Certifiable Code and KEEL® Technology

Compsim White Paper

Knowledge
Enhanced
Electronic
Logic

Objective: The objective of this paper is to offer a response to questions and concerns regarding KEEL[1] "code" certification. To respond to this question one needs to understand what KEEL is and what it is not.

The topic of code certification covers the entire lifecycle of a project:

- Planning
- Development
- Verification
- Configuration Management
- Quality Assurance
- Certification

It is important to understand that KEEL ("tool": the KEEL Toolkit) does not create an entire working project. It focuses solely on the creation of individual functions (or classes with methods) (called the KEEL Engine) that will be subsequently integrated into a broader development environment.

It is also important to understand that KEEL creates the KEEL Engine as a text file in the language of choice: ActionScript (Flash), C, C++, C#, Java, JavaScript, Octave (MATLAB), Python, Visual Basic (6), VB .NET, VBScript, and others. It is this code that is integrated (copied and pasted) into the user's project development environment to be compiled into the working system. This is the code that will ultimately need to be "certified".

The ultimate "intent" of code certification is to insure that the "code" performs as expected / desired, does not "fail", and can be "maintained".

It is also important to understand that KEEL Technology addresses some of the more complex problem sets that may not have been easy to address before: complex, dynamic, non-linear, inter-related, and multi-dimensional.

[1] KEEL® (Knowledge Enhanced Electronic Logic) "technology" is an umbrella term that encapsulates a new way of processing information in the form of a KEEL Engine, and the KEEL "Dynamic Graphical Language" which is used to create the KEEL Engine. KEEL Technology is Compsim LLC proprietary technology that is covered by a series of granted and pending patents.

The complexity of these tasks almost always utilize a spiral development process from the beginning, because the user (human) is probably incapable of describing exactly how the system is supposed to perform at the beginning. In this light, a critical aspect of KEEL Technology is the interactive development process, where the KEEL "dynamic graphical language" is used to develop the description of the problem solution at the same time as the solution (KEEL Engine) is being developed (behind the scenes) by the KEEL Toolkit.

Source Code:

The KEEL Dynamic Graphical Language (DGL) is "KEEL Source Code". It is unlike any textual source code like C or Java that can be edited with any text editor. Where one "could" take pencil and paper and write a C routine that could later be entered into a development environment where it is compiled / linked, this is not "the way" code is developed for KEEL.

The KEEL DGL makes use of the interactive nature of computer graphics. Icons representing elements of the KEEL DGL are dropped on the screen and linked together by wires representing specific functional relationships between data items. The user stimulates the system (through slider icons) and "observes" the system response.

A variety of features within the KEEL Toolkit are provided assist the user in viewing the system that has been created.

- Clicking on connection points illuminates wires connected to those points to make it easy to trace functional relationships.
- Selective "views" of the system can be created.
- Parent and Child views of inputs and outputs allow the users to see which data items are dependent on others (and how) and which data items influence others (and how).
- A variety of reports can be generated.
- 2 and 3 dimensional graphs can be created to show how the system will respond across a range of inputs.
- Data items can have embedded user documentation.
- Symbol names can be used to describe the external interface.

When a design is complete it is translated to "conventional" code as a text file in the language of choice. The structure of this code and the way it processes information is unique to KEEL.

Conventional Code:

Conventional code targeting von Neumann computers can be generalized as "IF | THEN | ELSE" sequential processing engines. Operation codes, instruction pointers and conditional branching are captured in programs that direct the computer through a series

of logic steps accumulating results. The concept of a program is to describe this process to accept a number of inputs and produce a number of outputs. The program can run one time or cycle forever, or it can be triggered to run from some "event" happening, or.... One can consider a computer program as a script for the operation of the computer. A key component of the von Neumann computer is the support for the conditional branch. One aspect of code certification is that there needs to be a way to test all of the conditions for all of the branches. Every design will be different as the code generated will be different.

The KEEL Engine:

Unlike conventional code, the KEEL Engine code (when translated from KEEL DGL source code to conventional code – example: C or Java) is almost always the same. We say "almost" because the code is optimized by removing instructions for functionality that is not included in the KEEL design. The project definition (what would be scripted in conventional languages) is maintained in "fixed size" tables of integers or floating point numbers.

The small KEEL Engine function that processes the information in the tables is always the same.

One might use the analogy that in a binary computer the 1 and 0 are always the same. It is just their combination that makes a complex system. With KEEL the way the information in the tables is processed is always the same. It is just the data items in the tables that are different.

