ARTICLE TYPE

Knowledge Representation and The Semantic Web: An Historical Overview of Influences on Emerging Tools

Michael DeBellis[a] and Robert Neches[b]

[a]mdebellissf@gmail.com *michaeldebellis.com, San Francisco, USA* [b] *Formerly University of Southern California, now retired, Los Angeles, USA*

ARTICLE HISTORY

Received:
Revised:
Accepted:

DOI:

Abstract: A suite of standards known as the Semantic Web is transforming the Internet to a semantic graph rather than a graph of hypertext links. This paper will describe how various ideas and initiatives in artificial intelligence knowledge representation influenced its design. We begin with the seminal work by Alan Turing and Alonzo Church that led to the definition of Turing Machines, enabled digital computing, and provided the mathematical theory of computation which has been one of the determining factors for Artificial Intelligence knowledge representation. We then provide a brief history of artificial intelligence knowledge representation starting with groundbreaking researchers such as Newell and Simon, then to the first "AI boom" driven primarily by rule-based expert systems followed by major initiatives such as Cyc and the DARPA Knowledge Sharing Initiative. We will discuss how innovations from these initiatives affected standards that in turn led to the suite of standards known as the Semantic Web. We conclude with a brief overview of the most important issues currently facing those who wish to see widespread adoption of Semantic Web technology in industry.

Keywords: Knowledge Representation, Semantic Web, Web Ontology Language, OWL, SPARQL, SHACL, SWRL, RDF/RDFS

1. INTRODUCTION

In 2001 Tim Berners-Lee, James Hendler, and Ora Lassila wrote one of the most influential papers in the history of Artificial Intelligence (AI) and Knowledge Representation [1]. The paper described efforts to develop knowledge representation languages such as the DARPA Agent Markup Language (DAML) that integrated with the Internet and provided it with a semantic layer [2]. Languages such as DAML provided the foundation for a new vision for the Internet called the *Semantic Web*. The Internet started as essentially a huge network graph formed by URLs and the various links to and from them. The Semantic Web is transforming it into a *semantic graph* rather than a graph of hypertext links. In hypertext, a link can have countless meanings. It can lead to a page where one can login to the site, to a document that describes the author of the page, to a page on a related topic, etc. The semantics of these links are not at all explicit. Rather, they are buried implicitly in the HTML and code that manipulates the pages. The Semantic Web captures the meanings of these various nodes and links and the metadata required to efficiently utilize them. It thus provides the foundation for intelligent agents and other new types of systems that go far beyond the conventional Internet.

This paper examines key AI research and how it influenced Semantic Web tools and standards. We begin with the mathematical foundations for all digital computers in the work of Alan Turing and Alonzo Church (section 2.1), and early work in AI by pioneers such as Newell and Simon (section 2.2). The advent of forward-chaining rule-based systems set the stage for the first practical applications (section 2.3). Limitations of the rule-based paradigm led to work on frames, which influenced both object-oriented programming languages and led to further research on representational power (sections 2.4-2.5). Differing philosophical approaches arose at this point (section 2.6). One approach is currently ascendant, but history and some critical observations make them important to understand because the pendulum may someday swing the other way (section 2.7).

The combination of growing power and a desire to scale to larger challenges engendered two major initiatives focused upon enabling very large reusable knowledge bases (Section 3) Resulting tools and lessons learned became essential influences on the core Semantic Web technologies that emerged, affecting how Semantic Web data is connected (section 4) and how its meaning is captured (section 5). Those tools and techniques initially focused upon open public use. The most recent initiative, spurred by work on Knowledge-Graphs at major corporations such as Google, Amazon, and Facebook bring in additional requirements for more closely controlled systems that provide structure and the elimination

2213-2759/19 $58.00+.00

of data silos to information behind the corporate firewall rather than on the public Internet (section 6).

We conclude with a brief discussion of significant issues and next steps facing those who want to see wide-spread adoption of Semantic Web technology to the Internet and industry (section 7). Figure 1 shows a timeline that summarizes some of the most significant events to be discussed in this paper.

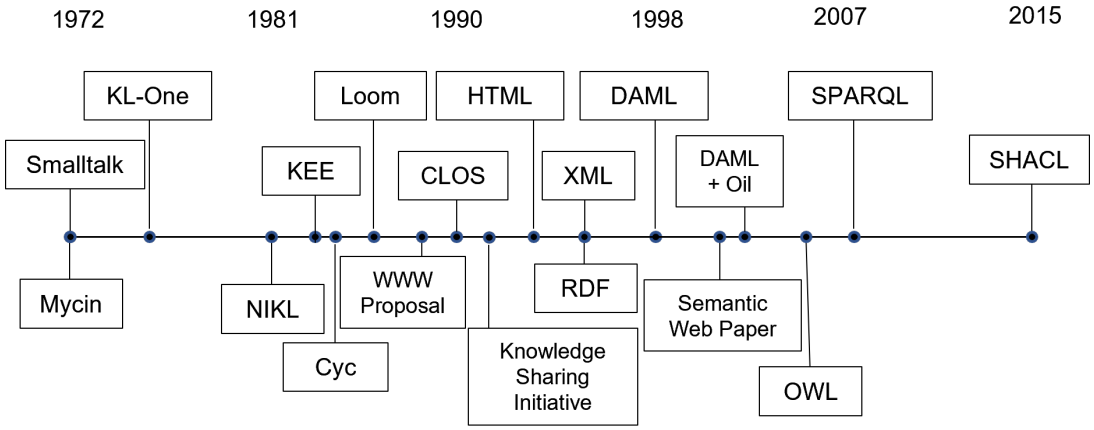

Figure 1. Knowledge Representation Timeline

2. Early History of Knowledge Representation

2.1. A Fundamental Tradeoff in Knowledge Representation and Reasoning

The roots of the semantic web and AI trace all the way back to the mathematical discoveries that provided the theoretical foundation for all modern digital computers. In 1936, Alan Turing published a paper [3] where he provided a solution to the *Entscheidungsproblem,* which had been described by David Hilbert as one of the most important unsolved problems in mathematics at that time.

The challenge of the Entscheidungsproblem was to create an algorithm for First Order Logic (FOL) that could take as input any set of formulas, and determine whether or not they were valid. An algorithm for propositional logic (Truth Tables) had existed for centuries. FOL extends propositional logic with only two quantifications: *existential* ("there exists some x such that…") and *universal* ("for all x such that…"). Thus, most mathematicians thought that it should be solvable via some equivalent algorithm.

As is not uncommon in math and science two researchers working independently hit on a solution to the same problem at almost the same time. In addition to Turing's proof, Alonzo Church created a very different proof a few months earlier. Both Turing and Church proved that -- contrary to previous expectation -- the Entscheidungsproblem is unsolvable [4]. Another way to say this is that they proved FOL was undecidable.

Turing defined the Turing machine model for his proof -- the mathematical model that all modern digital computers are based on. Church defined the Lambda Calculus. This, along with Newell, Shaw, and Simon's IPL (developed for their Logic Theory Machine), was the inspiration for John McCarthy's LISP programming language [5]. LISP is essentially a programming language implementation of Church's lambda calculus. It was one of the most important languages for AI research for several decades. Tree and graph based data structures trace their roots back to it, as do recursive programming techniques. LISP's ability to blur the distinction between data and code, and hence to write self-modifying programs, had a major influence on AI research.

The Turing/Church proofs defined one of the most important concepts for knowledge representation in artificial intelligence. First Order Logic was seen by most researchers as the most expressive language to describe an algorithm [6]. However, since FOL was undecidable, any knowledge representation language that had the full power of FOL would also be undecidable. I.e., any automated reasoner for a language with the full expressive power of FOL would not be guaranteed to terminate.

A paper by Hector Levesque and Ron Brachman [7] captured for AI researchers that a similar principle governs knowledge representation languages: the more expressive the language, the slower was any automated reasoner that could support it. This trade-off between the expressive power of a knowledge representation language vs. the performance of automated reasoners remains to this day as a driving force in designing AI knowledge representations.

2.2 Early Work in Theorem Proving and General Problem Solving

Newell, Shaw, and Simon's Logic Theory Machine, one of the first AI systems, emulates the proof process used to prove theorems in Whitehead and Russel's three volume Principia Mathematica. It was an early example of automated theorem proving and automated reasoning. It introduced concepts such as *forward-* and *backward- chaining* and of decomposing problems into sub-problems. These were essential to virtually all knowledge representation research to come.

Newell, Shaw, and Simon next worked on a General Problem Solver. Their goal was to "understand the information processes that underlie human intellectual, adaptive, and creative abilities" [8].

2.3 Rule-based systems

This research resulted in many useful insights about the structure of goals and plans but was overwhelmed by complexity with respect to real world problems. In addition,

solving general problems requires common sense reasoning which will be discussed in section 3. The first applied examples of AI research were expert systems, pioneered by projects such as Mycin. Mycin and other expert systems significantly reduced the expressiveness of their knowledge representation languages to languages based only on if-then rules and inferencing based on forward and backward chaining of rules [9].

