

Heuristic Reasoning in Batch Processing

Tom Keeley
President
Compsim LLC
Brookfield, WI 53045

Helena Keeley
CEO
Compsim LLC
Brookfield, WI 53045

KEYWORDS

Batch Processing, Human Error, Safety Critical Systems, Decision-Making, Human Reasoning, Cognitive Technology, Loosely Coupled Systems, KEEL Technology, Graphical Programming Language, Expert Systems, Auditable Processes

ABSTRACT

The batch processing industry is responsible for continuous flow manufacturing. The cost of human error in the operation of large batch processing facilities creates risk in both human and financial terms. The industry is constantly searching for ways to reduce human errors and minimize risk. There are also large liability issues faced by the batch processing industry if bad material makes its way to the market. Equipment can fail and be destroyed and people can be injured. Techniques to reduce that risk have involved the use of embedded safety mechanisms, improved training of personnel, and simulation and emulation systems.

This paper will focus on an approach that integrates a human-like methodology for interpreting complex situations that can then be integrated into the batch process as a parallel function. This approach can advise the human operators with suggested actions for consideration. It can also be used to integrate human-like reasoning directly into the batch process. It provides a graphical way to define and represent the situations or events, define the relationships between the items, and cause resultant decisions/actions based on the system data (which can change dynamically).

The system can easily be tuned as more information (or rules) are introduced. By providing "explainable" actions, the process can be audited. This paper introduces a framework for the creation of a system that will define dynamic rules for interpreting information and for documenting information inter-relationships that can be used to model how the "best" operator should respond. In addition to control functions, human-like reasoning can be applied to diagnostics and prognostics functions.

INTRODUCTION

Large batch and continuous manufacturing processes are commonly controlled by distributed control systems (DCS). These DCS systems contain custom Process Controllers and Programmable Logic Controllers (PLCs) orchestrated to work in a coordinated manner to control the system. Complex feedback loops are used to control the manufacturing process. These large batch processes can be compared to organic systems that adjust themselves in response to the system dynamics. In a perfect world, everything would be tightly coupled where a single control variable could be used to increase or decrease production for the manufacturing enterprise. Nothing would ever break. The raw material would be consistent and without flaw. And the manufacturing environment (temperature and humidity) would be consistent. Unfortunately, this isn't the way things work. Seemingly, every component or sub-process has a different time constant. Everything that can break will eventually break. This cycle demands a system that is closely monitored so all of the variables can be controlled in order to provide a consistent high quality product.

Humans play many roles in the batch manufacturing process. They load raw material, prepare and refine recipes, schedule production, and perform preventive maintenance. They also make mistakes. But it is not just the humans that make mistakes. Equipment fails, sensors fail to correctly detect, control valves fail to open or close, and pumps fail to pump or operate inefficiently.

When these failures happen in large complex systems, bad things can happen. There can be large economic impacts to the facility itself. There can be injuries and loss of life to the employees and sometimes to the surrounding communities. Studies from the late 1970's reveal that "operator error accounted for 50-70% of failures in electronic systems, 20-50% of missile system failures, and 60-70% of aircraft failures."¹

Human failures have been decomposed several ways based on characteristics humans employ while performing their jobs. One view identifies the following characteristics: Attention, Perception, Memory, and Logical Reasoning. The topic of Attention, also called Situation Awareness², is suggested as a specific problem in automation systems when humans act as monitors of automated systems. There is a suggestion that the human attention span is roughly 20 minutes. Beyond that mental fatigue sets in. Perception is the ability of a human to interpret sensed information and has many safety implications. Memory has to do with capacity, and short term memory is limited in that characteristic. In general, a human can handle seven (plus or minus two) data items at one time. It is suggested that humans don't do a very good job of "logical reasoning" and it also is suggested as a common source of problems in industry.³ One can further define logical reasoning as the application of judgment when interpreting information.

When humans participate in the overall operation of the system, they are often called to perform maintenance functions. These are often stressful situations that involve deadline pressures, system alarms, inconvenient schedules, poor environmental conditions, and so on. Human operator errors during maintenance are a significant source of system outages.⁴

There has been considerable research into the causes of human error. The objective of that research has been primarily to address human interaction with systems so that the human errors can be caught before causing a major problem.

