Cognitive Computing and Big Data Analytics
By Judith Hurwitz, Marcia Kaufman and Adrian Bowles

CHAPTER
5

Representing Knowledge in Taxonomies and Ontologies

Learning from data is at the heart of cognitive computing. If a system cannot use data to improve its own performance without reprogramming, it isn't considered to be a cognitive system. But to do that, there must be a wealth of data available at the heart of the environment, formats for representing the knowledge contained within that data, and a process for assimilating new knowledge. This is analogous to the way a child learns about the world through observation, experience, and perhaps instruction. This chapter looks at some simple knowledge representations before exploring more sophisticated and comprehensive approaches to knowledge representation: taxonomies and ontologies.

Representing Knowledge

In computer systems as in humans, knowledge may include facts or beliefs and general information. It should also include standard knowledge organizational structures such as ontologies and taxonomies—as well as relationships, rules, or properties that describe objects (nouns) and help to categorize them. For example, we may know that people are animals and Bob is a person, so Bob should have all the properties that we associate with animals. In people, we sometimes equate knowledge with understanding, but that's not the case with computers. Of course, in a computer, it is possible to "know" a lot without

"understanding" anything. In fact, that's the basic definition of a *database*: a collection of associated data organized within a computer environment so that it can be easily accessed.

Think for a minute about the smartest people you know. What makes someone smart or intelligent? It's much more than having an encyclopedic memory. Intelligence is the ability to acquire, retain, analyze, develop, communicate, and apply knowledge. One can be intelligent without knowing much—think of a precocious child, who may display signs of intelligence before acquiring much knowledge. Conversely, a person can know plenty of facts but not know how to use those facts to accomplish a goal.

Developing a Cognitive System

There are many different techniques that are useful in creating a cognitive system. One important technique is to leverage massive amounts of data and analyze the patterns that emerge from that data without providing an explicit query. This issue is covered in Chapter 6, "Applying Advanced Analytics to Cognitive Computing." In essence, you are not telling the system what answer you are looking for. In a 2012 experiment, Google researchers selected at random 10 million images from YouTube videos and used a network of 16,000 processors to look for patterns. Perhaps not surprisingly, this system found a distinct pattern (shading in various points of the image in the same proportion and relationship to other repeated subimages) from among more than 20,000 distinct items in the images. By analyzing these images in detail, looking for such a pattern that was repeated more often than random arrangements of pixels—regardless of color, background, image quality, and so on—it found one promising combination of shadings that appeared frequently enough to be flagged as unique. It "discovered" a generic pattern for images of cats.

Although this experiment verified that it was possible to detect patterns systematically in a big data sample, it was only a beginning. Most cognitive computing systems take a more focused approach. They are designed to learn and provide value to users in a specific field or domain, such as medical diagnosis or customer service. A challenge for these cognitive systems builders is to capture enough relevant knowledge to be useful and to represent it in a way that allows the system to add to the knowledge or refine it with experience.

Each industry and each domain within that industry has its own vocabulary and historical knowledge. These domains include a lot of different types of objects, from systems and parts of the body in medical systems to engine parts in a predictive aircraft maintenance system. Each of the object types may have specific rules that guide their interaction and behavior. For example, an X-ray may be a specific object type that has certain physical properties.

Likewise, a wing nut in an airplane part will be associated with specific rules governing how it may be installed and serviced with other physical components. The process of capturing and representing this knowledge requires experts who understand the vocabulary and rules of their industry well enough to explain them so that they can be codified for machine processing.

However, even with the support of industry experts, it is not possible to capture enough knowledge and nuance to design a system that replicates a complete understanding of an industry or market. Therefore, most cognitive systems start with a meaningful subset of domain knowledge and then dynamically—with experience or training—enhance and refine that basic model. The foundation of this approach is to define taxonomies and ontologies focused on a specific area of knowledge.

Another interesting aspect of creating cognitive systems is "cross-context" understanding. In order to achieve higher levels of cognition, people or systems must be able to correlate data from multiple corpora at the same time. Humans do this type of correlation early on in life with almost no effort. We learn how to ride a bicycle and then we process information about weather, traffic, road conditions, etc. so that we can ride safely and get where we plan to go. In our earlier example about aircraft parts, wouldn't it be appropriate for the cognitive system to connect the parts assembly with safety and historical weather data to better recommend materials or processes?

