Fundamentals of Knowledge-Based Techniques

Gerard T. Capraro
Capraro Technologies, Inc.
311 Turner Street – Suite 410
Utica, NY 13501
USA
gcapraro@caprarotechnologies.com

Abstract

This paper provides a brief overview of the fundamentals of Artificial Intelligence (AI) and Knowledge-Based (KB) techniques that we feel are necessary to understand the current research efforts being performed in knowledge base radar signal and data processing. A set of definitions and descriptions of some of the major areas of AI are presented. Examples are provided using radar terminology to illustrate concepts presented. Finally we present a description of those technologies being pursued by the World Wide Web Consortium (W3C) for building the Semantic Web or the next generation Internet. The Semantic Web is perceived by some as being a very large knowledge base.

Introduction

Current signal processing systems are built assuming Gaussian clutter and are optimized for their processing requirements whether the systems are mounted on an aircraft, a missile, a spacecraft, or at a ground based site. The algorithms are "hardwired" into the computer's architecture in order to meet the real-time requirements demanded by the sensor's operating parameters, e.g. scans per second and number of sensor elements. This approach to building radar systems is being assessed today by the radar research and development community because of its rigidity and high costs and will slowly change and evolve. This evolution will manifest itself such that different algorithms and/or their parameters will be modified by the radar's software as the environment changes. For instance if a radar is being jammed by a transmitter from a particular direction, then that radar could place a null in its antenna pattern in the direction of the jammer to reduce its negative affect. This and many more sophisticated algorithms have been studied and numerous research papers written.

Some of the most progressive work in employing artificial intelligence (AI) techniques has been pursued by the US Air Force Research Laboratory's Sensors Directorate. Some of their original efforts have been in the constant false alarm rate (CFAR) portion of a radar's signal processing chain. Work was performed (1, 14) to demonstrate that if the cell under test is near the boundary of two different clutter regions, then blindly applying a CFAR algorithm (like cell averaging) will not perform as well as choosing only those cells with the same type of clutter as the test cell and then performing cell averaging. This approach provides a better probability of detection and lower false alarm rates. However, to apply this approach for a radar looking for targets whose background is the Earth, requires that the registration of each cell on the earth be known and the type of clutter be categorized to determine which cells are the same type. If the radar is resident on a moving platform looking at the Earth then the algorithm must be dynamic in order to register the radar's beam on the Earth for each coherent processing interval (CPI). Laboratory experiments with radar data have shown good results especially when a radar is illuminating heterogeneous clutter such as land sea interface.

Capraro, G.T. (2006) Fundamentals of Knowledge-Based Techniques. In *Knowledge-Based Radar Signal and Data Processing* (pp. 2-1 – 2-18). Educational Notes RTO-EN-SET-063bis, Paper 2. Neuilly-sur-Seine, France: RTO. Available from: http://www.rto.nato.int/abstracts.asp.

This work was extended beyond the detection stage to the rest of a radar's processing chain under a US Air Force (USAF) effort dealing with knowledge based space time adaptive processing (KBSTAP) (2,3). This effort demonstrated the benefits of using outside data sources to affect the filtering, detection, and tracking stages of a surveillance radar sensor. Data from a side looking airborne radar system was used in demonstrating the performance enhancements over a conventional radar. The measurements were obtained from the multi-channel airborne radar measurement (MCARM) program (4) conducted by the USAF. Another program showed the benefits of using map data obtained from the US Geological Survey (USGS) to improve the performance of space-time adaptive processing (STAP) on an airborne radar selecting range rings based on computed criteria rather than blindly choosing the range rings surrounding the test range ring. This effort, KBMapSTAP (5,6), along with numerous researchers (e.g. Dr. Michael C. Wicks, Mr. William Baldygo, Mr. Gerard Genello, Dr. William Melvin, and Dr. Joseph Guerci) have laid the ground work for a new DARPA program. The Knowledge-Aided Sensor Signal Processing Expert Reasoning (KASSPER) program is to investigate the use of outside data sources to dynamically change a radar's signal processing chain to enhance a radar's performance.

Can we build new radar systems that can dynamically change its processing given information from other sensors, outside sources, weather data, etc.? We believe that we can. The computing clock rates for computers have been doubling approximately every 18 months. Today's commercial off the shelf computers have clock rates exceeding 3 GHz. We believe that the computing power is available to insert sophisticated "rules/logic" within radar signal and data processing.

This paper provides an overview of knowledge base technologies because we feel that therein lies the methods we will need to design radar systems that can dynamically change their algorithms as required to enhance their performance. The current work described in this lecture series contains examples of this relatively new research field. The knowledge base algorithms are currently only exercising simple rule-based logic. However, in the not too distant future we will be developing KB radar systems that will not only be able to dynamically change their algorithms but be able to explain why they did what they did and be able to learn from their own gathering of data, information, and by measuring their on going performance. The last paper of this series will discuss how KB techniques can be used to build an end-to-end radar signal and data processing system.

The following section provides an introduction to the field of AI and describes some of the major areas of investigation. The next section is devoted to knowledge base systems and methods for data representation and processing. The following section is devoted to some of the basic elements of the Semantic Web and how the W3C technologies along with basic knowledge base processing will allow us to build a knowledge base for radar signal processing. The last section provides a summary.

Artificial Intelligence

Modern day artificial intelligence has been around since the 1950s. It has been defined by Rich (7) as "the study of how to make computers do things at which, at the moment, people are better." Barr and Feigenbaum (8) define AI as "the part of computer science concerned with designing intelligent computer systems, that is, systems that exhibit the characteristics we associate with intelligence in human behavior." Buchanan and Shortliffe (9) define AI as "that branch of computer science dealing with symbolic, nonalgorithmic methods of problem solving."

