directions for good, but often conflicting reasons. By 2001, some components of the Semantic Web had been specified by W3C recommendations, but the only consensus on the overall architecture was the so-called "layer cake" at the left of Figure 1.



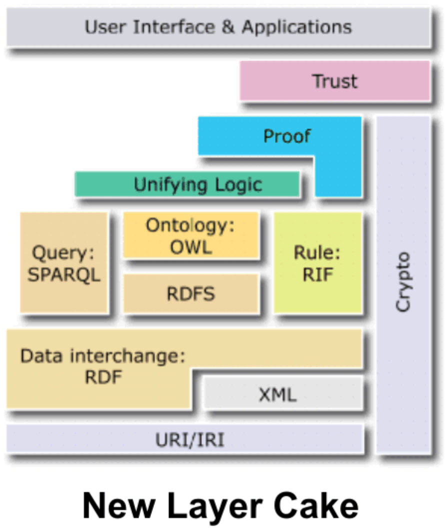


Figure 1: The architectural layer cakes for the Semantic Web

The new layer cake on the right of Figure 1 developed toward the end of the decade. The differences between the two cakes show significant changes in the evolution of the Semantic Web:

1. The original cake had clear layers that built on one another. The new cake adds more boxes, as one might expect, but it also lets some layers dip beneath their earlier foundations.
2. The Resource Description Framework (RDF), which was defined by XML Schema, now extends below XML. One extension allows RDFa, which consists of single attribute tags, to be placed in any XML area. But other variations have been used or proposed.
3. The digital signature pipe, which was supposed to be based on XML, is replaced by a cryptography pipe that goes beneath all layers, since XML and the browsers that process it cannot guarantee security.
4. The ontology vocabulary layer has been replaced by four loosely related boxes. The Web Ontology Language (OWL) could be used with the other boxes, but applications that use them are more likely to avoid OWL.
5. The logic layer has shrunk to a smaller box for a unifying logic, since the components beneath it use some sort of logic, but each of them has its own independently defined semantics.
6. Proof rests on top of the unifying logic, but it also dips beneath it to the Rule Interface Format (RIF).
7. Trust is the only layer that has not changed, primarily because nobody really knows how to achieve it.

The evolution of these components can clarify their interrelationships and suggest future directions. RDF began with a diagram by Tim Berners-Lee that showed how the links between documents formed a Giant Global Graph. The detailed specification evolved from an internal dispute in the Cyc Project. The director, Doug Lenat, wanted a single unified CycL language, but the associate director, R. V. Guha, considered CycL too complex for most users. Guha wanted a simpler subset of logic that would allow SMEs to read, write, and edit at least some of the knowledge base. Whatever the issues may be, Guha left Cyc to join Apple, where he designed a language called the Meta Content Framework (MCF). He later collaborated with Tim Bray to represent MCF in XML terms (Guha & Bray 1997). MCF was renamed RDF when it became a W3C recommendation, and Bray promoted it enthusiastically. But he later expressed serious concerns about the way it had developed (Bray 2003):

> Conceptually, nothing could be simpler than RDF. You have Resources, which by definition are identified by URIs. The resources have Properties, which by convention are identified by URIs. The properties have Values, which can be strings or numbers or Resources. Everything's a triple: (Resource, Property, Value)...
>
> Speaking only for myself, I have never actually managed to write down a chunk of RDF/XML correctly, even when I had the triples laid out quite clearly in my head. Furthermore — once again speaking for myself — I find most existing RDF/XML entirely unreadable. I think the Semantic Web people have taken on a job that's already tough, and are adding difficulty, and increasing the probability of failure, by sticking to the currently broken RDF/XML syntax.

Various tools provide more readable notations for triples, which are translated to and from the XML format. A popular alternative is the JavaScript Object Notation (JSON), which can represent an RDF triple as **`[R, P, V]`**. A collection of property-value pairs for the same resource could be written more compactly as **`{P1:V1, P2:V2, ..., Pn:Vn}`**. JSON is a humanly readable notation that is directly processed by JavaScript.

Although MCF had only a modest amount of logic, Guha and Bray noted that it could be used to describe its own structure and datatypes: "This self-description allows MCF to be its own schema definition language. This in turn allows MCF to be dynamically extended by an author or application." That principle was carried over to RDF: the base RDF notation has no built-in ontology, and RDF Schema (RDFS) contains a metalevel ontology for stating constraints on types and relations. The logic base (LBase) of RDF is simple, but quirky (Guha & Hayes 2002; Hayes 2003). A triple with three names (URIs or literals) represents a relation R applied to two arguments A and B: R(A,B). A collection of triples represents a conjunction:

$$R1(A1,B1) \land R2(A2,B2) \land R3(A3,B3).$$

But any argument slot could contain a URI that specifies a relation. In fact, a relation could even be applied to itself:

$$R1(A1,R2) \land R2(R1,B2) \land R3(R2,R3).$$

Furthermore, RDF allows "blank nodes" that represent anonymous entities: a triple of the form R(A,_) would say that something A is related by the relation R to some unspecified resource. In effect, a blank node represents an existentially quantified variable: $(\exists x)R(A,x)$. If a blank node happens to occur in the relation slot, then the quantifier ranges over relations: $(\exists r)r(A,B)$. That triple would say that there exists an unspecified relation r between A and B. The LBase semantics shows that this logic is consistent, but it allows combinations that go beyond the usual first-order logic.

To support reasoning, some version of logic with suitable rules of inference is required. For 2400 years, the most widely used version for representing and reasoning about ontology has been Aristotle's syllogisms. Description logics (DLs) are a family of formalisms that extend Aristotle's logic with features such as cardinality constraints and Boolean combinations of categories. McGuinness et al. (2002) showed how two description logics, DAML and OIL, could be adapted to the RDF notation to form OWL. But the combination of DLs with RDF exacerbated old controversies and created new ones.