Defining Taxonomies and Ontologies

Before getting into the details of how knowledge is managed, it is important to define taxonomies and ontologies. This is covered in more detail later in this chapter ("Explaining How to Represent Knowledge"), but definitions now will provide context. Taxonomies are a hierarchical way of capturing or codifying information within a particular field of study.

You can think of the categories of data within a taxonomy as a set of classification frameworks with common properties. *Hierarchical* refers to a structure or structures where a subcategory, like a subset, inherits all the properties defined in the superset or "top" of the category. A taxonomy typically has a formal way to specify the properties that apply to all elements within each category.

If you were interested in capturing everything that is known about motor vehicles, you could start with a set of motor vehicles, as shown in Figure 5-1, and perhaps divide that into subsets of passenger cars, motorcycles, commercial vehicles, and so on. You could create further categories or subsets for buses, taxis, and so on. This can become quite cumbersome, and the decision of which sets are at the highest levels should reflect the intended use. For example, is it

more important to the user that an electric bus carries a lot of people, or that it is powered by electricity? Fortunately, you can factor the data according to usage to simplify development.

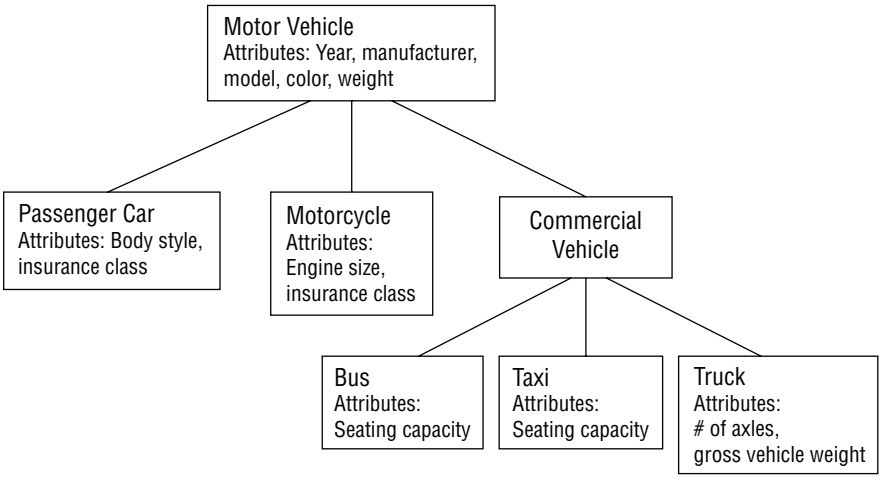

Figure 5-1: Motor vehicle types

For example, in a vehicle tracking system, there could be definitions for cars, boats, motorcycles, buses, and trucks. If you are a motor vehicle department and you are tracking registrations to determine ownership and assign fees, there is little information you need about each category. A car may be defined as a vehicle with two axles, and each "instance" of car—representing one actual car in the physical world—would have a weight assigned that could compute registration fees. If your state decides to tax based on other attributes, such as fuel type or EPA fuel ratings, the taxonomy might include those details in each specific car record, or create classes for "fuel efficient" and "gas guzzler" vehicles.

For other applications, the same vehicles may be categorized differently in a taxonomy. For example, an insurance company must calculate a liability class based on properties like horsepower versus weight and body type (convertible versus sedan). The taxonomy for an insurance company could organize the classes differently and have new subclasses tailored to their processing requirements. Certain industries have well-defined, mature taxonomies. The pharmaceutical industry, for example, has detailed taxonomies of compounds used to create a variety of drugs.

In contrast, an *ontology* generally includes all the information you would capture in a taxonomy, but also includes details about rules and relationships between the categories and about criteria for inclusion. An ontology is more likely to include semantic information that can be used to guide decision making. A richer, more fully specified ontology enables more ways that it can be applied to solving problems and decision making.