This paper focuses on another concept for addressing the human error component of these complex systems. The intent will be to show how some of the processes that are under the control of humans today can be integrated directly into the control system, thus eliminating these errors and reducing the associated costs. The modeling approach discussed in this paper applies equally well to the automation of situations where human operators can amplify the problems by incorrectly servicing them.

AUTOMATING JUDGMENTAL REASONING

Historically industry has automated systems in order to increase productivity, increase quality, and in some cases to keep humans out of harm's way. Today many manufacturing processes cannot be performed manually, because of the tolerances required in the processes. This might be a speed issue, or it might be a situation requiring precision not achievable with "human engines". Industry has done a good job of automating systems that have well-defined rules. Even complex feedback loops have been automated with process controllers and DCS systems. The segments that have yet to be automated have to do with the human's ability to interpret the situation and make judgmental decisions.

It is this requirement to integrate dynamic, non-linear, multi-dimensional, inter-related data sets that has made it primarily the domain of humans rather than control systems.

With regard to the characteristics that offer opportunities for human error, attentiveness is not a problem for automated solutions. Neither is perception, if responding to sensors in a well defined and documented method is used. Short term and long term memory is also not a problem for computerized solutions. What remains is the human capability to use judgment when performing tasks.

The primary reason for not automating human judgment has been the lack of an effective way to define human judgment in a way that satisfies some of the other demands of automated safety critical systems.

OTHER REQUIREMENTS OF BATCH (SAFETY CRITICAL) SYSTEMS

Safety critical systems must operate in a well understood, completely explainable manner. This may preclude some other approaches that can exhibit human-like reasoning. Two examples are artificial neural nets (ANN) and fuzzy logic.⁵ ANNs provide a mechanism for integrating multiple non-linear inputs in order to generate one or more non-linear outputs. But ANNs are pattern-matching techniques. The results are based on patterns that were taught, not on any specific logic. The ANN based solutions cannot provide an "explanation" of the reasoning for a decision or action. One has to assume that the system provides an answer that is close to what was taught. And one has to assume the ANN was taught the right patterns.

Another approach that might be considered for providing human-like judgmental reasoning would be “fuzzy logic”. Fuzzy logic is based on linguistic uncertainty, or the granularity of the human language. The fuzzification / defuzzification process provides a process that translates human terms into digital values that can be manipulated by computers. The fuzzification / defuzzification process blurs information and makes it difficult to explicitly define decisions and actions. While some automation suppliers provide tools to utilize ANN and fuzzy logic in their control systems that may provide repeatable results, they may not provide explainable / auditable results.

Conventional batch manufacturing systems have also utilized a variety of conventional programming languages to define the “rules” for operation. Conventional programming languages commonly incorporate sequential flow processing: IF – THEN – ELSE language constructs with the ability to manipulate conventional formulas. Examples of these languages would be C, C++, Java, and Assembly Language, just to name a few that have been developed over the years. When some of the logic is implemented in conventional PLCs, the languages may be Relay Ladder Logic, Sequential Function Charts, Instruction Lists, or Structured Text. These are still what one would call “sequential flow processing languages”. These languages do a good job of describing conventional manufacturing processes that have been automated in the past. The formulas (logic) created with these languages provide completely repeatable, explainable and auditable designs. They do not, however, provide an “easy” mechanism to define human-like judgment which requires the integration of non-linear dynamic data.

DEFINING HUMAN-LIKE JUDGMENT

Before one can automate human-like judgment, it must be defined. There are several aspects to human judgment and to problems addressed with human judgment.

The problems addressed with human judgment are all data fusion problems. Human judgment is used to combine multiple data sources to bear on the problem domain. Many times these appear to be complex, seemingly unstructured situations.

Human judgment is often applied to dynamic situations, allowing humans to continually correct to a changing environment.

Human judgment may be used to identify an optimal time to perform an act, or an optimal “value” of one or more variables.

Many times the data items being addressed by human judgment have a web of relationships within the problem domain. They are not well structured decision *trees*, but encompass situations where any piece of information may impact different parts of the problem domain with different levels of significance.