For thirty years, the DL community has been divided between practitioners who use highly expressive languages to implement applications and theoreticians who prove theorems about computational complexity. The Loom and PowerLoom systems, for example, have been widely used for practical applications (MacGregor 1991; Chalupsky et al. 2006). But the theoreticians ignored PowerLoom because it's too expressive: it's possible to state undecidable problems. Yet every major programming language is undecidable, and programmers want more expressiveness, not less. For any language, reducing the expressive power does not make the easy problems easier to define or faster to solve. It just makes the hard problems impossible to express. The PowerLoom language became undecidable because the users asked for more expressive power; none of them asked for decidability.

CycL is an extremely expressive language, but undecidability has never been an obstacle. On the contrary, OWL has created more obstacles by its draconian measures to enforce decidability: the constraints on OWL cause all models to be tree structured. A benzene molecule, for example, has a ring of six carbon atoms. In OWL, it's not possible to state or imply that they form a ring, because a ring is not a tree. In Cyc, a knowledge engineer can choose tree models when appropriate and thereby guarantee decidability. Chemists, architects, and airplane designers, however, require graphs with cycles. They have developed highly efficient ways of representing them in both procedural and logic-based languages. They can represent them in Cyc, but not in OWL.

Computational complexity is a critical issue, and software engineers have developed ways of addressing it. Structured programming, design patterns, and their associated methodologies help programmers in several ways: provide a toolkit of useful, repeatable techniques; guide a design toward structures that are known to be safe, decidable, and efficient; and support tests for detecting problematical aspects. Yet all these methods are optional. They don't stop creative programmers from exploring innovative ways of using their highly expressive languages to invent new patterns. Software engineers also observe a time-honored principle: "Premature optimization is the root of all evil." Fine tuning one component is irrelevant and even counterproductive before its relationships to all other components are thoroughly understood.

The fragmentation of the ontology layer in Figure 1 is the result of developing the components independently without considering their roles in an integrated system. In the 1980s, description logics were used as one part of a hybrid system: a DL would define concept types in the terminology component (T-Box) while a more general logic used those types in the assertional component (A-Box). By giving priority to definitions in the T-Box, the hybrid had a modal effect of making T=box statements necessarily true with respect to the statements in the A-Box. The hybrid structure also allowed tradeoffs that improved efficiency while simplifying the task of knowledge acquisition. Some systems had three levels: a T-Box for defining concept and relation types, and an A-Box that was split between a rule-based reasoning system and a database for storing ground-level facts. Cyc has similar levels internally, but all levels use different subsets of the very expressive CycL notation.

That point about databases raises another issue: commercial web sites usually include relational DBs, which are as important as any component in the layer cakes. Some people claim that RDBs are obsolescent, but they still run the world economy. Furthermore, vendors of RDBs provide SPARQL interfaces to the tables, and vendors of triplestores provide SQL interfaces to theirs. Back in the 1980s, query systems like TQA might have been successful if their semantics could be derived as a byproduct of the database design methodology. In fact, Figure 2 shows a proposal from that era that could have and should have supported such systems (Tsichritzis & Klug 1978).

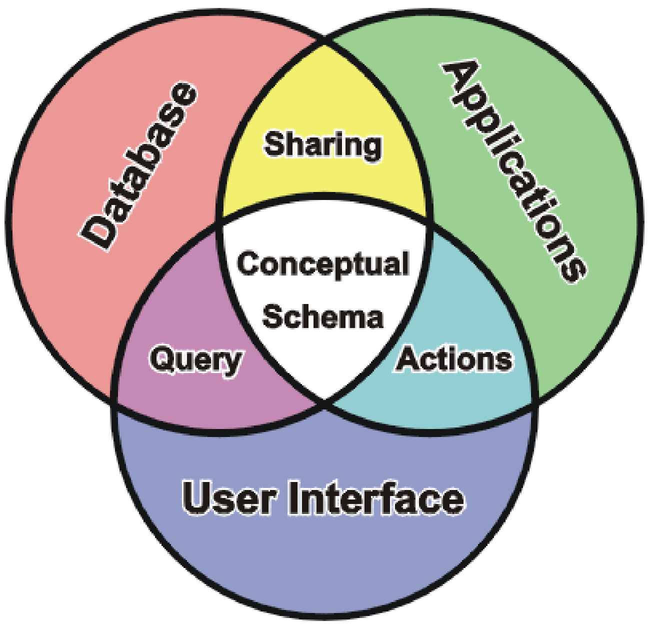


Figure 2: The ANSI/SPARC conceptual schema

The conceptual schema at the center of Figure 2 represents semantics, which includes logic and ontology. Unlike the layer cakes, Figure 2 makes semantics the foundation and relegates syntactic formats to the periphery. The same conceptual schema could represent the semantics of data organized in tables, networks, or hierarchies. And the meaning would remain constant under mappings to different application programs or user interfaces. Instead of leaving semantics for the comments, tools based on Figure 2 could begin with words and phrases the SMEs understand and create lexicons for systems like TQA. Unfortunately, the final C of SPARC represented a committee with conflicting experts pulling in different directions. The conceptual schema remained a technical report, and not a standard.

In summary, the goals of the Semantic Web were good, but the emphasis on syntax was a distraction. The strategy must begin with semantics: knowledge representation, reasoning methods, and knowledge acquisition. Guha had hoped to design a simpler notation than CycL, but the syntactic details of the components — RDF, RDFS, OWL, RIF, and SPARQL — dwarf the CycL manual in size and complexity. For special purposes, the semantics of a notation like SKOS (Simple Knowledge Organization System) is defined by a mapping to a larger component like OWL. That mapping enables OWL applications to use knowledge entered through SKOS. But there is no unified semantics that can define, relate, and share knowledge among all the components. Cyc, for example, allows each item of knowledge to be entered once and be reused in as many different ways as necessary. But the components of the layer cake have overlapping semantics, they tend to compete with one another, and any sharing that might occur is on an ad hoc basis. A coherent strategy should build on a unified semantic foundation, simplify knowledge acquisition, and promote the original goals of sharing and reusing knowledge among all systems connected through the WWW.

