Towards Building Active Knowledge Systems With Conceptual Graphs

Harry S. Delugach[1]

Abstract. This paper outlines a vision for using conceptual graphs to build active knowledge systems that have the capability to solve practical and complex problems. A key ingredient in an active knowledge system is its ability to interact (not just interface) with the real world. Basic features of such systems go beyond logic to include support for data mining, intelligent agents, temporal actors, active sensors, a system for knowledge interchange and finally, support for knowledge-in-the-large.

1 Introduction

I am a system builder. I am interested in solving real-world problems through the use of automated knowledge-based systems. This paper outlines a framework for building systems *active knowledge systems*: systems that have the capability to solve practical and complex problems. A key ingredient in an active knowledge system is its ability to inter*act* (not just inter*face*) with the real (and changeable) world. If a system can do that, it becomes more of an intelligent system, whose full potential we are just beginning to understand. This paper is intended to encourage and promote effective techniques for building these systems.

It is my claim that conceptual graphs have the potential – much of it unexplored – to support active knowledge systems that will explicitly model processes, and interact with the real world. The power of these capabilities becomes apparent when serious developers begin building large-scale and robust systems for general use. Several systems based on conceptual graphs (CGs) have recently become available e.g., PROLOG+CG [8], Notio [14], WebKB [10], CoGITaNT [6] and CharGer [4] [5]. Researchers are beginning to formulate practical guidelines as to how such systems should be designed and implemented [9]. In addition, interest in the Semantic Web [2] and support for distributed communities [12] [13] make it essential to find ways for systems to involve themselves in the real world.

While these existing conceptual graph systems each have their own clear strengths, I have found that these systems lack some features that give a system rich semantics. Because such systems generally support the declarative core of conceptual graphs, they aim to represent knowledge mostly about *things*; only a few (e.g., [12], [5]) are interested in semantic modeling of *procedures* and *activities*. The CG core, based on first-order predicate calculus, can certainly support useful results, especially within a closed, monotonic (i.e., unchanging) environment; however, to be more generally useful, a system must have additional features that lie outside of the core.

[1] Computer Science Dept., Univ. of Alabama in Huntsville, Huntsville, AL 35899 U.S.A. Email: delugach@cs.uah.edu.

My interest in building useful systems places me at the juxtaposition of several areas of research. I am not a philosopher who studies the limits of interpretability, yet I require formal ways to interpret the results of the systems I want to build. I am not a theorist who studies the soundness or completeness of conceptual graph systems, yet I require a system to be sound and complete with respect to some portion of the real world I am modeling. It is these struggles (and others) which have convinced me to propose something more.

2 Practical Active Knowledge Systems

In this paper, I propose using conceptual graphs to support a full-fledged active knowledge system. Many of the features described here can, of course, be employed with other approaches besides conceptual graphs; they are gathered here in order to suggest the capabilities that a system ought to have. The notion of an active knowledge system is not new – a number of researchers have been exploring agents in distributed systems, most notably the Semantic Web [2] and the DAML effort (www.daml.org).

In this paper, I argue that useful knowledge (e.g., in the sense of the Semantic Web [2] or Interspace [13]) is coupled implicitly with activity or procedures that are rooted in the real-world. Every piece of knowledge encoded in a formal system must reflect some actual knowledge that real people have, if the system is to be practical.

I use the term *practical system* to mean a knowledge-based system that has the following properties. Though these descriptions are somewhat subjective, I propose them as a starting point for discussion.

A practical system is a system that:

- Addresses one or more needs that can be articulated in the real world,
- Produces results that are meaningful and useful to people in the real world in addressing their needs,
- Can be easily (re-)configured to provide meaningful and useful results in various domains,
- Can incorporate and accommodate new information,
- Can reason about processes and activities, as well as their steps and goals,
- Can recognize nonsense,
- Has the capability to explain how it operates, including where its knowledge comes from, and how it reached its conclusions,
- Is scalable to human-sized problems.

It would be a great leap forward for us to identify a single feature or technique that would unify all these characteristics, but no single feature or technique is going to suddenly make a system useful and intelligent. A collection of useful features and techniques can be very powerful, however, which is why this paper proposes a unifying framework to support a collection of useful features. Conceptual graphs, because of their demonstrated semantic modeling capabilities [15], [16] and their intrinsic ability to support reasoning about semantics [3] are a reasonable candidate for a representation.