Finally, human judgment is a balancing act, where the human balances a series of inputs and a series of inter-related outputs to achieve the best set of outputs for the problem domain.

Since human judgment is applied to dynamic situations, human judgment is a repetitive act that continues to rebalance itself as new information is available.

When humans apply “judgment”, this is mostly a “right-brain, image processing” problem and is different from the left-brain language / number processing segment.⁶ In this light, conventional programming techniques for process controllers and PLCs have addressed the left-brain portion of the problem domain, but have yet to sufficiently address the right-brain types of problems.

SUMMARIZING THE OBJECTIVE OF AUTOMATING JUDGMENTAL REASONING

- Automating human judgmental reasoning can help overcome the primary roadblock to the elimination of human error in these complex systems. Attention, perception, and memory can easily be addressed by the very process of encapsulating them in computerized equipment.
- Any solution that automates human judgmental reasoning must be auditable. For a solution to be auditable, it must be completely explainable.
- If judgmental reasoning is more of an image processing function than a sequential formula processing function, then it can best be addressed with a graphical language.
- The language must be dynamic, since the solution needs to address dynamic, non-linear situations. This is necessary to develop solutions without decoupling development and test functions.
- The mechanism for designing systems capable of exhibiting human-like reasoning must be cost effective. The primary reason that explainable, auditable human judgment has not been automated in the past has been that this reasoning has been difficult (or impossible) to model. These topics will be addressed with the remainder of the paper.

KNOWLEDGE ENHANCED ELECTRONIC LOGIC (KEEL[®]) TECHNOLOGY

KEEL is an umbrella term for a collection of services used to encapsulate human-like reasoning in devices or software applications. It is suggested as an alternative model for manufacturing systems compared to complex, mathematical, formula-based logic solutions. It is also suggested as an alternative to ANN or fuzzy logic when completely explainable and auditable solutions are required. KEEL technology provides: 1. a mechanism for describing judgmental policies that can be explicitly explained and audited, 2. a web model for fusing multiple information sources, 3. an architecture for implementing human-like reasoning in devices and software applications, and 4. a programming paradigm for rapid development and test of human-like reasoning.

KEEL technology incorporates a graphical methodology for documenting policies that define the human-like reasoning that maps closely to the process that the right brain uses to evaluate information.

If judgmental reasoning is an image processing type of mental function⁷, then a graphical approach is the most appropriate way to define policies that describe the rules for judgmental reasoning.

Graphical languages have been around for a long time. Relay Ladder Logic (for PLCs) was one of the first graphical languages. It consists of graphical representations of relays that are processed much like their physical counterparts. There has also been a number of data flow languages developed since the introduction of windows-based computers provided a graphical human interface for their operating systems. These graphical languages, however, focused primarily on the sequence of instruction execution. Basic programming constructs are based around IF-THEN-ELSE structures where the program flow is determined by testing, comparing and branching.

The KEEL graphical language provides a new “type” of language; one without IF-THEN-ELSE language constructs. Rather than processing information sequentially, the data is processed during a “cognitive cycle”. With this type of processing, a snapshot of input information is collected at the beginning of the cycle. This information is iteratively processed until all of the outputs stabilize, at which time the outputs are exposed to external logic. (Figure 1) This is somewhat similar to a PLC scan cycle, where a program scan is bounded with input and output sections. A PLC program scan, however, runs from beginning to end and does not go through a stabilization process. With the KEEL language, one is implementing the functionality of an analog computer. Language elements show inputs and outputs as graphical data items. Inputs and outputs can be visible to the outside world, or they can be internal integration points used to fuse information items together. The size of the output graphical elements defines the importance of information. Wires are used to define relationships between information items. The collection of inputs, outputs and wires, that defines a cognitive problem solving domain, is called a KEEL Engine. (Figure 2)

In complex systems, a single piece of information can carry different weights in different parts of the problem domain. In dynamic systems, the importance of a piece of information can change as the environment changes. The dynamic nature of the KEEL graphical language is provided by using scroll bars to indicate the instantaneous values of inputs. The cognitive designer “sees the system think” by moving the scroll bars and observing the system re-stabilize. In many cases the designer will create two and three dimensional graphs to show data interactions. The user can see the impact of any change by observing the impact of the change propagate through the system.