Edward Feigenbaum, an early student of Simon's (Simon was Feigenbaum's PhD thesis advisor at CMU), [10] was a key player in this shift. Feigenbaum had two major insights:

1. Knowledge is as – if not more – important as reasoning methods in performing tasks requiring intelligence.
2. The best way to explore this was to refocus from general intelligence to what might effectively be considered idiot savants that were narrowly focused on a single domain such as medical diagnosis.

Feigenbaum and his colleague's success had the technical result of cementing the view of systems as divided into knowledge bases and inference engines, and the practical effect of establishing the feasibility of commercial, practical AI applications [6].

Rule-based systems fueled the first AI boom in the 1980's [11]. The rule-based languages and inference engines of researchers' expert systems soon transitioned from the laboratory to industry. This was fueled by the development of new commercial products called expert system *shells* that refined the rule-based languages and inference engines from academic research into industrial-strength tools. Over and above specific applications, the benefits of these rule-based shells for rapid prototyping was a very significant – and somewhat underappreciated -- accomplishment of this first wave of AI.

In the 1980's Information Technology (IT) projects typically entailed months of specification followed by months before the first version of a system was available. A process graph for that approach resembled a waterfall, with each step (Analysis, Design, Development, etc.) requiring completion before the next began, leading to it being called the Waterfall Software Development model. Providing some of the first major impetus to change this model is one of the underappreciated impacts of the first wave of AI. Not only did AI bring new technologies into IT systems, it exposed industry to new development models, enabling rapid iterations in weeks rather than months and emphasizing performing many phases (e.g., design, development, and testing) in parallel rather than sequentially [11].

This new approach to software development dovetailed with research and industry experience in software engineering, particularly Barry Boehm's paper describing an alternative to the Waterfall Model known as the Spiral Model employed at TRW Defense Systems to build embedded software for satellites [12]. These threads eventually culminated in the Agile model, now practiced by many industry leading software development groups, which has shown significant benefits to reducing risk, decreasing cost, and increasing quality of software [13].

It is common to view Expert Systems as a "failed" idea that led to an "AI winter" [14]. There is an alternative interpretation of this history: rule-based systems succeeded, but *AI-centric* tools failed. As rule-based systems were integrated with other industry tools, it soon became apparent that rules were a powerful tool for defining complex business logic of any kind. The rapid prototyping capabilities of rule-based shells allowed rapid reviews of the logic to validate it was correct. The high level of the rules (i.e., that they were a very limited subset of FOL) meant that with minimal training, end users could review and, in some cases, even modify the rules themselves – a big leap over filing change request forms.

During this time, one of the authors, Michael DeBellis, was a member of the AI group in Accenture's[1] Technology Services Organization. In that experience, deployed systems were virtually never classic standalone expert systems. Instead, rule-based shells were used to understand complex logic and then integrated with larger mainframe systems and databases. Alternatively, the rules replaced traditional specifications and were re-coded in conventional languages of the time such as COBOL.

For example, one of the first AI projects that gained significant attention within Accenture was for an Oil and Gas client. The accounting and tax rules for Oil and Gas are extremely complex. An Accenture team had been at the client site for apx. 6 months using Accenture's Method/1 Waterfall methodology to capture these requirements. However, they were unable to satisfy the client that the documentation adequately defined the complex business logic. A team of one staff and one manager from the AI group spent 2 months developing a prototype with a PC rule-based shell. This satisfied the client that the rules were adequately defined and the rules from the PC shell were then reimplemented in COBOL.[2]

As a result, enterprise systems began including rule-based systems as part of their toolkit. Enterprise Resource Planning (ERP) products such as SAP included rules for defining business logic [15]. Customer Relationship Management (CRM) tools such as Siebel included rules for defining promotions to targeted customers [16]. Enterprise Application Integration (EAI) middleware tools such as Tibco included rules to automate event processing [17]. In addition, most standards and vendors for enterprise programming environments began to include rules as a standard component of the tools that IT developers had at their disposal [18] [19].

In this interpretation, the first wave of AI hardly failed. On the contrary, it succeeded so well that the technology was absorbed into mainstream enterprise products -- which killed the market for standalone rule-based tools. One observation supporting this revisionist history is that several AI tool vendors refocused or were absorbed into enterprise tools. For example, Intellicorp, a company founded by Feigenbaum and the vendor of KEE (Knowledge Engineering Environment, one of the most sophisticated first wave AI tools), shifted from

[1] At the time Andersen Consulting.

[2] This and the previous paragraphs are supported by conversations with Chunka Mui a colleague in the Accenture AI group.

being a general purpose AI tool to a strategic partnership with the largest ERP vendor in the world, SAP [20].

2.4 Semantic Nets to Frames

While rule-based systems were catching commercial attention, representational research began focusing upon Semantic Networks. A Semantic Network is an undirected graph of concepts (nodes) and connections (links). Semantic networks were first utilized by Quillian as a way to model human memory [21].

Lindsay, Norman, and Rumelhart developed a knowledge representation system called MeMod (Memory Model) that employed semantic network graph representations with typed *is-a* hierarchies in which concepts were defined as nodes within a graph of *is-a*'s, with other typed links connecting them to other typed nodes, inheritance of attributes defined as links to other types, and a notion of concepts as type definitions and instances as type instances [22] [23]. These semantic graphs were called *is-a* hierarchies because they modelled concepts starting from very general (e.g., *Animal*) progressing to more specific (e.g., *Dog*) and with the leaf nodes of the graph as instances of concepts. E.g., *Dog is-a Animal* and *Fido is-a Dog.*

The definition of certain types of links in a semantic network to represent specific kinds of knowledge such as *is-a* was described by Ron Brachman as the epistemological layer for a semantic network [24].

This led to a new model for knowledge representation organized around *is-a* hierarchies first called Schemas by researchers such as Don Norman and David Rumelhart.[25] [26] and later called Scripts and Frames by researchers such as Roger Schank [27].

2.5 Objects and Frames

One of the next advances in industrial use of AI was to integrate Frame research with the rule-based inferencing of expert systems. Much of the inferencing in expert systems that was represented as rules could be modeled as classification of a node into a Frame hierarchy. For example, determining the diagnosis for a disease could be viewed as finding the appropriate position for a *Diagnosis* instance in an *is-a* graph starting from very general concepts such as *BloodDisease* and navigating to specific concepts such as *SickleCellAnemia*.

The data that was queried and set in rules were typically no longer simply variables but rather slots on frames. The data stored in frame languages was described as an ontology. Eventually, virtually all of the expert system shell vendors included some capability to define Frame hierarchies in addition to rules.

At the same time Object-Oriented Programming (OOP) languages such as Smalltalk were beginning to be used in industry [28].

The distinction between frame-based languages and OOP is full of ambiguity. Message-passing was incorporated in several frame-based languages, such as the Knowledge Engineering Environment (KEE), [29] and in OOP languages such as Smalltalk. There was a large amount of cross pollination between the two communities in both academia and industry, facilitated by common roots to MIT's AI Lab and Xerox PARC. Mark Stefik, one such bridge, brought his work on OOP in Common LISP to Smalltalk [30].

Object-oriented programmers were more influenced by software engineering. Object-oriented programming was essentially a natural extension of the theory of abstract data types and the design goal of encapsulation [31]. Frame-based systems were focused on AI research, embracing a wide array of various techniques for different domains and problems. Frame systems thus tended to be much more eclectic and diverse. Some of the capabilities of frame systems included:

1. **Facets on slots.** Slots were the equivalent of properties in OOP. However, unlike those languages a slot could also have additional information stored on it. For example, when was the slot last accessed, who accessed it, a certainty value representing how certain the reasoning was that resulted in the value of the slot, etc. This additional information was stored on a data structure known as a facet. Each slot could have as many facets associated with it as the designer required. This allowed slots with facets to essentially be n-ary relations rather than just binary. For example, *hasEmployer* could be a ternary relation between an *Employee*, the *Employer* and the *startDate* when they were first employed. This capability is very similar to property graphs and RDF*. This and other Semantic Web analogs of Frame language capabilities will be discussed below in sections 5-7.
2. **Default values.** A default value for each slot could be defined, either as a facet or as a separate feature. When a new instance of a class was created, any slot with a default value would automatically be filled with that value. Default values were usually inheritable. E.g., if the default value for *hasCovering* on *Mammal* was *Fur* then all subclasses of *Mammal* such as *Dog* would inherit that default value unless it was over-ridden in the same way as methods can be over-ridden by new sub-classes.
3. **Triggers**. Triggers could be defined for slots. This was code that would be executed whenever a value was put and/or retrieved from the slot.
4. **Constraints**. Slots could include various data integrity constraints such as the maximum or minimum values required for the slot, the datatype, a range of legal numeric values, etc. In Frame languages there was no distinction between using constraints for reasoning and using them to validate data integrity.
5. **Strong rule integration**. Frame systems often had a strong integration with rule-based inference engines. Not only could they invoke a rule base to infer over an ontology, they could also include capabilities such as truth maintenance (when a fact changed that would impact information inferred by rules the rules would automatically be re-invoked), possible worlds (the ability to create different versions of an ontology based on alternative assumptions), and explanation engines (the ability to explain how a certain value was inferred using the trace of rules that fired).

