Tallinn Technical University

An Open and Dynamic User Interface to the CAD-system “Turbo Tester”

By

Priidu Paomets

A Master Thesis

Submitted to the Chair of Computer Engineering and Diagnostics of the

Department of Computer Engineering

In fulfillment of the requrements for the

Degree of Master of Science

Computer Engineering

Tallinn

May 1998

Table of Contents

1 INTRODUCTION 1

1.1 THE NEED FOR CAD FRAMEWORKS AND GOOD USER INTERFACES 1
1.2 THE NEED FOR TEST GENERATION 2
1.3 THE NEED FOR AUTOMATED TEST PATTERN GENERATION IN EDUCATION 3
1.4 THESIS LAYOUT 3

2 OVERVIEW OF RELATED WORK 4

2.1 HISTORY AND EVOLUTION OF CAD FRAMEWORKS 4
2.1.1 File-and-Translator Based Systems 4
2.1.2 First Role: Design Database 4
2.1.3 Second Role: Design Data Manager 6
2.1.4 Third Role: Design Process Manager 7
2.2 THE EVOLUTION OF TURBO TESTER 8
2.2.1 Alternative Graphs 10
2.3 TESTING SOFTWARE ON THE MARKET 10
2.3.1 System HILO 11
2.3.2 SUNRISE 12
2.3.3 SYNOPSYS 13
2.3.4 CADENCE 13
2.3.5 LogicBIST 14
2.3.6 Mentor Graphics 14

3 DESIGNING NEW CAD FRAMEWORK 16

3.1 MOTIVATION 16
3.2 REQUIREMENTS FOR EDUCATIONAL TOOLS 16
3.3 CONFORMANCE WITH CAD FRAMEWORK DESIGN PRINCIPLES 16
3.4 DESIGN IMPLEMENTATIONS 17

4 SINGLE-USER GRAPHICAL ENVIRONMENT 18

4.1 MOTIVATION 18
4.2 SYSTEM REQUIREMENTS 18
4.3 HIERARCHICAL ARCHITECTURE 18
4.3.1 Requirements for CAD tools 19
4.3.2 Parameter passing to tools 20
4.3.3 Alternative ways to integrate tools and transfer information 23
4.4 GRAPHICAL USER INTERFACE 24
4.4.1 Command shell console 25
4.4.2 Text editor 26
4.4.3 Customizable toolbars and menus 27
4.5 SIMPLE DESIGN PROCESS MANAGER 27
4.6 CONFIGURABILITY AND EXTENDIBILITY OPTIONS 29
4.6.1 GUI extensibility options 29
4.6.2 Easy CAD tool integration 31
4.7 SHELL AND PROGRAMMABILITY OPTIONS 32
4.8 INTER-PROCESS COMMUNICATION AND AUTOMATION 33
4.9 DISCUSSION 34

5 INTERNET-BASED MULTI-USER ENVIRONMENT 35

5.1 MOTIVATION 35
5.2 HISTORY OF NETWORK COMPUTING 36
5.3 POSSIBLE SOLUTIONS 38
5.4 VIRTUAL LABORATORY 38
5.4.1 Levels of intricacy 39

5.4.2 User-Tool interface ... *41*
5.4.3 Web-server extension ... *42*
5.4.4 Tool registration with central server ... *43*
5.4.5 Service Description File ... *43*
5.4.6 Central Server ... *44*
5.4.7 Security ... *45*
5.4.8 Conclusion ... *45*
5.5 THE FORM-BASED INTERFACE ... 45
5.5.1 System Requirements ... *46*
5.5.2 The architecture ... *46*
5.5.3 Conclusion ... *51*
5.6 THE INTERACTIVE CLIENT ... 51
5.7 DISCUSSION ... 52
6 CONCLUSIONS ... 53
6.1 SUMMARY ... 53
6.2 FUTURE WORK ... 53
6.3 ACKNOWLEDGEMENTS ... 53
BIBLIOGRAPHY ... 54
LIST OF PUBLICATIONS ... 55
APPENDIX A. TT3 CONFIGURATION FILE (TESTER.INI) ... 58
APPENDIX B. TT3 TOOL DESCRIPTION FILE ... 61
APPENDIX C. TT3 DESIGN FLOW DESCRIPTION FILE ... 63
APPENDIX D. TT3 IMPORTER DESCRIPTION FILE ... 65
APPENDIX E. SAMPLE OF THE SDF FORMAT ... 67
APPENDIX F. VILAB PYTHON SCRIPT FILES ... 68