KEEL technology is in the “expert system” domain, since it requires a human expert to define the rules. It is not a pattern-matching system like ANN systems, although it does provide for webs or “nets” of inter-related data items. During the development phase, the designer “tags” inputs and outputs with names. When exported as a KEEL Engine for deployment in a Process Controller or a PLC, the text descriptor tags are used only to comment the code as the policy description itself is maintained only in tables of values and relationships.

KEEL DECISIONS AND RELATIONSHIPS

In its simplest form, KEEL technology provides a way to integrate multiple sensors (or data items) in order to create a single “relative” analog output. The inputs in KEEL can provide a driving or

blocking component to the integration process. The method behind the handling of the integration is beyond the scope of this paper.⁸ Once data is brought into a KEEL Engine, it can be used for various dynamic services. It can be used to control the “importance” of other information. It can be used to contribute to another decision or action. It can be used to trigger other decisions by turning outputs or internal control variables on or off.

For example: a human is often asked to evaluate a situation and make a decision to initiate one of several actions (A, B, or C). A simple IF-THEN-ELSE structure may support this function. On the other hand, the difficulty for a human may be to integrate the data contributing to A, B, or C in order to determine which action to select.

DESIGN PHILOSOPHY

The integration of heuristic reasoning capability into batch processing systems is likely to be an evolutionary process. In other words, it is likely that high priority / high risk situations would be integrated first. Because the KEEL API is very simple, it is conducive to evolutionary development.

The development process is likely to proceed through the following steps: 1. Create the initial design and test using the KEEL graphical language within the KEEL development environment. 2. Incorporate the KEEL Engine(s) in a simulator and test again. 3. Deploy the KEEL Engines in the production environment and observe performance. 4. Audit the autonomous decisions / actions and update the policies as appropriate. 5. Extend and refine the reasoning process over time (spiral development model).

KEEL technology supports this spiral model with several key services: 1. The KEEL graphical language can be deployed in multiple languages without “touching” the deployed code. This allows the same policy model to be operated on multiple platforms if necessary. 2. The API is very simple; making it easy to integrate and test. 3. The policies are auditable by using snapshots of real-world data to drive the KEEL graphical language so the users can “see” why decisions and actions are made.

DEPLOYMENT OF KEEL ENGINES

KEEL Engines are self-contained data analysis functions. When they are scheduled to analyze a situation, they do it with the information available at that instant in time. In dynamic situations, which are common in complex batch processing systems, this means that the analysis is continuously repeated, or repeated at some known update rate. In other situations, the analysis can be driven by change of state or by manual intervention.

DCS systems will contain many dynamic elements that require a number of analysis functions working together. Since KEEL Engines are self contained, the system architect is responsible for coordinating the information flow and determining the consequences of the timeliness of data items. This may mean that old information is not used, or it could mean that the old information will carry less weight in the process, or it could mean that some type of interpolated pseudo information is used in place of

the real information. In this way, KEEL Engines operate just like a group of human operators that work together to control a system. Human operators exchange information periodically to update their peers on the status of their part of the system. Updates from different operators occur at different times. During the times between updates, the operators receiving the updated information perform their work based on the last available information. The same is true with KEEL Engines.

The KEEL graphical language provides a way to define and evaluate the judgmental policies. However, this language cannot be directly executed by any process controller or PLC. So, to deploy the policies to process controllers and PLCs, the graphical language is translated to the conventional language of choice. For PLCs, this language is commonly Structured Text, although some PLCs support embedded C functions. For process controllers, the language of choice could be C, Java, C#, or perhaps even Visual Basic. This source code is provided as a text file that can be pasted into the user's development environment for the system build.

SAMPLE PROBLEM MODEL

It is common to use human operators to "monitor" DCS systems through complex HMI systems. Visual aids are used to highlight system or component failures or operational parameters. One example might be to monitor a boiler as it is filled with chemicals and heated. The chemicals are added at different times and the heat is adjusted according to a recipe. Using conventional logic, the system might generate alarm conditions if certain events take place or fail to take place. A human operator may be wandering around the process, observing several gages showing pressure, flow, etc. The human may be expected to interpolate the change in temperature over time as he/she monitors the process. Of course, if the human is monitoring several gages, then the interpolation process is more complex, AND more granular as the human is probably considering each gage at some human scan rate.