THE ROLE OF STATE IN A COGNITIVE SYSTEM

Before getting into a discussion about how to represent knowledge and how to build a model that can make the connections between elements, you need to understand the concept of state. *State* is the condition of a system at a particular point in time or in a specific situation. As a simple example, a body of water could be in one of three states: solid, liquid, or gas. The state variable is the temperature. To determine what state it was in last month, knowing the temperature at the time (and for a period leading up to that time) would be sufficient. For a cognitive computing system, the state may include many variables, from values for stored knowledge to user logs to configuration data (which modules were actually in place at a particular time). The ability to determine state information or restore a system to a particular state may be an auditing requirement (for example, in a financial services or medical diagnostic system). If the system is used as a cognitive platform for applications, the platform may not need to track state information at all if that is left to the applications themselves.

Explaining How to Represent Knowledge

Deciding on a knowledge representation scheme—such as taxonomies and ontologies—is a critical step in planning a cognitive computing solution. Simplicity is always a good design goal, but some problem domains are inherently complex, with relationships that are imprecise or that cannot be specified completely. As a general rule, we want to at least capture all the known object types or classes. A *class* defines the properties of a set of elements or instances. The class definition accomplishes the goal of providing information about relationships or behaviors as the system learns. A more robust representation requires more work in the beginning, but it is more flexible when the system is operational.

Domain knowledge within a cognitive computing application may be captured and stored in a variety of data structures, from simple lists, to conventional databases, to documents, to multidimensional purpose-built structures. Cognitive computing system designers can use procedural, list-processing, functional, or object-oriented programming languages to specify and implement these structures. They may use data modeling tools or even specify the knowledge model in a language created just for this purpose. The choice of tools and representations should reflect the types of operations the system will have to perform on the data. As is the case in any application, there may be trade-offs that dictate one approach over another. Optimizing a medical system for fast diagnosis of toxins (poison control, for example) may make it suboptimal for recommending appropriate lifestyle changes based on similar input. It is important to consider typical scenarios and rare but conceivable test cases when defining the knowledge model and the structure you will use to implement it in software.

A single system may actually contain several knowledge repositories, partitioned by the task or by some attribute of a particular class of objects. For example, a predictive maintenance system for a large manufacturing firm may have

separate instances for problems with toasters and MRI machines. Knowledge for each machine type could be organized by attributes of the machine, or perhaps by types of fault and their frequency of occurrence.

The logical design for the collection of data structures used to represent knowledge acts as a model for the domain. Some domains are straightforward to model. For example, the game of chess is centuries old and easy to explain and represent but difficult to master. Chess is a two-person, perfect-information, zero-sum game. Players see the same board and can be expected to know the rules and therefore mentally compute possible next moves, limited by their ability to store all the alternatives in their brain's memory.

A system that plays chess has to model the following elements: an 8x8 chessboard and 32 chess pieces, grouped into 6 categories or object types (pawn, rook, knight, bishop, queen, and king). Each category has its own point value and set of permissible behaviors. The system must know the starting location and color of each piece. (Actually, if it "knows" the starting location, it "knows" the color, too.) As the game progresses, the system must ensure or enforce that only legal moves are made and calculate its own best move based on the state of the game. Figure 5-2 shows a variety of representations that could capture the state of a chess game.

	A	B	C	D	E	F	G	H
5								
4								
3								
2	WP	WP	WP	WP	WP	WP	WP	WP
1	WR	WK	WB	W King	WQ	WB	WK	WR

Possible first moves

A2 A3, A2 A4, B2 B3, B2 B4, C2 C3, C2 C4, D2 D3, D2 D4
B1 A3, B1 C3

	Piece	Place
1	WR	A 1
2	WK	B 1
3	WB	C 1
4	W King	D 1
	⋮	⋮
30	BK	F 8
31	BB	G 8
32	BR	H 8

Rules

Piece	Moves
Rook	Vertical, horizontal, 1-n
Knight	Row +/–2, col +/–1 Row +/–1, col +/–2
Bishop	Diagonal 1-n
	⋮

Figure 5-2: Representing a chess game

Chess notation has evolved over the years. Today, a Standard Algebraic Notation is commonly used to record physical games by the World Chess Federation. The notation may be used internally in a chess program to capture knowledge about the "state" of the game. This means that the knowledge of the parts and permissible behavior is straightforward. In fact, a program that considers only the current state of the board and evaluates several or even every possible future state before making its next move could beat human champions (who have a limited ability to evaluate future states) without being cognitive—all the rules could be instantiated in code before the first move is made. The brute force approach is possible because the domain is completely specified. At any point in the game, both sides have perfect information and can—in theory—calculate and evaluate every possible subsequent move given sufficient time and memory.

