Leveraging a Commercial Mutation Analysis Tool For Research

Mark Hampton
Chief Technology Officer, Certess Inc.
mark.hampton@certess.com

Stéphane Petithomme
Manager, Software Development, Certess Inc.
stephane.petithomme@certess.com

Abstract

A presentation of Certitude, which is a commercial software tool performing mutation analysis (MA). The product has been deployed within the microelectronics industry. Certitude is presented with specific emphasis on the product's standard processing model and feature extension mechanisms to support mutation analysis research an extension of Certitude to news languages.

1 Introduction

Certess[1] develops a commercial software product which performs mutation analysis[2]. This paper introduces the product with the intention of facilitating future mutation analysis research.

There have been a number of software tools developed by the software research community including Mothra[3] and MuJava[4]. Each implementation requires a large amount of common infrastructure such as a databases for storing results and mechanisms for presenting data from the database.

The product developed by Certess is the result of many men years of development and testing efforts. This product is targeting the microelectronics industry but has also been used for the mutation analysis of software systems. Through extending this product it is hoped that conducting some aspects of mutation analysis research will be simplified.

This paper will introduce the Certitude product and details of the standard processing model, followed by an overview of extending the tool for supporting new languages and modifying the standard processing models.

The paper focuses on some product aspects that are of particular interest for the research community in Mutation Analysis (MA). Researchers may leverage Certitude's back-end capability to facilitate research, including the following domains:

- Fault model research. Certitude can be extended to support new languages and fault model. APIs for doing so are described.
- Data mining on fault results and result presentation. Certitude presents extensive facilities to query all data related to a fault. This facilitates development of result analysis tools and validation of the usage model.

2 Introducing Certitude

Certitude is an EDA (Electronic Design Automation) software product developed and sold by Certess. This paper is intended for an audience not familiar with the EDA market, therefore a brief overview of this market will be included first.

Worldwide EDA product and maintenance revenue for 2006 totaled around $US5 billion. Functional verification is a segment of the EDA market with software license revenues of approximately $US700 million. Functional verification in the microelectronic development process is similar to software testing in the software development process.

Certitude introduced a new category of EDA product called functional qualification. Functional qualification measures the quality of the functional verification. A key technology underlying functional qualification is mutation analysis.

EDA tools are typically significantly more expensive than tools used in the software development process. EDA products are often leased through a renewable one year license that may cost from tens of thousands to hundreds of thousands of US dollars, before volume discounting. EDA products are typically sold by a direct sales force, field application engineers are often present at the client site to facilitate tool adoption. EDA users expect a level of support and service in line with the cost of EDA tools.

The end user of the Certitude product is a verifier or verification engineer, this role is analogous to a tester in the software development process.

0-7695-2984-4/07 $25.00 © 2007 IEEE
DOI 10.1109/TAIC.PART.2007.39

Certess was founded in November of 2003 with a prototype implementing functional qualification. This code base was used within the context of a services business model initially. Feedback from the use of this prototype was continuously integrated and provided a specification for a commercial software development project. The Certitude product was a complete rewrite in C++ and TCL of this early prototype and is currently released as version 2.3.

Certitude is developed in Certess' R&D center in Grenoble, France. The development team consists of 10 engineers divided into separate test, development and usability teams. The test team uses an extension of the Certitude product that enables the functional qualification of C++ programs. In this way the Certitude test suite is "qualified" by the Certitude product.

3 Introducing Functional Qualification

Functional Qualification can be thought of as an implementation of mutation analysis. This section of the paper will introduce some specific terms used in functional qualification.

The term "fault" is used to refer to a mutation of the original design code. A fault can have different status:

- "**Activation**" which means that the code corresponding to the fault is executed.
- "**Weak**" means that the fault does not cause a change in behavior of the statement in which the fault is injected. For example if the original code was "a = a and b" then a potential fault could be "a = a or b" if the value of 'a' and the value of 'b' is always true in every test, then it is not possible for this fault to be detected by the provided test and the fault will be classified as Weak.
- "**Detected**" which means the fault causes at least one test to fail, this is analogous to the notion of a killed mutation.
- "**Non-Detected**" means the fault cannot be detected by any of the tests in the non-regression suite this is analogous to the notion of a live mutation.