A KEEL based design would allow the monitoring to be much more controlled. At the same time, much more complex relationships can be monitored. A model of the expected performance would be developed to satisfy this capability. This would include the definition of curves defining the expected behavior in two dimensional views (Figures 3 and 4). An example might be to model how flow rate might change as the temperature of a fluid changes. A similar process might model how viscosity changes impact the process. These relationships would be integrated to model how flow rate might change as temperature and viscosity change. These relationships might be fused with a vibration sensor to determine an accumulated risk factor. Once the input data is fused, the operator response is defined: either by adjusting controllable variables, by promoting alarms to other systems, or by scheduling preventative maintenance.

To program KEEL based systems the designer does not think in terms of events or in terms of comparing values. The designer thinks in terms of the importance of information and how each piece of information impacts other pieces of information. In this way, the designer is considering how information is integrated rather than as a sequence of tests (Figure 5). The development process, like the cognitive cycle, is iterative. The designer identifies data items and relationships and tests them using the dynamic language (Figure 7). The designer then observes the reaction of the system (Figure

6). If it performs as expected, the designer adds the next component. If not, the shape of the curves is modified and tested again.

The test phase for this kind of heuristic system is commonly the longest phase as complex conditions need to be evaluated. The graphical language allows complex models to be rapidly created and modified, but the design still needs to go through extensive testing to insure that the judgmental actions operate as desired. There is one additional benefit from the process of automating human reasoning. As the model is created, the systems engineer is developing a better understanding of the process. In many cases, this may be the first time that human reasoning will be documented at this level of detail. The KEEL development process itself may expose some of the reasons that human error can contribute to problems with the manufacturing process.

SUMMARY

Complex batch manufacturing systems process conventional hard-coded logic to address the deterministic situations that exist in any control system. If humans perform duties that can easily be documented with formula based rules, this technique should be used to automate these functions as well. There are situations, however, when it is difficult or inefficient to develop complex formulas to define the policies that define how human judgment should be applied to DCS applications. These situations should be considered for automation, especially when human error can cause significant economic and social consequences. KEEL technology is offered as an alternative to artificial neural nets and fuzzy logic when *completely explainable actions* are required. The development environment which allows immediate feedback into the design process is an added benefit when the economics of managing a new technology needs to be considered. The addition of human-like reasoning will seldom be addressed with a single technical approach (i.e., just conventional logic, just neural networks, or just fuzzy logic). It is also not a service that will be offered to completely eliminate humans as part of the manufacturing process. KEEL technology should be considered one additional approach that could be applied when the situation demands explainable actions in a dynamic environment.

