Giving Devices the Ability to Exercise Reason

Thomas KEELEY
Compsim LLC
Brookfield, WI 53045, US

ABSTRACT

One of the capabilities that separates humans from computers has been the ability to exercise "reason / judgment". Computers and computerized devices have provided excellent platforms for following rules. Computer programs provide the scripts for processing the rules. The exercise of reason, however, is more of an image processing function than a function composed of a series of rules. The exercise of reason is more right brain than left brain. It involves the interpretation of information and balancing inter-related alternatives. This paper will discuss a new way to define and process information that will give devices the ability to exercise human-like reasoning and judgment. The paper will discuss the characteristics of a "dynamic graphical language" in the context of addressing judgment, since judgment is often required to adjust rules when operating in a dynamic environment. The paper will touch on architecture issues and how judgment is integrated with rule processing.

Keywords: Artificial Intelligence, Expert Systems, Reasoning, KEEL Technology, Dynamic Graphical Language

1. INTRODUCTION

The intent of this paper is to introduce a technical approach that can be used to extend the behavior of devices beyond conventional rule based systems to systems that can exercise reason and judgment. We will avoid any psychological discussions of how human beings exercise reason: explicit drivers, fight / flight drivers, specific stress related impacts to decision-making, etc. But we will lay out the capabilities of a technology called Knowledge Enhanced Electronic Logic[1] (or KEEL®) that can model and execute such behavior.

In this light, we are discussing a new form of mathematics and what one can do with it, not the specific solution to a particular problem.

2. REASON & JUDGMENT

Dr. Horst Rittel (UC Berkley) coined the concept of "wicked problems" in the 1970's with his definition of Issues Based Information System (IBIS)[1]. He contrasted wicked problems with tame problems. With tame problems, you can use formulas to calculate a "correct" answer. With wicked problems it would be impossible or inefficient to define a formula. With wicked problems you are looking for a "best" answer. One uses "judgment" and "reason" to address wicked problems.

There are two types of problems where reason and judgment can be utilized. 1. The selection of an option or options, which is accomplished by balancing alternatives. In some cases the alternatives are inter-related, so they cannot be addressed in isolation. For example, the selection of treatment options by a medical practitioner. 2. The allocation of resources, which can also be described as the balancing levels of control. This can be a continuous process. For example, driving a car requires continuous adjustment to steering, brakes, and accelerator in response to continuously changing road conditions.

A key factor in any discussion of reason and judgment is the concept of "adaptation". Adaptive control systems are common in batch processing and continuous processing systems, such as food processing, mixing, rolling mills, distilling. These systems commonly use PID (proportional, integral, derivative) control loops for very specific applications. In these industrial automation systems, the designers have very specific measurable objectives and can justify the significant effort to tune the PID control loops to perform exactly the way they want.

Human reasoning and judgment is often required to address much more complex problems. Humans are continuously tasked with addressing problems with many inter-related inputs and outputs. They balance their decisions and actions by continuously evaluating risks and rewards.

3. HUMAN EXPERTS

Human "experts" exert reason and judgment as they dispense their experience. Through training and experience they "learn" how to interpret information and what to do when they observe information in their domain. During the learning process they go through a trial and error process. This allows them to appropriately interpret the value of different information items. This valuing of information has been likened to an image processing function. Discussions of left-brain / right-brain reasoning suggest that the right brain handles impressions or feelings that might be termed value judgments.[2] The concept that the human brain separates rules from reason is the basis for the model discussed in this paper.

If a human expert "just" followed a set of rules, then computers would have already exhibited the capabilities of humans for many years. It is this reasoning and interpretation of information that has been the differentiator.

When we want to package human-like expertise in devices or software applications, we do not want these devices to have to develop the expertise on their own. There may be some cases

[1] KEEL Technology is a proprietary technology covered by a series of granted and pending patents that is licensable from Compsim LLC.

where it may be acceptable for devices to learn by trial and error, but when we potentially mass produce these devices we would suggest humans must remain in control. We want all of these devices to perform in the same way. We also want them to be completely explainable and auditable, in case they need to be fixed or adjusted.