The Certitude tool breaks the functional qualification into a four steps process:

1. The **model** step will analyze the design and prepare the instrumented code necessary for future steps.
2. The **activation** step will analyze the behavior of the tests by running each test once using an instrumented version of the design which captures the behavior of the test without modifying the design's behavior.
3. The **detection** step will perform a mutation analysis using an instrumented version of the design which provides a mutant schemas [4].
4. An HTML **report** can be generated to view the results of an ongoing or completed qualification run.

Certitude provides a TCL shell interface with an extended set of commands adapted to functional qualification. The HTML report allows for a convenient analysis of the results in the context of the design's original source code.

The Certitude product is designed to allow for simple integration into diverse functional verification environments. Functional verification environments are often referred to as "testbenches".

Certitude supports designs described using the Verilog or VHDL Hardware Description Languages (HDL). The user must specify the location of the design files to be qualified. Certitude will parse the design files and produce instrumented versions of these files. The user must provide a script which will compile the instrumented design and test environment. This script is called by Certitude at the appropriate time in the activation and detection phases.

Certitude requires that the user provides a list of test names that represent the non-regression test suite being qualified. The user also provides a script that can run a test and return a pass or fail result. This script will be called by Certitude and the name of the test to execute will be passed as an argument to the script.

The compilation and execute scripts isolate Certitude from the implementation details of the verification environment so Certitude can be used with any functional verification strategy.

For the remainder of the paper, we will make use of the terminology define in this section

4 Tool Flow

In order to perform the MA of a design, Certitude performs four different steps:

1. Model
2. Activation
3. Detection
4. Result Presentation

These steps are explained in more detail.

4.1 Model

The model phase is the first step in the MA of the design. Its purpose is to analyze the design under test to determine which faults should be inserted. The tool is using an adequate language parser to build an internal representations of the language to be manipulated. Presently, the VHDL and Verilog languages have built-in support. Then, the tool determines the list of all possible faults that can be inserted in the design (the fault model is not configurable, user can only select the wanted fault set).

Certitude uses a selective mutation strategy [5].This paper will not provide a full explanation of the fault model.

The faults are modeled using an internal database. Typically, the following information defines a fault:

- Fault type
- Fault location in original design
- Fault structural context definition (statement, block structure,...)
- Fault property (like asynchronous reset for hardware descriptions)

From this point, the tool does not need any further reference to the design and works independently of the design source code.

4.2 Activation

Activation is the second step in the Functional Qualification of the design. The purpose is to determine the Weak and Activation status of the modeled faults for later usage by the detection algorithm. During this step, the tool will execute a full regression of the testbench (At this point, the user must have provided a script which individually executes tests in the testbench See 4.6 for detail). In order to improve performance, Certitude is able to parallelize the execution of the tests [6].

A complete sample of code instrumentation is provided chapter 7.

4.3 Detection

Detection is the third step in the MA of the design. Its purpose is to determine the final status of the modeled fault: Detected or Non-Detected. Faults which are known as Weak or Non-Activated from the Activation step are not processed here.

During this step, Certitude will execute a partial optimized regression for each fault: test ordering for each fault is dynamically chosen to improve the overall performance and chance to detect a fault with minimal computing effort. Once again, to improve performance, Certitude is able to parallelize the execution of the tests.

4.4 Result Presentation

Certitude provides all necessary features to check the results computed by the model, activation or detection steps. This includes:

- A TCL interface that allows complex manipulation of the data to be performed through built-in query commands and the manipulation capabilities provided by TCL itself.
- The capability to generate an HTML report that allows browsing all results through a convenient GUI-like interface.

These two interfaces are presented later in this paper.

4.5 Iterative Support

Certitude's primary industrial usage is to help verifiers to improve the quality of the testbench, thus the typical usage flow consist of running model, activation and detection.

Once results are analyzed with specific attention to Non-Detected faults, the verifier may add new tests to the testbench or modify existing tests. Certitude can easily integrate these new tests. The user needs to rerun the activation step (only newly added test are run instead of the full regression). Then, detection can either be rerun on the full design (to check all impacts) or on specific faults to validate the capability of the new tests to detect faults.

4.6 Batch System

Certitude is able to parallelize the execution of each individual test during activation or each individual “fault-test” pair during detection. Certitude is a highly multi-threaded system that incorporates a batch system interface that allows distribution of the computational workload on large computing farms. A typical deployment environment in the EDA industry is based on a batch system where common data are accessed over a shared file system e.g. NFS.

Certitude supports hundreds of simulations returning per second and per CPU on its main server.