5. Language Analysis and Reasoning

The Holy Grail of knowledge acquisition is to design computer systems that can read a textbook and map it to logic. But that task is difficult, even for logicians. Hans Kamp, for example, was a graduate student at UCLA when he got a summer job at the RAND Corporation to translate an article from the *Scientific American* to logic. His thesis advisor, Richard Montague (1970), had claimed that

"Section 5 shows how the VivoMind Language Processor (VLP) uses all three kinds of semantics for language analysis and reasoning."

> There is in my opinion no important theoretical difference between natural languages and the artificial languages of logicians; indeed, I consider it possible to comprehend the syntax and semantics of both kinds of languages within a single natural and mathematically precise theory.

But when Kamp tried to translate the article from English to logic, he found that the mapping was far more difficult than anyone had thought. Some factual statements were fairly straightforward. But most sentences required new ontological assumptions, new translation rules, and sometimes *ad hoc* decisions about word senses and anaphoric references. That experience led him to develop *discourse representation theory* (DRT) on formal principles that went beyond Montague's (Kamp & Reyle 1993). Other linguists and logicians added more details and variations. But by the early 21st century, most of them agreed with Kamp that the basic principles "have to be thought through anew."

Even though a direct translation from language to logic or other computable form is not always possible, the opposite mapping from a formal language to a natural language is much easier and more systematic. It is also possible to relate formal programs to the documents that describe them, but the task requires a looser kind of analogy rather than a direct translation. The VivoMind Analogy Engine (VAE), for example, was used in legacy re-engineering: analyze and compare the programs and documentation of software in daily use that was up to forty years old (LeClerc & Majumdar 2002; Sowa & Majumdar 2003). Although the documents specified how the programs were supposed to work, nobody knew what errors, discrepancies, and obsolete business procedures might be buried in the code. Following is an excerpt from one of them:

> The input file that is used to create this piece of the Billing Interface for the General Ledger is an extract from the 61 byte file that is created by the COBOL program BILLCRUA in the Billing History production run. This file is used instead of the history file for time efficiency. This file contains the billing transaction codes (types of records) that are to be interfaced to General Ledger for the given month.
>
> For this process the following transaction codes are used: 32 — loss on unbilled, 72 — gain on uncollected, and 85 — loss on uncollected. Any of these records that are actually taxes are bypassed. Only client types 01 — Mar, 05 — Internal Non/Billable, 06 — Internal Billable, and 08 — BAS are selected. This is determined by a GETBDATA call to the client file.

No computer program could translate that text to an executable program, and even professional programmers would need much more analysis before deciding how to design the system. The problem of comparing previously written programs to the documents that describe them is much easier, but not trivial. Note that the text contains a large amount of jargon, and it mixes English words with the names of programs, files, and variables. Instead of references by name, some files are mentioned by descriptions such as "the 61 byte file that is created by the COBOL program BILLCRUA." The text also uses ad hoc syntax, such as "32 — loss on unbilled."

The project required an analysis of 100 megabytes of English, 1.5 million lines of COBOL programs, and several hundred JCL scripts, which called the programs and specified the data files and formats. Over time, the English terminology, computer formats, and file names had changed. Some of the format changes were caused by new computer systems and business practices, and others were mandated by different versions of federal regulations. The goal was to generate an English glossary of all processes and data, to note and generate cross references for all changes of terminology and definitions over time, to define the specifications for a data dictionary, to create dataflow diagrams of all processes, and to detect inconsistencies between the documentation and the implementation. Off-the-shelf software was available for analyzing COBOL programs, but not for analyzing the documentation and relating it to the programs. A major consulting firm estimated that the project would require 40 people for two years to read all the documentation, relate it to the software, create all the cross references, and produce the desired results.

For this project, Arun Majumdar and André LeClerc produced those results in 15 person weeks instead of 80 person years. To do that, they used VAE combined with a language analyzer called Intellitex, which translated English to conceptual graphs. The elapsed time was 8 weeks: 4 weeks for design, ontology, and additional programming for I/O formats; 3 weeks to run Intellitex, VAE, and the new programs on all the data; and 1 week to produce a CD-ROM with the results, which were exactly what the company had asked the consulting firm to produce.

During the computation, the combination of VAE and Intellitex analyzed the English documentation in terms of the semantic patterns specified by the COBOL programs and JCL scripts. Key to that analysis was a common knowledge representation in conceptual graphs (Sowa 2008). Even more important were the strategy and tools for using CGs:

- The first step is to use off-the-shelf grammars to analyze COBOL and JCL and add a back-end for generating conceptual graphs instead of executable instructions. That analysis also generates a lexicon of all the names of programs, files, and variables with cross references to the text sources and the CG translations.
- The next step uses VAE to index the CGs from COBOL and JCL to make them accessible while Intellitex is analyzing English. For N graphs, the indexing time is proportional to (N log N), but the time for VAE to find all graphs within a given semantic distance of a particular graph is proportional to (log N).
- Since English sentences are frequently ambiguous, many different CGs can be derived from the same sentence. During the analysis, VAE checks each option against the previously generated CGs to determine which ones are the most likely. Any CGs that match something derived from the programs are saved and indexed. The others are discarded as irrelevant.
- Pronouns and other anaphoric references are resolved by matching the newly generated CGs to other CGs derived from the same document. Names and vague references like “the 61 byte file” can be matched to any CGs derived from the entire corpus. Context surrounding the coreferent nodes can also be used to resolve ambiguities.