4. CHARACTERISTICS OF JUDGMENT/REASONING

Judgment requires the fusion of multiple data sources. One could say that judgment and reason are all information fusion problems.

Any data source can impact different parts of a problem domain with different levels of importance. For example: a human's body temperature could have different impacts to different bodily functions.

Data sources must be measurable. In some cases it may be more important to understand how information is perceived (trust, fear, frustration…). In these cases, other inputs might be used to qualify specific data sources if one desires a "reasonable" result. In this manner, judgment itself evolves as information becomes more or less validated.

Some data sources are only used to validate or invalidate other data sources, or to eventually establish a value of another data source.

In many cases, judgment has a time or distance value. Tactical decisions and actions are often real-time activities, or just need to happen "now". Strategic decisions often have a broader view and take into consideration future desires or expectations. Both tactical and strategic decisions can benefit from the application of judgment.

5. DEMAND FOR PACKAGED EXPERTISE

There are several reasons that one might want to package human-like expertise into devices or software applications:

Avoid human error
Humans make errors because of lack of attention, failure to perceive or recognize a situation, limited short term memory or the inability to handle complex situations, and poor judgment. In many cases, when humans make errors they cannot explain why they made the errors. This is especially true when the errors are errors in judgment. To support this demand, it is mandatory that judgmental decisions and actions are explicitly auditable when they are packaged in an "expert system".

Autonomous devices
There is a demand for more devices that can take on more complex responsibilities. In some cases this is to keep humans out of harm's way while still performing complex tasks (military, disaster recovery). In other cases the objective is to replicate human expertise (security, health care of the aged). In still other cases it may be for size reasons (in-vivo medical equipment, robotic equipment required to operate in small spaces, light weight UAVs). These devices cannot perform the duties that are expected of them without the ability to exercise human-like judgment and reasoning. [3]

Human support
The market for business intelligence services is growing rapidly. Businesses and organizations that can make better decisions faster will gain market share and be more competitive. Augmenting existing systems with more intelligence (reasoning and judgment) will increase their flexibility and value. [4]

6. INTRODUCTION TO KEEL® TECHNOLOGY[5]

We call KEEL (Knowledge Enhanced Electronic Logic) a technology, because it is a new way to process information. It is supported with a set of tools, including a "dynamic graphical language" that is used to define the policies that are executed in KEEL cognitive engines. In this context a technology is a methodology to solve a problem, and a tool is something used to facilitate the implementation of the technology. The KEEL "Engine" is the result of deploying the technology to provide a specific service.

Pros and Cons
In order to introduce the technology, one needs to think in terms of pros and cons. This is a common process used when evaluating decisions and actions. The concept was utilized by Dr. Rittel in his IBIS definition [1]. He used the terms "Issues", "Positions" and "Arguments. One responds to "Issues" with some number of "Positions" (or options). One argues the validity of "Positions" with "Supporting" and "Objecting" Arguments (pros and cons). We retain the terminology of Positions, Supporting Arguments, and Objecting Arguments in our definition of KEEL Technology.

The fundamental KEEL algorithm for accumulating the pros and cons of any position uses normalized values (between 0 and 100) for each supporting and objecting argument. Each position has a "potential" value that is an indication of its importance with a maximum value of 100. Supporting arguments are accumulated first. The maximum support that can be accumulated is 100 (absolute support). Objecting arguments detract from accumulated support. The maximum rejection is 100 (absolute rejection). Supporting arguments cannot accumulate more then the position "potential" value. Any completely blocking objecting argument can completely block a position. This accumulation methodology has the following characteristics. A position with no support is the same as a position with any amount of support, but is completely blocked, because a single objecting argument indicates impossibility (absolute objecting argument). The resultant accumulation will be somewhere between 0 and the position's "potential" value. The potential (importance) value is significant when integrating multiple information items in a broader problem domain.