It is conjectured that there are varied definitions of AI because the field contains many sub-fields or areas of interest within its general domain. Some of these areas are planning, robotics, speech recognition, natural language processing, and expert systems.

Planning is that field where we wish to analyze a significant amount of information in order to develop a methodology to achieve some well defined goal. This area of AI is pursued by business and the military for achieving such goals as maximizing profit and the planning for the delivery of the proper ordinances on targets while conserving life and fuel. These systems are usually interactive and aid decision makers in managing their enterprises.

Robotics is that field that is concerned with developing "thinking" forms of devices that can function within a changing environment. This field is not interested in robotic manufacturing devices where the electromechanical devices only perform a pre-programmed set of actions. This field is interested in developing devices that can achieve its predefined goals by adapting to a dynamic environment, without human intervention.

Speech recognition is concerned about computers understanding human spoken language. The research is part of the general field of natural language understanding whether the communications media is speech or written communications. The goal is to have a computer not only understand a human's language of communications but also to generate responses, usually in the same media.

Expert systems is that field that tries to capture and emulate the actions of an expert within a particular domain of interest. Expert systems are sometimes called knowledge based systems. They are composed of facts about a domain of interest along with heuristics or rules that operate upon these facts.

As can be seen from above the areas of AI that are closest to the radar domain are expert systems and robotics. We chose robotics because we want the radar signal processing to change both on transmit and receive, depending upon its goals and changing environment and operate somewhat autonomously. We chose expert systems and knowledge base systems because we want to change the transmitting and receiving signal processing based upon a changing environment that would be based upon an expert. However, we don't have an "expert", i.e. there is no human that currently modifies the signal processing software chain in real-time based upon the changing environment. We in the radar community are just beginning to develop the rules or heuristics for determining how and when the processing chain should be changed. This is why we emphasize the knowledge base and not the expert system or robotics portions of the AI field. As we become more knowledgeable and have proven techniques we will advance to building our radar sensors as robots. First, we will develop the knowledge base that operates with human intervention. Second, once a knowledge base approach is proven in fielded systems then we may emulate the human intervention portion of the system as an expert system. When this approach is achieved for multiple sensors, possibly operating on an aircraft platform, then our next step will be to have these sensors operate autonomously as a robot.

Knowledge Base Systems (KBS)

As noted above a knowledge base system consists of facts about a particular domain and heuristics or rules that operate upon these facts. A KBS has three main components and can be built in numerous ways. We will highlight some of the main aspects of two out of three of the main components. The main components are the user interface, the knowledge base, and the inference engine. The user interface is that part of the KBS that allows the user to communicate

with the computer. This communication can be implemented using voice, keyboard, touch screen, etc. for both input and in some cases output from the computer. This component is not a major concern for the purposes of this paper since the current research in KBS and radar systems is dealing with embedding the KBS within a computer software process. We are not at the point in our research where we are concerned with how a human will operate with the KBS.

The knowledge base and the inference engine are the major components that this paper addresses. The knowledge base representation is a key element in understanding KB systems and how they function. Three methods that have been used are: predicate logic, semantic nets and frames. The following condensed description of knowledge representation was obtained from different sources (7 - 10) and radar examples added to help understand some of the basic concepts.

Predicate Logic

Predicate logic knowledge representation models what are known as facts within a domain and represents these facts so that an inference engine or computer software can operate upon their representation. The basic element is a fact for example "The ANxx is a radar." This can be represented as Radar(ANxx). We can also state that all radar systems have an antenna (i.e. If X is a radar then X has an antenna.) and therefore by using deductive logic we can deduce that ANxx has an antenna, i.e. ANxxHas(antenna). From this we can generate the following fact in English representation that "The ANxx has an antenna." This approach allows us to traverse the different mappings between facts and their representations. See figure 1.

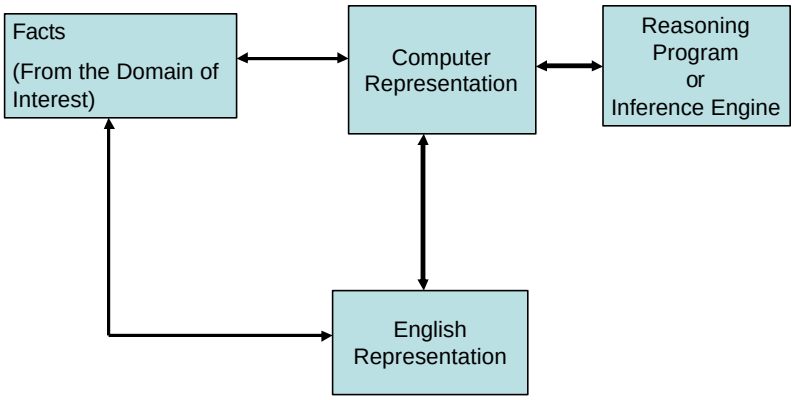

Figure 1. Mappings Between Facts and Their Representations.

Note that these mapping functions or relations may be one to one, many to one, or many to many. For example, "All radars have antennas" and "Every radar has an antenna" could map to the same fact. In predicate logic we can represent domain facts as statements written in well formed formulas. In so doing, this knowledge can be represented in software and inference engines can

process them and draw conclusions and learn. Some example facts and clauses are provided in figure 2.