Although Frame languages haven't died out completely, OOP is now the dominant paradigm. In industrial software development OOP has moved from a leading-edge technology to a best practice followed by most IT organizations for new software development projects with mainstream languages such as Java and Python. Although frame languages are still used in academia, even there they have mostly given way to OOP.

OOP came to dominate because of elegance and simplicity, coupled with a larger support base due to interest outside AI. Features 2-4 in the list of Frame capabilities can be achieved by the use of constructor, get, and set methods, a standard practice of OOP [32]. A constructor is a method used to create a new instance of a class. Among other things, it can set default values for properties. A get or set method can check constraints or trigger arbitrary code every time a property is accessed or set [33].

Facets can be implemented in OOP via a design pattern that creates a new class that stores the information that would be stored in facets. For example, the ternary relation *hasEmployer* cited above could be implemented in OOP by creating a new class called *Employment* with properties *hasEmployer* and *startDate*. The result would be that the property on the *Employee* class would be *hasEmployment* with its range being the *Employment* class.

When designing large systems, the simplicity of OOP made collaboration between different developers and re-use of code simpler. The diverse capabilities of Frame languages meant that different developers would choose different approaches to implement the same functionality and would make the code of one developer different than that of another. By limiting the options of the language to methods OOP made code more uniform and hence more maintainable and reusable [33].

In addition, OOP enabled encapsulation. Because the data of a class was only available via get and set methods, the implementation details of the class could be hidden from developers that re-used it. Thus, the implementation could be changed and as long as the public interfaces (methods) didn't change, the code of those who reused the class would still work. A classic example is a *Dictionary* class might store its elements at first using an array and then later convert to a hash table for better performance. Encapsulation means that the code that reuses the *Dictionary* class won't need to be modified because the implementation detail of using an array or hash table will be hidden from the objects that use the class [33].

2.6 Formalist (Neat) vs. Heuristic (Scruffy) Thinking

In parallel with these technical developments, researchers began noting two loose and not cleanly differentiated philosophical approaches. These had many names, but common underlying themes. One theme contrasted starting points: tackling AI challenges and asking what representations helped vs. starting with a representation and exploring extensions and uses. This closely correlated with focus on demonstrating a capability vs. concern for decidability, performance, and confidence in outcomes. This correlated with lesser or greater concern with the ability to analyze and predict behavior and performance of the tools used. Fuzzy as these distinctions were, they are worth understanding because of implications they still hold for current and future systems.

Within the Frame community, the two different approaches were labelled by Roger Schank as "Scruffies" vs. "Neats". The Neats were researchers such as John McCarthy and Ron Brachman who wanted to start from a principled (formal) framework and ask what could be accomplished within it. The Scruffies were people such as Schank who took a more bottom-up approach and utilized whatever technology they thought was most appropriate for their specific problem.

These attempts to differentiate into schools of thought were not well-defined. Very influential researchers, e.g., Herb Simon, bridged them. Simon appreciated the power of formal tools, but he also took a very pragmatic approach and advocated that the principles governing intelligent systems were best studied by trying to build them and learning from what did and didn't work [34].

Similar distinctions arose in the Linguistics community under the rubric of "west coast" vs. "east coast". West coast linguists were influenced by people such as George Lakoff at the University of California at Berkeley.[3] Lakoff's students study semantic concepts such as metaphor and are significantly influenced by research in cognitive science such as Ellen Rosche's experiments that demonstrate how human concepts don't always conform to formal set theory [35]. There was a significant amount of overlap between the Scruffy approach to AI and the West Coast approach to Linguistics. For example, in his book, "Women, Fire, and Dangerous Things," Lakoff goes into significant detail about the concept of Frames and Roger Schank's work [36].

The east coast linguists were most influenced by Noam Chomsky at MIT. . Chomsky extended Turing and Church's computational theory work by defining a hierarchy of mathematical computation models from the most simple (Finite State Machines) to the most complex (Turing Machines), along with the kind of languages that they could parse. E.g., Finite State Machines can only parse context free languages. Recursively enumerable languages such as all human natural languages require a Turing Machine. In addition to linguistics, Chomsky's language hierarchy had significant impact on computer science such as compiler design [37].

Chomsky and his students use logic and set theory models such as X-bar theory to explore mathematical models for the syntax rules of human languages. Chomsky's current work hypothesizes that ultimately all language transformations can be decomposed into basic set formation capabilities: adding an element to a set or taking the union of two sets [38].

Returning to AI, some of the leaders in addition to Schank closer to the "Scruffy" mindset were Norman, and Rumelhart [39]. They defined Frames (also called Scripts) as a way to partially explain human language processing. The core idea was that part of understanding human language were high

[3] Personal communication between Michael DeBellis and Schuyler Laparle, graduate student in Linguistics at UC Berkeley.

level concepts that set expectations. Computationally, this was a way to prune the huge search space of possible parsing options for any sentence. For example, when a person enters a restaurant, they invoke the *restaurant script*. This is a frame with various slots such as *server*, *menu*, *meal*, etc. This frame is used to guide the parsing of sentences within that context.

On the more formal side, Ronald Brachman was one of the people whose research contributed substantially to what would become OWL. In work that began with his 1977 Ph.D. thesis, Brachman extended semantic networks into the first formal Frame language called KL-One. KL-One had an automated reasoner called a classifier. An issue with Scruffy frame languages and OOP languages is that while there are guidelines as to what it means for one class to be a subclass of another, these guidelines are only defined in documentation. They are not formalized in the language itself. KL-One was the first language to change this. In previous languages the user manually defines a hierarchy of classes from general to more specific and then finally to the leaf nodes which are instances. In KL-One the user defines a set of axioms (logical statements) and the classifier automatically defines the class hierarchy and other aspects of the ontology that would normally be defined manually. The classifier is a type of automated theorem prover. Similar to inference engines for rule-based systems, classifiers reason over a subset of FOL. However, the classifiers were able to cover a larger (hence more expressive) subset of FOL than rule-based inference engines because the language was not restricted to if-then rules.

2.7. Description Logic

Brachman found that the range of expressivity in KL-One was too great for practical use. This led to research by Brachman and others into the theoretical foundations for formal Frame languages, and consequent definition of *Description Logic*. Recall that Turing and Church proved that First Order Logic (FOL) was undecidable. The primary goals of Description Logic research were:

1. Find the maximal subset of FOL that was still decidable.
2. Formally describe the various levels of complexity that Frame language could have and the implications each level had on the performance of the reasoner [40].

As will be seen below, Brachman's approach is currently ascendant. Nevertheless, considerations raised in the preceding section linger. Note that virtually all common programming (e.g., Java, Python, Lisp) and query languages (e.g., SQL) are not decidable -- almost all programmers have accidentally written code with infinite loops. Researchers such as John Sowa have pointed to this to argue that what really matters is having a maximally expressive language that allows developers to define knowledge in the most powerful and intuitive format [41]. This opinion is not restricted to Sowa. For example, in a workshop on Term Subsumption languages (the technical name for languages such as KL-One and Loom) the question of decidability was a frequent topic of discussion [42]. In the real world, decidability is not the ultimate issue, performance is. A language that is decidable but takes a decade to return a solution is useless.[4] On the other hand, some languages such as SPARQL are potentially undecidable but good implementations (e.g., the SPARQL implementation in AllegroGraph) provide warnings to developers when queries are at risk of causing excessive time or infinite loops [43]. OWL itself includes a profile (OWL 2 RDF Semantics, previously OWL Full) that is not decidable [44]. OWL Full is avoided by virtually all Semantic Web developers and the emphasis in the community at this point in time is clearly on decidability. Whether the pendulum will swing back is unknown, but history suggests the possibility.

3. Building AI Systems at Scale: Very Large Ontologies

The next phase of AI research in knowledge representation was to a great extent driven by the problem of how to build, manage, and use very large ontologies. Up to this point, research tended to be confined to individual problem domains such as diagnosing diseases. The goal of managing large ontologies was primarily driven by three requirements to scale AI to larger problems:

1. The need to solve the problem of common sense reasoning.
2. The success of OOP and the resulting emphasis on building systems via re-using predefined components as opposed to developing custom software from scratch.
3. Combining structured (machine readable) and unstructured (human readable) information in an integrated representation and reasoning environment.

Common sense reasoning has been and still is one of the most critical unsolved problems of AI [45]. In the pop science press it is common to hear people discuss "General AI". Most AI researchers think that General AI will, at a minimum, require addressing Common Sense reasoning. I.e., in any typical human conversation there are countless examples where in order to understand a sentence the listener must apply common sense knowledge that even children simply take for granted. Facts such as that things fall down and not up, that fire is hot, that water is wet, etc. While humans take this for granted, AI researchers have found it to be a very difficult problem.