Figure 1 represents the logical schema used to distribute simulation to a batch system: Certitude dispatches the simulation to the batch system through usage of an execution wrapper (“Execute Wrapper”)

that controls the batched execution of the simulation. The wrapper is handling the communication protocol between Certitude and the running test. When completed, the test calls a specific *cer_simulation_result* command to return its execution status back to Certitude. Thus interfacing existing tests with Certitude is straightforward: the user needs to provide an execution script that:

- takes as input the test case id
- returns the simulation result through the provided command.

The handling of the mutant id is transparent to the user, thanks to the provided API protocol that hide all the internal mechanism.

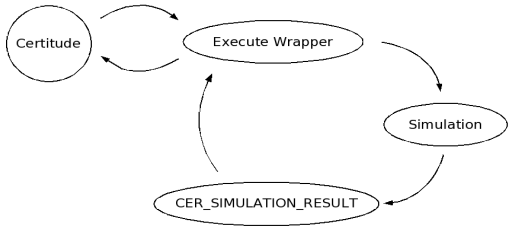


Fig 1 Batch system model

5 TCL Interface

Certitude appears as a TCL shell to the end user. This TCL shell provides a full programming language feature set, including possible user specific extension through libraries, plus Certitude specific add-ons. Besides running the 3 qualification steps and generating a report, the built-in commands allow the user to manipulate the following information:

- full access to fault definition and fault query using more than 15 different criteria
- full access to test definition and test query using more than 10 different criteria
- Activation and Detection status results

Furthermore, through TCL, the user can gain full control over the Batch System interface for debugging purposes as well as for individual simulation runs.

Finally, through TCL, the user can store and retrieve data related to faults, testcases or HDL files in the Certitude database.

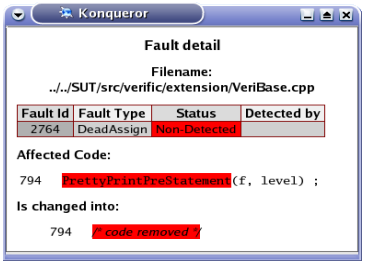
6 HTML Reporting

Certitude can generate an HTML report that allows navigation through all results using a convenient GUI-like interface. The report shown in Figure 2 has dedicated pages that provide:

- Overall status summary
- Fault status per file
- Not Detected fault summary
- Test case summary status
- Individual Test statistics (efficiency, detected fault ...)
- Highlighted source code with color based status showing each fault.
- Fault detail page.

7 Adding Language Support

Certitude is provided with built-in support for external language extension. The extensions are implemented through a general purpose mechanism using the TCL interface. Figure 2 is illustrating the overall API model used for such an extension implementation.



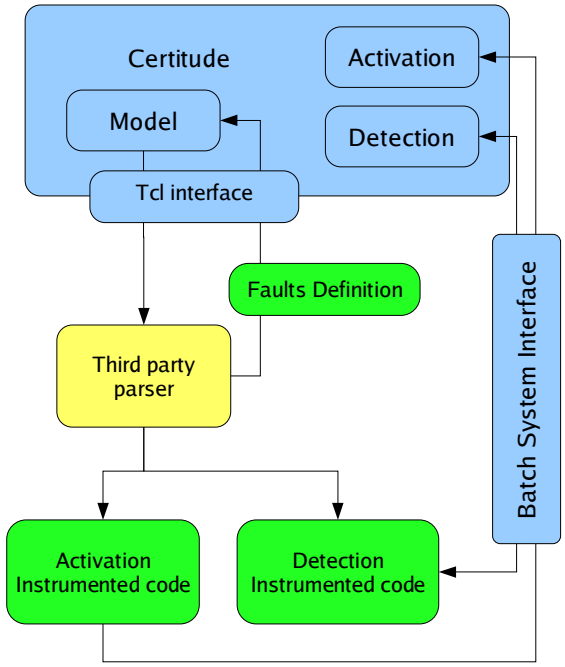


Fig 2 New Language API

In order to add new language support, there are two TCL commands that will form the interface with Certitude. The first command is a model analyzer (defined by the user, following a well defined API) that is capable of parsing each individual source code to extract the fault models. Once a fault is identified, the command needs to:

- Provide Certitude with the relevant fault definition using Certitude's dedicated command
- Generate the instrumented code for the said file for the Activation phase. This instrumented code has the capability to compute at run time the Activation status and the Weakness for each fault. Certitude provides an API (C++) that can be used to interface with the batch system and return the faults status to Certitude (simulation status is still returned using *cer_simulation_result* in the test wrapper).
- Generate the instrumented code for the said file for the Detection phase. This instrumented code is assumed to be a Meta Mutant Schemata [4] that can enable on request each individual fault (A run time API (C++) with the batch system allows the user to have access to which fault needs to be enabled).