RadarNom(ANxxx)	LocLatTime(ANsss, xxxxN, Time1)
RadarHas(Tx)	LocLonTime(ANsss, xxxxE), Time1)
RadarHas(Rx)	LocLatTime(ANyyy, xxxxN, Time2)
RadarHas(RxAnt)	LocLonTime(ANyyy, xxxxE, Time2)
RadarHas(TxAnt)	
TxHas(RadiatedPowerLevel)	RxNoiseFloor(ANsss, -110dBM)
RxHas(CFARThreasholdLevel)	RxNoiseFloor(Anyyy, -100dBM)
RxHas(NoiseFloor)	
RxHas(Target)	ReflectiveTime(T1, xxxx)
AntHas(Gain)	ReflectiveTIme(T2, xxxx)
AntHas(3dBBeamWidthElv)	
AntHas(3dBBeamWidthAz)	CFARThreasholdLevel(ANsss, xxxx)
RadarHas(LocLon)	CFARThreasholdLevel(ANyyyy, xxxx)
RadarHas(LocLat)	
TargetHas(Range)	Target(ANsss, T1)
TargetHas(RxPowerLevel)	Target(ANyyy, T2)
TargetHas(ReflectiveTime)	
RangeTarget(Units, Meters)	RxPowerLevel(T1, xxxx)
ReflectiveTime(Units, Microsec)	RxPowerLevel(T2, xxxx)
OneSecondHas(10^6 Microsec)	

Figure 2. Some Hypothesized Radar Facts and Clauses

If the knowledge base defined for a particular domain is small relative to a computer's main memory, then all of the facts and clauses can be contained within main memory. This will allow for very fast processing. If however, for large knowledge bases that cannot be contained within main memory, many developers have made use of database management systems (DBMS) to store the facts while the rules were maintained within main memory with the inference engine. DBMS are built to manage large databases with many users accessing their contents. It has optimized processing functions to manage data stored on secondary storage devices (e.g. hard drives) thereby allowing knowledge base systems to process their rules while maintaining the facts. It should be noted that currently there are some large scale DBMS that contain inference engines within their structure, e.g. Oracle 9i.

For relational DBMS the definition of simple facts such as AntHas(xxx) and RadarHas(xxx) can easily be mapped to an attribute value pair in a relation Has. The attributes are Ant and Radar and their corresponding values are the values in the parenthesis of each of the respective facts. The process of building these facts are similar to the building of entity relationship diagrams for databases. The above facts and clauses can be represented as relations in a DBMS as shown in figure 3.

Has
Relation [Radar, Tx]
[Radar, Rx]
[Radar, RxAnt]
[Radar, TxAnt]
[Tx, RadiatedPowerLevel]
[Rx, NoiseFloor]
[Rx, Target]
[Rx, CFARThreasholdLevel]
[Ant, Gain]
[Ant, 3dBBeamwidthElv]
[Ant, 3dBBeamwidthAz]
[Radar, LocLon]
[Radar, LocLat]
[Target, RxPowerLevel]
[OneSec, 10^6MicroSec]
[Target, RxPowerLevel]
[Target, Range]
[Target, ReflectiveTime]

IsA
Relation [TxAnt, Ant]
[RxAnt, Ant]

LocLatTime
Relation [ANsss, xxxxN, Time1]
[ANyyy, xxxxN), Time2]

LocLonTime
Relation [ANsss, xxxxE, Time1]
[ANyyy, xxxxE, Time2]

RxNoiseFloor
Relation [ANsss, -110dBM]
[ANyyy, -100dBM]

ReflectiveTime
Relation [T1, xxxx]
[T2, xxxx]

CFARThreasholdLevel
Relation [ANsss, xxxx]
[ANyyy, xxxx]

Target
Relation [ANsss, T1]
[ANyyy, T2]

RxPower
Relation [T1, xxxx]
[T2, xxxx]

Figure 3. Relational DBMS Model of Facts and Causes

Semantic Nets

Originally semantic nets were developed for the purpose of modeling the English language (7), for a computer to understand. A method used by engineers and scientist to understand complex relationships, is to draw pictures. This is true in the DBMS world with regard to designing databases with the use of entity relationship diagrams. It is also true in knowledge representation with the use of similar graphs called semantic nets. Semantic nets are networks composed of nodes and arcs. Nodes represent facts and the arcs or edges represent relationships. For entity relationship diagrams the nodes represent entities and the arcs represent relationships. The nets also help in simplifying the deduction process. Consider the following partial semantic net of our radar example shown in figure 4.