A more sophisticated approach might be adapted to an individual opponent by reviewing every move she played in past games to predict her next move based on how she moved in similar game states (configurations of pieces). Using historical behavior to defend against unconventional strategies would require much more of this historical knowledge. The pieces, behaviors, and game state would be the same with or without considering historical behavior of a particular opponent. All could be represented with simple data structures—a reasonable assignment for a college freshman studying computer science. The difference is in the complexity of knowledge required to select a move based on context. Although the possible moves are always the same for a given state, the choice of moves might differ based on context.

Writing a chess program that can adapt to an individual opponent by evaluating the context of a particular move by comparing it to historical behavior to defend against unconventional strategies would be more complex, but the underlying principles are the same.

Now look at a more difficult domain to represent: automotive diagnostics and repair (Figure 5-3). Every automobile powered by an internal combustion engine (ICE) shares certain properties, components, and major subsystems. Electrical, fuel, ignition, cooling, and exhaust are but a few of the dozen or so commonly recognized subsystems. A cognitive solution for automotive diagnostics would have to represent each component of each system, and possible interactions between them.

Here, classification begins to blur the lines; for example, is an electric fuel pump part of the electrical system or part of the fuel system? Is a coil part of the electrical system or ignition system? For diagnostics, common symptoms must be codified before the system can suggest a cause. Is black smoke coming from the exhaust pipe an indication of an exhaust system failure or a fuel failure? (It is typically fuel.) Is white smoke an exhaust or fuel problem? (Usually neither—it's an indication of a bad gasket allowing water in the combustion chamber.)

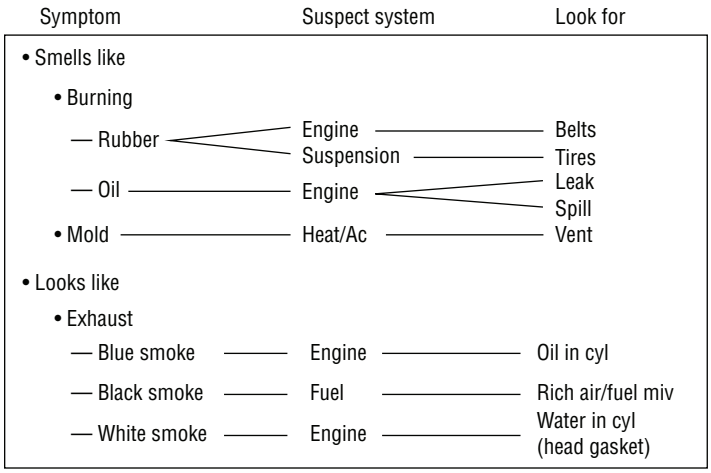

Figure 5-3: Automotive diagnostics and repair

When it is impractical to capture all the relationships between components, and between component condition and symptoms of failure in advance, a cognitive system can accept feedback and improve its performance. However, this is only practical if the knowledge representation is robust enough to cover all conditions. Decisions made about classification impact how knowledge is stored, which may dictate what types of problems can be easily solved.

So far we have been dealing with relatively simple domains. It is much more difficult to navigate when a domain's experts do not agree on knowledge and rules. For example, in drug discovery there are too many factors and complications to create a straightforward knowledge base. Therefore, it is not surprising that there are practical limits as to what we can process effectively today.

