

Application of KEEL Technology in the Area of Computational Mathematics Whitepaper

**Tom Keeley
Compsim LLC
(262) 797-0418**

tmkeeley@compsim.com

Abstract:

The Air Force has indicated interest in innovative methods and algorithms that improve modeling and simulation capabilities to enable understanding, prediction and control of complex physical phenomena. Compsim has invented a technology called “Knowledge Enhanced Electronic Logic” a.k.a. KEEL[®] Technology that allows a designer to capture complex situations with a dynamic graphical language that can be packaged as a KEEL Engine in conventional source code language like Java, or C. Unlike conventional processing of data, however, all information items are processed during a “cognitive cycle” where the system combines the various non-linear information items that “change in importance dynamically” and “balance” to obtain the “one or more solutions”. An example of the type of problem would be to “explain” how an expert driver drives a car around a corner, constantly balancing speed, road surface conditions, on-coming traffic, pedestrian traffic, comfort, detection of sliding...where the result would be control of speed, braking, and turning radius. It would be difficult to write the formula for this activity using conventional “coded algorithms”; however, this can be developed with relative ease using the KEEL graphical language. The intent of this paper is to expose the reader to the fundamentals of KEEL Technology and to the types of problems it can address.

History of KEEL Technology

KEEL Technology evolved from Compsim’s patented technology for addressing business and organizational problems that were termed “wicked” problems by Dr. Horst Rittel (UC Berkley) in the 1970’s. His focus was on city planning, where he suggested it was difficult or impossible to write a formula to run a city. He suggested the best one could hope for was a “best” solution to the problem, not a “correct” solution. His concept called Issues Based Information System (IBIS) focused on collecting positions (or options) as well as arguments (pro and con). He suggested that if one collected and organized all the appropriate information about a problem, one should be able to make the best decision or set of decisions. Compsim studied the data and patented a mechanism for evaluating these complex situations and creating explainable decisions.

Our background at Compsim is in the area of real-time control systems. We recognized that our model for human reasoning could be applied to those same real-time control systems. We realized that providing computer systems with the ability to exercise reasoning and make judgmental, subjective decisions, could transform the computing paradigm. KEEL Technology evolved from these basic concepts.

Copyright 2004 Compsim LLC
Patent Pending KEEL[®] Technology

Characteristics of Complex Problems

When we focused our technology on business and organizational problems we were working on relatively structured situations. A “decision tree” model was used. When we began applying KEEL Technology to embedded device applications and complex software problems, we realized that we were dealing with “webs of relationships” (not simple tree structures).

We also realized that intelligent “devices” would have to address dynamic situations and continually adapt to real-time circumstances. This differed from most business and organizational situations where information is static by comparison: acquire a business, develop a new product, outsource a function... where there is often time allocated to collect and evaluate information before making a decision. In this vein, KEEL was developed to handle continuously changing dynamic data.

Fundamental to KEEL Technology is the “importance (or significance) of information”. In KEEL, the importance of an information item is its maximum value. In complex systems, this importance can be fixed, or it can change dynamically as the environment changes.

Intelligent devices have to address complex time and space problems as they plan strategy (future) or react to real-time events (now).

In an information “web”, a single piece of information can have different impacts on different parts of the problem domain.

The impact of one piece of information on another might be binary, linear or non-linear following any “curve” relationship.

In an information “web”, one piece of information may control the importance of another or others.

Solutions to complex problems might be to select the best option, or might be to set some number of binary or relative valued outputs.

Inputs to complex problems, like the outputs, are analog as well as binary.

Solving Complex Problems with Conventional Formulas

We suggest that it is difficult to write formulas for complex, multi-variable, non-linear systems that control multiple non-linear outputs. In some cases the domain expert is a different person from the “coder” or formula developer. It may be difficult to insure that to correct information is modeled. Also, complex “formulas”, written using conventional programming languages, are often “brittle” (break easily) and subject to coding errors. Alternative solutions to complex problem, such as artificial neural nets (ANN), might be

used. This technique uses trained patterns that are matched with real world data. This may be the best solution when the problem is not understood well enough to write the formula. ANN based solutions, however, suffers from a high cost of the pattern training. They also create solutions that cannot be audited. When neural net based solutions make really wrong decisions, catastrophic results can occur. Some complex solutions demand auditable solutions so they can be corrected if necessary.