Issue 2 was due to the fact that the business world and US Department of Defense both experienced great benefits from the use of OOP. However, one of the promises of OOP, that systems could be composed from reusable components, while significantly aided by OOP languages still failed to be completely realized due to problems related to finding and re-using software components. It was thought that techniques from knowledge representation might help to address these problems and finally realize the goal of software construction via composition rather than duplicative development.

Issue 3 was a result of an evolution in the understanding of how AI could best work in the real world. One of the lessons learned from Expert Systems was that standalone systems that simply took in input and provided answers had limitations.

[4] Email communication Martin O'Connor to Michael DeBellis.

Most humans were reticent to blindly turn over decision making to computers [46]. Users have much more trust and desire to utilize systems that are interactive and can explain their reasoning [47]. Another driver for 3 was the wide-spread adoption of knowledge management systems such as Lotus Notes through many US corporations. These systems demonstrated that there was a market for systems that could represent and reason over structured and unstructured knowledge. Also, the growing realization of the major impact that the Internet was having on reshaping modern culture and business was a key driver to the requirement for systems that could represent and manage both machine and human readable knowledge.

Tom Malone was among the first to attempt to bridge structured knowledge representations and unstructured data, as part of an intelligent email system that supported frames and allowed rules about structured elements but allowed slots to be filled with either structured elements or free text [48]. Neches' research group tried to take this further, attempting to build a human-in-the-loop reasoner that would seek assistance to put it back on track when a typed slot was filled with untyped data [49] [50].

There were two major efforts in the US focused on these issues. The Cyc system focused on common sense reasoning. The DARPA Knowledge Sharing Initiative focused on reusable components and structured/unstructured knowledge. Table 1 summarizes the different approaches of these two programs which will be described next in sections 3.1 and 3.2.

Research Focus and Approach	Research Program	
	Cyc	Knowledge Sharing Initiative (KSI}
Very large ontologies	✓	✓
Common Sense Reasoning	✓	
Assembling Systems from Reusable Modules		✓
Structured and unstructured knowledge		✓
Model Integration	Common Upper Model	Distributed Agents & Ontology Mapping
Architecture	Tightly Coupled	Loosely Coupled

Table 1. Summary of Cyc and KSI Approaches

3.1 Cyc: Representing Common Sense Knowledge

Cyc was a system initially developed at the Microelectronics Computer Consortium (MCC) in Austin Texas. MCC was put together as a reaction to a similar large program called the Fifth Generation project out of Japan where major technology companies decided to pool their investments in AI research in order to spur new business applications. MCC had investment from IBM. Dell Computer, Sun Microsystems, Motorola, and many others [51].

Cyc was led by Doug Lenat (a former student of Feigenbaum) and was an attempt to develop a very large collection of ontologies to represent common sense and encyclopedic knowledge. The Cyc project created their own Frame based language and inference engine, as well as tools to enter information into their ontologies [52].

The Cyc project was the largest and most ambitious effort to address representing common sense knowledge. They had a large number of developers as well as content creators whose main job was to create entries for both common sense and encyclopedic knowledge.

The approach used for the Cyc environment was to tightly couple all the various component tools. I.e., each tool was designed specifically for Cyc and with little regard for how to separate the tool out to be used independently or to replace one component with an alternative. The advantage of tight coupling is that it often allows more features and better performance than loose coupling. The disadvantage is that it is an "all or nothing" approach. It is difficult to take one tool from the suite of tools and use it on its own and is also difficult to swap out one tool with another (e.g., a commercial product).

3.2 The DARPA Knowledge Sharing Initiative

Starting more or less at the same time as Cyc was another project to deal with large ontologies. This project was sponsored by the US Defense Advanced Research Projects Agency (DARPA) and was called the Knowledge Sharing Initiative [53].

Whereas Cyc was driven by the goal of representing common sense reasoning the knowledge sharing initiative was driven by the goal of facilitating reuse of software and knowledge. Consequently, the Knowledge Sharing Initiative took a diametrically opposite approach to their architecture as Cyc. Whereas Cyc was tightly coupled, the Knowledge Sharing Initiative was loosely coupled. Tools were developed for different aspects of the problem in a manner such that they could both be used independently from the project and so that they defined interface specifications that would facilitate swapping out one tool and replacing it with another as long as it implemented the appropriate interfaces. The Knowledge Sharing initiative also had a requirement to facilitate agent-based systems and supported a range of efforts to build publicly shared ontologies. For a detailed comparison of the two initiatives, see [54]. There were several tools that resulted from the Knowledge Sharing Initiative that had significant impacts on the Semantic Web. These included:

- **KRSL**. The Knowledge Representation Standard Language was an attempt to get community support for a knowledge representation language specification. It did not lead to an implementation and was subsumed by LOOM [55] for practical applications but was influential in thinking that fed into DAML. Loom was a frame-based knowledge representation language that was modeled after KL-One. It included a classifier that could restructure the model as a result of inferences about the axioms in the ontology [56]. These systems are discussed below.
- **KIF**, the Knowledge Interchange Format spearheaded by Mike Genesereth and Richard Fikes [57] provided a basic knowledge representation language rooted in FOL facilitating knowledge exchange among various knowledge representation

systems such as Loom and KEE. KIF was not meant to be a language for developers but rather for agents and other systems transmitting knowledge encoded in differing representation languages. The designers of KIF described it as analogous to PostScript. PostScript was not a language used by word processors but a common interchange language so that documents in Microsoft's RTF format, HTML, etc. could have a common standard for sending documents to printers.

- The **Shared Ontology** Working Group, led by Tom Gruber [58], supported collaborations on topic-specific ontologies, in stark contrast to the broad "upper model" goals of Cyc.
- **KQML**, the Knowledge Query Manipulation Language spearheaded by Tim Finin [[59]] was a protocol for enabling agents to query and/or modify ontologies and knowledge bases controlled by external systems. Several of the same developers worked on both KQML and SPARQL and KQML significantly influenced the design of SPARQL. SPARQL will be discussed in section 5.

3.3 Lessons Learned

Despite very different approaches, what both Cyc and the Knowledge Sharing Initiative ultimately demonstrated is that widespread adoption of any shared ontology is at best slow and difficult. The fundamental problem is two-fold: inherent complexity and engineering compromises.

The inherent complexity problem starts with the fact that the world is complex and multi-faceted. Consequently, attempts to model it in symbolic structures grow large.

An example of the difficulty of defining one comprehensive upper model can be found in research funded by the DARPA Knowledge Sharing Initiative and conducted by Jerry Hobbs in the early '90s, first at SRI and later at USC/ISI. Hobbs' goal was to develop a sharable ontology of time. Time is fundamental to many applications. Issues as diverse as time-stamping an email or scheduling the launch of satellite depend upon it. The effort was regarded as simple. It was expected to take weeks to months. It was dropped after several years. It was simply too complicated to build an all-encompassing model, in part because practical resolution depends on context[5]. For example, consider modeling time ranges. What should happen when, "I'm free from noon to 1:00 for an hour meeting" meets, "Well, I'm free from 12:01 to 1:01"? One might want those comparisons to be somewhat flexible. Not so, however, if the time range in question is an orbital launch window, where accepting even a multi-decimal fraction of a second outside the range could lead to an expensive disaster. But when one says 1:00 pm, do you mean local or Greenwich? Does 1:00:00.001 pm match? How about 1:00:00.000000001?

Considerations like these inevitably led to engineering compromises driven by a range of considerations. Practical applications necessitate decisions to do the best possible within available time and budget, and/or doing no more than needed for intended uses. This has important implications for the reusability of ontologies and directions for future work.

An immediate implication is that evaluating the reusability of a particular ontology for a particular purpose is much easier if one understands multiple factors. These include: the context(s) of prior use, the context(s) of intended future use, the underlying formal but highly philosophical thinking behind modeling decisions, and the impacts of engineering trade-offs taken upon the form of the models. Thus, both inherent complexity and engineering considerations impact sharing and reuse of ontological models.

A direct consequence of this is a problem analogous to code re-use in traditional software engineering. There is a strong force toward proliferation of duplicative and overlapping alternatives. Little has changed here since these risks were pointed out in 1993 (see [54]). The barrier cost of the intellectual effort required for understanding, evaluating, and adopting a pre-existing ontology creates a strong temptation to simply build one's own from scratch. Another factor is the strong (not entirely baseless) fear that re-use, being a close neighbor to standardization, can both become a barrier to innovation and a weapon of control over markets.

These concerns appear to be shared by major companies. It is worth noting that Google, Microsoft, Yahoo and Yandex have chosen to co-fund what is effectively a crowdsourced community-driven ontology development [60] in a bottom-up, grassroots fashion – much as advocated by graduates of the DARPA Knowledge Sharing Initiative [61, 62, 63].