List of Figures

Figure 1 Tools having proprietary data representations. Translation of formats is required for sharing data among tools 4
Figure 2 Tools integrated on top of design database 5
Figure 3 Turbo Tester 1.0 User Interface 8
Figure 4 Turbo Tester 2.0 User Interface 8
Figure 5 Turbo Tester 2.5 User Interface 9
Figure 6 Hierarchical architecture of a CAD framework 19
Figure 7 General command-line format 20
Figure 8 Sample of environment variables 22
Figure 9 Sample INI file 23
Figure 10 Turbo Tester 3.0 GUI 25
Figure 11 TT 3.0 Console 26
Figure 12 TT 3.0 Text Editor 26
Figure 13 TT 3.0 Menu and Toolbars 27
Figure 14 Simple Design Process Manager 28
Figure 15 Design Process Manager's Tool Icon 28
Figure 16 State chart of Design Process Manager’s Tool 29
Figure 17 Design Process Manager Toolbar 29
Figure 18 Mainframe-based distributed computing 36
Figure 19 Client-Server architecture 37
Figure 20 3-tier network computing 37
Figure 21 The Architecture of Virtual Laboratory 39
Figure 22 User-Tool interaction 41
Figure 23 SDF Format’s hierarchy tree 43
Figure 24 VILAB Welcome page 46
Figure 25 Turbo Tester VILAB front page 47
Figure 26 Page displayed on successful design import 47
Figure 27 Selection of VILAB tools 48
Figure 28 Setting properties for a VILAB tool 48
Figure 29 Results shown when tool has finished 49

List of Tables

Table 1 Commertially available test generation tools 11
Table 2 Predefined tool exit codes 20
Table 3 Evolution of network computing architectures... 37

1 Introduction

Everyone has an experience with applications that are easy and intuitive to use. Unfortunately there are also lot of programs that are so complex that it takes ages to start using them, and it may eventually happen that one rejects them completely and tries to find another, which is easier to master.

Historically, first user interfaces (UI) were plain character-mode command-prompt applications and users had to type in commands in order to carry out their job. Later, simple character-mode pseudo-graphical interfaces appeared that allowed users to work more productively, eliminating need for constant typing of commands. Nowadays, a highly interactive[1] graphical user interface (GUI) is a must.

Since the early 1980's, the topic of CAD frameworks has become increasingly popular, both in the research community and in the commercial arena [WOL94]. So what is all that sudden interest about CAD frameworks? An advanced CAD framework can turn collection of individual tools into effective and user-friendly design environment. CAD framework differs from any conventional applications, which are usually monolithic programs, by a set of individual tools that are tightly integrated under the hood of (graphical) user interface.

In this paper, we are going to develop two User Interfaces to a CAD-system called "Turbo Tester". Turbo Tester (TT) is a set of software tools for teaching diagnostics of digital devices and design for testability [JER98]. The first version will be interactive graphical Windows 95™ user interface that allows users selecting tools for a job and setting their properties visually. The second version is network-enabled user interface that allows using educational tools over intranet/Internet.

1.1 The need for CAD frameworks and good user interfaces

With each day, the number of CAD tools is increasing rapidly. From the point of view of an engineer, it is good to have more tools at hand to automate his/her job, but at the same time, it gets very difficult to become a master of all these new tools. It is evident that we cannot increase the productivity of a designer with just increasing the number of CAD tools. We also need to find a way to hide the complexity that accompanies the myriad of new tools. We need a CAD framework that integrates all these tools and hides the complexity of manipulating with them.

We adopt the following definition of CAD framework, as originally given by the CAD Framework Initiative (CFI), the international consortium developing framework standards:

Definition 1.1 A CAD framework *is a software infrastructure that provides a common operating environment for CAD tools*.

CAD frameworks play a role in *building* as well as *operating* integrated design environments. First, a CAD framework has to provide facilities for conveniently

[1] Interactive in this context means that GUI helps user to boost his/her productivity using visualization techniques.

integrating multiple CAD tools into a coherent design environment. It is a basis for *tool integration*.