Using a Dynamic Graphical Language to Describe Complex Systems

Using the KEEL Graphical Language to describe complex systems utilizes a completely new paradigm, compared to one that focuses on writing a formula.

Using the KEEL Graphical Language, the domain expert describes information items with graphical representations of their importance. Information items are wired together to show how one information item impacts others. Supporting and Blocking Drivers are accumulated to provide answers to “sub problems”.

Using scroll bars to indicate the level of support or the level of blocking to individual decisions, the user can simulate inputs to the system while it is being developed and “see the system think” (or balance) as the results of individual decisions propagate through the system until it stabilizes. We call this “stabilization phase” a “cognitive cycle”.

The design is constantly being reviewed by the development environment to insure that unstable situations are not incorporated. An example of this would be to have a circular reference in a design that could potentially get into a never ending loop.

Because the graphical language is “explicit”, there is never a situation that cannot be resolved. This means that any given set of analog inputs will resolve to a fixed set of analog outputs. By tracing the wires between information items, the user can see exactly why any decision or action is taken.

A New Development Mindset

In conventional programming languages, one considers how to transform data. This usually involves deciding how to combine information in some kind of sequential process to determine an answer.

With KEEL, the model is different. It is commonly looked at from a higher level. First the designer identifies the outputs (or what controls might the system provide). The next step is to define the inputs (or what pieces of information contribute to the outputs). Finally, the user identifies how the inputs relate to the outputs. This may require that intermediate decisions are used to combine inputs to establish intermediate values. This last step requires the user to consider the importance of combined data items in non-linear situations. The user “thinks in curves”. The user wires relationships together and tests the system and observes the results. Using just a few basic constructs, very complex relationships can be modeled and observed. The result of this modeling concept is an

“engine” that can be scheduled to run periodically, on change of state, or continuously, where inputs are monitored and acted upon according to the performance requirements of the system.

KEEL Technology works with normalized data (all data items are normalized to values between 0 and 100). A built in “dashboard” allows the user / domain expert to work with data in real-world scales. A number of other tools are provided to the user to assist in issues of importance to the system integrator. When the design is complete and ready for deployment, it can be translated to conventional source code (For example: C, Java, Microsoft C#, Macromedia Flash Action Script, Microsoft Visual Basic, etc) and a series of data tables. This source code becomes the design of the KEEL Engine (encapsulation of the functional cognitive engine). The system *design* is maintained as table data. The one or two functions or methods (depending on the language) process the table data to obtain results that are returned as table data. The only way to understand the processing model is to return to the graphical language where the design can be visualized.

Integrating KEEL Technology with Existing Systems

While KEEL Technology can be used directly to address non-linear system design and simulation problems, it can also work with other technical solutions. A systems engineer has the responsibility to choose the right techniques for each problem. Because KEEL Technology, when deployed as KEEL Engines, has a very simple API, it can easily be integrated into almost any system configuration. So, just like a simple formula processed within a subroutine or class method, a KEEL engine can be integrated into almost any architecture.

Summary

KEEL Technology can be used to package solutions to complex non-linear multi-variable systems. The dynamic graphical language allows these complex systems to be modeled rapidly in a manner that allows the systems to be audited for their performance. One example can be viewed on Compsim’s web site (<http://www.compsim.com>) where 2 UAVs (each with their own KEEL Engine) maneuver over a simulated map. Each UAV is hunting for a target. Each UAV has its own policy engine (KEEL Engine) that balances its mission drivers against risk. Compsim has another version of this same demonstration where one UAV publishes its view of its environment. This published data can then be used to drive a copy of the KEEL Toolkit in a manner that allows the user to “see the UAV” interpret its environment while it is in simulated flight. This demonstration can be obtained from Compsim. In another version, external intelligence is integrated into the UAV’s decision-making model, allowing mission control to change policy in real time. Creating this model using conventional programming languages would be extremely difficult.