The rest of this paper describes a few of the issues to consider in building practical systems.

2.1 Logic Support

The basis for most of our current systems is first-order logic. This is essential for a workable system, as some have argued for many years, starting with [15]; there is widespread agreement that logic is an important foundation for any knowledge-based system. For symbolic reasoning, we need logic, and if logical formulae are to be passed among different systems, then we must have a complete, consistent and practical standard for the interchange of logical expressions. One important point is that practicality is not a feature to be added to a standard after it is established; a standard should already include those best parts of a practical system and attempt to generalize them. As an ISO working group [1] considers a common logic standard, we may finally have a standard with which system builders can exchange knowledge between systems, as well as gauge their systems' logical consistency and completeness. I support current efforts to work out the details so that a standard can be agreed upon.

While nearly everyone agrees that logic is necessary, there is some disagreement as to whether it is sufficient. I believe that intelligence is composed of many elements, only one of which is logic. I further argue in this paper that, with logic as a basis, additional capabilities can support useful and practical intelligent systems based on conceptual graphs.

2.2 Support For Denotational Semantics

A logic system provides predicates and functions, which allow the representation of relationships between elements in a system. The rules of logic ensure that the truth of these predicates is preserved; e.g., if I assert that **dog(Spot)** is true and then perform any number of logical transformations, a (consistent) system should never derive the expression **not dog(Spot)**. This is a necessary property of any system that is meant to preserve truth. Most knowledge-based systems therefore operate as in Figure 1. Elements in the model represent predicates, functions, etc.

Figure 1. Knowledge Based System Model.

There is a larger question that often goes unanswered (at least in a formal way): what do the predicates mean? Logicians remind us that a predicate is assumed to be a primitive relation between symbols; a logical system's only obligation is to preserve those relationships throughout its logical operations. The symbols themselves, however, are arbitrary (at least as far as the logic operations are concerned). This presents a difficulty for a system builder, whose symbols are not arbitrary at all! Symbols used in a knowledge base stand for something, usually in the real world.

Figure 2 depicts a model as it relates to the real world. Things in the real world are represented in a knowledge-based system (KBS) through a mapping **E** (for "encoding") that allows their representations to be transformed within the system. Once transformed, new representations can arise within the system. These new representations' symbols can then be mapped back to the real world through a mapping **D** (for "decoding") that captures their interpretation. This model is intentionally simplistic,

ignoring some important philosophical issues, in order to provide a basis for the following discussion.

It is crucial for a knowledge system to preserve truth internally, and of course the system builder requires the same of any system they build. Furthermore, the system builder needs for the system to preserve the symbols' meanings as they exist in the real world. The mapping D in Figure 2 represents this need.

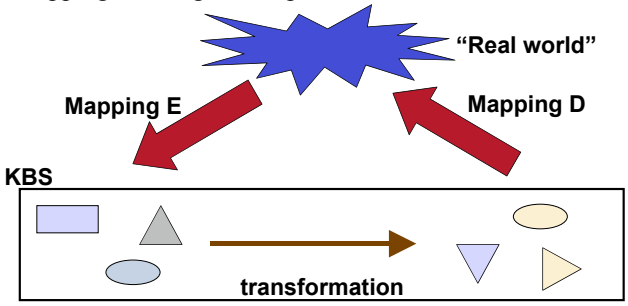

Figure 2. Knowledge Based System Model's Relationship To Reality.

To be fair, logicians freely acknowledge this limitation, reminding us that the power of logic is that symbols and predicates can come from anywhere. No matter where the symbols come from, our logic system will preserve them and their relationships through a series of well-founded transformation steps. Of course, this is an appropriate use of logic and a clearly powerful feature for a knowledge system. One of my purposes in this paper is to describe clearly what are the limitations of logic and what additional features a practical knowledge system needs.

Figure 3 adds an essential characteristic of any useful knowledge model – namely that the real mapping to/from symbols depends upon a human's interpretation. This dependency was clearly articulated over a century ago by Charles S. Peirce (e.g., see [17, Lecture Two]) and reinforced by others. Indeed, much of the difficulty in building effective knowledge systems stems from our inability to formalize these mappings. In any knowledge-based system (whether formal or informal), there exist boundaries where formality can no longer apply. Even for a formal system, these boundaries constitute formality's "end-of-the-world" beyond which lies the *terra incognita* of interpretations.