The artifacts of the KEEL Engine are the KEEL "dodecisions" function, the KEEL "tables" (1 and 2 dimension arrays; and 1 multi-dimension array for the Accelerated Code – See next section). Depending on the conventional language selected, there may be an initialization routine (initializefixedtables) or a constructor if an object oriented language is selected. Object oriented languages will have methods to load and retrieve data items to/from tables.

In summary, a KEEL Engine includes the code to process tables of information in a consistent way. The "code" (not including the tables) is 381 lines of uncommented code (for C). Commented code has more lines. While this is conventional code, it is of no use in understanding the entire problem being solved, because the problem definition is maintained in the tables, not the code. The resultant code "manipulates" the data in the tables according to the domain expert's design that is packaged with the KEEL Toolkit.

This differentiates the problem of certifying the project from conventional scripted code. The 381 lines of KEEL code has to be certified only one time because it is "always" the same.[2]

[2] Optimized code will eliminate some lines of code associated with functional relationships not used.

Two Types of KEEL Engines:

KEEL Technology was originally developed to target embedded systems. Embedded systems sometimes have additional drivers:

- Smallest possible memory space
- Fastest possible operation and more deterministic (as far as processing time)

To achieve these two goals, two different KEEL Engines can be created. There is very little difference between the two approaches. They are labeled Normal and Accelerated. The Normal code is slightly smaller, and slightly slower, with slightly more jitter. The Accelerated code is slightly larger, slightly faster with less jitter.

The difference between the two versions is that the Accelerated code includes one additional multi-dimensional table that defines an optimal way to process the information. This takes somewhat more memory space, but accelerates the processing (in most, but not all, cases).

Code Verification:

The following code verification process is used to verify the "conventional" code generated by the KEEL Toolkit. It includes services built into the KEEL Toolkit and external applications hosting the KEEL Engine code in the language of choice.

First Level of Code Verification:

When a designer is developing a KEEL-based design and dropping icons representing language components on the display (Positions and Arguments in KEEL lingo) and wiring them together to define functional relationships, the user gets immediate feedback as far as the functionality being developed. Behind the scenes a "real" KEEL Engine is being created that tracks the design. Tables are automatically being created and manipulated. The tables and KEEL Engine structure are the same as what will eventually be produced. The developer can "see" how the information is being processed. There is no "hidden" information.

Second Level of Code Verification:

The KEEL Toolkit is a conventional code generator. So the second level of code verification is a "code walkthrough". Because all KEEL code (any language) is structured the same, a formal code walkthrough document exists, explaining every line of code for both Normal and Accelerated models. The approach used to explain the code uses both the KEEL DGL (showing graphically how the functional relationships are modeled) and a textual description to explain the functional relationships that are encoded. This code walkthrough is provided to developers during KEEL Workshops and as part of the licensing of KEEL Technology.

Third Level of Code Verification:

A menu option in the KEEL Toolkit called "Snap" is provided to support the code verification function. When this menu option is clicked, the KEEL Toolkit creates an XML file of all of the table entries that describe the input and output variables of the system. Project information is also included. This is a snapshot that is taken every time the user clicks the menu option.

For each conventional language, a Test Program is developed. It has the following capability- as a minimum:

- A method for the user to click that causes the program to read the XML file created with the KEEL Toolkit "Snap" function.
- A place to insert the KEEL Engine code created by the KEEL Toolkit.
- A method to display error information.

The Test Program will be compiled with its new KEEL Engine linked in.

For languages where there is a "Windows" development capability, there may be other features that make it easier to use.

To operate this "third level" of code verification, the user will be running both the KEEL Toolkit and the Test Program at the same time. The user will click the Snap menu item on the KEEL Toolkit and use the Test Program to read the XML file and use the inputs to drive its own KEEL Engine. The Test Program will then compare its results with the results of the KEEL Toolkit. There may be some very small differences because translating floating point output values to text looses some of the precision (10^{-14} or 10^{-15}). If there are any larger problems an error in the code is detected.

This level of testing insures that the code functions exactly as it is designed with the KEEL DGL.

Fourth Level of Verification (Functional Verification):

It is common to take more complex systems through a series of steps:

Develop and test within the KEEL Toolkit by manually stimulating inputs and observing outputs to insure the DGL appropriately describes the problem solution.