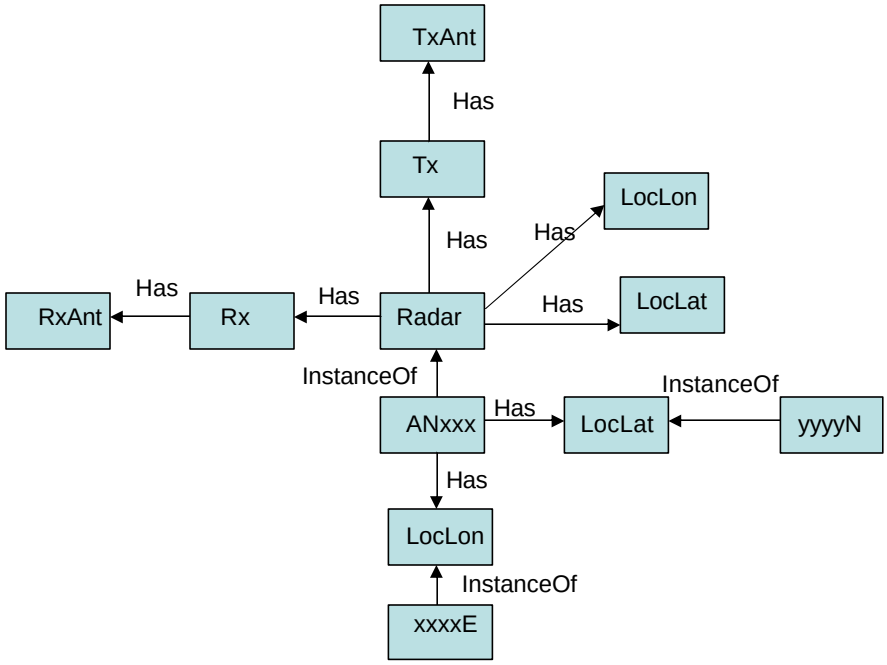

Figure 4. Example Radar Semantic Net

In a semantic net the relations similar to the predicate logic model are seen in our example as the Has relation. To designate an occurrence or instantiation of a fact we used the relation "InstanceOf". One can see from the semantic net that we can deduce facts not explicitly shown or directly connected. For example we can deduce that ANxxx is a radar and since it's a radar, it has both a Rx and a Tx, even though there is no direct nodal connections between ANxxx and Rx or ANxxx and the Tx node. This capability is called by some (10) as the inheritance property. From our simple semantic net a radar entity inherits the property that it has both a Rx and a Tx and that each of these entities has an antenna.

Implementing a semantic net in a computer however, would not be represented as a graphic, but would be broken down into tuples or relations. A tuple is an ordered set of values, e.g. attribute value pair within a DBMS or the attributes that compose a relation in a relational DBMS. For example the Has edge would be a relation and some of the occurrences of the relation would be [Tx, TxAnt], [Rx, RxAnt], [Radar, Tx] and [Radar, Rx]. One can see the similarity between the predicate logic and semantic nets once one tries to implement them in software or within a DBMS. These can also be written in a DBMS format as Has(Tx, TxAnt), Has(Rx, RxAnt) and Has(Radar, Tx) and Has(Radar, Rx) or more directly as TxHas(TxAnt), RxHas(RxAnt), RadarHas(Tx) and RadarHas(Rx). This last representation is identical to those relations shown in figure 2. How we build these relations or knowledge representation is designer dependent and there is no formal science that says there is only one way to represent knowledge whether one uses predicate logic, semantic nets or frames (see below). There are however, better representations than others as there are better ways to write software. The same is true in developing knowledge representations and of course their software instantiations.

Frames

Frames try to capture what is conjured up in one's mind when they think of a particular object such as a radar or transmitter or antenna. When we think of one of these objects we cluster in our minds a set of attributes that describe this object. In frame terminology each of these attributes are called slots and each of these slots contain one or more values. This concept was proposed by Marvin Minsky (10) early in the 1970s. If we try and develop a knowledge representation for our radar example we may come up with frames as shown in figure 5.

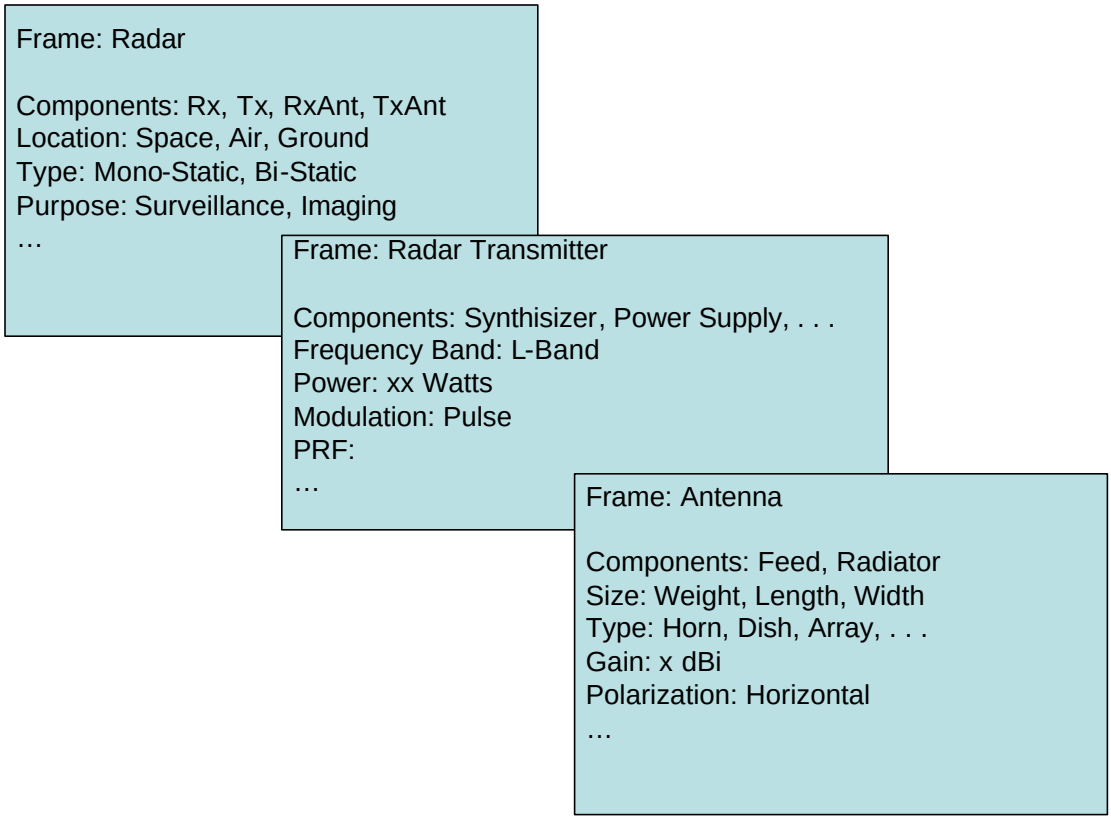

Figure 5. Frame Knowledge Representation Example

Inference Engines

The description and design of the knowledge base is the most important part of building a KB system along with choosing how one is going to describe it, i.e. predicate logic, semantic nets, or frames. This choice is obviously dependent upon the domain of interest, the purpose of the knowledge base, and of course the inference engine or inference engine tool that is best for building the solution. There has been primarily two basic methods used for building KBS. Back in the 1970 to approximately 1985 time frame engineers and scientists either built their solutions using Prolog or Lisp. Since that time people have been purchasing software tools that contain the inference engines and one only needs to define the knowledge base and the tools will help one build a knowledge base or expert system. In this paper we will provide some of the basic elements of both Prolog and Lisp so that the reader will have an appreciation of each.