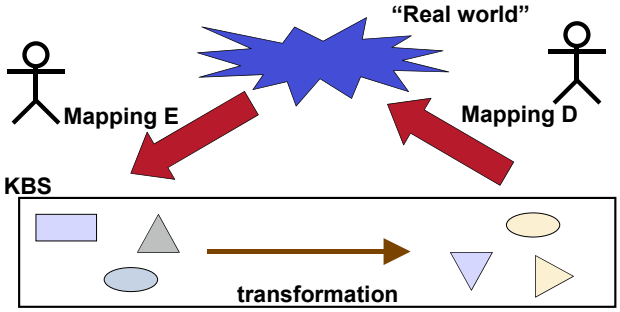

Figure 3. Human Role In Interpretation.

We often overlook (or take for granted) this significant boundary. We have become so accustomed to using symbols for the actual things they represent that we forget the symbols themselves have no inherent meaning. English speakers see **cat** and immediately envision a small fluffy animal with a tail that is often treated as a human pet. One way to remind ourselves of this boundary is to pick up any document written in a foreign language that we do not know. When we see a sentence in such a language, we not only do not know what the symbols represent, we do not even know what is being written about! That is why foreign travelers spend a lot of time looking at the pictures accompanying written text.

I do not propose that we will ever be able to formalize interpretations from inside our models all the way into a person's mind; the best we can attain is the ability to formalize enough of the interpretation process so that we can share them among multiple persons and multiple systems.

One step toward that formalization is already in place. Conceptual graph theory provides a mechanism for getting at least part way toward this interpretation through *individual markers*. Markers are identifiers for real things, so that we can establish some facts about individuals in the real world, most notably what type they are. This part of conceptual graph theory therefore does attempt to show a formal model of the real world (albeit a shallow one) as follows: *the world consists of individuals, each of which can be identified using a unique individual marker*. Essentially this is a naming approach, whereby every thing is given a name. A system therefore contains an associative lookup between unique identifiers and particular things.

Of course this approach does not solve the problem of what symbols stand for; it merely defers the question (or restates it): namely, how do we construct the map that associates individual markers with actual real-world individuals? It also does not directly address the question of how to relate things (such as aggregates) that may not be denoted by a single typed concept. Constructing the map constitutes a *procedure*; it is not a purely logical or structural feature.

The real-world things to which a model refers are called its *denotation*. I will use that notion of how a knowledge system relates to the real world by describing a set of features to support denotational semantics. Denotational semantics can be supported by several technologies, such as:

- Data mining with augmented semantics
- Intelligent agent operations
- Temporal reasoning
- Sensor operation and interpretation

Figure 4 shows the "heaven" of a conceptual graph sheet of assertion, with the real world below it. Within the plane of the assertion sheet, the rules of logic hold; outside of the plane of assertion are found links to the real world. In a practical system, there must be actuators to cause information to flow along these links. The links shown in Figure 4 are not meant as a definitive structure. They are meant to illustrate some examples of the range in behavior we would expect from a practical system that is linked to the real world.

The sheet of assertion contains our knowledge based system model, with appropriate concepts, relations, contexts and (most importantly) actors. These actors provide "hooks" to the real world, giving our system its eyes, ears, fingers, and feet (and even a sense of "smell"!). Different kinds of actors provide different kinds of senses and connection to the world; they are briefly described in the following sections.

Within the sheet of assertion, traditional rules of logic can apply; however, since actors will necessarily react to a changing world, the contents of the sheet will also change. Though it is possible to build a system without actors (and actually solve some problems with it), such a system would only be able to model those parts of the world that do not change.