Although some domains are straightforward, others are broad and complicated. For example, though we think of "medicine" as a single domain, it actually consists of a huge number of disciplines. Therefore, you would not expect that there could be a single representation of the knowledge in that domain. You can't represent everything relevant to medicine in one system—that would be like having a 1:1 scale map of the world with views for political boundaries, roads, topography, and weather. In medicine, there are well-defined subsystems (such as circulatory, respiratory, nervous, and digestive), and well-defined diseases, medical conditions, and pathologies that cross these systemic boundaries.

For representing and codifying knowledge, one-half the battle is deciding what to ignore in a specific model. As Marvin Minsky, a pioneer in artificial intelligence and knowledge management, noted in *Artificial Intelligence at MIT: Expanding Frontiers* (Patrick H. Winston, Ed., vol. 1, MIT Press, 1990. Reprinted in *AI Magazine,* Summer 1991):

To solve really hard problems, we'll have to use several different representations. This is because each particular kind of data structure has its own virtues and deficiencies, and none by itself would seem adequate for all the different functions involved with what we call common sense.

Therefore, some of the earliest cognitive systems are focused on a single branch of medicine, such as oncology. By focusing on an area of medicine, you can start to partition the domain into manageable and meaningful segments.

As with all branches of medicine, the comprehensive study of oncology requires an understanding of diagnosis, care or treatments, and prevention. Each of these subcategories may be further decomposed and studied in isolation. In practice, that is what leads to professional specialization (knowing more and more about a smaller subset of the field), but to "understand" oncology you must understand the interrelationships between these subcategories. Table 5-1 lists common types of cancers. Each subcategory or cancer type would be associated with disease-specific paradigms for diagnosis and treatment.

Table 5-1: Common Types of Cancers

Common Solid Tumors	Lung, colon, breast, reproductive, stomach, brain
Hematologic Tumors	Leukemia, lymphoma
Connective Tissue Tumors	Sarcoma

Managing Multiple Views of Knowledge

In the automobile diagnostic example, you could factor the available knowledge into separate models for each subsystem. Many expert mechanics and maintenance manuals organize their knowledge by subsystems. Asking qualifying questions to rule out one system or another before going deeper into diagnosing a problem within a single system works well and may appear to be an obvious approach. However, there is a danger in partitioning knowledge into subsystems. Partitioning may make it difficult to correctly identify problems when they span multiple subsystems. For example, for healthcare, there might be a subsystem about diagnosing high blood pressure and a second subsystem focused on diabetes. In fact, there is often a correlation between these two subsystems that needs to be taken into account.

In a complex domain like oncology, in which it is more common to have complications involving several subsystems, you may need more than one view or representation to capture all the relevant knowledge. A professional may use books, journals, case notes, and communications with peers to fully understand a new case. Of course, the stored knowledge in the doctor's brain is an amalgam of historical references to these same types of resources.

In a cognitive computing system, you can also capture this type of knowledge and segment it into views that can be linked together to present a more complete view. New discoveries may change the way professionals think

about problems, and that in turn may change the way you choose to segment knowledge in your cognitive systems. For example, cancer types and treatments were historically described by parts of the body where they were found. Someone studying a case of liver cancer might not immediately think to try a treatment previously tested and approved for use on another organ. Recently, however, it has become possible to analyze vast quantities of data across organ boundaries and compare cases based on attributes of the patient's genome. Similarity analysis for healthcare diagnostics is an important application of machine-learning algorithms. This has led to the discovery of promising patterns between patients, genomes, cancers, treatments, and outcomes. As a result, new relationships have been found and new treatments have been applied. This type of discovery points out the value in deferring partitioning as long as possible to prevent missing relationships.

Models for Knowledge Representation

There are many different ways of representing knowledge. It could be as simple as a chart on the wall or as complicated as a full lexicon of terms used in a field along with their representations and definitions. This section provides an overview of taxonomies and ontologies. In addition, it provides some insights into additional knowledge representations that are important in a cognitive system. Within cognitive systems, knowledge representations range from simple trees to ontologies, taxonomies, and semantic webs.

Taxonomies

A *taxonomy* is a representation of the formal structure of classes or types of objects within a domain. Taxonomies are generally hierarchical and provide names for each class in the domain. They may also capture the membership properties of each object in relation to the other objects. The rules of a specific taxonomy are used to classify or categorize any object in the domain, so they must be complete, consistent, and unambiguous. This rigor in specification should ensure that any newly discovered object must fit into one, and only one, category or object class.