Prolog

Prolog is an acronym for PROgramming in LOGic. It is (7) "a rule-based language built on top of a predicate-logic theorem prover". It was developed in France during the 1970s and was used heavily in Europe and Japan. It is also used in the USA but not as much in the early years as Lisp. Prolog is a language but also a theorem prover based upon predicate logic. Some of the information presented here was obtained from (11).

Programming in prolog requires one to describe facts and relationships about objects or entities. One also must describe rules about these objects and their relationships. Once these are defined then questions regarding these objects and relationships can be evaluated.

Facts are described similarly as shown in figure 2's radar facts and clauses. Consider the following hypothetical radar data where their position and their established tracks are presented. Note that we changed are naming convention from upper case to lower case for the first letter of every fact, clause, or occurrences. We did this to be consistent with prolog's identification of variables i.e. any name beginning with a capital letter is taken to be a variable in Prolog.

Radar Nomenclatures	**Locations and Times for Different Objects Based Upon CPI Time Emissions.**	**Established Tracks**
radarNom(aNxxx)	locLatTime(aNsss, xxxxN, time1)	target(aNsss, t1)
radarNom(aNyyy)	locLonTime(aNsss, xxxxE, time1)	target(aNyyy, t2)
radarNom(aNzzz)	locLatTime(aNyyy, xxxxN, time2)	
	locLonTime(aNyyy, xxxxE, time2)	
	locLatTime(t1, xxxxN, time3)	
	locLonTime(t1, xxxxE, time3)	
	locLatTime(t2, xxxxN, time4)	
	locLonTime(t2, xxxxE, time4)	
	locLatTime(aNxxx, xxxxN, time5)	
	locLonTime(aNxxx, xxxxE, time5)	

Table 1. Example Prolog Facts

Once facts are defined within prolog then one may ask questions of the fact base. For example: "Is t2 a target being tracked by aNyyy?" In prolog it would be written as:

```
?-target(aNyyy, t2).
```

If this were typed into prolog with the above facts in its knowledge base then it would respond with "yes". If the following question were asked:

```
?-target(aNyyy, t1).
```

The response from prolog would be "no". This process is very similar to asking a query to a DBMS where you would ask how many records have the values as stated for the relationship "target". The response from the DBMS would be the count of zero if the answer was no or the count of the number of records that contained the values of the attributes as stated in the query.

Prolog allows the use of variables. Suppose we wish to know the locations of all the radars when emitting their CPIs.

?- radarNom(X), locLatTime(X, Z, Y), locLonTime(X, W, Y).

The variable X is placed in the fact for which we want the system to find a value that will satisfy each part of the conjunctive goal. The commas between the three goals represents the logical AND. First prolog will search the radarNom facts to find the first one with a value, i.e. aNxxx. It will then instantiate the variable X with the value aNxxx. It has satisfied the first goal. It will then search the fact base for locLatTime(aNxxx, Z, Y). If and when it finds an occurrence then it will have satisfied the second goal. It then will search the fact base for locLonTime(aNxxx, W, time5). If it satisfies this goal it will return with a "yes". If not it will backtrack to the second goal and see if there is another occurrence of locLatTime(aNxxx, Z, Y) with a different value that Y may be instantiated to and begin the process again until either the question is answered with a "yes" or a "no". If the second goal cannot be achieved then it will proceed to the first goal and seek out the next radarNom(X) it can instantiate the X to and begin the process over again. It will answer with a "no" if it exercises all of the relevant facts and no correct response is found.

If we rewrote the question above as the following what would we be asking?

?- locLatTime(X, Z, Y), locLonTime(X, W, Y).

It would be asking for all those objects (i.e. radars or target/tracks) and their locations for all times that are recorded within the knowledge base. If we wanted to know if multiple radars are tracking the same targets then the radar's location of the targets at approximately the same time should be in approximately the same location. For illustrative purposes let's assume that all radars transmit at exactly the same time and their target locations contain no errors (note in reality we can compute the differences based upon radar system errors and build this into our logic). Therefore, two different radars are tracking the same target if two different target tracks are located at the same time and at the same location. Using prolog we can now search for potential common tracks between multiple radars.

Consider the following rule.

commonTracks(X,Y) :- target(A, X), target(E, Y), (E\=A), locLatTime(X, B, C), locLonTime(X, D, C), locLatTime(Y, B, C), locLonTime(Y, D, C).

The ":-" is read "If". The first three rules state that two different radars are tracking two different targets. The next four rules state that the two different targets/tracks should be at the same place at the same time. If these are true then we can say that radars A and E are tracking the same target. (Note E\=A states that A is not equal to E)

Lisp

Lisp is an acronym for LISt Processing. The language is one of the older computer languages and is used by some even today. In many circles, especially in the USA, in 1984 it was considered (12) "the linqua franca of artificial intelligence research". Many AI systems were built using Lisp. However, because of its basic processing of lists conventional computers at the time were not "fast enough" for AI researchers to demonstrate the power of their work. This led to companies like LMI to build special computers designed for processing the Lisp programming language.