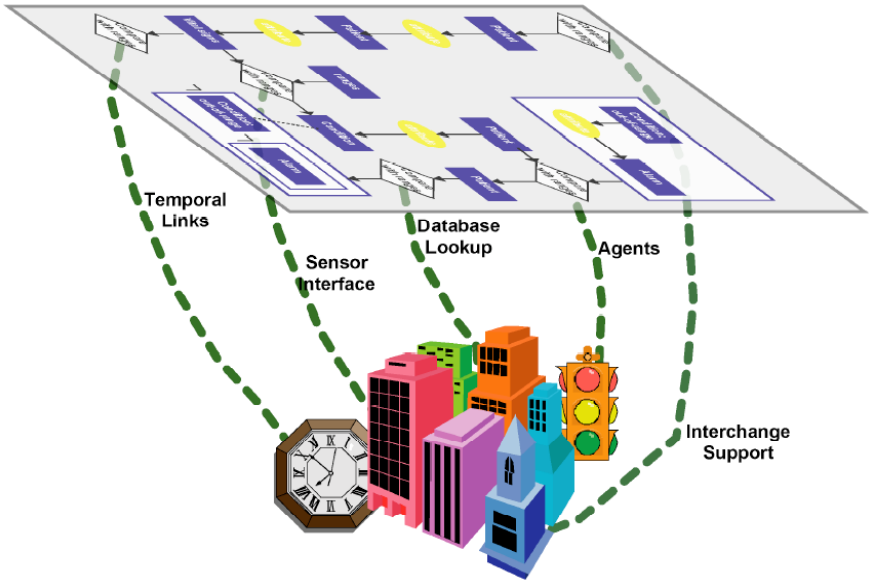

Figure 4. Active Knowledge System Framework.

2.3 Data Mining With Augmented Semantics

There is much useful information currently stored around the world in conventional databases. A key ingredient in a practical system is its ability to acquire information from these databases. While a database itself possesses insufficient semantics for real knowledge based reasoning, there is much to be gained from the data if we use a semantically driven data mining process. A knowledge engineer can model the database's semantics by building conceptual graph models in a knowledge system, while populating the models with the actual identities of concepts that are retrieved from a conventional (relational) database as needed.

Here is a simple illustration. Suppose a database contains company records and has an employee relation as in Table 1. Because we humans can read and understand the column (field) labels, and because the first few entries' values seem to match our understanding, we tend to think that the relation table has meaning, but it really has very little inherent semantics. For example, each row appears to capture information about a different person, each of whom is employed by the organization that is being modeled by the database. As another example, "Yrs Experience" might mean experience in business environments, or experience in that particular position, or experience at that particular organization. And of course, replacing the (apparently meaningful)

names with arbitrary identifiers like “bq598” would not alter the formal characteristics of database accesses at all.

Table 1. A relational database table.

Name	Position	Yrs Experience	Degree	Major	Percent Stock
Karen Jones	VP Marketing	18	MBA	Marketing	3
Kevin Smith	VP Technology	12	MSE	Engineering	4
Keith Williams	VP Finance	15	BS	Accounting	3
…	…	…	…	…	…

We can easily construct a conceptual graph with actors that will retrieve values from the database and insert them into a graph; in fact, CharGer already does this [4]. The fact that it can be constructed automatically tells us that there is no particular meaning in either the values themselves, or in the column labels (i.e., field names). Figure 5 shows how data values are acquired from a database and inserted into a knowledge base. The **<lookup>** actor is a procedural element, which performs the association between attribute names and a lookup key.

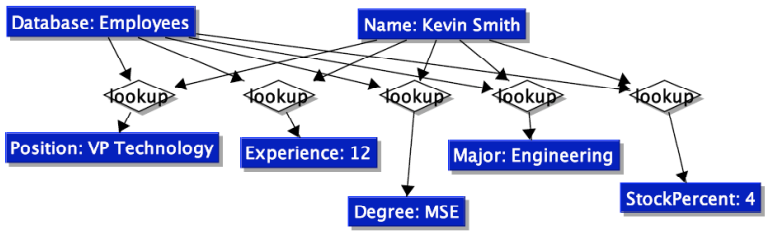

Figure 5. Database Information Without Semantics.

We have shown how limited are the semantics the database possesses by itself. In order to show semantics, we want to know what all those values mean. A richer representation would be found in Figure 6 which shows the conceptual relationships (beyond the “relations” in a conventional database) that exist in the real world between the values and a particular employee. Now we are getting closer to what an intelligent system needs to operate.

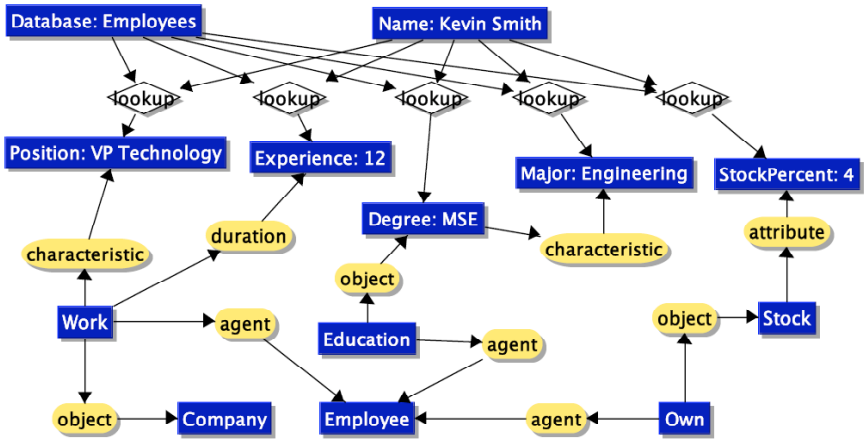

Figure 6. Database Information With Augmented Semantics.