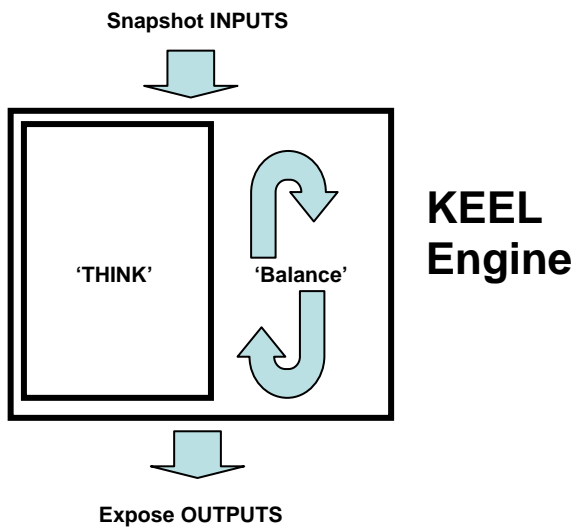


FIG. 1 – KEEL COGNITIVE CYCLE

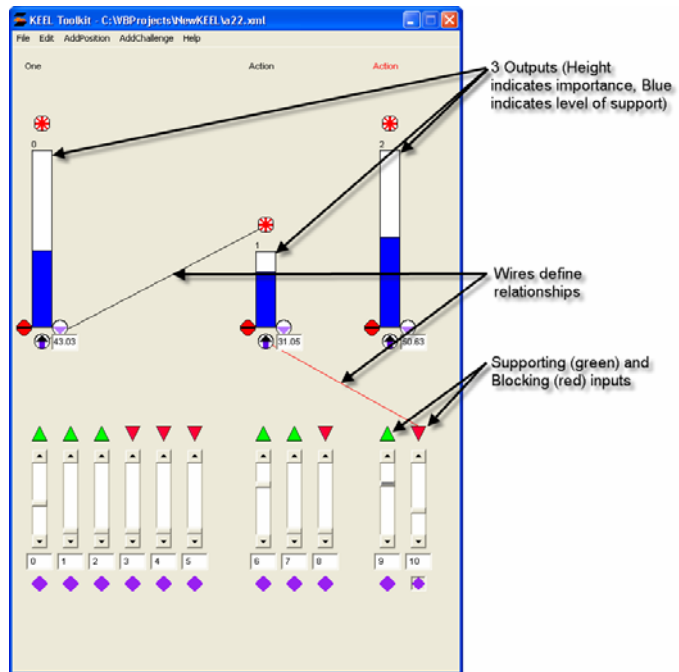


FIG. 2 – GRAPHICAL PROGRAMMING LANGUAGE

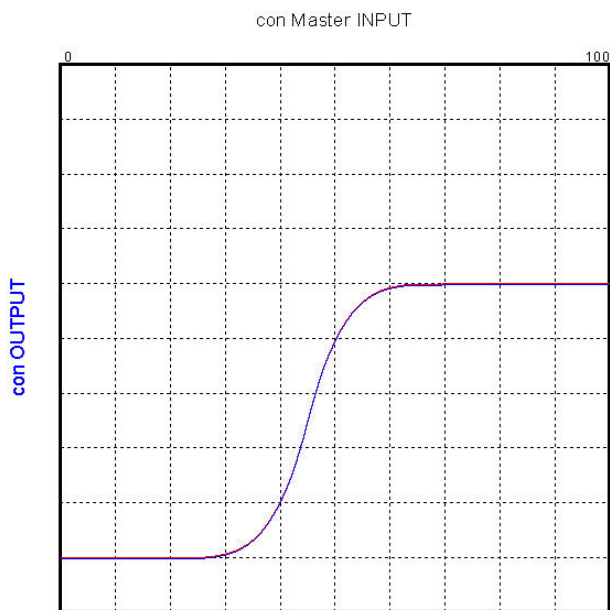