Complexity
Dr. Rittel focused on tree structured problems, where one starts out with a top level "Issue", that is addressed with "Positions", that are supported or rejected with "Arguments", that can spawn new "Issues", that can be addressed with "Positions"… This works where humans have the opportunity to work on one problem at a time. When we began applying this kind of reasoning into autonomous devices we realized that these

devices and applications may not have the luxury of working on well-structured (decision tree structured) problems. We observed that these devices need to solve problems that have webs of inter-related data items. An autonomous device cannot address problems in isolation as it balances (sometimes conflicting) goals.

Graphical Context

Given that reasoning and judgment are more image processing functions than mathematical rules, we decided that, in many cases, numeric values are secondary to graphical values. Numbers can still exist, behind the scene, but humans assigning judgmental values to data items can be more effective using graphical techniques. When they compare multiple values they can look at graphical items and see (visually) how they compare without translating to and from specific numeric values. It is the right-brain interpreting the relationships. The concept of "significance" can be observed by the "size" of a bar in a bar chart. It is easy to see that one bar is larger than another.

Mathematics and Language

"Mathematics" has been defined as a way to describe measurement.

One might suggest that a characteristic of most mathematical representations is that mathematics has been effectively "written" on a plane surface with a "number system" and a set of "symbols". Mathematicians have developed techniques to describe non-linear, dynamic functional relationships by writing formulas using "paper and pencil".

Representing "mathematics" on a computer screen has primarily been accomplished to present formulas on a new media for display purposes.

Another characteristic of mathematics is that it needs context to have value. 1+2=3 has little value, when one doesn't know what 1 represents, what 2 represents, and what 3 represents.

Judgment has more to do with "enough" or "too much" or "not enough". Fuzzy logic was developed to address "linguistic uncertainty" which is part of the problem, especially when dealing with the description of problems in human terms. But "devices" do not normally communicate in human language terms.

If two "devices" need to communicate, and both understand the context of the problem and share a particular protocol, then they can effectively communicate just by exchanging structured data. There is no need to translate between human terms to numbers and back to human terms in this case, by either device.

With KEEL, we use graphics to display "measurement". In this case, we use graphics to show the measurement of the importance of information and the measurement of the impact of supporting and objecting arguments. We use wires between connection points define functionality, rather than symbols. The dynamic capabilities of the computer screen are used in place of "pencil and paper" to describe dynamic behavior. The ability to interact with the functional relationships is key to the development of KEEL models.

Targeting Devices

The KEEL "dynamic graphical language" was developed to address dynamic, non-linear, inter-related, multi-dimensional problem sets, without resorting to complex mathematics. It was derived from a model of how humans integrate supporting and objecting factors when making judgmental decisions. The KEEL dynamic graphical language evolved as web-structured problems were addressed, while taking advantage of dynamically available graphics on a computer screen.

Graphical Computer Languages

Graphical computer languages have been around for a long time. A common trait of these languages has been to suggest data "flow". Boxes indicate collections of functionality and wires show how data moves from box to box. These languages map to the structure of computers with sequential processing of instructions. As these graphical languages map closer to the instruction set of the computer it is common to see graphical elements that equate to IF | THEN | ELSE logic of the hardware.

The KEEL Dynamic Graphical Language

The KEEL "dynamic graphical language", unlike data flow graphical languages, has no concept of data flow. It more closely equates to a "formula". It provides a method of describing functional relationships graphically. While KEEL Engines cannot ADD two numbers together (i.e. 1+2=3, they can ACCUMULATE the impact of an unlimited number of supporting and blocking inputs. The KEEL "dynamic graphical language" is used to create KEEL Engines that can be deployed into conventional programming structures. KEEL Engines are functions or class methods that are integrated into conventional programming languages. They are processed during what we term a "cognitive cycle".