Additional semantics can be provided to Figure 6 by adding additional graph elements and by including a type or relation hierarchy.

2.4 Intelligent Agent Operations

Researchers are already working on conceptual graph models of intelligent agents (e.g., [11] [7]). An intelligent agent is an autonomous entity, capable of controlling its own communication, deciding which other agents should be contacted, and evaluating the quality of the knowledge gained. Agents are well suited for supporting a knowledge system, particularly if they can use the knowledge system's semantics to formulate queries and communicate with other agents.

One might argue that conceptual graph actors are already intelligent agents, since they interact with the outside world. Unfortunately, an actor is too limited (in general): its interactions with the outside are carefully pre-planned and constrained. The **<lookup>** actor in the previous section is a good example**. <lookup>** is "hard-wired" to a specific database by the graph itself and has a very limited function (e.g., it is not clear what the **<lookup>** actor should do if it cannot find a value to match its lookup key).

An intelligent agent is an autonomous entity that can interact with other knowledge based systems and/or their agents, making decisions along the way. A knowledge-based system will need to create new agents that are sent off into the real work to collect information that will be sent back to its creator. How would this work?

The active knowledge system first creates an intelligent agent, and may start it off with a "briefcase" of knowledge to work with. The knowledge system may also create an interface (or use a previously created one) through which the agent can communicate with the system. At a minimum, an agent must be able to report back its results. It should also be able to report back preliminary findings and have its search refined via interaction with the knowledge base. Of course, the agent could be supplied with all the knowledge it needs beforehand, but an adaptive agent cannot know all the content it needs in advance.

An agent is much more "intelligent" than an actor, which might merely be a functional relation. Whereas an actor takes a set of referents and returns one or more new referents, an agent has the ability to evaluate a set of graphs and return one or more graphs, as well as having the capability to alter existing graphs in the set it is given.

2.5 Temporal Reasoning (What time is it?)

By temporal support, I mean access to clocks and other time-keeping mechanisms so that the system will maintain some notion of what time it is in the real world. Just as most computer operating systems have access to a wall-clock time, so should the knowledge system have access to the world's actual time.

The notion of real-world time need not always mean the local wall clock. One advantage to having temporal support is that we can use it for models operating in time frames other than the one we all share in the real world. For example, a simulation can be supported by a knowledge system that operates on "simulator" time. Past scenarios or event records can be re-played using time values corresponding to when things actually happened. Demographic or geological models can use a much coarser granularity of time.

Temporal reasoning encompasses more than just having a clock. Temporal concepts include reasoning about duration, intervals, and relationships between them.

Unlike pure logic, which does not care the order in which expressions are evaluated, real world events often have constraints on which ones must precede others. An active knowledge system can use its temporal reasoning abilities to help solve problems dealing with processes and time.

2.6 Sensor operation and interpretation

Regardless of how sophisticated we make our models, they remain closed off from things in the real world. A closed system relies completely on the decisions of people and other systems for any knowledge it acquires. These external decisions restrict a system's potential to autonomously gather new information.

I consider here a general notion of a sensor: the eyes and ears (fingers, toes, etc.) of a system which can continually update locations, states, and processes that can only be statically represented by concepts in the representation.

The temporal support in the previous section is a simple example of a sensor that merely reads a clock. Other sensors can be attached to entities in the real world so that we may get a "reading" on them – their location, state, etc. In this way, some concepts' referent values (or the concepts themselves) become changeable, reflecting the real world, which the model represents.

Sensors are only placed on those entities and attributes that we choose to track. These choices already constrain a system's ability to effectively use the sensor's information, especially if the structure of the environment changes. This is a persistent problem in current (passive) monitoring systems. For example, there is a feature in human beings (*selective attention*), whereby we are able to focus our senses more sharply on some parts of our surroundings and reduce the focus on others. This is not just a matter of efficiency in applying the resources of our sensory apparatus; it is also a matter of selecting priorities about what is important to see. Active knowledge systems need support for a kind of "selective attention" in the same way.