FIG. 3 – CONTROL SIGNAL

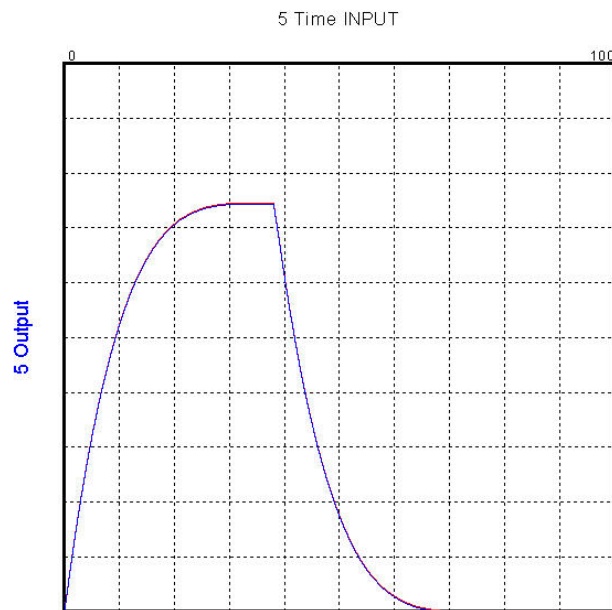


FIG. 4 – CONTROLLED SIGNAL

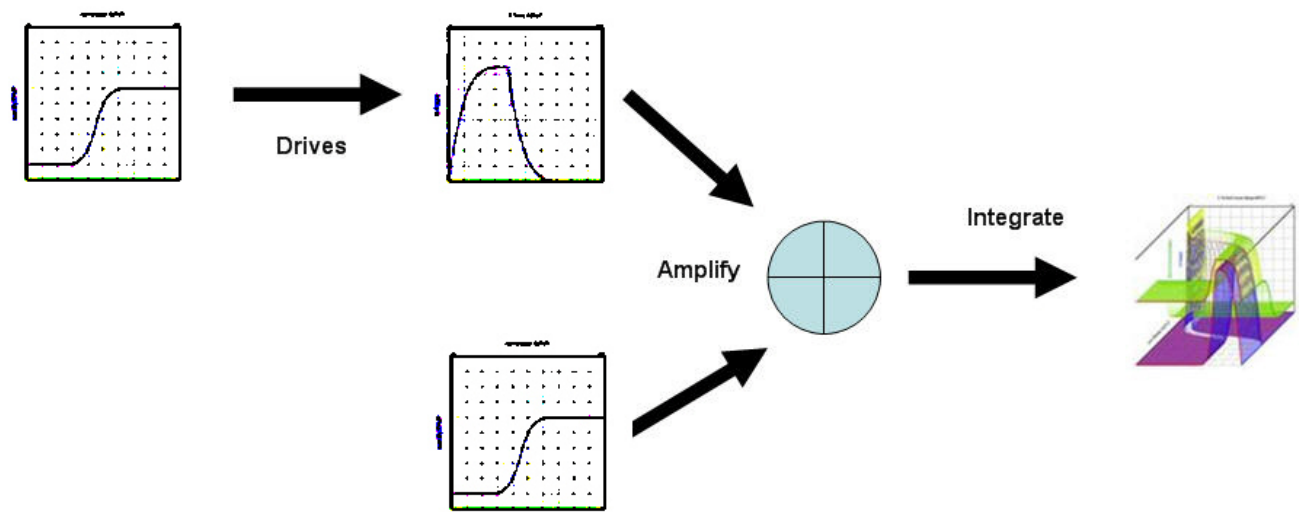


FIG. 5 – CONCEPT MODELING (THINKING IN CURVES)