When VAE compares a CG derived from the current sentence to CGs derived from the programs or other documents, an exact match confirms the accuracy. If one CG has more or less detail than the others, there is no contradiction. Perhaps some program didn’t need all the detail, or some document didn’t mention it. But sometimes two CGs might represent different pathways through the background knowledge. Figure 3 shows different paths through the company’s database from market to location.

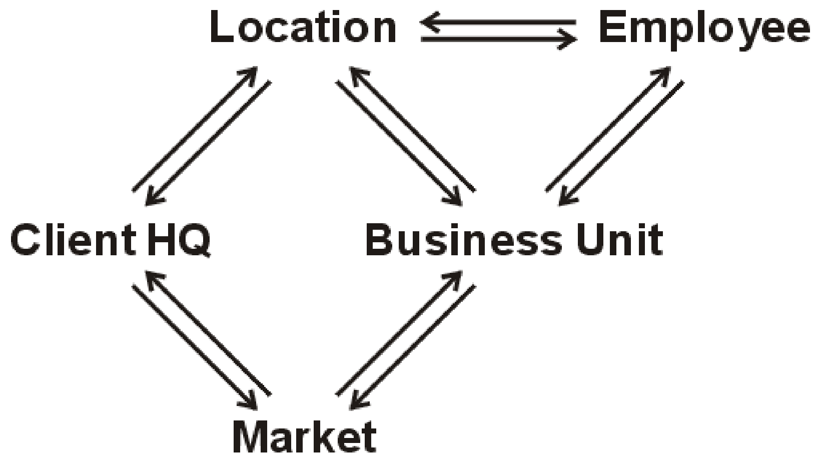


Figure 3: Different paths for relating market to location

VAE discovered that the CG derived from the documentation showed that the company's market regions are determined by the location of its business units. Sony Pictures, for example, would be in the California market, where the company has a business unit. But the CG derived from the COBOL programs shows that the market region is computed from the location of the client headquarters. Sony Pictures would therefore be in the Japan market. Some programmer had made a mistake, and management was making decisions based on incorrect assumptions. Nobody noticed the discrepancy until VAE discovered it.

As another example, the ontology implied that every employee is a human being and no human being is a computer. But CGs derived from COBOL showed that some employees were computers. The trail of pointers from those CGs led to a comment buried in a COBOL program that described an ad hoc patch. Back in 1979, two computers were used to assist human consultants. The company had standard procedures to bill customers for time spent by their employees, but there was no provision to bill for computer time. Therefore, the programmer named the computers Bob and Sally and assigned employee IDs to them. This was a "temporary" patch that would be removed when the project was finished. But few people clean up after finished projects. As a result, Bob and Sally remained employees for over 20 years before VAE discovered them.

Intellitex has a simple grammar, but it always produces some CG for any sentence. If a word is not in its lexicon, Intellitex capitalizes the word as a starting hypothesis about the associated concept type. If no parse is found for some string of words, it uses the completely unspecified relation **`(Link)`** to connect adjacent words. As an example, Intellitex would translate the phrase "32 — loss on unbilled." to a conceptual graph of the following form:

```
[Integer: 32]→(Link)→[Punctuation: "—"]→(Link)→
    [Loss]→(On)→[Entity]←(Thme)←[Unbilled]
```

The first line contains a concept of the integer 32 linked to some punctuation linked to a CG for the phrase *loss on unbilled*. The concept **`[Entity]`** in the second line is derived from a *canonical graph* for the participle *unbilled*, which by default would have some unspecified entity as its theme **`(Thme)`**. Then VAE would compare this graph to the previously generated graphs to find anything similar. For this example, VAE found comments in the data division of a COBOL program that mentioned "transaction code" and other comments that related 32 to the phrase "loss on unbilled." Similar phrases, such as "72 — loss on uncollected," used the same punctuation for the same semantics. But VAE also found that the syntactically similar phrase "06 — Internal Billable" was related to client types rather than transaction codes. To derive generalizations, detect exceptions, and refine hypotheses, some learning algorithms were later combined with VAE.

More recently, a new VivoMind Language Processor (VLP) has replaced the old Intellitex (Majumdar et al. 2008, 2009). One of the first applications was for analyzing documents about oil and gas fields, and answering extended queries by geologists who wanted to evaluate the potential for exploring new regions. Two geologists visited the VivoMind offices on a Monday morning and brought a collection of 79 documents in the geosciences domain. They ranged in size from 1 to 50 pages, some described sites of interest for oil and gas exploration, and others were chapters from a textbook on geology that VLP could use to extract background knowledge. The documents were not tagged or annotated in any way, except for the usual formatting tags intended for human readability.

The first test was to run the documents through VLP without adding any domain ontology and let the geologists ask questions. The answers were not bad, but they weren't much better than a typical search engine applied to the same documents. As a result of the analysis, VLP also produced a list of all terms that were not found in its lexicon. For the next four days, the geologists worked with the VivoMind staff to generate a domain ontology with lightweight and middleweight semantics. The first task was to classify the unknown words in several categories, such as Rock, RockFormation, Hydrocarbon, and GeologicalAge. Another task was to add domain-dependent word senses for common words, such as *basin*, *cap*, *corridor*, *fan*, *feeder*, *field*, and *reservoir*. The third task was to add a modest amount of background knowledge for resolving some of the ambiguities. The first task, which used lightweight semantics, was completed in about two days. It made a major improvement in the quality of the answers.