Second, a CAD framework can support the end-user in conveniently operating the design environment. Being the infrastructure that binds the tools together, the framework is the proper place to incorporate facilities for organizing the design information and managing the design process. A CAD framework is to become the *electronic assistant of the designer*.

From the above description we identify two categories of framework users: *developers* (e.g. CAD tool developers, CAD tool integrators) and *end-users* (e.g. design engineers, administrators, project managers).

To combine all above we can draw following key-points that CAD framework can help with:

- Cut down design time.
- Make the design process less error prone.
- Master the increasing complexity.
- Increase performance and quality.

To provide an intuitive, uniform and user-friendly interface to CAD framework and its tools, we need advanced user interfaces, which do all this. Such UIs should preferably be graphical and interactive.

1.2 The need for test generation

As the complexity of today's integrated circuits (IC), VLSI, ASIC and digital systems based on components increases, the hardware test generatin and fault diagnosis gets more and more expensive and time-consuming. No-one working in this area cannot blindly ignore and overlook the need to carefully test the design against all possible misbehaviours. Thus, the portion of production costs required for testing today is in many cases well over 70%. The difficulty in creating test for new designs, contributes to delays in getting products into the maket.

To overcome problems of testing, new algorithms, methods and software tools are constantly being developed. To cover all phases of testing and fault diagnosis the following procedures are required:

- Test generation
- Fault simulation
- Test quality analysis and fault localization

These procedures could be broken down into more detailed lists, but it is already evident that testing is an extremely important part of IC design. There is not many tools available for public use that allow test generation. For a detailed list of these tools, refer to Chapter 2.3. Most companies develop their own proprietary methods, or use these few available. Furthermore, there is no toolset aimed at teaching of digital test.

1.3 The need for automated test pattern generation in education

Although it would be possible to generate test for tiny circuits manually, it is unthinkable for complex digital systems. All the above-mentioned methods are implemented in software tools called Automated Test Pattern Generators (ATPGs). As already mentioned, there are only few tools available. In order to use them for education, they need to meet following criterions:

- Low cost
- Small footprint (in terms of resource requirements)
- Large collection of different test generation and analysis methodologies
- Open architecture with dynamic customization
- Easy to adopt to actual course needs
- Comprehensive help (preferrably online)
- Easy to use for a newcomer – short learning time

Unfortunately none of these systems have all of the features, and this is a quite good reason for creating a systems that fulfills the requirements.

1.4 Thesis layout

The contents of the thesis is organized as follows.

Chapter 2 gives an overview of provious works that are related to user interfaces and testing. Here one also will find a history of Turbo Tester and a list of other tools on the market that allow for test generation.

Chapter 3 set requirements for design of educational tools and conformance with CAD framework design priciples.

Chapter 4 focuses on the design and implementation of a graphical single-user framwork that runs under Windows 95 operating system.

Chapter 5 discusses multi-user version that can be used over network, namely Internet.

Chapter 6 concludes the work and sets milestones for the future.

2 Overview of Related Work

2.1 History and evolution of CAD frameworks

The history of CAD frameworks starts in the early eighties when people realized that with the growing number of CAD tools, data transfer between them became an ever-growing pain.

The following is a description of aspects of CAD framework design.

2.1.1 File-and-Translator Based Systems

At the time of CAD framework appearance, tools typically used proprietary and tool-dependent ASCII or binary formats to represent the design data. Communication between tools was possible only if a translator for the respective formats was available (see Figure 1). Bringing the growing number of tools together into an integrated design environment involved writing large number of translators. The emergence of de-facto standard formats, allowing the number of translators to be reduced, eased this pain a bit.