The first feature built into the KEEL Toolkit is a constant monitoring of the design for unstable designs. This is similar to features built into spreadsheet programs that check for circular loops. The KEEL Toolkit checks for dependencies every time the design is changed to insure there are no circular dependencies: A dependent on B, B dependent on C, C dependent on A for example. Issues are immediately flagged so the user can work around potential problems before they are embedded in the design.

An additional feature built into the KEEL Toolkit is the ability to read an XML file (defined with KEELDataSchemaxml.xml) with externally produced input variables. Using this technique (if one has access to real world input data), the user can read those inputs into the KEEL DGL and use them to drive the inputs rather than manually driving them. This allows the user to trace "visually" the impact of the inputs as they propagate through the design and drive outputs.

To support this activity, the KEEL Toolkit can also drive an external "poor-man's HMI or a custom HMI program to show how the KEEL design is performing.

Fifth Level of Verification (Functional Verification):

The next step may be to integrate the KEEL engine into a simulated environment. An optional routine that can be included in the "conventional" code created by the KEEL Toolkit is one that takes the inputs as seen by the device and writes them to an XML file. This is the same file format (KEELDataSchemaxml.xml).

If this XML file is created by the simulation during operation, it can be used to drive another feature of the KEEL Toolkit: Design Animation.

The animation feature allows the external system to drive the KEEL DGL. The KEEL Toolkit can be set to read this file at a periodic rate and animate the DGL. If the user wants to monitor the system for different decisions and actions the system can be set to "trap" on them so they can be analyzed in detail (tracing wires, etc).

Alternatively, the simulator could be set to log the data for later playback.

The KEEL Toolkit can read the inputs and animate the language as long as it is provided data according to the KEELDataSchemaxml.xml file. Project information is included to insure that the source matches the DGL in the KEEL Toolkit.

It is possible that the simulation will be running in a different language than the final product. If so then an emulation phase may be appropriate.

Final Level (Functional Verification / Auditing):

If the production device has included some means of recording the inputs to the KEEL Engine, either continuously, or when critical decisions were made, then it should always be possible to translate that information into an XML file compatible with the KEELDataSchemaxml.xml file.

This level of auditing is critical for safety critical systems as well as weapon systems that have the potential to make autonomous kill decisions on their own. With KEEL-based systems the decisions and actions will be traceable to the designer.

Testing the KEEL Toolkit:

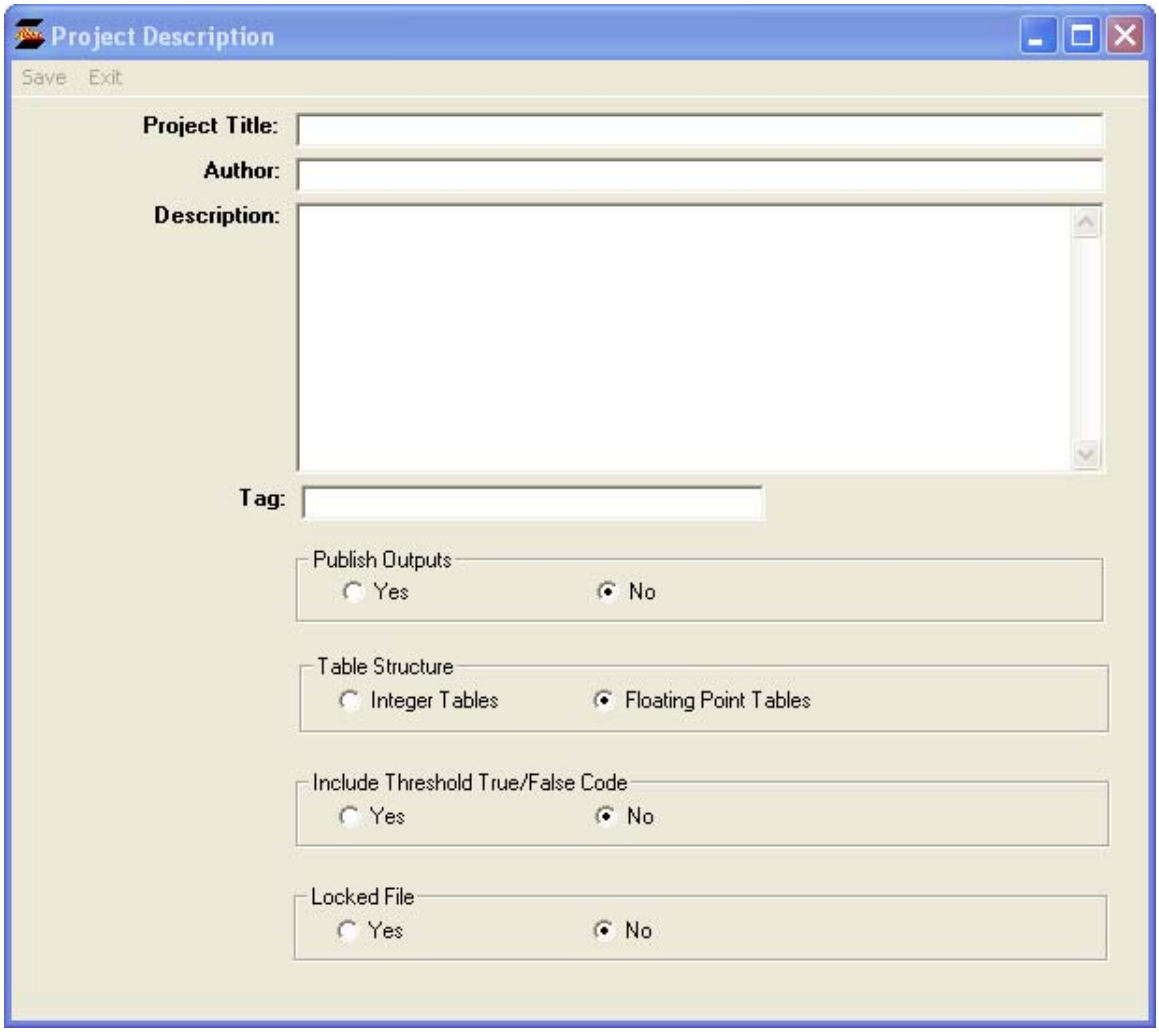
Another part of the verification process is the verification of the internal KEEL Engine that is being dynamically created as one builds a KEEL-based application. To support this function the KEEL Toolkit has what is called the "CheckA" feature.