3 Other practical considerations

There are some other considerations that are not necessarily part of a system's interaction with the outside world, but are nonetheless essential to a practical usable system. These considerations are:

- Interchange mechanisms and standards
- Support for knowledge-in-the-large
- Explanation facilities

3.1 Interchange mechanisms and standards

Since constructing a knowledge model is both labor-intensive and time-consuming, we would like to re-use our models on other projects and also share our models with other systems. There are many issues involved in both re-use and sharing that must be addressed, or else each developer will find himself building little isolated solar systems of knowledge instead of contributing to a galaxy of inter-related knowledge.

One key issue is that of an agreed-upon interchange format (see sec. 2.1). It is obviously inefficient for each pair of system developers to get together and create a one-time interchange scheme for their systems. It seems clear that formal standards are the

best solution to the interchange problem, and such standards are being developed. A standard does not solve all the problems associated with exchanging information; it merely makes a solution possible.

A larger problem when hooking systems to the real world is that when exchanging knowledge, the interpretive mapping from symbol to actual thing may be lost. One solution is to specify some procedure that performs the mapping, and have the capability to exchange such procedures as well. This means that whenever a particular symbol appears in the model, there is an associated procedure that can be invoked to get values for a set of attributes associated with that symbol.

3.2 Support for knowledge-in-the-large

The previous sections dealt with specific technologies that provide a range of different capabilities for an active system. Regardless of the capability being considered, however each such feature must be practical for very large models that may be tightly coupled to the real world through actors. All of the technologies mentioned here must be scalable if a system is to solve practical problems.

Knowledge-in-the-large does not just refer to the sheer number of things. If we simply had to perform linear searches through long lists of things, there are well-known ways to process these lists efficiently. The real problems in scalability fall into two categories:

- Computational complexity: looking for partial matches, mining for new relationships, etc.
- Concurrency complexity: large numbers of agents all operating simultaneously on large numbers of graphs
- Conceptual complexity: the explanation of a result may turn out to be difficult or impossible for a person to understand, regardless of how reliable is the process by which the results are obtained

The first of these problems has received considerable attention in conceptual graphs, most notably the efforts of Levinson [9]. The second problem is only now beginning to be understood in terms of what is expected of agents; their computational aspects are not yet being addressed. The third problem will be discussed in a moment.

3.3 Explanation facilities

A practical system must have a way of explaining itself to its human users. It must be able to explain its own internal knowledge, its lines of reasoning in answering queries, and even its own internal operations. Most current systems require an in-depth explanation as to what they do; these explanations often must come from the programmers themselves or from a detailed examination of the design or source code, since the intelligent system itself does not have access to its own functioning.

In order for a system to explain itself, it must have direct and explicit access to all of its internal inference and reasoning procedures. “Hard wiring” these procedures (usually through program code) has the effect of removing them from the actual knowledge model; yet, these reasoning mechanisms are what gives the system its power!

3.4 Limitations on our understanding

The problem of conceptual complexity is an interesting one whose issues are ultimately influenced by the limits of our ability as human beings to understand our world. In our daily lives, our economic lives and our political decisions, we encounter situations where our intuition can lead us to an understanding that is counter to what logic or actual observation tells us. Though automated reasoning machines may be able to gain our confidence and trust, we still may not really be able to comprehend what they tell us. This makes it even more important for us to design and build knowledge-based systems with great care, since they may one day surpass our ability to understand them.

4 Conclusion

Can this become real? The support features described here may seem ambitious – can we really build them all? And do we really need them all? To be sure, they may go beyond what any particular system builder would need. Instead I am proposing that architectures and algorithms be developed for all these features, so that the system builder need not start from the beginning to develop them.

Though the list may seem to include unnecessary features, they may help a system builder to see beyond their immediate needs and thereby design new capabilities that might not have seemed possible without a pre-existing architecture.

This paper has been a brief attempt at illuminating the landscape of intelligent systems' possibilities, in order to help others as they begin to explore it. I look forward to seeing these features begin to take shape in practical knowledge-based development so that active knowledge systems can begin to help us solve more difficult and interesting problems.

Acknowledgments

I thank Bernhard Ganter, Aldo de Moor and the reviewers for their careful reading as well as their thoughtful and helpful comments on drafts of this paper. I am also grateful to Dan Rochowiak for our many stimulating conversations on these topics.