In order to make the above explanation more explicit, the following pseudo-code sample is provided:

```
if (a < b) then
    [some statements]
end if
```

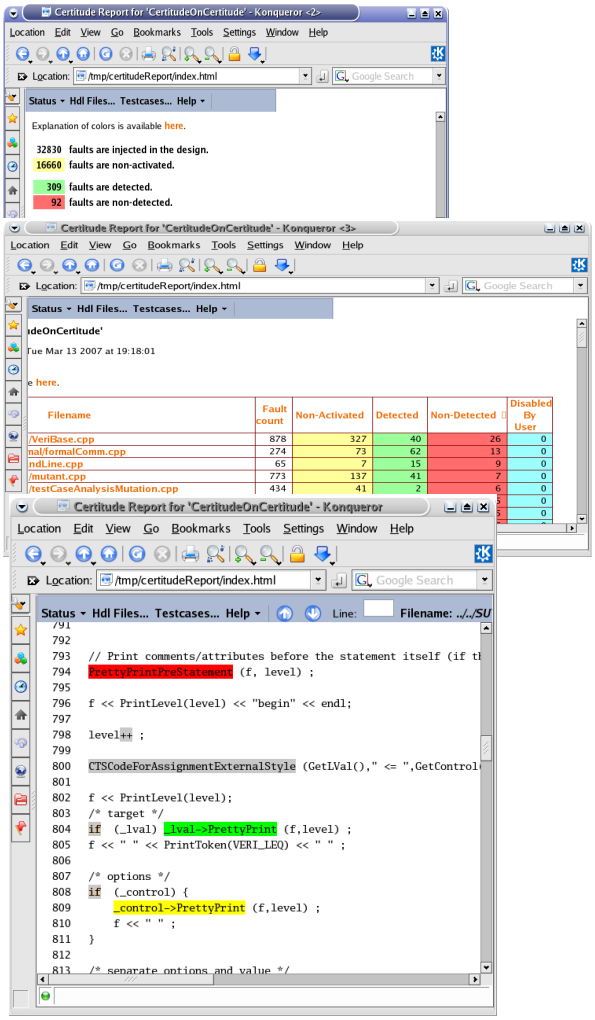


Fig 3 HTML Report

Faults could include “condition true” (1), “condition false” (2) “negated condition” (3) and “operator changes < to >” (4).

Instrumented code for Activation will look like:

```
[mark fault 1,2,3,4 as activated]
if ( a < b != true) [mark 1 as not Weak]
if ( a < b != false) [mark 2 as not Weak]
[mark 3 as not Weak]
if ( a < b != a > b) [mark 4 as not Weak]
if (a < b) then
    [some statements]
end if
```

Instrumented code for Detection will look like:

```
if (    ([fault enabled is 1] and true)
     or ([fault enabled is 2] and false)
     or ([fault enabled is 3] and !(a < b))
     or ([fault enabled is 4] and (a > b))
     or ([fault enabled isn't 1,2,3,4] and (a < b))
   ) then
    [some statements]
end if
```

7.1 Case Study: C++

Certess has implemented a trivial C++ fault operator set in pure TCL using a simple regular expression based parser for C++ in less than 250 lines of TCL code plus 200 lines of interface code with the batch system. This prototype is able to parse Certitude's own source code. It has support for multi-threaded applications and has been deployed to provide a basic mutation analysis on hardware design using SystemC modeling (a layer built on top of C++). Complete implementation took less than one week and is still very functional.

A more complex and robust solution based on a commercial parser is now available. It has an extended mutation operator set and represents one man month of development. The source of this solution is available as a case study to show how to extend Certitude with new language support.

Furthermore, the C++ support presents some interesting additions to Certitude by allowing investigation of C++ language specific mutation models. C++ is a versatile language where the notion of Weakness is more difficult to define compared to the domain specific HDL languages. This is an ongoing area of development at Certess.

8 Extending or Modifying Algorithms

Certitude's processing algorithm can be altered through TCL callbacks where the user can overwrite the standard processing of Certitude. These callbacks are divided int two groups:

- Batch system interface.
- Fault processing.