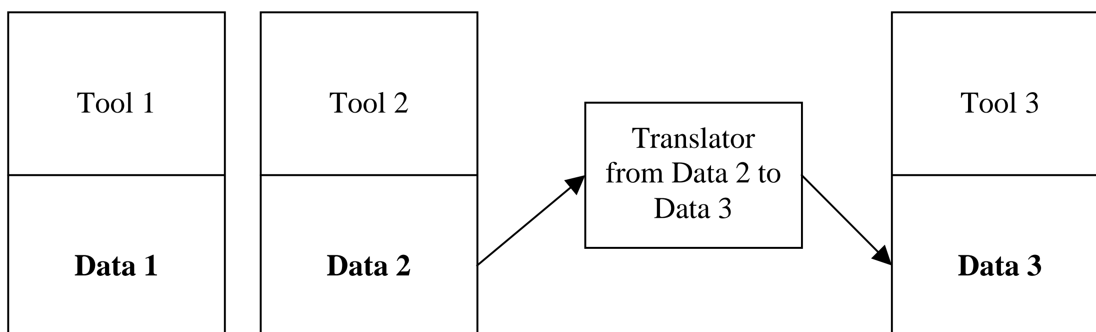

Figure 1 Tools having proprietary data representations. Translation of formats is required for sharing data among tools

These simple integration facilities can be termed the first primitive CAD frameworks.

2.1.2 First Role: Design Database

The first step towards the better integration was to create a *common data repository* for tools that share the same data format. This eliminated the need for duplicate data for each tool. For tools that have not been written to operate directly on the common data repository, reformatting may be performed on input and output. The common repository has come to be called 'integrated design database', or just '*design database*'. It is the key to *tool integration*, and in particular to *tool interoperability*: the ability of CAD tools to communicate and share data (see Figure 2).

Advantages of this approach are increased consistency and increased efficiency. For the end-user there is increased convenience as his/her design data is stored in a structured way in a single place. Additional database utilities support the end-user in managing his/her design data.

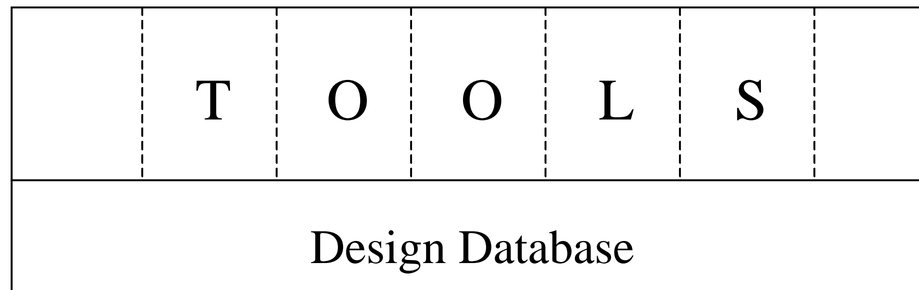

Figure 2 Tools integrated on top of design database

2.1.2.1 File Based Systems

In many CAD systems the design database is a file based system, implemented as a small layer on top of the host operating system. This layer augments the operating system with specific support for managing the design files. Design files are stored in a structured file organization, for example, employing a hierarchical file system and pre-defined naming conventions. Access methods provided for storage and retrival of design data.

2.1.2.2 Use of Conventional DBMSs

Another approach has been to employ conventional (record-oriented) database management systems (DBMSs) to fulfil the role of design database. The conventional DBMSs are typically used as storage and *transaction* processing components in business applications. The interest in DBMSs was driven mainly by the appearent 'keyword-level' match between DBMS functions and facilities and the emerging requirements for design databases. A DBMS provides mechanisms for the reliable storage of data, including recovery facilities, it protects data from unauthorized access, and it provides concurrency control and integrity maintenance.

Drawback of this approach is the commertial and business orientation of conventional DBMSs which does not specifically address the problems encountered in a design environment. There are no known successful applications of classical business DBMS for the purpose of a CAD framework design database, and we can consider this to be a dead end.

Successes have been reported in the application of conventional DBMSs for meta data handling only, that is, to hold administrative data including references to the actual design data stored in files.

2.1.2.3 Object-Oriented DBMSs

The database community has recognized the fundamental nature of the requirements posed by engineering applications and other new database applications such as multimedia databases and knowledge bases. Work was started on next-genaration DBMSs targeted at these applications: *object-oriented DBMSs* (OODBMSs). In contrast to conventional DBMSs, object-oriented DBMSs offer far more flexibility for handling highly interrelated data of different granularities on which different types of accesses are performed. For engineering applications, the DBMS must be capable of handling many different data types and large number of instances of each type. Moreover, flexibility must be provided for modifying or extending the conceptual schemas to match the different views of data operated by different application programs.