Images Introducing a Graphical Language

The following figures attempt to give the reader some idea of how the graphical language works. Unfortunately, because this text is a static document, the user cannot see the dynamic nature of the language. Given access to the KEEL Toolkit, the user loads outputs and inputs on the screen, establishes their importance, and wires relationships. All the time during the development process, the Toolkit is accepting inputs from the user (through the scroll bars) to stimulate the system so the model can be tested. The user is also given the opportunity to graph relationships to assist in the visualization process. Because the graphical language can also be packaged as various conventional source code languages, the KEEL Engines can be used in devices, simulators, training systems and demonstrated on the web without manually recoding the KEEL algorithm.

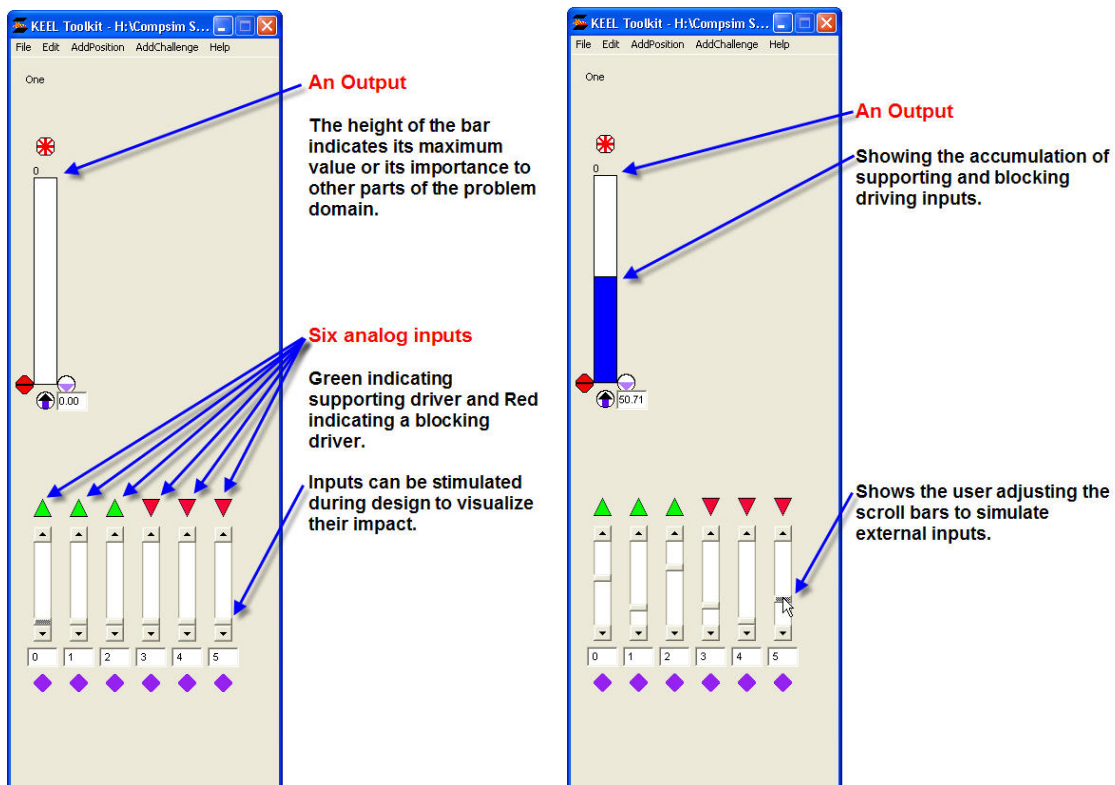


Figure 1

A single output driven by 6 inputs showing the user manipulating scroll bars to simulate inputs