References

[1] "Metadata Standards", ISO/IEC JTC1 SC32 WG2, http://metadata-stds.org/

[2] Tim Berners-Lee, James Hendler, and Ora Lassila, "The Semantic Web," in *Scientific American*, vol. 284, 2001.

[3] Dan Corbett, *Reasoning and Unification over Conceptual Graphs*. New York: Kluwer Academic, 2003.

[4] Harry Delugach, "CharGer: Some Lessons Learned and New Directions," in *Working with Conceptual Structures: Contributions to ICCS 2000*, G. Stumme, Ed. Aachen, Germany: Shaker Verlag, 2000, pp. 306-309.

[5] "CharGer - A Conceptual Graph Editor", Univ. of Alabama in Huntsville, http://www.cs.uah.edu/~delugach/CharGer

[6] "GoGITaNT", LIRMM - Montpellier, http://cogitant.sourceforge.net/index.html

[7] Lois W. Harper and Harry S. Delugach, "Using Conceptual Graphs to Capture Semantics of Agent Communication," in *Conceptual Structures for Knowledge Creation and Communication*, *Lecture Notes in Artificial Intelligence*, B. Ganter, A. d. Moor, and W. Lex, Eds. Berlin: Springer-Verlag, 2003.

[8] Adil Kabbaj and Martin Janta-Polcynzki, "From Prolog++ to Prolog+CG: A CG Object-Oriented Logic Programming Language," in *Conceptual Structures: Logical, Linguistic and Computational Issues*, vol. 1867, *Lecture Notes in Artificial Intelligence*, B. Ganter and G. W. Mineau, Eds. Berlin: Springer-Verlag, 2000, pp. 540-554.

[9] Robert Levinson, "Symmetry and the Computation of Conceptual Structures," in *Conceptual Structures: Logical, Linguistic and Computational Issues*, vol. 1867, *Lecture Notes in Artificial Intelligence*, B. Ganter and G. W. Mineau, Eds. Berlin: Springer-Verlag, 2000, pp. 496-509.

[10] Philippe Martin, "The WebKB set of tools: a common scheme for shared WWW Annotations, shared knowledge bases and information retrieval," in *Conceptual Structures: Fulfilling Peirce's Dream*, vol. 1257, *Lecture Notes in Artificial Intelligence*, D. Lukose, H. S. Delugach, M. Keeler, L. Searle, and J. F. Sowa, Eds.: Springer-Verlag, 1997, pp. 585-588.

[11] Guy W. Mineau, "A first step toward the Knowledge Web: Interoperability Issues among Conceptual Graph Based Software Agents," in *Conceptual Structures: Integration and Interfaces*, vol. 2363, *Lecture Notes in Artificial Intelligence*, U. Priss, D. Corbett, and G. Angelova, Eds. Berlin: Springer-Verlag, 2002, pp. 250-260.

[12] Aldo de Moor, "Applying Conceptual Graph Theory to the User-Driven Specification of Network Information Systems," in *Conceptual Structures: Fulfilling Peirce's Dream*, vol. 1257, *Lecture Notes in Artificial Intelligence*, D. Lukose, H. S. Delugach, M. Keeler, L. Searle, and J. F. Sowa, Eds.: Springer-Verlag, 1997, pp. 536-550.

[13] Bruce R. Schatz, "The Interspace: Concept Navigation Across Distributed Communities," *IEEE Computer*, vol. 35, pp. 54-62, 2002.

[14] Finnegan Southey and James G. Linders, "NOTIO - A Java API for Developing CG Tools," in *Conceptual Structures: Standards and Practices*, vol. 1640, *Lecture Notes in Artificial Intelligence*, W. Tepfenhart and W. Cyre, Eds. Berlin: Springer-Verlag, 1999, pp. 262-271.

[15] J.F. Sowa, *Conceptual Structures: Information Processing in Mind and Machine*. Reading, Mass.: Addison-Wesley, 1984.

[16] J.F. Sowa, *Knowledge Representation: Logical, Philosophical, and Computational Foundations*: Brooks/Cole, 2000.

[17] Patricia Ann Turrisi, *Pragmatism as a Principle and Method of Right Thinking, by Charles Sanders Peirce: The 1903 Harvard Lectures on Pragmatism*. Albany, NY, U.S.A.: State University of New York Press, 1997.