The next two tasks used Common Logic Controlled English (CLCE) to state some middleweight semantics. Instead of formal definitions, new word senses for the common words were introduced by stating simple CLCE sentences that use them in the new sense:

```
A cap on a well is a barrier.
A field that contains a hydrocarbon is under ground.
A reservoir that contains a hydrocarbon is under ground.
```

The background knowledge was also stated in CLCE:

```
Some ground is under water.
No city is under water.
Every reservoir that contains a hydrocarbon is in a field
that contains a hydrocarbon.
```

The geologists learned to write such statements during their visit, and they continued to add more background knowledge after they left. CLCE is general enough to represent full first-order logic (Sowa 2004), but this level of detail was sufficient for the task. Following is a sample sentence:

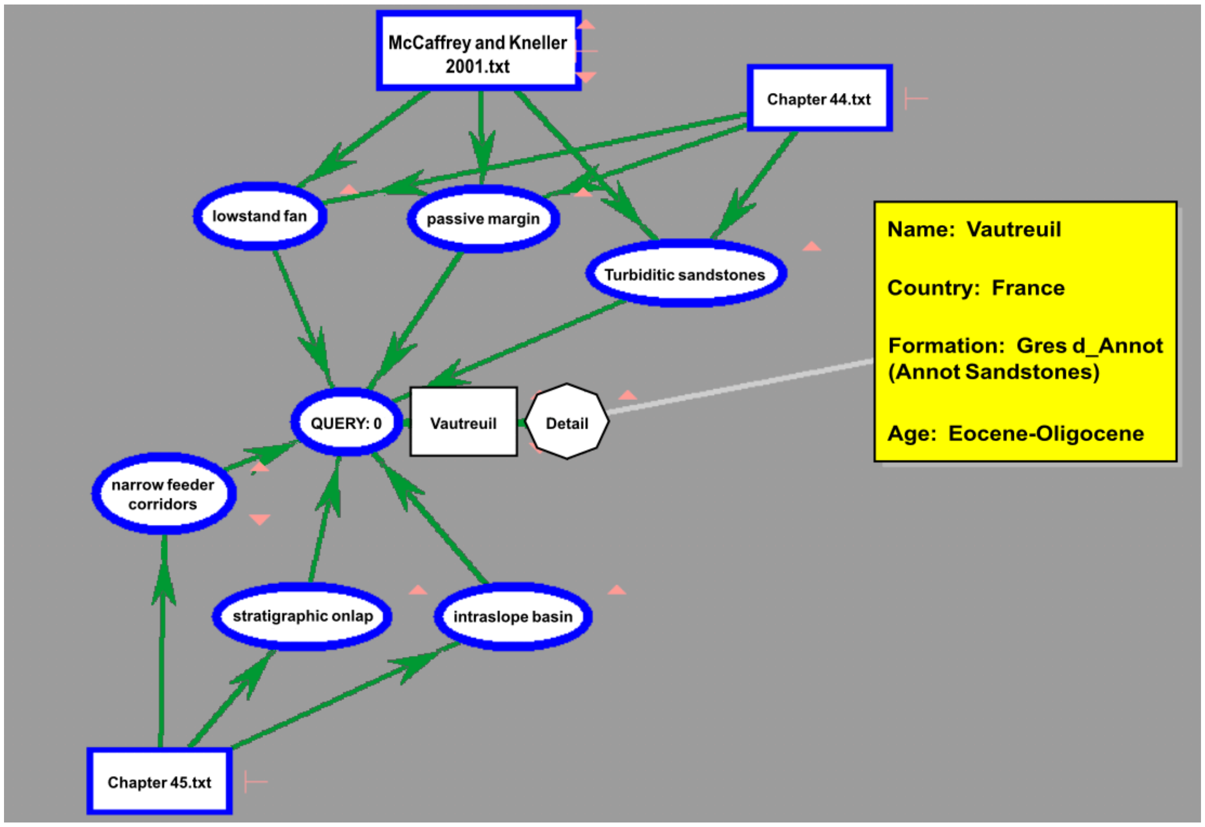
> The Diana field is situated in the western Gulf of Mexico
> 260 km (160 mi) south of Galveston
> in approximately 1430 m (4700 ft) of water.

If the sentence had ended with the word *Mexico*, the syntax would be unambiguous. But the measures in the next two lines, the parenthetical expressions, and the points for attaching prepositional phrases create ambiguities. Is Diana field or the Gulf of Mexico south of Galveston? What is in the water? Diana field, the Gulf of Mexico, or Galveston? After a devastating hurricane, Galveston was under water, but background knowledge should imply that cities are usually not under water. Majumdar et al. (2008) describe the VLP organization and how it uses background knowledge to resolve such ambiguities.

After they had added sufficient semantics to the domain ontology, the geologists who developed it invited another geologist from an oil company to test the system. He brought a small file that described a prospective site. He wanted VLP to compare all the sites it had analyzed to the following description, rank the sites that were most similar, and determine both the similarities and the differences for each site:

> Turbiditic sandstones and mudstones deposited as a passive margin lowstand fan in an intraslope basin setting. Hydrocarbons are trapped by a combination of structural and stratigraphic onlap with a large gas cap. Low relief basin consists of two narrow feeder corridors that open into a large low-relief basin approximately 32 km wide and 32 km long.

From the 79 documents it had analyzed, VLP found 17 sites that had some similarity. The most similar was in the Vautreuil region of France. It based that evaluation on three of the 79 documents. The report that described the Vautreuil site was essential, but VLP also extracted information from two chapters of the geology textbook in order to relate the geologist's query to that report. The screen shot in Figure 4 shows how those documents are related to the query.



Name: Vautreuil

Country: France

Formation: Gres d_Annot (Annot Sandstones)

Age: Eocene-Oligocene

Figure 4: Relating documents to a query

The oval at the center of Figure 4 represents the query. At the right is a short description of the Vautreuil region. That summary was extracted from a site report about the Vautreuil region written by McCaffrey and Kneller, represented by the box at the top. The six ovals surrounding the query oval contain English phrases, whose translations to conceptual graphs led to the documents used to answer the query. Three of those phrases were not found in the site report, but they led to the box for Chapter 45, from which VLP extracted CGs with background information that it used to interpret the site report. The three phrases above the query oval occurred in both the site report and Chapter 44 of the textbook. The result is a network of conceptual graphs that connect the geologist's query to the Vautreuil report via background graphs derived from chapters 44 and 45. The little red triangles in Figure 4 are links to windows that display relevant paragraphs from the source documents. Clicking on detail leads to a side-by-side comparison of the similarities and differences between the Vautreuil site and the site described by the geologist's query.