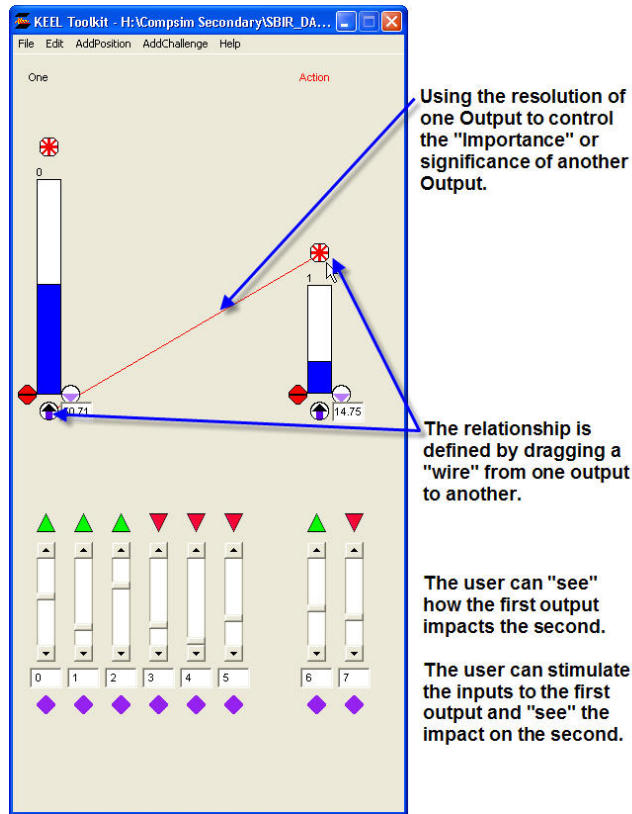
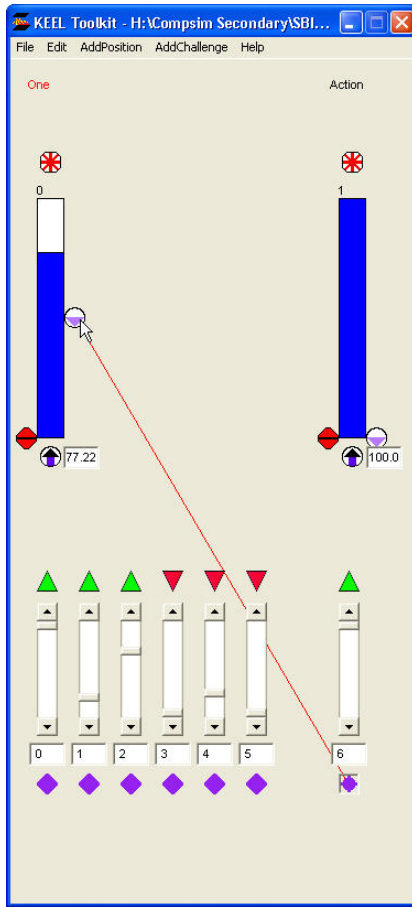


Figure 2

Shows the results of the user wiring one output (representing a decision or action) to control the importance of another.

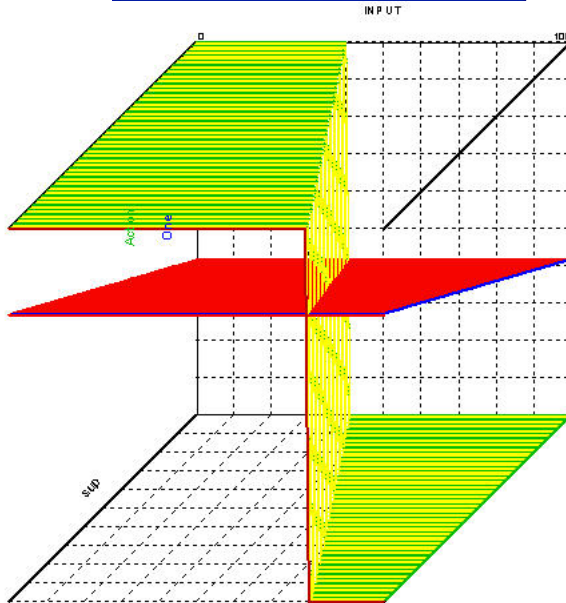


A Threshold related to an output can be used to turn on and off logic.

In this case the Threshold has been "wired" to the input of the second Output.

When the first Output is resolved to a value higher than the Threshold, the input to the second Output is driven from 0 to 100.