Code re-use has grown significantly, in part because software engineering principles of modularity, encapsulation, and interface specification have been absorbed. In coding, also, the productivity benefits have been clearly shown to be too significant to ignore. Reuse of ontological models will not parallel that growth until some software engineering principles are developed, and appropriate development environments are provided.

These considerations define some important directions for future work. We need to think very hard about ontology development environments. The importance of capturing and reasoning about design rationale has been long recognized in other areas of design and engineering [[64]] [[65]]. Ontology developers need to find ways to do the same.

A complementary alternative is to look more closely into ontology mapping as a research and development direction. Rather than trying to build all-encompassing models, this direction would focus on tools that automatically or semi-automatically try to identify equivalences and differences between related ontologies in order to bridge between them. This approach was first introduced by Gio Wiederhold [[66]] [[67]] [[68]]. As ontologies transition from the lab to industry the problem of ontology mapping has been the impetus for a startup in Europe called Dynaccurate[6] that has shown significant time and money savings by automating the mapping of various healthcare and defense ontologies [[69]].

A particularly innovative extension of this approach was demonstrated by Baoshi Yan.[[70]] Yan built an end-user oriented environment supporting "bottom-up ontology

[5] Jerry Hobbs, personal communication to Robert Neches, 1997.

[6] https://www.dynaccurate.com/

alignment" in which users collecting documents picked up semantic annotations associated with them. As users organized those documents, Yan's tool helped them create their own ontology in which to place them. At the same time, it compared their personal ontology to the ones associated with the documents' original semantic annotations. These comparisons were used both to suggest mappings between users' ontologies and others, and to suggest refinements to users' personal ontologies.

One can imagine that, if tools like Yan's were in widespread use, "grassroots ontologies" might emerge which gained broad acceptance (or, at the very least, acceptance within communities of like-minded collaborators). The process by which they would emerge and evolve might go a long way toward ensuring utility, as well, since they would be tested through usage at every step.

There seems to be relatively little recent work picking up on these threads. In the recent 2021 International Semantic Web Conference, for example, only one out of 54 papers focused on ontology translation [71], and one on ontology alignment [72]. A smattering of others dealt with automated generation of ontologies from less-structured sources. However, those, although they do accelerate availability of ontologies, do not address the challenges of understanding, re-using, extending, or adapting them. It is very much time to get back to the challenges of software engineering and development environments supporting sharing and re-use. Without more work on these topics, significant cost, productivity, and maintenance barriers to widespread ontology adoption will remain.

4. Internet Research: From Hypertext to Linked Data.

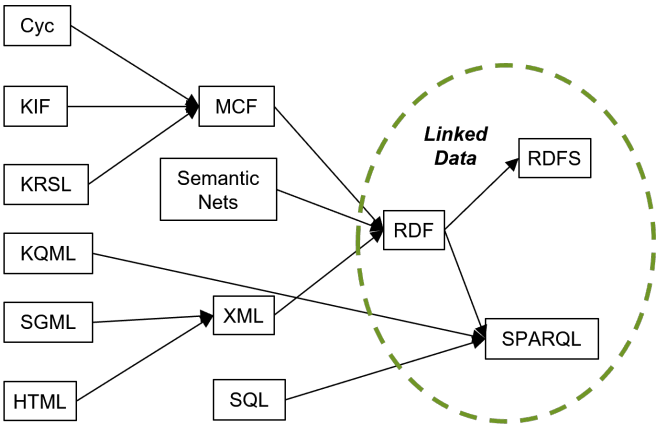

Figure 2. Research Leading to Linked Data

As these AI knowledge research initiatives were proceeding, Internet researchers began developing new technologies to add semantics to the Internet that significantly leveraged them. Figure 2 shows the connections between various research programs leading towards the standards that provide the foundation for the Semantic Web known as Linked Data.

The first technology that addressed semantics was XML. XML is a powerful meta-language that allows developers to define domain specific tags and structure for their documents. XML was the first and is still one of the most widely used languages to go beyond simple hypertext and to add basic semantics to HTML pages that could be accessed by bots for better searches and to enable agent like systems that can perform functions such as search for the lowest price for a purchased item.

XML is a subset of the ISO 8879 Standardized General Markup Language (SGML). SGML was designed for document sharing within large organizations such as governments and the military. XML is designed to be simpler than SGML and with Internet documents as one of the most important driving use cases [73].

XML provided basic semantics but is still oriented around the concept of a document. The next step after XML was the definition of the Meta-Content Framework (MCF). MCF was originally a research project funded by Apple Computer and was highly influenced by the Knowledge Representation community, specifically Cyc, KRSL, and KIF. Unlike XML, MCF had an object rather than a document focus. It described objects with attributes and relations to other objects [74].

MCF was a main influence to the Resource Description Framework (RDF). RDF is the foundation for the Semantic Web. It has two primary datatypes: resources and literals. A resource has a unique identifier that is an Internationalized Resource Identifier (IRI). An IRI looks similar to a URL. In fact, all URLs are IRIs. The difference is that a URL is typically meant to be a document or some resource that is designed to be viewed in a browser. An IRI can be any resource such as a class, property, or instance. Every object in an OWL ontology is a resource and hence has a unique IRI. A literal is a simple datatype such as a string, integer, or date. The fundamental representation scheme in RDF is the triple. A triple has a Subject, a Predicate, and an Object. The subject and predicate must be resources (IRIs). The object can be either a resource or a literal. RDF results in network graphs because the subject of one triple can be the object of another and vice versa [75].

RDF is a basic semantic network language. The next step was to add what Brachman defined as the *epistemological layer* to RDF. The creation of nodes such as classes and predicates such as properties and type. This layer is called the RDF Schema language (RDFS). Note that although RDFS adds basic meta-model concepts and links it does not support a formal semantics such as Description Logic.

The graph structures of RDF/RDFS requires a query language to traverse, retrieve, and modify information in the graph. This language is SPARQL. SPARQL is a recursive acronym: SPARQL RDF Query Language. SPARQL was influenced by SQL in order to make it more accessible to the large community of database developers. The two most important differences between SPARQL and SQL are:

1. SPARQL can match (have a wildcard) for any or all parts of a triple. A SPARQL query with wildcards for all three parts of the triple will retrieve the entire graph.
2. SPARQL has built-in capabilities to link to and request data from any resources on the Internet. Queries can match and retrieve data from many different sources (e.g., DBpedia and Geonames) even though those sources were designed independently with neither system knowing about the design of the other. This in effect gives every SPARQL user a huge, distributed database of all

RDF data on the global Internet accessible from their local client machine.

This concept of defining queries that match and retrieve data from multiple heterogeneous sources on the Internet is known as *Linked Data* [75].

5. Knowledge Representation Meets the Internet

This brings us to the most powerful semantic language in the Semantic Web: The Web Ontology Language (OWL). Figure 3 shows a graph of the most important knowledge representation languages and their relations to each other and eventually to OWL.

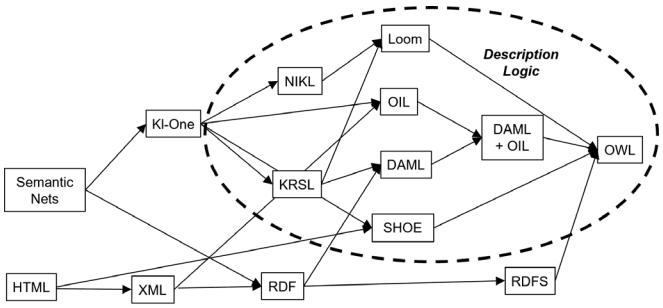

Figure 3. Research Leading toward OWL

KL-One was a groundbreaking language, but it was too slow for application to real problems. New Implementation of KL-One (NIKL) was developed at USC/ISI and was an attempt to develop a Description Logic (DL) language with similar power as KL-One but with performance that could be applied to real problems. It was followed by the Loom language also developed at USC/ISI by a team led by Bob McGregor. Loom was a very powerful language implemented in LISP. Later versions were implemented on top of the Common LISP Object System (CLOS) which gave it good performance and portability. Loom was one of the first DL languages used to develop applications that were utilized by actual users from the Defense Logistics Agency and Aerospace and Defense contractors taking part in the USC/ISI FAST project. The FAST project was far ahead of its time, demonstrating Business to Business (B2B) commerce at least a decade before anyone in the commercial space had conceived of the idea [50].

The Knowledge-Representation Standard Language (KRSL) discussed above was part of the Knowledge Sharing initiative. While it never resulted in an actual development language it shaped the goals of the Knowledge Sharing initiative to encourage knowledge re-use via interoperability and mapping rather than by simply mandating use of one knowledge representation language or shared model. Also, as discussed in section 5, KRSL was one of the influences on the MCF language which was a direct influence on RDF.

SHOE[7] was a very innovative language developed at the University of Maryland by Jim Hendler. It was one of the first languages to attempt to directly add DL semantics to HTML pages.