Normally when a KEEL design is being created the internal engine that is being developed is a Normal model. This is slightly faster to execute when adding and removing Positions, Arguments, and Wires since the one additional "sequence" table does not have to be developed. (The exception is drawing the 2 and 3 dimensional graphs where an Accelerated version is actually created.) Any time the CheckA menu option is selected, an Accelerated Engine is created internally and stimulated with the same inputs that the Normal Engine has been provided. The results of the two engines are compared to insure they calculate the same answers.

This provides another level of validation that the KEEL Engines are in sync and they process the information as expected.

Project Documentation:

The KEEL Toolkit provides several services to assist in the documentation of KEEL-based projects:



Project Title: Textual Name of the project. The Project Title is incorporated in packets used to animate the KEEL DGL and in project documentation and in conventional source code generated by the KEEL Toolkit.

Project Author: The Project Author is incorporated into project documentation and conventional source code generated by the KEEL Toolkit.

Project Description: The Project Description is incorporated into project documentation and conventional source code generated by the KEEL Toolkit.

Tag: Used to tag KEEL Engine conventional code so that multiple KEEL Engines can be included in a single compile.

Publish Outputs: Tells the KEEL Toolkit to publish an XML file for a local HMI display.

Table Structures: Use Integer Tables for smallest possible implementation.

Threshold True / False Table: Used for very small number of applications to offload external software.

Locked File: Once projects are put into production, provides an extra level of security to insure that they are not inadvertently modified. User should still use a code management system for production releases.

Project Reports:

General Report: Prepares a report including general project information: Title, Author, Description, Project Details, Sequence Table (For Accelerated Code).

Connection Chart: Prepares a report of Positions (outputs) with the Arguments (inputs) that drive them.

Input Connection Chart: Prepares a report of Arguments (inputs) with the Positions (outputs) they directly drive.

Wire Report: Prepares a report of the different functional relationships incorporated in the design.

Child Tree: Prepares a report showing internal dependencies of an output.

Parent Tree: Prepares a report showing internal information items driving a selected output.

Summary:

The KEEL Technology "umbrella" contains numerous services and features to support the certification of KEEL Engine code. The responsibility for insuring that the KEEL Engines perform safely and as expected resides solely with the organization creating production applications using KEEL Technology. Broader system definition tools (like UML) focus on broader system functionality. KEEL Technology (tools and services) focus only on the KEEL related functionality. In this vein, certifying KEEL code is only a portion of the overall certification process. Scheduling of KEEL Engine calls would be handled like any other function calls in the broader system perspective of the project.

Compsim LLC
PO Box 532
Brookfield, Wisconsin 53008
(262) 797-0418