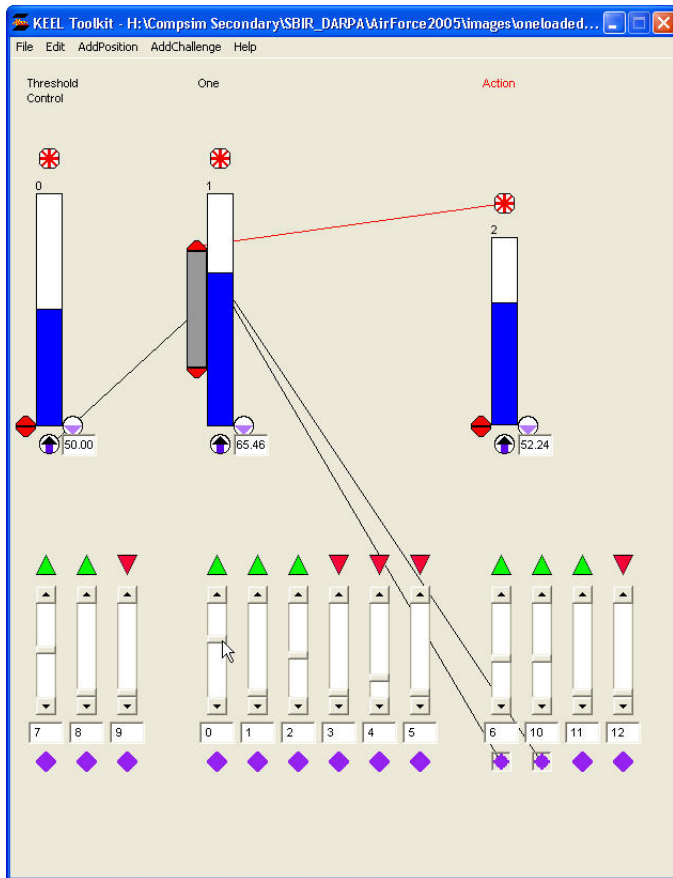
The user can stimulate inputs to the first Output and "see" the reaction of the second. This model can turn on and off external components or can trigger events.



Graphing just 2 of the relationships

Figure 3

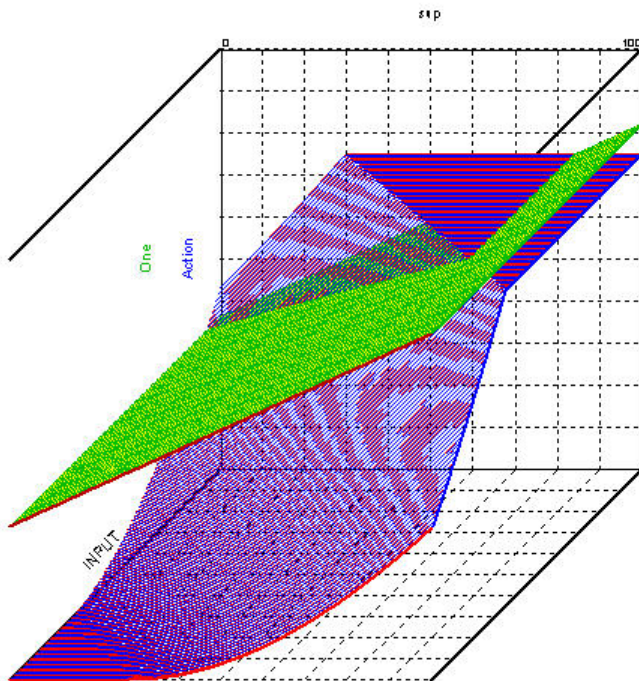
Shows the user wiring the output of a Threshold of one output to turn another segment of the system on/off. A graph of the relationship is shown for 2 of the inputs and the 2 outputs



Using a "Clipper" to control the importance of another Output, while at the same contributing to its resolution.

Using an Output to control the position of the "Clipper" window dynamically.

The user can stimulate any of the inputs and "see" the impact on the overall system.



A slightly more complicated relationship.

Figure 4

Shows a slightly more complex relationship