The VLP analysis goes into greater depth and precision than current systems for information retrieval (IR) and information extraction (IE). For each of the 17 sites related to the query, VLP found multiple documents that contained CGs derived from the query, CGs from the site report, and CGs from the textbook or other reports that contained background information. In a sense, VLP "learns" new information by reading a book. But for each query, it focuses only on those parts of the book that are useful for relating the query to the answer. This method is very different from current IR, IE, and DB systems:

- IR systems typically use a "bag of words" method to measure the similarity of a query to a document that might contain an answer to that query. But they don't extract the information and summarize it in a table or paragraph. It's possible to apply IR methods to individual paragraphs, but that technique would miss documents in which the significant words are scattered in different paragraphs. And no IR systems connect partial information from multiple documents.
- IE systems extract particular pieces of information, and some can link multiple pieces from different documents. Typical IE systems use predefined templates that specify expected syntactic and semantic patterns, but they have stagnated at about 60% accuracy. Hobbs and Riloff (2010) noted "it is not clear what we can do to overcome [that barrier], short of solving the general natural language problem in a way that exploits the implicit relations among the elements of a text." As Figure 4 shows, VLP doesn't need predefined templates. CGs derived from the query enable it to find implicit relations in a textbook and exploit them to generate precise answers.
- DB systems can relate and combine information from multiple sources, but they use query languages like SQL and SPARQL. Some support English-like front ends, but they face the same customization problems as TQA and Cyc. Furthermore, all the information they access must be predigested and translated to whatever format the database system requires.

Although conceptual graphs are defined as a formal logic, precise logic cannot be derived from a vague sentence. The CG that represents a sentence is actually derived by combining CGs from previously acquired knowledge. The precision of the result is determined by the precision of the original CGs. This method violates Frege's principle of *compositionality*, which says that the meaning of a sentence is derived from the meaning of the words it contains and the grammar rules for combining words. Montague was a strict adherent: each word is defined by one or more logical expressions, and each grammar rule has an associated semantic rule for combining those expressions. Montague allowed some words to have multiple meanings, but the grammar rules check semantic constraints to determine the correct option in each case. To support context-dependent references, Kamp's DRT uses information outside the sentence to determine interconnections. Both neat and scruffy systems make tradeoffs between the amount of meaning stored in the lexicon and the amount derived from context or general background knowledge. The high-speed analogy engine enables VLP to find and use much more background knowledge than most NLP systems.

In summary, VLP uses a combination of lightweight, middleweight, and heavyweight semantics. For any text, the broad outline of meaning comes from lightweight resources such as WordNet combined with middleweight ontologies with few axioms and definitions. The detail comes from background knowledge represented in conceptual graphs. At the heavyweight extreme, those CGs may be derived from formal logics, programming languages, and highly structured databases. The Common Logic standard (ISO/IEC 2007) specifies a model-theoretic semantics for CGs. But CGs have extensions beyond the CL standard (Sowa 2006, 2008), and they can also be used with “scruffy” heuristic methods.

6. Integrating Semantic Systems

Semantic systems have different interfaces for people with different requirements and skills. People with no training in programming or artificial intelligence, either casual users or subject-matter experts, should have interfaces that take advantage of their knowledge of the subject and their knowledge of their native language. Conventional programming tools and AI languages require different kinds of skills, and developers with experience in either or both should be able to collaborate. Automated and semi-automated tools should assist all developers in the stages of design, implementation, testing, and integration with other systems. The examples of Systran, TQA, Tesco, Cyc, and VLP illustrate the issues:

- Systran required highly trained linguists to design the dictionaries, but the resulting translations could be read by large numbers of people with no special training. TQA required DB administrators with special skills to customize the system for every application, even though the number of users of an application might be small. The cost of customization per TQA user was much higher than the cost per Systran user.

- Tesco wanted a more flexible system that could provide helpful suggestions to customers who visited their website. The software vendor designed a reasoning system based on OWL, but Tesco employees could not modify it. Ellis designed a new system with the same kind of interface for Tesco customers, but with a simpler interface for Tesco employees. Ellis’s design reduced the cost for updates by Tesco, but many software vendors don’t have employees with PhDs in computer science.

- Cyc was developed by 10 person-centuries of programmers and PhDs in several different fields. The cost of customizing Cyc is similar to the cost of customizing TQA, and success depends on the number of users per application. Each client must pay Cyc experts to customize the AI technology. That is a source of revenue for Cycorp, but it limits their market to large clients that can afford to pay.

- The Tesco system used a version of controlled English to enable employees with no AI training to read and update the knowledge base. The language for TQA users was called English, and it had greater expressive power than Tesco English, but it was just as tightly controlled. VLP was designed to process unrestricted natural languages, but it also supports controlled English for SMEs who update, supplement, and correct its knowledge base. All three systems show that users who know the subject matter can adapt to controlled NLs and use them effectively.

- People with every level of skills find diagrams helpful as a supplement to both natural and artificial languages. For the legacy re-engineering project, the VivoMind system translated the internal conceptual graphs to the more conventional dataflow diagrams and system structure diagrams used by programmers and systems analysts. The screen shot in Figure 4 is one of several kinds of diagrams that VLP generates from the internal CGs. They enable a geologist or other SMEs to explore the network of inferences and associations at different levels of detail. At each step, the system can follow the pointers from any CG to display the paragraph or paragraphs from which it was derived.