The Ontology Interchange Language (OIL) was a European initiative that defined DL semantics on top of XML. The DARPA Agent Markup Language (DAML) was part of the Knowledge Sharing initiative and was the first attempt to add DL semantics to RDF. The European researchers working on OIL and the American researchers working on DAML soon realized that their ultimate goals were very similar and pooled their efforts to develop the DAML + OIL language which was one of the most significant influences on the Web Ontology Language (OWL).

The endpoint (at least so far) of this research was of course OWL.[8] OWL is built on RDFS and was standardized by the W3C. OWL is supported by the Protégé ontology editor from Stanford [76]. Protégé far exceeds the expectations for most research tools. Its robustness, documentation, and support from an active user community, equals many commercial tools. It also continues to be a platform for innovation via many plugins developed by researchers, which seamlessly integrate with it and provide tools for SPARQL, SHACL, and many other features. In addition to Protégé, OWL is also supported in commercial offerings from vendors such as Franz Inc., Ontotext, Pool Party, Stardog, and Top Quadrant.

The primary difference between OWL and languages such as Loom is that in order to achieve scale up to what is known in industry as "big data" OWL focuses only on representing the data of a model. Languages such as Loom and KEE could model hierarchies of classes and also include features such as message passing to implement process. They were self-contained tools that could be used to build entire applications. That is not the case with OWL. OWL only models data.[9] To implement process developers must utilize APIs to programming languages such as Java, Python, and Lisp. For examples and more detail on OWL and other Semantic Web standards we recommend the book [77]. For a hands-on tutorial on Protégé and several of the most important plugins see [78].

6. Shapes Constraint Language (SHACL)

The newest Semantic Web standard is the Shapes Constraint Language (SHACL). The original vision of the Semantic Web was to provide a semantic layer for the public Internet. There has been success with this in open crowd sourced knowledge graphs[10] such as DBpedia, Wikidata, and Geonames. However, the technology is also having significant impact behind corporate firewalls, and this has created a requirement for a new language focused on data integrity constraints. SHACL seeks to address that need and is gaining great traction as one of the most significant Semantic Web standards for use in Industry [79].

SHACL and OWL seem similar at a first glance. Any axiom that can be defined in OWL can be defined in SHACL and almost any constraint that can be defined in SHACL can be

[7] https://www.cs.umd.edu/projects/plus/SHOE/
[8] https://www.w3.org/TR/owl2-overview/
[9] As with modern databases (e.g., triggers) the distinction between data and process is not clear cut. The Semantic Web Rule Language (SWRL) provides the capability to implement many types of business logic. However, it has limitations and for any real application a programming language must be integrated with OWL.
[10] In industry *knowledge graph* is the more popular term and in academia *ontology* is. Although there are some subtle differences, for the purposes of this paper, we will treat them as synonyms.

defined in OWL. The difference isn't what can be defined, but rather how the logic is utilized.

OWL ontologies are meant for reasoning on valid data. For example, if the range for a property is defined as the class *Person,* and an instance of the class *ElectronicPart* is entered as the value of that property, the OWL reasoner won't trigger an error but will instead wrongly infer that the *ElectronicPart* is also an instance of *Person*. If the two classes are defined as disjoint in OWL (i.e., their intersection is the empty set), the OWL reasoner will trigger an error but in so doing will make the ontology inconsistent and unable to utilize any inferences until the inconsistency is resolved.

Also, the OWL reasoner adopts the Open World Assumption (OWA). This is arguably essential for use with the public Internet. Its design rationale is that no ontology could encompass all the data on the Internet. Thus, if a value for a particular property is missing, the reasoner shouldn't infer that there is no such value (as systems built upon the far more common Closed World Assumption would). The OWA means that the reasoner won't trigger errors for certain kinds of axioms that require a minimum number of values for a property. It assumes that such values could exist somewhere in the Internet but just haven't been found yet.

The considerations that make OWL work for the public Internet, don't work for systems behind corporate firewalls. Data integrity constraints are essential for enterprise data. If each customer must have an email address in their account, there must be a way to alert the enterprise that something is wrong if an email is not defined for a customer.

These concerns are what drove the creation of SHACL. SHACL uses the Closed World Assumption rather than the OWA. It will signal errors if there is a constraint that requires a minimum number of values for a property and the proper number of values are not currently in the knowledge graph. Also, SHACL does not make the entire knowledge graph inconsistent when a data integrity violation is found. SHACL provides the developer with the option to define various kinds of errors as well as with the option to define code that may attempt to repair certain types of data integrity violations. E.g., if a data property has a range of integer and the string "1" is a value for some individual, SHACL can attempt to coerce the string into an integer datatype.

In relation to knowledge representation, SHACL brings back some of the capabilities of the original Frame languages such as constraint checking and default values.

7. Current & Future Developments

As often happens in research, the original vision of the researchers tends to evolve as concepts transition from the lab to industry. The initial vision of the Semantic Web was focused primarily on providing a semantic layer for the public Internet. There have been great achievements toward this goal with Linked Data. However, the technology only began to achieve significant recognition in industry when Google coined the term "knowledge graph" [80] and began to discuss their knowledge graph project, which they use to provide direct answers rather than just links in response to queries. Google's trail blazing created a flood of interest as technology leaders such as Facebook, LinkedIn, Amazon, and others began creating knowledge graph projects behind their firewalls to better understand, control, and utilize their data [81].

The technology is still in its infancy in terms of industry use. The opportunities, as well as the challenges, are enormous. Along with research issues touched upon in Section 4.3, some of the most important of these are:

1. **Privacy**. Knowledge graphs and visualization tools make explicit information that is left implicit in documents. An example of this is the CODO project [82], which created a knowledge graph for information about the Covid-19 pandemic in India. The information that the researchers used was all from publicly available spreadsheets published on the Internet. However, when certain information (e.g., tracing paths of infection from one patient to another) that was implicit in the spreadsheets became explicit in the knowledge graph, owners of the data became deeply concerned and stopped releasing their data.

2. **Alternatives to W3C Standards.** It is inevitable that there will always be deviations from most standards. Vendors want to include new features that differentiate their products, and specific use cases can encourage alternative techniques from the accepted standards. This is currently happening with Semantic Web standards. Property graphs are a competitor to RDF/RDFS. There is currently no accepted standard for property graphs, it is a term used for various implementations of the concept. The main difference between property graphs and RDF is that the links (properties) in a property graph can also have attributes. I.e., property graphs provide a capability similar to facets on slots in Frame languages. There are also alternatives to SPARQL that are designed to query property graphs rather than RDF. One of the most popular is the Cypher query language from Neo4J [83]. Another alternative to the current W3C Semantic Web stack is RDF* which is being proposed as a W3C standard. It essentially provides similar features as property graphs but built on top of RDF. However, it would not be compatible with OWL or the current implementation of SPARQL. Several knowledge graph vendors that adhere to W3C standards such as Franz Inc. also provide proprietary extensions that enable the same capabilities as property graphs.

3. **OWL and Big Data.** Most of the large, frequently used knowledge graph implementations to date have utilized RDF/RDFS or Property Graphs. One reason is that OWL is newer than RDF. Many large projects such as DBpedia began in RDF and continued to use that technology. In addition, there remains skepticism among many in industry as to whether the OWL reasoner can scale to very large data. However, as Jim Hendler points out in his presentation "Whither OWL" [84] it may be that only subsets of OWL can provide the performance needed for Big Data, but those subsets may be enough for most use cases. This is supported by the fact that many of the current vendors of triplestore

products[11] such as AllegroGraph from Franz Inc. support a large but not complete subset of OWL and achieve very fast performance for very large knowledge graphs [85].

4. **Development environments for building, finding, and re-using ontologies.** Excellent tools such as the Protégé ontology editor from Stanford are robust and widely used. Nevertheless, as discussed in section 4.3, the problem of defining modular ontologies, mapping between ontologies, and capturing information about the context and design decisions behind an ontology still exist only in the lab. Achieving the same level of knowledge re-use as the software engineering community has in code re-use will require significant effort. E.g., consider sophisticated automated OOP build and testing tools such as Apache Ant and Maven. Nothing comparable (in terms of scale and robustness) currently exist for Semantic Web technology.

5. **Automation of Knowledge Graph Creation and Maintenance.** Crowd sourced Linked Data assets such as DBpedia are enormous. DBpedia has approximately 20 billion RDF triples.[86] Manually creating and maintaining resources that large is a daunting task. One of the most interesting areas of research is the use of techniques such as machine learning to develop, review, maintain, and extend large knowledge graphs [87].

Knowledge representation techniques that date to the very beginning of the field are only now coming to fruition in a way that makes them practical for large scale usage on the Internet and industry. The future potential is enormous. It will yield whole new ways of leveraging data to enhance human knowledge and understanding. Given the challenges we face as a world community, it will be a resource that is sorely needed.

CONFLICT OF INTEREST

Michael DeBellis is an independent consultant and researcher. He is on the board of directors for Dynaccurate, a company mentioned in the paper but otherwise has no other potential conflicts of interest. Robert Neches is on the board of the Innovation Infrastructure Utility, LLC. He is otherwise retired and has no conflicts of interest.