In the broader overall system context, conventional programming languages perform the general processing of the system; the left-brain, scripted part of the system. When appropriate, the judgmental portions (implemented as the KEEL Engines) are called. The judgmental processing could be triggered by an event; periodically scheduled, polled, or run continuously as a separate task. However it is triggered, a snapshot of real-world information is supplied to the KEEL Engine at the beginning of the cognitive cycle. Within the cognitive cycle it is iteratively processed until complete. The results are then available to the calling application.

A KEEL Engine has a very small memory footprint. It is usually smaller than 3K words in size, no matter how large the problem set. The problem structure is defined as structured data (tables). The size of the structured data area is determined by the complexity of the problem domain (the number of data items being processed and the number of functional relationships defined).

Defining Policies

The process of defining policies for the execution of judgment and reasoning is somewhat different than writing formulas using conventional paper and pencil techniques. One might say that writing formulas is accomplished by defining an answer and explaining how to arrive at that answer. (X = A + B + C) Policy definition is accomplished by (1) identifying the potential decisions and actions, (2) defining the information items that contribute to the decisions and actions, and (3) describing how the information items are inter-related. This is done in the KEEL language by using graphical icons to represent the outputs and the inputs and then using wires between connection points to describe functionality. Using the dynamic characteristic of the language, one then observes how the

system will respond as inputs are stimulated. Since one is often dealing with complex, non-linear relationships the developer is "thinking in curves". The developer is considering how different pieces of information are interacting. This is a subtle difference in the mindset of the domain expert that is developing the policy from a more conventional mathematical approach. Since the modeling can be done completely without resorting to complex (conventional) mathematics, the models can often be created directly by the domain expert; not the mathematician or the software engineer.

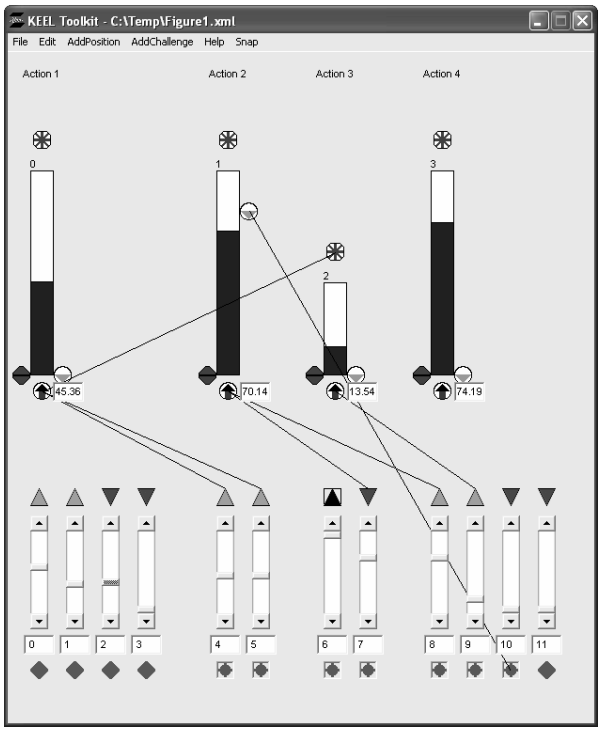

Figure 1.

The KEEL Language Structure

Figure 1 shows the KEEL dynamic graphical language "source code". The four vertical bars in the top half of the window are termed "Positions" and represent outputs. The importance of each Position is shown graphically by its size, and numerically by holding the mouse over the icon at the top of the Position. The vertical scroll bars at the bottom half of the window are termed "Supporting Arguments" and "Objecting Arguments" and represent inputs that drive or block the positions. A green triangle above a vertical scroll bar pointing up indicates a "Supporting Argument". A red triangle above a vertical scroll bar pointing down indicates an "Objecting Argument". The dark area (blue) in the vertical bars (Positions) at the top shows the results of the accumulation of the driving and blocking signals associated with each position. This is called the "Modified Value" in the language. The numeric value for the Modified Value is shown in the small window to the lower right of each position.