The concept of using a taxonomy to organize knowledge in science is well established. In fact, if you've ever started a guessing game by asking "is it animal, mineral, or vegetable?" you were using Linnaeus' 1737 taxonomy of nature. Linnaeus called those three categories the "kingdoms" in his taxonomy (Figure 5-4). Everything in nature had to fall into one of those categories, which he further divided into class, order, genus, species, and variety. At any level in a taxonomy, there can be no common elements between classes. If there are, a new, common higher-level category is required.

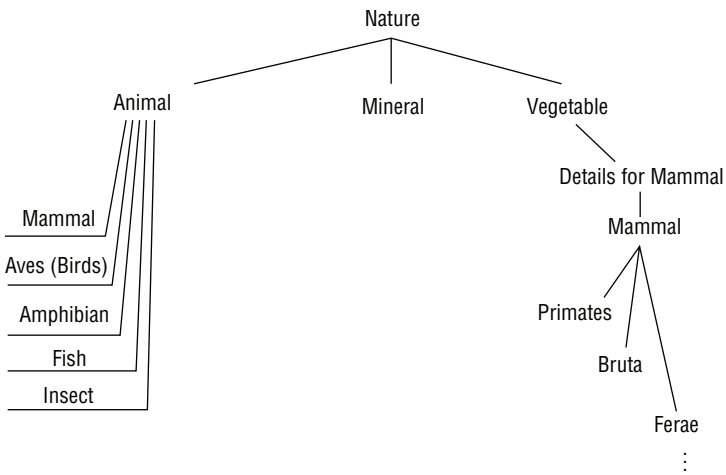

Figure 5-4: Taxonomy of nature

Members of any class in the taxonomy inherit all the properties of their ancestor classes. For example, if you know that humans are mammals, you know that they are endothermic (warm-blooded) vertebrates with body hair and produce milk to feed their offspring. Of course, you also know that humans breathe, but you know that because everything in the class mammals belong to the phylum chordata, which are all animals, and animals respire. Inheritance simplifies representation in a taxonomy because common properties need be specified only at the highest level of commonality.

In a cognitive computing system, the reference taxonomy may be represented as objects in an object-oriented programming language or in common data structures such as tables and trees. These taxonomies consist of rules and constructs that will not likely change over time.

Ontologies

An *ontology* provides more detail than a taxonomy, although the boundary between them in practice is somewhat fuzzy. An ontology should comprehensively capture the common understanding—vocabulary, definitions, and rules—of a community as it applies to a specific domain. The process of developing an ontology often reveals inconsistent assumptions, beliefs, and practices within the community. It is important in general that consensus is reached, or at least that areas of disagreement in emerging fields be surfaced for discussion. In many fields, professional associations codify their knowledge to facilitate communications and common understanding. These documents may be used as the basis of an ontology for a cognitive computing system.

For example, the codes in the Diagnostic and Statistical Manual of Mental Disorders (DSM) of the American Psychiatric Association classify all disorders recognized by the APA. These definitions may change over time. For example, in DSM-5 the diagnosis of attention-deficit/hyperactivity disorder (ADHD) was updated to state that symptoms can occur by age 12 instead of the older view that they had to occur by age 6. That would be a small change in an ontology based on the DSM. A bigger change, as shown in Figure 5-5, was the replacement of four specific disorders (autism, Asperger's, childhood disintegrative disorder, and pervasive developmental disorder not otherwise specified) with a single condition called autism spectrum disorder (ASD). Another major change was the elimination of a "multi-axial" reference system that distinguished between medical and mental disorders. Structural changes like that reflect a change in thinking, which needs to be reflected in any system based on this classification scheme.