All five systems connect AI technology with conventional software, but the esoteric theories, languages, and methodologies of AI limit their use by most programmers and webmasters. With relational databases, Codd (1970, 1979) introduced first-order logic as the semantic foundation for database query languages. The conceptual schema illustrated in Figure 2 was an attempt to introduce even more semantics into database systems. It stimulated thirty years of research and collaboration between the AI and database communities, but most of that technology remains isolated from mainstream IT. To be successful, AI developers must find ways to simplify their development tools and integrate them with commercial software.

In contrast with the slow transfer of AI research to applications, the original World Wide Web addressed a specific problem with a well-defined goal: combine hypertext with the Internet, and let physicists get any report by clicking on a citation. It worked so well that everybody wanted to use it. The Semantic Web, however, began with the vague idea of adding semantics to the links. AI researchers, who were eager to promote their tools and theories, proposed them to the W3C. Those proposals led to the boxes in the layer cakes of Figure 1. Meanwhile, skeptics like Shirky (2005) claimed that "ontology is overrated." Like the conceptual schema, ontology is a fertile field for research, but most programmers who use XML for data interchange ignore OWL.

Yet the Semantic Web has been developing a valuable set of tools, and they should be better integrated with both AI technology and more conventional software. One way to begin is to fill the box for "unifying logic" in Figure 1 with Common Logic (ISO/IEC 2007). The semantic foundation for Common Logic is based on a proposal by Hayes and Menzel (2001). Guha and Hayes (2002) adopted that semantics for RDF, and it is compatible with every logic in the layer cakes. Common Logic has also been adopted as the unifying logic for the UML diagrams, which are widely used to specify conventional software (OMG 2010). A unifying logic can support development tools with precisely defined mappings between components from different communities with different notations and methodologies.

A promising opportunity for semantic applications is the movement toward Linked Open Data (LOD), and the largest single collection of data is held by the US Government. Vivek Kundra, the chief information officer, summarized the issues:

Kundra Kendra ?

> Just consider the huge experience gap that Americans have when they go online to make a hotel reservation or buy a book through Amazon versus how they interact with the public sector — whether it's paying taxes, applying for student aid or applying for Social Security benefits. (Quoted by Moyer 2010)

Yet Kendra's examples are far more complex than buying a book or reserving a hotel room. Commercial websites handle those routine transactions very well, just by using conventional programs and databases. For complex reservations, travelers prefer to talk with a human agent, even if they have to pay a service charge. For most commercial sites, the help facilities are notoriously poor, and it's unlikely that they could answer the kinds of questions people would ask about taxes, student aid, or social security. Search engines are popular because they're easy to use for finding documents, but some knowledgeable person or computer would have to read and understand them to answer such questions.

In the interview, Kendra used the term *data sets*, not *documents*, and most of those data sets are stored in databases. The initial goals are to make the data accessible by interfaces to the web, but those interfaces need not use AI technology. For many of them, tags that mark the datatypes are the only semantics. Such tags are useful for both conventional software and AI technology, but detailed reasoning requires more semantics. Fortunately, AI technology can also derive the semantics. The US Government is the world's largest publisher, and every data set is described or mentioned in many documents. A user-friendly interface should relate that data to the documents and answer questions by extracting related paragraphs. But most LOD projects aren't using NLP methods to process the documents and relate them to the data sets.

The VivoMind examples in Section 5 show how an integrated system of semantic tools can process both structured data and unstructured texts. The legacy re-engineering project shows that formal semantics derived from software can be used to interpret reports and manuals about the software. The method of answering a geologist's query shows how NLP systems can integrate semantics derived from multiple sources to analyze documents and answer questions. These methods are at the cutting edge of applied research, but they are likely to evolve rapidly during the coming decade. That evolution is inevitable, and better tools can facilitate the transition.

References

Berners-Lee, Tim (1994) W3 future directions, Keynote speech, First International Conference on the World-Wide Web, May 25-27, Geneva: CERN. http://www.w3.org/Talks/WWW94Tim/

Bray, Tim (2003) The RDF.net challenge, http://www.tbray.org/ongoing/When/200x/2003/05/21/RDFNet

Chalupsky, Hans, Robert M. MacGregor, & Thomas Russ (2006) *PowerLoom Manual*, ISI, Marina Del Rey, CA.

Codd, Edgar F. (1970) A relational model of data for large shared data banks, *Comm. ACM* **13:6**, 377-387.

Codd, Edgar F. (1979) Extending the relational model to capture more meaning, *ACM Transactions on Database Systems* **4:4**, 397-434.

Compton, Paul, Lindsay Peters, Glenn Edwards, & Tim G. Lavers (2006) Experience with ripple-down rules, *Knowledge Based Systems* **19:5**, 356-362.

Damerau, Fred J. (1981) Operating statistics for the Transformational Question Answering System, *American Journal of Computational Linguistics* **7:1**, 30-42.

Damerau, Fred J. (1988) Prospects for knowledge-based customization of natural language query systems, *Information Processing and Management* **24:6**, 651-664.

Ellis, Gerard, Robert A. Levinson, & Peter J. Robinson (1994) Managing complex objects in Peirce, *International J. of Human-Computer Studies* **41**, 109-148.

Feigenbaum, Edward A., & Pamela McCorduck (1983) *The Fifth Generation*, Addison-Wesley, Reading, MA.

Guha, R. V. (1991) *Contexts: A Formalization and Some Applications*, PhD dissertation, Stanford, and Technical Report ACT-CYC-423-91, MCC, Austin, TX.