Wires define functional relationships between information items (inputs and outputs). More specifically, connection points acting as the source of the wire and connection points acting as the sink (end) of the wire together define specific functional relationships. The most common "source connection point" is represented by the (blue) arrow in a circle below each position bar. This is the Modified Value connection point. Wiring this point to a sink connection point defines a specific functional relationship. Some of the more common functional relationships driven from the Modified Value connection point (expressed in human terms) are "controls the importance of", "contributes to", "controls the threshold position of". Expressed in more stand-alone terms, the accumulation of the arguments of one position can control the importance of another (or other) position(s). The accumulation of the arguments of one position can contribute to the inputs to another (or other) position(s). The accumulation of the arguments of one position can control the position of a threshold on another (or other) position(s). Thresholds (as indicated graphically by the small triangle in a circle to the right of the Position bars) are used to detect Modified Values above or below the Threshold point. Wires from the Threshold can be used to turn on / off Arguments (inputs) to other Positions.

Programming is accomplished by dropping Positions on the screen and assigning Supporting and Objecting inputs that will drive them. As soon as they are dropped on the screen, they are active. By manipulating the inputs represented by the scroll bars, the entire model is updated. This is why we call this a "dynamic" graphical language. Using common drag and drop techniques, the information items are wired together creating the desired functional relationships. As soon as they are wired on the screen, the entire model is updated. Again stimulating any of the inputs causes all of the functional relationships to be exercised.

Other icons (connection points), not described in this document, allow sub-ranges to be defined and manipulated within the scale of the Position importance. This allows complex curves to be created from multiple curve segments. These features allow the definition of models with time and distance characteristics. For example, there may be optimal instants to make decisions. It is easy to integrate time utility functions (TUFs) into the decision making model. Figure 2 below shows an example of one TUF with characteristics that can be modified in real-time. One example of this is a UAV making a time and distance decision about the optimal instant to take a picture while balancing risk and reward.

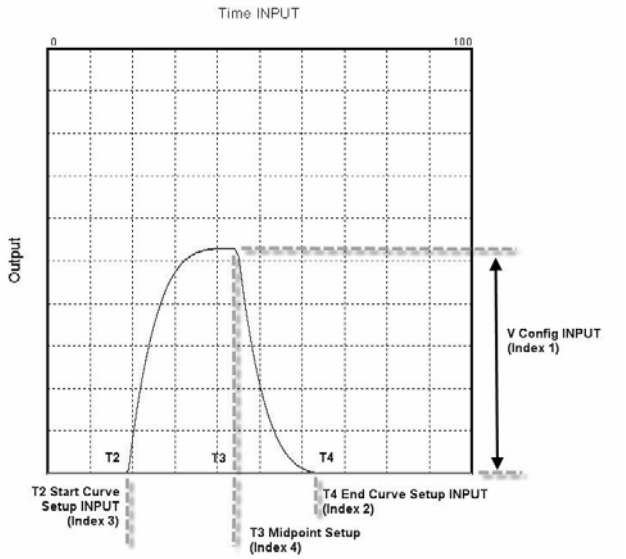

Figure 2.

By allowing external factors to control the shape of the curves very complex relationships can be defined. From a judgmental aspect, this means that we can create policies that can adapt to change.

2-D and 3-D graphs allow the user to incrementally adjust inputs and observe the continuous outputs. Figure 3 shows the impact of stepping two Arguments through their range and observing the impact on two of the Positions in the abstract problem space defined in Figure 1.