The basic element of Lisp is its ability to process lists. Lists are a data structure that is a key element in modeling natural language processing and speech understanding. It is also a key element for creating theorem proving algorithms necessary for knowledge base and expert system reasoning systems which we will need to capitalize upon when we build an end-to-end KB radar system. It would be too cumbersome to discuss Lisp in any great depth, without describing its syntax as we have above with Prolog. Therefore we will provide a brief overview of lists and recursion, two main ingredients in building knowledge base systems, using Prolog for demonstration purposes. A basic understanding of Lisp can be obtained from (12). We will intersperse some of the LISP terms taken from (12) throughout some of the description provided below.

A list (11) "is an ordered sequence of elements that can have any length". The elements of a list can be any acceptable entity, i.e. an element, a variable, even another list. A list can be an empty list i.e. a list with no elements. A list has two elements the head of the list and the tail of the list. (In LISP the head is obtained by using the CAR function and the rest of the list is obtained using the CDR function.) The end of a list is called the tail and is set to the empty list when the list contains no elements and is written as [ ]. (In LISP it is called NIL.)

In list processing a common operation is to split a list into its head and tail. Prolog represents this as [X|Y]. For example if we had a list of facts about radar nomenclatures denoted as relationship n, then note the following:

n([radarNom(aNxxx), radarNom(aNyyy), radarNom(aNzzz)]).

If we then ask the question to find the value of the head and tail, it would be stated as:

?- n([X|Y])

This will set X = radarNom(aNxxx) and Y = [radarNom(aNyyy), radarNom(aNzzz)]. This type of processing is performed throughout Prolog and LISP programs where one needs to process one element of the list at a time. A method which performs this process is called recursion. To demonstrate this we define the member predicate in Prolog. In any list there is a relationship we can define as membership about any object and whether it appears in any particular list. We can write this as member(X,Y). This relationship is true if X is contained in a list named Y. If X is a member of Y then X is a member of the head or it is a member in the tail.

member(X, [X|_]), where the underscore or anonymous variable is used for the tail of the list to signify that the tail of the list is not used or is not important in this fact. This fact states that X is a member of a list if X is the same as the head of the list.

member(X, [_|Y]) :- member(X,Y). This rule states that if X is a member of a list if X is a member of the tail of the list. These two rules together define the membership predicate and are an example of recursion where the definition of the rule contains the rule.

Stated together the membership rule is:
member(X, [X|_])
member(X, [_|Y]) :- member(X,Y).

Note that if we wanted to know if an element occurs in a list we can state it in Prolog as:

?- member(radarNom(aNxxx), [radarNom(aNxxx), radarNom(aNyyy), radarNom(aNzzz)]).

The first part of the rule would be exercised and the answer would be ”yes”. If the question were changed to the following:

?- member(radarNom(aNzzz), [radarNom(aNxxx), radarNom(aNyyy), radarNom(aNzzz)]).

The first part of the rule would come back with “no” and then the second part of the rule would keep X as instantiated with radarNom(aNzzz) and instantiate Y with the tail of the previous list, i.e. [radarNom(aNyyy), radarNom(aNzzz)], thereby re-asking the original question as:

?- member(radarNom(aNzzz), [radarNom(aNyyy), radarNom(aNzzz)]).

Processing this question will result again with the first part of the rule returning ”no” and the second part of the rule would keep X as instantiated with radarNom(aNzzz) and instantiate Y with the tail of the previous list, i.e. [radarNom(aNzzz)], thereby re-asking the original question as:

?- member(radarNom(aNzzz), [radarNom(aNzzz)]).

Processing this question will yield a ”yes” from the first part of the rule. If however, this part failed because the last element in the list was not equal to the head of the list then the second part of the rule would come into play and the following question would be:

?- member(radarNom(aNxxx), [ ]).

If this occurs then the rule fails and a ”no” is returned. It should be noted that each time Prolog uses the second clause and the member relation is exercised again that the system keeps a different copy of the member relation. This is necessary so that the processing does not get confused with which variables are used within which instantiation of a clause.

Keeping track of the processing within Prolog and LISP based programs is important within the knowledge base and AI community. Many recursive algorithms are written and are important in solving complex problems. The opportunity of keeping this type of information in knowledge base systems is important for it is needed in explaining how different questions are answered. It can be used in diagnosing errors in programming but more importantly it is used by the system to explain to the user how it derived the solution it provided. In addition, as a system is utilized it also “learns” from its past experiences and can evolve over time.

Now that we spent some time describing how a KBS functions it should be noted that you really don’t need to be an expert at writing Prolog or LISP code in order to build a KBS. You do however need to know how to build a knowledge base and recognize the best methods of representing the knowledge base given the domain of interest and purpose of the KBS. With that in mind there are numerous expert system building shells or tools that one can obtain that will help you in building a KBS. These tools will help one acquire the knowledge and present it to the system. Some are built upon Prolog, some with LISP and others even in C code. One only needs to use a search engine such as Google (http://www.google.com/) and type in “expert system tool” or ‘knowledge base tool”, and numerous sites are presented that will point you to all kinds of tools that can help one build a KBS. These systems are usually easy to use and will allow one to build simple rules that can be used along with the facts provided. Some simple examples are:

If ReceiverTargetLevel(xxx) < ReceiverNoiseFloor(yyy)
Then ~Detect(Target)

If ReceiverTargetLevel(xxx) > RadarThreasholdLevel(yyy)
Then Detect(Target)