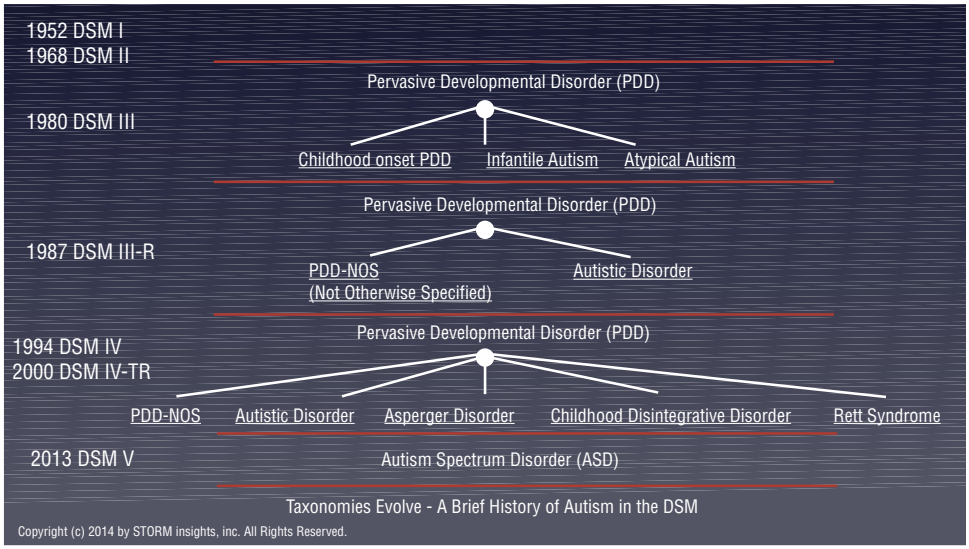

Figure 5-5: Taxonomies Evolve—Autism in the Diagnostic and Statistical Manual of Mental Disorders

A practitioner needs to keep up with changing definitions and standards of care, and a cognitive computing system designed to aid a practitioner must update its knowledge base accordingly. For example, a system tracking patient care must account for the fact that a specific disorder no longer exists but has been superseded by another.

As mentioned, sometimes multiple views are necessary, and they must be compatible. In healthcare, we think of three major constituencies: providers, payers, and patients. Their ability to communicate is critical. Continuing with the mental health example, provider/payer communication is accomplished by the DSM mapping to the ICD-9-CM codification used by insurance companies. It also uses codes from the U.S. Clinical Modifications of the World

Health Organization's International Classification of Diseases (ICD) to provide global compatibility.

The process of updating from DSM IV to DSM V required three drafts and generated more than 13,000 comments from the community. That is an extreme case, but virtually all established professions and disciplines have some common codification that can be used as the basis for a cognitive computing ontology.

For a cognitive computing solution, it is critical that industry experts agree on the underlying ontology. If there are disagreements with the structure or content of an ontology, the output of the system is suspect. Practitioners need to determine what information means in context with their requirements; otherwise, the system will not be meaningful. Early ontologies for learning systems were often specified in programming notation, predominantly LISP (LISt Processing, the original lingua franca of artificial intelligence). LISP is still in use, but as larger domains are being specified in greater detail, developers are increasingly turning to a special purpose language. Web Ontology Language (OWL) is a formal ontology specification language supported by open source tools. As with any knowledge representation, the chosen structure imposes limits on what is captured, and what is captured imposes limits on what can be answered.

Other Methods of Knowledge Representation

In addition to ontologies, there are other approaches to knowledge representation. Two examples are described in the following sections.

Simple Trees

A *simple tree* is a logical data structure that captures parent-child relationships. In a model where the relationships are rigid and formalized, a simple tree, implemented as a table with (element, parent) fields in every row, is an efficient way to represent knowledge. Simple trees are used frequently in data analytic tools and in catalogs. For example, a retailer's catalog may have 30 or 40 categories of products that it offers. Each category would have a series of elements that are members of that category.

The Semantic Web

Some members of the World Wide Web Consortium (W3C) are attempting to evolve the current web into a "semantic web" as described by Tim Berners-Lee, et al in *Scientific American* in 2001. In a *semantic web*, everything would be machine usable because all data would have semantic attributes, as described in the W3C's Resource Description Framework (RDF). The current web is basically a collection of structures or documents addressed by uniform resource locators (URLs). When you find something at an address, there is no required uniformity about how it is represented. By adding semantics, you would force structure

into everything on the web. If we did have a semantic web, we could use more of what is on the web for a cognitive system without extensive preprocessing to uncover structural information.