ACKNOWLEDGEMENTS

Jim Hendler provided extremely useful resources and guidance on the history of the Semantic Web. Chunka Mui provided valuable help related to work done by Accenture's AI group. Doug Robbins provided invaluable assistance helping us deal with formatting and graphics issues. Thanks to the members of the Ontolog Forum: https://groups.google.com/g/ontolog-forum especially John Sowa for insight and feedback on several issues.

REFERENCES

[1]. T. Berners-Lee, J. Hendler and O. Lassila, "The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities". Scientific American. Vol. 284, Issue 5. May 2001.
[2]. J. Hendler and D. L. McGuinness, "The DARPA Agent Markup Ontology Language", IEEE Intelligent Systems, Vol. 15, No. 6, November/December 2000.
[3]. A.M. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem", Proceedings of the London Mathematical Society, Series 2, Vol 42, pp. 230–265, 1936–37.
[4]. A. Church, "An Unsolvable Problem of Elementary Number Theory", American Journal of Mathematics, Vol. 58,pp. 345–363. 1936.
[5]. J. McCarthy "LISP prehistory - Summer 1956 through Summer 1958." In History of Lisp, 1979.
[6]. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach, 3rd Edition*. p. 274. Prentice Hall Series in Artificial Intelligence, 2010.
[7]. H. Levesque and R. Brachman, "A Fundamental Tradeoff In Knowledge Representation and Reasoning", In Readings in Knowledge Representation, R. Brachman and H. J. Levesque, Ed. Morgan Kaufmann. 1985.
[8]. A. Newell, J.C. Shaw and H.A. Simon, "Report on a general problem-solving program", Proceedings of the International Conference on Information Processing, pp. 256–264, 1959.
[9]. F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, Building Expert Systems. Addison-Wesley, 1983. Hayes-Roth, Frederick. Building Expert Systems. 1983.
[10]. H. A. Simon and E. A. Feigenbaum, "An information-processing theory of some effects of similarity, familiarization, and meaningfulness in verbal learning." *Journal of Verbal Learning and Verbal Behavior*, vol. 3, no. 5, pp. 385-396, 1964, doi: 10.1016/s0022-5371(64)80007-4.
[11]. D. T. Connors, "Software development methodologies and traditional and modern information systems." *ACM SIGSOFT Software Engineering Notes,* vol. 17, no. 2, pp. 43-49, 1992, doi: 10.1145/130840.130843.
[12]. B. W. Boehm, "A spiral model of software development and enhancement." Computer, vol. 21, no. 5, pp. 61-72, 1988, doi: 10.1109/2.59.

[11] A triplestore is a database that natively stores information as triples rather than in tables as a relational database does.

[13]. K. Beck and C. Andres, Extreme Programming Explained. Addison-Wesley Professional, 2004.
[14] B. Dickson, "What is the AI Winter?", [Online] Available From: https://bdtechtalks.com/2018/11/12/artificial-intelligence-winter-history/ [Acessed Feb. 9, 2022].
[15]. SAP Help Portal, "SAP Rule Engine", [Online] Available From: https://help.sap.com/viewer/9d346683b0084da2938be8a285c0c27a/2011/en-US/e30ec2e429e84d22b9045b84c366ab76.html [Acessed Feb. 9, 2022].
[16]. Oracle Docs, "About Seibel Business Rules Benefits", [Online] Available From: https://docs.oracle.com/cd/B40099_02/books/BusRulesAdm/BusRulesAdm_About2.html [Acessed Feb. 9, 2022].
[17]. Tibco White Paper, "Event Processing with Inference Rules", [Online] Available From: https://www.tibco.com/sites/tibco/files/resources/wp-event-processing-inference-rules_0.pdf [Acessed Nov. 17, 2021].
[18]. IBM Operational Decision Manager documentation, "Business rule applications based on COBOL for z/OS platformsS", [Online] Available From: https://www.ibm.com/docs/en/odm/8.0.1?topic=development-business-rule-applications-based-cobol-zos-platforms [Acessed Feb. 9, 2022].
[19]. E. Friedman-Hill, *Jess in Action*. Manning Publications Company, 2003.
[20]. SFGATE, "Europe's SAP Buys 14% Stake In IntelliCorp", Aug. 10, 1996. [Online] Available From: https://www.sfgate.com/business/article/Europe-s-SAP-Buys-14-Stake-In-IntelliCorp-2970378.php [Acessed Nov. 17, 2021].
[21]. M.R. Quillian, "Semantic Memory." Report AFCRL-66-189, Bolt, Beranek, and Newman, 1966.
[22]. D. E. Rumelhart, P. H. Lindsay and D.A. Norman, "A process model for long-term memory", In Organization of Memory, E. Tulving and W. Donaldson Eds., New York: Academic Press, 1972.
[23]. Rumelhart, D. E., & Norman, D. A. (1973). Active semantic networks as a model of human memory. Proceedings of the Third International Joint Conference on Artificial Intelligence. Stanford, CA, 450-457.
[24]. R. Brachman, "On the Epistemological Status of Semantic Networks", In Readings in Knowledge Representation, R. Brachman and H. J. Levesque, Eds. Morgan Kaufmann, 1985.
[25]. D.A. Norman and D. E. Rumelhart, Explorations in Cognition, San Francisco: Freeman, 1975.
[26]. D. E. Rumelhart and D.A. Norman, "Analogical Processes in Learning". In Cognitive Skills and Their Acquisition, J.R. Anderson, Ed. Hillsdale, NJ: Erlbaum, 1981..
[27]. R. C. Schank and R. P. Abelson, Scripts, Plans, Goals, and Understanding. Psychology Press, 2013.
[28]. L. Bass and R. Kazman, "Object-Oriented Development at Brooklyn Union Gas", IEEE SoftwareVol. 10, Issue 1 January 1993, pp. 67–74.
[29]. J.-P. Laurent, J. Ayel, F. Thome, and D. Ziebelin, "Comparative Evaluation of Three Expert System Development Tools: Kee, Knowledge Craft, Art," *The Knowledge Engineering Review,* vol. 1, no. 4, pp. 18–29, 1984.
30. M. Stefik, D.G. Bobrow, *Object-oriented programming: Themes and Variations.* AI Magazine 6:4, pp. 40-62, Winter 1986.
[31]. D. Kafura, "Object-Oriented Programming and Software Engineering", [Online] Available From: https://people.cs.vt.edu/kafura/cs2704/oop.swe.html [Acessed Feb. 9, 2022].
[32]. D. Terletskyi, "Inheritance in Object-Oriented Knowledge Representation." *Communications in Computer and Information Science*, pp. 293-305, 2015.
[33]. B. Meyer, *Object-oriented Software Construction*. Prentice Hall, 1997.
[34]. H.A. Simon, Herbert A. 1996. The Sciences of the Artificial (3rd ed.). Cambridge, MA: MIT Press, 1996.
[35]. E. H. Rosch, "Natural Categories", Cognitive Psychology, Vol. 4 (3), pp. 328–350, May 1973.
[36]. G. Lakoff, Women, Fire, and Dangerous Things. University of Chicago Press, 2008.
[37]. J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2007.
[38]. Chomsky, Noam. The Minimalist Program. The MIT Press; 20th Anniversary ed. edition (December 19, 2014). ISBN-13: 978-0262527347.
[39]. Eisenstadt, Marc (1979) Schank/Riesbeck vs. Norman/Rumelhart: What's the Difference? 17th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, pp 15-16
[40]. F. Baader, D. Calvanese, D. McGuinness, P. Patel-Schneider, and D. Nardi, The Description Logic Handbook. Cambridge University Press, 2003.
[41]. J. Sowa. "Fads and Fantasies about Logic", IEEE Intelligent Systems, 22:2, pp. 84-87, March 2007.
[42]. R. Fikes and T. Garvey, " Term Subsumption Languages in Knowledge Representation" AI Magazine, vol. 11, no. 2, 1990.
[43]. Franz Inc. SPARQL Execution Warnings, [Online] Available From: https://franz.com/agraph/support/documentation/current/sparql-reference.html#query-warnings [Acessed Feb. 10, 2022].
[44] W3C Recommendation, OWL 2 Web Ontology Language Profiles (Second Edition), [Online] Available From: https://www.w3.org/TR/2012/REC-owl2-profiles-20121211/ [Acessed Feb. 8, 2022].
[45]. J.F. Allen and D.J. Litman, "Discourse Processing and Common sense Plans." In Intentions and Communication, P.R. Cohen, J. Morgan, and M. Pollack, Eds. MIT Press, 1990.
[46]. G. Fischer, "Symbiotic, Knowledge-Based Computer Support Systems" In Proceedings of the Conference on Analysis, Design and Evaluation of Man-Machine Systems, Pergamon Press, Baden-Baden, pp. 351-358, 1982.
[47]. R. Neches, W.R. Swartout and J.D. Moore, "Explainable (and Maintainable) Expert Systems", Proceedings of the Ninth International Joint Conference on Artificial