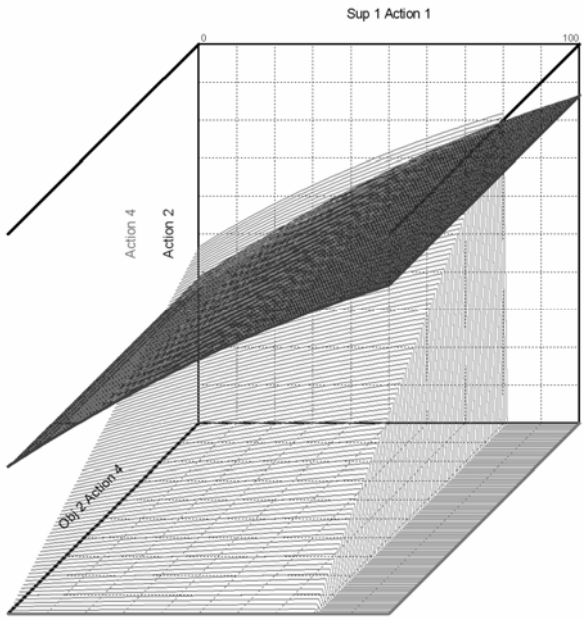

Figure 3.

Satisfying Complexity
Complex systems have a large number of inputs and outputs. Different "views" of a design can be exposed to simplify what is displayed on the screen at any one time. Features like "Child View" allows the selection of a particular Position in order to display all parts of the model that are impacted by that Position. "Parent View" allows the selection of a Position in order to display all of the other Positions that impact it. Custom views are also available.

To accelerate developments of complex models, the KEEL Toolkit provides a library of static and dynamic structures that can be merged into existing designs. The KEEL structure that defines the curve shown in Figure 2 is an example. All structures are defined with the simple components described above. All functional relationships are visible.

Another common practice, when addressing any complex problem, is to decompose it into smaller, more manageable sub-sets. This is supported with a Function Block Diagram (FBD) configuration tool.

The FBD application allows each KEEL Engine to be exposed as a box representing a function block with inputs on the left side and outputs on the right. Good I/O naming conventions allow all connections between KEEL Engines to be created automatically. The primary value of this tool is to eliminate coding errors by automatically generating the conventional code (C, C++, Java…) to move data values between engines and control the processing order for the scripted (left-brain) portion of the design.

7. A REASONING ARCHITECTURE

We have already discussed separating the rule sets (left-brain) from the interpretive, judgmental (right-brain) portions. Additionally, there are reasons to segment the judgmental functions into different levels of service. [6]

The scripted left-brain functionality includes the operating system and conventional run time rule sets that are triggered to run in different situations. The right-brain cognitive segments are triggered to interpret new information (situations where judgment is required).

When one considers an autonomous device, there will probably be (at least) two levels where judgment might be required. The first level will be used in the pursuit of specific goals that are actively being pursued. This is likely to be continuous, or at least tightly coupled with an activity. In this case, judgment is causing the right-brain scripted rules to adapt. An example of this is an autonomous aircraft attempting to avoid another aircraft in a shared airspace. The lower level(s) will be interpreting outside influences. This is implemented as a stack of judgmental observations. In a human, this might be a subconscious observation of surrounding activity that, at any instant, may not have any relationship with an action that is being performed. An example of this might be recognizing a dog near the side of a highway. In these cases, this stack or queue of observations is periodically updated because of a change of status (the dog jumps into the road). Judgment again is used to determine the significance of the action and what should be done about it. Judgment is used to select a course of action from a set of options. In some cases it can disrupt the original run-time goal and replace it with another. An example of this would be a UAV encountering a new target of opportunity. In this case a single KEEL engine might be used to process multiple secondary observations.

Interpretation of situations is likely to include a hierarchy of judgmental (information fusion) engines operating asynchronously. At the lowest level, information from multiple sensors will be integrated to form information items. At this level "judgmental functions" can be used to integrate features supporting and rejecting different observations. As they are transferred up the hierarchy they can carry a confidence factor. When those observations are integrated with other observations the confidence factors can be used to control the importance of the integrated information items. The result is a completely traceable / auditable, judgmental model.

8. REQUIREMENTS

A number of requirements drove the development of KEEL Technology. This section identifies the requirements and describes the response.

- A methodology must be provided that allows a domain expert to define the policy with sufficient granularity so

that it can be exactly translated into a form that can be explicitly executed by a device or software application.

The functionality described with the KEEL dynamic graphical language is explicit. The resulting functional relationships are executed as if they were conventional mathematical formulas. The functionality can be traced by observing values and relationships. Specific values can be viewed and graphed. A built in dashboard allows the domain expert to manipulate inputs as floating point numbers, if needed.