Guha, R. V., & Tim Bray (1997) Meta Content Framework using XML, W3C, http://www.w3.org/TR/NOTE-MCF-XML-970624

Guha, R. V., & Patrick J. Hayes (2003) LBase: Semantics for languages of the semantic web, http://www.w3.org/TR/2003/NOTE-lbase-20031010/

Hayes, Patrick J., & Chris Menzel (2001) A semantics for the Knowledge Interchange Format, IJCAI'2001 Workshop on the IEEE Standard Upper Ontology, http:reliant.teknowledge.com/IJCAI01/HayesMenzel-SKIF-IJCAI2001.pdf

Hayes, Patrick, ed. (2004) *RDF Semantics*, W3C Recommendation, http://www.w3.org/TR/rdf-mt/

Hobbs, Jerry R., & Ellen Riloff (2010) Information extraction, *Handbook of Natural Language Processing*, 2nd edition, edited by N. Indurkhya & F. J. Damerau, CRC Press.

Hutchins, W. John (1995) Machine translation: a brief history, in E. F. K. Koerner & R. E. Asher, eds., *Concise History of the Language Sciences: from the Sumerians to the Cognitivists*, Oxford: Pergamon Press, pp. 431-445.

ISO/IEC (2007) *Common Logic (CL) — A Framework for a family of Logic-Based Languages*, IS 24707, International Organisation for Standardisation, Geneva.

Kamp, Hans, & Uwe Reyle (1993) *From Discourse to Logic*, Kluwer, Dordrecht.

Kamp, Hans (2001) Levels of linguistic meaning and the logic of natural language, http://www.illc.uva.nl/lia/farewell_kamp.html

Kassoff, Michael, & Michael R. Genesereth (2007) PrediCalc: a logical spreadsheet management system, *Knowledge Engineering Review* **22:3**, 281-295.

Moyer, Michael (2010) Digitizer in chief, an interview with Vivek Kundra, *Scientific American* **303:4**, October, 90-94.

LeClerc, André, & Arun Majumdar (2002) "Legacy revaluation and the making of LegacyWorks," *Distributed Enterprise Architecture* **5:9**, Cutter Consortium, Arlington, MA.

Lenat, Douglas B., & Edward A. Feigenbaum (1987) On the thresholds of knowledge, *Proc. IJCAI'87*, pp. 1173-1182.

Lenat, Douglas B., & R. V. Guha (1990) *Building Large Knowledge-Based Systems*, Addison-Wesley, Reading, MA.

Majumdar, Arun K., John F. Sowa, & John Stewart (2008) Pursuing the goal of language understanding, in P. Eklund LNAI 5113, Springer, Berlin, 2008, pp. 21-42. http://www.jfsowa.com/pubs/pursuing.pdf

Majumdar, Arun K., & John F. Sowa (2009) Two paradigms are better than one and multiple paradigms are even better, in S. Rudolph, F. Dau, and S.O. Kuznetsov, eds., http://www.jfsowa.com/pubs/pursuing.pdf

MacGregor, Robert A. (1991) The evolving technology of classification-based knowledge representation systems, in J. F. Sowa, ed., *Principles of Semantic Networks*, San Mateo, CA: Morgan Kauffmann, pp. 385-400.

McGuinness, D. L., R. Fikes, J. Hendler, & L. A. Stein (2002) DAML+OIL: An ontology language for the Semantic Web, *IEEE Intelligent Systems* **17:5**.

Montague, Richard (1970) Universal grammar, reprinted in R. Montague, *Formal Philosophy*, New Haven: Yale University Press, pp. 222-246.

Mueckstein, Eva-Maria M. (1983) Q-Trans: Query translation into English, *Proc. IJCAI 83*, pp. 660-662.

OMG (2010) *Semantics of a Foundational Subset for Executable UML Models*, Object Management Group, http://www.omg.org/spec/FUML/1.0

Peterson, Brian J., William A. Andersen, & Joshua Engel (1998) Knowledge bus: generating application-focused databases from large ontologies, *Proc. 5th KRDB Workshop*, Seattle, WA. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-10/"

Petrick, Stanley R. (1965) A recognition procedure for transformational grammars, PhD dissertation, MIT, Cambridge, MA.

Petrick, Stanley R. (1981) Field testing the TQA System, *Proc. 19th Annual Meeting of the ACL*, pp. 35-36.

Quinlan, J.R., ed. (1993). *C4.5: Programs for Machine Learning*, San Mateo, CA: Morgan-Kaufmann.

Sarraf, Qusai, & Gerard Ellis (2006) Business rules in retail: the Tesco.com story, *Business Rules Journal* **7:6**.

Shirky, Clay (2005) Ontology is overrated, talk presented at the O'Reilly Emerging Technology Conference, San Diego. http://www.shirky.com/writings/ontology_overrated.html

Sowa, John F. (2004) Graphics and languages for the flexible modular framework, in K. E. Wolff, H. D. Pfeiffer, & H. S. Delugach, eds., *Conceptual Structures at Work*, LNAI 3127, Springer-Verlag, Berlin, pp. 31-51.

Sowa, John F. (2006) Worlds, models, and descriptions, *Studia Logica*, Special Issue *Ways of Worlds II*, **84:2**, 323-360.

Sowa, John F. (2008) Conceptual graphs, in F. van Harmelen, V. Lifschitz, and B. Porter, eds., *Handbook of Knowledge Representation*, Elsevier, Amsterdam, pp. 213-237.

Sowa, John F., & Arun K. Majumdar (2003) "Analogical reasoning," in A. de Moor, W. Lex, & B. Ganter, eds., *Conceptual Structures for Knowledge Creation and Communication*, LNAI 2746, Springer-Verlag, Berlin, pp. 16-36. http://www.jfsowa.com/pubs/analog.htm

Tsichritzis, Dionysios C., & Anthony Klug, eds. (1978) The ANSI/X3/SPARC DBMS framework, *Information Systems* **3**, 173-191.

Wang, Hao (1960) Toward mechanical mathematics, *IBM Journal of Research and Development* **4**, 2-22.

Whitehead, Alfred North, & Bertrand Russell (1910) *Principia Mathematica*, 2nd edition, Cambridge University Press, Cambridge, 1925.