TargetRange = (300X10^6 Meters/Sec)X(TargetReflectiveTime(xx)/2

World Wide Web Consortium (W3C)

If one visits the www.w3c.org Internet site they will obtain a definition of who they are: "The World Wide Web Consortium (W3C) develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential. W3C is a forum for information, commerce, communication, and collective understanding." They, along with the Defense Advanced Research Project Agency's (DARPA) Agent Markup Language (DAML) program, are building the next generation Internet or the Semantic Web. The Semantic Web will allow one to develop Web pages that are written such that software can read and understand the contents of Web pages. Our current Web pages are developed for human consumption. They are not built for software to read and understand their contents. This is why when using search engines the responses are numerous. For example if one puts in the words "radar signal processing" then the response pages are those pages that contain one or more of these words in any order and in any place within the page. The next generation Web is being designed in a manner similar to a large knowledge base such that one can define ontologies for different interested domains, like radar or sensors in general. An ontology is best defined for our use by what motivated the development of ontologies for the Web. The following definition was taken from http://www-ksl.stanford.edu/kst/what-is-an-ontology.html and authored by Dr. Tom Gruber.

"An ontology is a specification of a conceptualization….What is important is what an ontology is *for*. … For pragmatic reasons, we choose to write an ontology as a set of definitions of formal vocabulary. Although this isn't the only way to specify a conceptualization, it has some nice properties for knowledge sharing among AI software (e.g., semantics independent of reader and context). Practically, an ontological commitment is an agreement to use a vocabulary (i.e., ask queries and make assertions) in a way that is consistent (but not complete) with respect to the theory specified by an ontology. We build agents that commit to ontologies. We design ontologies so we can share knowledge with and among these agents."

The goal of an ontology is to define those terms within a domain and reference other ontologies necessary for machine understanding. Using this, a software agent built to understand that ontology will be able to collate and cross reference content across resources, and draw conclusions from information found in more than one source. As an example one could define an ontology for sensors where one would define the tags necessary to describe a sensor. Each sensor would then describe itself in a document using the terms from that ontology and other higher-level ontologies and software would then be able to look at many sensors from many manufacturers and be able to collate the data from all the sensors and provide a coherent analysis. These higher level ontologies will come from prominent organizations. For example if the National Institute of Standards and Technology (NIST) writes an ontology defining the conversion of Hertz to Megahertz and frequency to wavelength, then an ontology about sensors need only refer to those specific classes within the NIST ontology. Sample ontologies can be found at DARPA's Agent Markup Language (DAML) web site.

The W3C's XML family of technologies is the standard for data exchange. XML is a formal mechanism for exchanging data between platforms and programs. XML is defined by the W3C:

Extensible Markup Language, abbreviated XML, describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them… XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure. [http://www.w3.org/TR/2004/REC-xml-20040204/]

XML is a formal Recommendation of the W3C. A W3C Recommendation implies that:

It has been reviewed by W3C Members and other interested parties, and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web. [http://www.w3.org/TR/2004/REC-xml-20040204/]

To build documents for representing sensors, we would use another standard, built upon XML, for its structure: RDF (Resource Description Format), a W3C Recommendation that provides a standard method for simple descriptions of accessible resources. The W3C describes RDF as:

The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web. It is particularly intended for representing metadata about Web resources, such as the title, author, and modification date of a Web page, copyright and licensing information about a Web document, or the availability schedule for some shared resource. However, by generalizing the concept of a "Web resource", RDF can also be used to represent information about things that can be identified on the Web, even when they cannot be directly retrieved on the Web. … RDF is intended for situations in which this information needs to be processed by applications, rather than being only displayed to people. RDF provides a common framework for expressing this information so it can be exchanged between applications without loss of meaning. [http://www.w3.org/TR/2004/REC-rdf-primer-20040210/]

Another level above RDF is OWL, the Web Ontology Language, a descendant of DARPA's DAML program's DAML+OIL (Ontology Inference Layer) ontology language. Descriptions of sensors would be written in the OWL format.

The OWL Web Ontology Language is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics. [http://www.w3.org/2001/sw/WebOnt/]

The W3C moved OWL first to a Candidate Recommendation as of 19 August 2003, and then the community gave final approval to OWL as a Recommendation on 10 February 2004. There are other formats that have been proposed for ontologies, but OWL's status as a W3C Recommendation is an important reason why it is provided herein.

The elements of a RDF document are triples: structures which consist of a subject, predicate and object. For example, the tag

<cml:INTEGER rdf:value= “3”>

is actually a triple with the following parts: “cml:INTEGER” is the subject (the cml represents the Uniform Resource Indicator (URI) of the ontology where the term INTEGER is defined for this domain); the “rdf:value” is the predicate (the term “value” is defined by the RDF ontology) and represents a property of the subject; and the object is the literal numeric value “3”. This is very similar to Prolog concepts, where a relationship such as “Target has range” would be represented as a triple where “Target” is the subject of the triple, “has” is the predicate (property) and “range” is the object (value). We could also represent those data as a relation in a relational database where the name of the relation is the predicate’s name and the attributes in the relation are the subject and the object.

In any valid RDF document, every structure can be represented as a triple, and its triples are a set of individual entities, without any intrinsic order, similar to the way a relational DBMS stores data in relations, relying on the value of the instance to provide a sorting entity for queries. In many cases of RDF, the order of the triples does not matter, but in the case of a sequential program for instance, order is a fundamental property of a program. Order can be preserved in the set of triples by creating other triples that explicitly define this order as a property in the RDF by the use of a subclass of the RDFS-defined container class called a sequence. The RDF ontology requires that a sequence have a property for each node in the structure that defines its relative position within that container. Each subject will have a property (i.e. predicate) that is its relative position within the representation of the program or function being defined.