- The methodology for describing the policy must support the efficient development of policies when addressing complex, non-linear scenarios.

  The dynamic graphical language allows models to be created and tested by the domain expert before handing the design to a software engineer for integration in a complete system. By allowing the model to be tested during development, the time to market is reduced. By automatically generating the code (C, C#, C++, VB, Java, Flash, PLC Structured Text…) coding errors are avoided.

- The execution engine for the device or software application that will execute the policies must be suitable for embedded real-time operation.

  KEEL Engines are small, high performance functions. Two processing approaches are provided: One that is optimized for size and another that is optimized for speed. Capabilities of the language that are not utilized are optimized out of the conventional source code. KEEL Engines can also be implemented as analog circuits when even higher performance is required

.

- The methodology must be completely understandable so it can be efficiently tested before deployment.

  The ability to visually trace the impact of different variables allows one to see how complex judgmental reasoning situations are resolved. 2-D and 3-D graphs support the analysis. The development environment can also be used to animate external applications during the development environment to facilitate the analysis.

- Device or software application performance needs to be audited after deployment.

  Capabilities are built into the KEEL Toolkit that allow the language to be animated by models that have been deployed into devices or simulations. Black box information recording techniques can be used to audit judgmental performance in an off-line mode when exact details need to be reviewed. One can watch a rendering of the actual devices interpreting information.

- The efficiency of the entire policy life cycle must be considered (design, test, deploy, audit, extend).

  The dynamic graphical language supports the rapid development, test and deployment. The ability to animate the language from external sources supports the auditing and reverse engineering of specific situations. Extending existing designs can be accomplished by just dropping new positions and arguments into the design and wiring them into the system. System engineering features are also integrated into the language, which allow extended designs to be integrated into broader systems with little impact.

- The methodology must be architecture independent so it can be deployed on a variety of platforms and in a variety of situations.

  KEEL Engines are platform and architecture independent. The same cognitive model can be deployed in a variety of situations without re-engineering. The system engineer is responsible for the system architecture.

9. SUMMARY

Adding the ability for devices to exercise human-like reasoning will be mandatory to achieve the performance goals expected of the devices in the near future. Judgment and reasoning are information interpretation / information fusion problems that can be displayed and manipulated with graphical techniques characterized by the KEEL dynamic graphical language. The ability to visualize the importance of information and functional relationships is a significant advantage, as is the ability to interact with the model as it is being developed.

Since we are asking devices to take on more critical duties, it is absolutely mandatory that they be auditable. The KEEL graphical language allows policy-based decisions and actions to be visually traced through the models so one can see exactly what drove the specific decision or action.

KEEL technology is an “expert system” technology driven by the understanding of humans and completely under their control.

10. REFERENCES

[1] Rittel, Horst and Melvin Webber. “Dilemmas in general Theory of Planning” **Report**, University of California, Berkeley 1973

[2] McCrone, John, “’Right Brain’ or ‘Left Brain’ Myth Or Reality?” **The New Scientist**; http://www.rense.com/general2/rb.htm

[3] Clark, Admiral Vern; “Sea Power 21” **Proceedings**, October 2002; http://www.navy.mil/navydata/cno/proceedings.html

[4] Andrus, D. Calvin “The Wiki and the Blog – Toward a Complex Adaptive Intelligence Community”, https://www.cia.gov/csi/studies/vol49no3/html_files/Wik_and_%20Blog_7.htm

[5] Compsim paper; “Knowledge Enhanced Electronic Logic for Embedded Intelligence” http://www.compsim.com/papers/About_KEEL.pdf

[6] Albus, James S. “A Reference Model Architecture for Intelligent Unmanned Ground Vehicles”, **Proceedings of the SPIE 16th Annual International Symposium on Aerospace/Defense Sensing, Simulation and Controls**, Orlando, FL, April 1-5, 2002