8.1 Batch system interface

Certitude provides the following callbacks that can be used to replace the existing built-in batch system interface, thus allowing users to have direct connection facilities with a custom batch system:

- ::CerTclCallback::JobSubmitCommand

  Mechanism used to submit a new job to the batch system
- ::CerTclCallback::JobStatusCommand

  Mechanism used to query the job status in the batch system
- ::CerTclCallback::JobKillCommand

  Mechanism used to force cancellation of a job in the batch system
- ::CerTclCallback::SimulationResult

  Mechanism used to implement some post processing regarding the simulation result. The result itself (detected/not detected) can be overwritten.

Besides this process, Certitude has a "fake" simulator where the simulation results are either random (to test new algorithms on random data) or extracted from a pre-determined result matrix (allowing quick testing of the algorithm with sampled data). The matrix is a simple text format.

8.2 Fault processing

Certitudes provides several hooks that can execute specific processing when special events are reached. These are related to the detection phase. Callbacks allow specific processing when the fault status is determined by the application based on the actual status of a specific fault (Fault Not Yet Detected, Fault Detected, Fault Non-Detected).

Other extensions can be used to:

- Change the fault ordering
- Change the test case ordering for each fault

While callbacks are executed, the user can have access to the full interface of Certitude's internal objects through ad-hoc TCL queries.

Note that despite Certitude being multi-threaded, the callback can be executed in a protected TCL environment: the programmer does not need to consider any of the issues related to multi-threaded programming under TCL.

TCL interface callbacks can be extended by Certess support to provide desired APIs to the community if needed.

8.3 Completely new algorithm

Entirely new algorithms can be written under Certitude by taking advantage of single execution mode (one fault/test pair) and the combination of TCL queries. User centric data related to a fault can be

registered in the database through the TCL interface. Still, if the interface is not adequate, TCL interfaces to other databases can be used. Certess has successfully used this approach in the past to experiment with metric computation of the testbench quality.

9 Future Direction

As a commercial product with a growing user base, Certitude is continually incorporating new features and refinements.

Certess is currently :

- extending the supported language base to System Verilog (yet another HDL language),
- deploying result management features based on customer feedback. The target is to facilitate the analysis of the data provided by MA technology for end-users.
- Include fault models for system interfaces, to target MA for system integration testing.
-

In addition to these ongoing efforts, support for additional languages in cooperation with the researcher community would be welcome. This could allow for the use of Certitude in software programming classes where mutation analysis is being taught.

Certess is exploring new ways of exploiting MA results. For example the optimization/reduction of regression suits is of interest to a certain class of users.

Exploring the application of MA to traditional software development is also an active topic at Certess. The non-regression suite for the Certitude product is improved by exploiting the results from the functional qualification of the Certitude product's non-regression suite.

Further extensions to the existing C++ frontend such as the automated removal of equivalent faults are also of particular interest.

The use of MA in embedded software development is another area of interest.

10 Conclusion

Certitude is a robust infrastructure for conducting research in mutation analysis. The tool provides several features for simple extension intended to facilitate research.

Certess can provide access to the Certitude product for research teams or classes which are furthering the field of mutation analysis.

The application of mutation analysis on large commercial systems offers many technical challenges and will ensure that mutation analysis continues to be an exciting area of software research.

11 References

[1] Certess Inc., http://www.certess.com

[2] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, “Hints on test data selection: Help for the practicing programmer” *IEEE Computer*, vol. 11, pp. 34-41, April 1978.

[3] A. J. Offutt and K. N. King, “A Fortran 77 interpreter for mutation analysis", *1987 Symposium on Interpreters and Interpretive Techniques*, (St. Paul MN), pp. 177-188, ACM SIGPLAN, June 1987.

[4] Yu-Seung Ma, Jeff Offutt and Yong-Rae Kwon, “MuJava : An Automated Class Mutation System”, *Journal of Software Testing*, Verification and Reliability, 15(2):97-133, June 2005.

[5] Roland H. Untch, “Mutation-based software testing using program schemata”, *Proceedings of the 30th annual Southeast regional conference*, ACM Southeast Regional Conference, April 08-10, 1992

[6] A. J. Offutt, G. Rothermel, and C. Zapf, “An experimental evaluation of selective mutation," *Proceedings of the Fifteenth International Conference on Software Engineering*, (Baltimore, MD), pp. 100-107, IEEE Computer Society Press, May 1993.

[7] C. N. Zapf, “Medusamothra - a distributed interpreter for the mothra mutation testing system," M.S. Thesis, Clemson University, Clemson, SC, August 1993.