Intelligence, Palo Alto: Morgan Kaufman Publishing Co., 1985.
[48]. T.W. Malone, K.R. Grant, K.Y. Lai, R. Rao and D.A. Rosenblitt, "Semi-structured messages are surprisingly useful for computer-supported coordination", Proceedings of the Conference on Computer-Supported Cooperative Work, Austin, Texas, December 3 – 5, 1986.
[49]. B. Harp and R. Neches, "Model Formality in Human/Computer Collaboration", AAAI '93 Fall Symposium Series Workshop on Human-Computer Collaboration: Reconciling Theory, Synthesizing Practice, October, 1993.
[50]. J. Yen, R. Neches, M. DeBellis, P. Szekely, P. Aberg, "BACKBORD: An Implementation of Specification by Reformulation", Chapter 18 In Intelligent User Interfaces J.W. Sullivan and S.W. Tyler, Eds. New York : Addison-Wesley ACM Press, pp. 421-444, 1991.
[51]. S. Higginbotham, "Final bell ringing for MCC", Nov. 3, 2004, [Online] Available From: https://www.bizjournals.com/austin/stories/2004/11/01/story3.html [Acessed Feb. 7, 2022].
[52]. D. B. Lenat, R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd, "Cyc: toward programs with common sense." *Communications of the ACM*, vol. 33, no. 8, pp. 30-49, 1990, doi: 10.1145/79173.79176.
[53]. R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator and W. Swartout, "Enabling Technology for Knowledge Sharing", AI Magazine, 12(3), pp. 36-56, 1991.
[54]. R. Neches, "Review of D.B. Lenat and R. Guha, Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project", Journal of Artificial Intelligence, Vol. 61, pp. 65-79, 1993.
55. R. M. MacGregor. "A Description Classifier for the Predicate Calculus" in *Proceedings of the Twelfth National Conference on Artificial Intelligence, (AAAI 94)*, 1994 pp. 213-220.
[56]. R. M. MacGregor, "Retrospective on Loom", [Online] Available From: https://www.isi.edu/isd/LOOM/papers/macgregor/Loom_Retrospective.html [Accessed: Feb. 7, 2022].
57. M. Genesereth and R. Fikes; "Knowledge Interchange Format", *Version 3.0 Reference Manual*; Technical Report Logic-92-1, Computer Science Department, Stanford University, Stanford, CA, 1992.
58. T. R. Gruber. The Role of Common Ontology in Achieving Sharable, Reusable Knowledge Bases. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference*, Cambridge, MA, pp. 601-602. Morgan Kaufmann, 1991.
59. T. Finin, R. Fritzson, D. McKay, R. McEntire, "KQML as an agent communication language". In *Proceedings of the third international conference on Information and knowledge management - CIKM '94*. 1994, p. 456.
60. Schema.org [Online] Available From: https://schema.org/, [Acessed: February 3, 2022].
61. K. Yao, I. Ko, R. Neches , R. MacGregor, "Semantic Interoperability Scripting and Measurements", In *Proceedings of the Working Conference on Complex and Dynamic Systems Architecture*, Brisbane , Australia, December 2001.
62. M. Frank, P. Szekely, R. Neches, B. Yan, and J. Lopez, "WebScripter: World-Wide Grass-roots Ontology Translation via Implicit End-User Alignment",In *Proceedings of the WWW-2002 Semantic Web Workshop*, Honolulu , Hawaii, May 2002.
63 B. Yan, M. Frank, P. Szekely,R. Neches and J. Lopez, "WebScripter: Grass-roots Ontology Alignment via End-User Report Creation", In *2nd International Semantic Web Conference (ISWC2003)*, Sanibel Island , Florida, Oct 20-23, 2003, pp 676-689.
[64]. R. Neches and G. Arango, "Design Capture, Information Technology Issues" In *Concurrent Design and Engineering* G. Zack and J. Hopcroft eds., Academic Press, 1993.
[65]. C. Green, D. Luckham; R. Balzer; T. Cheatham; C. Rich "Report on a Knowledge-Based Software Assistant", Kestrel Institute. A996431, Aug. 1983.
[66]. G. Wiederhold, "Future Needs In Integration Of Information" International Journal of Cooperative Information Systems, vol. 9, no. 4, pp. 449-472, 2000, doi: 10.1142/s0218843000000211.
[67]. Wiederhold, Gio and Michael Genesereth: "The Conceptual Basis for Mediation Services"; IEEE Expert, Vol.12 No.5, Sep.-Oct. 1997, pages 38-47.
[68]. G. Wiederhold, "Mediators in the architecture of future information systems." *Computer*, vol. 25, no. 3, pp. 38-49, 1992, doi: 10.1109/2.121508.
[69]. J. C. Dos Reis, C. Pruski, M. Da Silveira and C. Reynaud-Delaître, "DyKOSMap: A framework for mapping adaptation between biomedical knowledge organization systems", Journal of Biomedical Informatics,Volume 55, pp. 153-173, ISSN 1532-0464, 2015.
[70]. B. Yan, "Enabling Laymen to Contribute Content to the Semantic Web: A Bottom-up Approach to Creating and Aligning Diversely Structured Data". Ph.D. Dissertation, University of Southern California, Department of Computer Science, 2006.
[71]. T. Hahmann, R.W Powell II, "Automatically Extracting OWL Versions of FOL Ontologies", In *The Semantic Web – ISWC 2021* A. Hotho Ed. Lecture Notes in Computer Science, vol 12922. Springer, Cham. 2021.
[72]. J. Portisch, M. Hladik and H. Paulheim "Background Knowledge in Schema Matching: Strategy vs. Data", In *The Semantic Web – ISWC 2021. ISWC 2021. Lecture Notes in Computer Science*, A. Hotho Ed. vol 12922. Springer, Cham, 2021.
[73]. W3C. "Extensible Markup Language (XML) 1.0 (Fifth Edition)" W3C Recommendation. November 26, 2008. https://www.w3.org/TR/REC-xml/ [Acessed Nov. 18, 2021].
[74]. B. Hammersley, *Content Syndication with RSS*. Sebastopol: O'Reilly. ISBN 978-0-596-00383-8. 2003.
[75]. B. DuCharme, *Learning SPARQL*. Oreilly & Associates Incorporated, 2013.
[76]. M. Musen, "The Protégé Project: A Look Back and a Look Forward", A.I. Matters 1(4), 2015.
[77]. M. Uschold, *Demystifying OWL for the Enterprise*. Morgan & Claypool Publishers, 2018.

[78]. M. DeBellis, "A Practical Guide to Building OWL Ontologies Using Protégé 5.5 and Plugins Edition 3.2", Sept. 2021. [Online] Available From: https://www.michaeldebellis.com/post/new-protege-pizza-tutorial [Accessed: Feb. 7, 2022].
[79]. M. Figuera, P. D. Rohde, and M. Vidal. "Trav-SHACL: Efficiently Validating Networks of SHACL Constraints", Proceedings of the Web Conference 2021. Association for Computing Machinery, New York, NY.
[80]. A. Singhal, Amit, "Introducing the Knowledge Graph: things, not strings", May 16, 2012. . [Online] Available From: https://www.blog.google/products/search/introducing-knowledge-graph-things-not/ [Accessed: Feb. 9, 2022].
[81]. N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor. "Industry-Scale Knowledge Graphs: Lessons and Challenges", Communications of the ACM. Volume 62, Issue 8. pp 36–43, August 2019.
[82]. M. DeBellis and B. Dutta. "The Covid-19 CODO Development Process: An Agile Approach to Knowledge Graph Development" In *Second Indo-American Knowledge Graph and Semantic Web Conference*, Vardhaman College of Engineering, Hyderabad, Telangana, India. 22-24 November 2021.
[83]. I. Polikoff, "Knowledge Graphs vs. Property Graphs – Part I", August 19, 2020. [Online] Available From: https://tdan.com/knowledge-graphs-vs-property-graphs-part-1/27140 [Acessed Nov. 18, 2021].
[84]. J. Hendler. "Whither OWL", May 31, 2016. [Online] Available From: https://www.slideshare.net/jahendler/wither-owl [Acessed: Nov. 18, 2021].
[85]. Franz Inc. "AllegroGraph 7.2.0 Materialized Reasoner", [Online] Available From: https://franz.com/agraph/support/documentation/current/materializer.html#Rule-Sets [Acessed Nov. 18, 2021].
[86]. J. Holze, "DBpedia Snapshot 2021-09 Release", October 22, 2021. [Online] Available From: https://www.dbpedia.org/blog/snapshot-2021-09-release/ [Acessed Nov. 18, 2021].
[87]. H. Paulheim, "Knowledge graph refinement: A survey of approaches and evaluation methods." *Semantic Web*, vol. 8, no. 3, pp. 489-508, 2016, doi: 10.3233/sw-160218.