It is important to differentiate between semantics and syntax. *Syntax* describes the legal structural relationships between elements. For example, one legal form of a sentence is <subject><predicate>, where <subject> may be a noun phrase and <predicate> may be a verb phrase. According to that rule, "Bob runs quickly" is a valid sentence. ("Bob" is the noun phrase, "runs quickly" is the verb phrase.) However, because syntax is structural, we can substitute any noun for another and still be syntactically valid. "Nose runs quickly" or "The blue glass runs quickly" are similarly valid. That's where semantics—the rules for interpreting or attributing meaning to language—come into play.

When processing natural language or even programming languages, you look at syntax first and then semantics. Does the concept follow logically, and then what is the meaning of those terms as they apply to a specific topic? Syntactically, you are looking for the right word type, but semantically you need to look for meaningful words in context. To identify meaning within data requires that there is a hierarchy that takes enough data so that hidden meanings begin to emerge from the usage and context within the corpus of data. The end result has to be the meaning and intent of the data.

The Importance of Persistence and State

The concept of state was discussed in terms of modeling and remembering the placement of pieces on a chessboard after a particular move. Without capturing and recording the state of a game, it would be impossible to stop the application and resume it later. Cognitive computing applications—and platforms—may also be "stateful" and remember details about their last interaction with a user, or a cumulative history, or they may be "stateless" and start each session without preconceptions. At a higher level, they may also capture new knowledge each time and preserve it for future sessions with the same or different users. Alternatively, some knowledge may be preserved, whereas some that is related to a specific user may not be retained.

In many situations, knowledge—or beliefs that we treat as facts—changes as we learn more about a field. For example, what we assumed a decade ago about treatments for lung cancer is radically different from what we know today. Therefore, as a field of study matures, early assumptions about relationships may ultimately be proved false, or confidence levels may change. That is an argument for preserving some state attributes or auditing data to reconstruct the state of the system when something was captured as a "fact." For example, a nutrition system giving advice in 2010 would likely have reported that butter was dangerous, but in 2014 new evidence emerged and we "knew" that was false—based on the best available data at the time. In 2020, the state of the practice

may swing back to an antifat stance based on new evidence, so it is critical that we capture time information with knowledge.

This concept of statefulness is especially important in a cognitive system in which asking a single question will not resolve an issue. In medical diagnosis of a type of cancer, the practitioner may need to ask a series of related questions. Each question will lead to a follow up question. The system needs to not only know the current question but the context of the previous question. A simple example is instructive. When you use a voice recognition system such as Apple's Siri, you might ask, "Is there a restaurant near where I am walking?" Siri will tell you, "Yes, there is a restaurant within the next block." If you now ask, "Does it serve pizza?" Siri will not know the context and will not provide you with a correct answer.

Implementation Considerations

The primary consideration in choosing which representations to use requires that you understand what you need to capture for the types of queries you want answers to. For example, if you want to search quickly for part numbers while performing aircraft maintenance, a tree structure might be sufficient. You may not need to know the history of the origins of all the parts that are used. If you need to trace back the manufacturing history of specific parts to identify likely failures based on other issues with parts made in the same batch, you would need a more detailed representation. And if you are looking for possible relationships between diabetes and a specific skin condition—crossing typical knowledge boundaries—you may need to use a sophisticated and more comprehensive ontology.

Summary

When an organization creates and captures its representation of relevant domain knowledge, that knowledge has to be stored and managed on an ongoing basis. There is no method that works best in every situation. The method used depends on the type of structure within the knowledge, the industry, and many other factors. One certainty is that the size of the data sources will continue to expand and explode over time. Therefore, scalability is a foundational requirement to create cognitive systems that are both trusted and reliable.

As soldiers quickly learn in basic training, when the map and the terrain don't agree, they must believe the terrain. In a cognitive system, when the system's knowledge doesn't reflect reality, the system must be changed. The time to plan for operational changes is in the design phase where knowledge representation decisions are made.