The concept of an ontology is exactly what we need in our overall pursuit of having sensors operate in cooperation and eventually having sensor platforms operating autonomously as a robot. For them to operate cooperatively they must be able to communicate, share data and information, and understand each other and their environment. If we tried to do this with each sensor system building their own knowledge base with different knowledge base representations it would be difficult for them to communicate and understand each other. Each system would have to build software translators to understand each other. Each sensor system would have N-1 translators for a system with N sensors. This would be expensive to build, it would be processor intensive, and would generate a high maintenance cost over the life of the sensor systems.

Leveraging the approach and technology of the W3C will allow us to develop an ontology for sensors thereby having one knowledge base that can be understood by all new knowledge base sensor systems added to the overall domain including communications, radar, electro-optical, infrared, acoustic, etcetera. This approach will allow multiple sensors on one platform to inference and fuse data and information from all its sensors on board. It will also allow for this platform to share and fuse data and information between sensors on multiple platforms located nearby or miles away within a command center. The building of ontologies is going on today. They can easily be found on the Web and can be used to build and share information within the community and domain of interest. The approach we recommend and used (13) is to not build one’s own ontology from scratch but to leverage the object oriented feature of inheritance and reference the resource descriptive framework (RDF) (i.e. an instantiation of an ontology) of those ontologies that already exist and then add those additional facts and rules required for one’s own needs. For example if a respective organization has built an RDF describing facts and rules for a transmitter, a receiver, and an antenna then if their facts and rules meet your needs then they should be referenced in the radar ontology that one is building, rather than one building their own.

In this manner one needs only to refer to the ontology where they wish to use these rules and facts and they can add additional rules and facts as required.

Summary

This paper has provided a brief discussion of artificial intelligence and why it can play a major role in the next generation of radar systems. The important areas of AI that seem to fit well for radar signal processing are knowledge based processing and robotics. We have provided a brief overview of knowledge bases and the processes of building a knowledge base. Some of the most recent efforts in this field is being pursued in the Internet field for building intelligent agent software that can understand Web pages. We wish to leverage that technology and their success for building knowledge based radar systems. We are currently investigating this technology and have begun building a prototype ontology for radar systems. A lecture on the second day will have more examples of knowledge bases and their uses in building an end-to-end radar signal processing system.

Acknowledgements

The authors would like to recognize the efforts of the following people. We would like to thank Mr. Gerard Genello, Mr. William Baldygo, and Dr. Michael C. Wicks for providing the resources, encouragement, guidance, and the opportunity in the pursuit of our goals. We would also like to thank Mr. John Spina for chairing this lecture series and Mr. Christopher Capraro and Mr. Gerald Berdan of Capraro Technologies, Inc. for their help throughout. Thank you all.

References

[1] W. Baldygo, M. Wicks, R. Brown, P. Antonik, G. Capraro, and L. Hennington, “Artificial intelligence applications to constant false alarm rate (CFAR) processing”, Proceedings of the IEEE 1993 National Radar Conference, Boston, MA, April 1993.
[2] R. Senn, “Knowledge Base Applications To Adaptive Space-Time Processing”, Unpublished Final Report, July 1999.
[3] P. Antonik, H. Shuman, P. Li, W. Melvin, and M. Wicks, “Knowledge-Based Space-Time Adaptive Processing”, Proceedings of the IEEE 1997 National Radar Conference, Syracuse, NY, May 1997.
[4] Multi-Channel Airborne Radar Measurement (MCARM) Final Report, Volume 1 of 4, MCARM Flight Test, Contract F30602-92-C-0161, for Rome Laboratory/USAF, by Westinghouse Electronic Systems.
[5] G. T. Capraro, C. T. Capraro, and D. D. Weiner, “Knowledge Based Map Space Time Adaptive Processing (KBMapSTAP)”, Unpublished Final Report, March 2000.
[6] C. T. Capraro, G. T. Capraro, D. D. Weiner, and M. Wicks, “Knowledge Based Map Space Time Adaptive Processing (KBMapSTAP),” Proceedings of the 2001 International Conference on Imaging Science, Systems, and Technology, June 2001, Las Vegas, Nevada.
[7] E. Rich, “Artificial Intelligence”, New York, NY, McGraw-Hill, 1983
[8 A.Barr and E. A. Feigenbaum, “The Handbook of Artificial Intelligence, 3 vols.”, Los Altos, CA: William Kaufman, 1981-1982
[9] B. G. Buchanan and E. H. Shortliffe, “Rule-Based Expert Systems”, Reading, MA: Addison-Wesley, 1984.

[10] H. C. Mishkoff, “Understanding Artificial Intelligence”, Dallas, Texas: Texas Instruments Incorporated, 1985.
[11] W. F. Clocksin and C. S. Mellish, “Programming in Prolog”, Spriner-Verlag Berlin Heidelberg: Beltz, Offsetdruck, Hemsgach, 1981.
[12] D. S. Touretzky, “LISP A Gentle Intrduction to Symbolic Computation”, New York, NY: Harper & Row, 1984.
[13] G. B. Berdan, G. T. Capraro, J. Spina, and R. A. Liuzzi, “Building an Ontology for Computing Devices”, Proceedings of the International Conference Information and Knowledge Engineering, June 2003.
[14] M. C. Wicks, W. Baldygo, and R. D. Brown, “US Patent 5,499,030 Expert System Constant False Alarm Rate (CFAR) Processor”, filed March 18, 1994 issued March 12, 1996