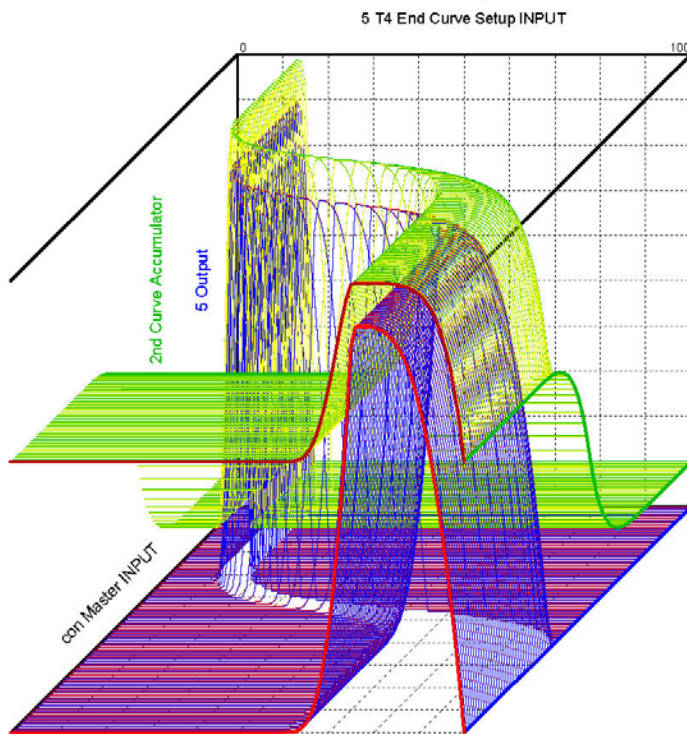


FIG. 6 – COMBINED COGNITIVE MODEL

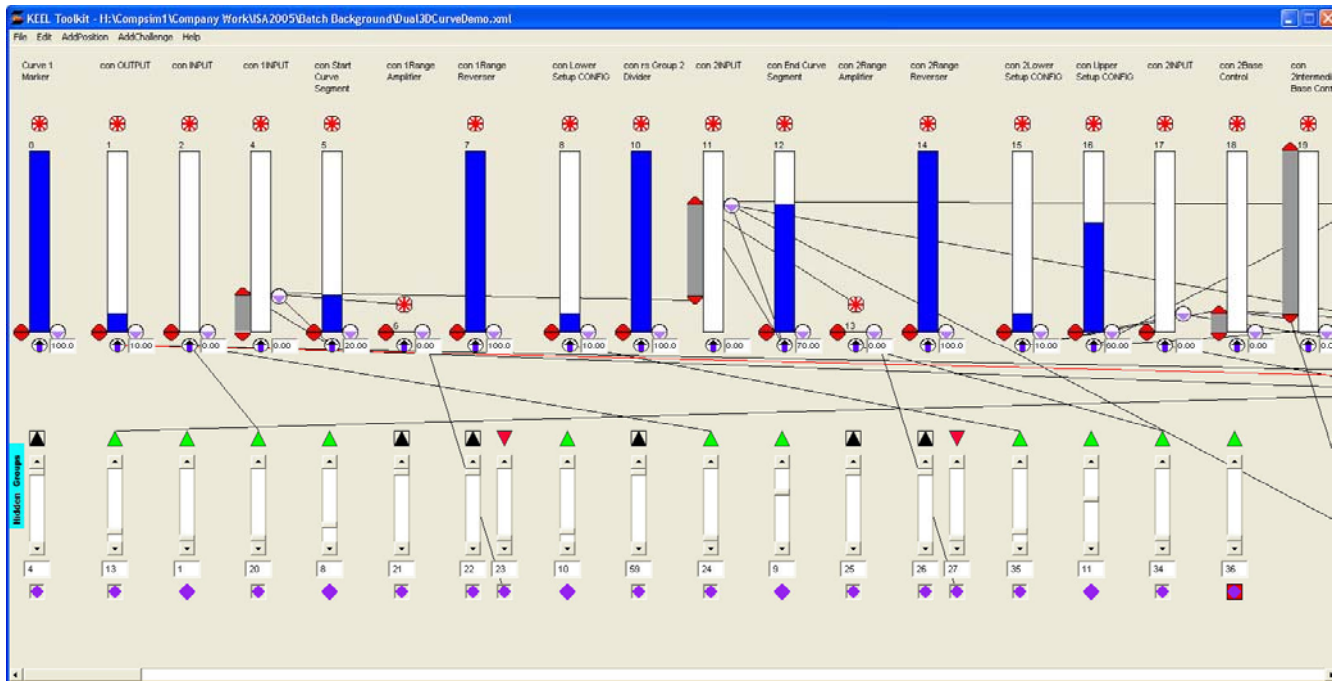


FIG. 7 – POLICY DEFINITION

REFERENCES

¹ Brown, A. B., Patterson, D. A.; “To Err is Human”; Computer Science Division, University of California of Berkley; <http://roc.cs.berkeley.edu/papers/easy01.pdf>

² Endsley, M. R.; Department of Industrial Engineering, Texas Tech University; “Automation and Situation Awareness”; 1996; <http://www.satechnologies.com/Papers/pdf/SA&Auto-Chp.pdf>

³ Parliamentary Office of Science and Technology; “postnote; Managing Human Error”; June 2001 Number 156; <http://www.parliament.uk/post/pn156.pdf>

⁴ Brown, A. B., Patterson, D. A.; “To Err is Human”; Computer Science Division, University of California of Berkley; <http://roc.cs.berkeley.edu/papers/easy01.pdf>

⁵ Von Altrock, Constantin; “NeuroFuzzy Technologies”; Fuzzy Logic & NeuroFuzzy Applications Explained; Prentice Hall; 1995; pg 64.

⁶ Keeley, T.; Compsim paper; “Right Brain Programming”; http://www.compsim.com/papers/Right_Brain_Programming.pdf

⁷ McCrone, J. “‘Right Brain’ or ‘Left Brain’ Myth or Reality”; The New Scientist; <http://www.rense.com/general2/rb.htm>

⁸ Compsim paper; “Decisions and Actions in KEEL”; 2003; http://www.compsim.com/papers/Decisions_and_Actions_in_KEEL.pdf