Databases typically found in Artificial Intelligence (AI) systems tend to provide flexibility for many different data types, which may also be defined dynamically. These systems, however, have not been geared towards the large number of instances of each type, as found in engineering applications. More-over, they typically are single-user systems. They may be used for purposes of prototyping, but are not suitable for production systems.

The data models provided by powerful OODBMSs permit arbitrary types of design information to be represented and accessed conviniently. Typically the deisgn information is modeled as a web of highly interrealted objects, granularity of which may vary significantly; from a simple integer attribute to a complete design file.

A number of OODBMSs have been realized and have become the base components of some of today's frameworks. These developments signal a move by the EDA software industry away from proprietary design databases to the use of standard commertially available OODBMSs.

2.1.3 Second Role: Design Data Manager

The second role is the use of *design data manager*. A design database, in the sense of common data repository, provides a facility for storing the design data, but provides no support for *managing* the data. A design data management system uses knowledge of the structure and status of design information to provide management support and enforce constraints on the design process.

Clearly, system integration can be, and should be, much more than the definition of common formats for the purpose of tool communication. With the tool being interfaced to the framework, common data management services can be identified and incorporated into the framework. Such data management systems may, for example, organize design information across representations, provide versioning capabilities to support evolutionary designs, support consistent operation on hierarchical designs, control concurrent access and facilitate teamwork. A user interface may then be added to enable the end-user to interact with the system to get informed about the structure and status of his design. "Which copy is the latest version?", "Has this layout been extracted since it was updated, and if so, which circuit description was derived from it?", "If I change this layout, which other parts of the design will be affected?". It is the ability to answer such questions that differentiate a true data management system from asimple data repository.

Many of the data management solutions presented in today's CAD systems appear to be ad-hoc extensions rather than a coherent set of facilities based on a well-defined conceptual foundation. This is illustrated by missing features or serious restrictions found in these systems. For example, in some systems design hierarchies can be traversed only in the downward direction or design hierarchies have to be isomorphic across representations. Data management solutions adding value to today's CAD systems do exist, but too often they lack important functionality, are too slow, or miss required flexibility.

2.1.4 Third Role: Design Process Manager

The third major role allotted to CAD frameworks is that of design process manager. With the increasing number of tools found in today's CAD systems, there is a growing need to support the design engineer in correctly executing these tools to perform his design tasks. Framework services may help the design engineer in correctly invoking the individual tools, as well as provide support for executing the tools in the correct order, according to a pre-defined design procedure. A next step is to have the framework automatically execute design tools, for example, when valid output data is required for a subsequent tool run or the verification status of the design is to be enhanced. Ultimately, design process management facilities will help offer a design environment in which the framework actively supports the design engineer in meeting his design goals.

The terms *design methodology management* and *design flow management* are both used to denote framework servicies that help the design engineer to correctly perform design activities according to a pre-defined design procedure. The term *design methodology* is typically used to refer to the definition of a design procedure in terms of abstract design stages, such as 'circuit design', 'circuit verification', and 'layout design'. The term *design flow* typically refers to the definition of a design procedure in terms of individual tools and dependencies between tools.

Design flow management is one of the key framework topics today. As demonstrated by several prototypes, a pre-defined design flow can be presented graphically to the end-user, to actually guide him through the design process. On some aspects of design flow management there appears to be quite natural agreement: a design flow can be represented by a graph describing how data can flow from one tool to the other, and validity of data can be indicated in such a graph with appropriate primitives. At a more detailed level, however, the known approaches differ significantly in their capabilities; for example, in the types of tools that can be supported and the restrictions put on data management functions.

Many more developments can be expected in the area of design process management. Design flow management is now primarily concerned with correct use of data and the execution of tools in the correct order, but higher level facilities for design methodology management and design task scheduling will certanly follow. Further, more emphasis will be put on design decision support services, which will help design engineers in their decision-making processes, for example, by estimating the cost related to alternative design solutions and checking these against the design objectives. New developments are also expected in the area of *project management*. These will include computerized facilities for the planning of projects, selection of tools and methodologies, allocation of task to team-members, evaluation and presentation of project progress, etc.

2.2 The evolution of Turbo Tester

In the late 70s it become evident that in order to teach diagnostics and design for testability, a tool or set of tools that let students to get aquanted with different approaches of test generation was needed. First tools from diagnostics group of TTU appeared in early 80's and soon formed a package that was called "Turbo Tester".

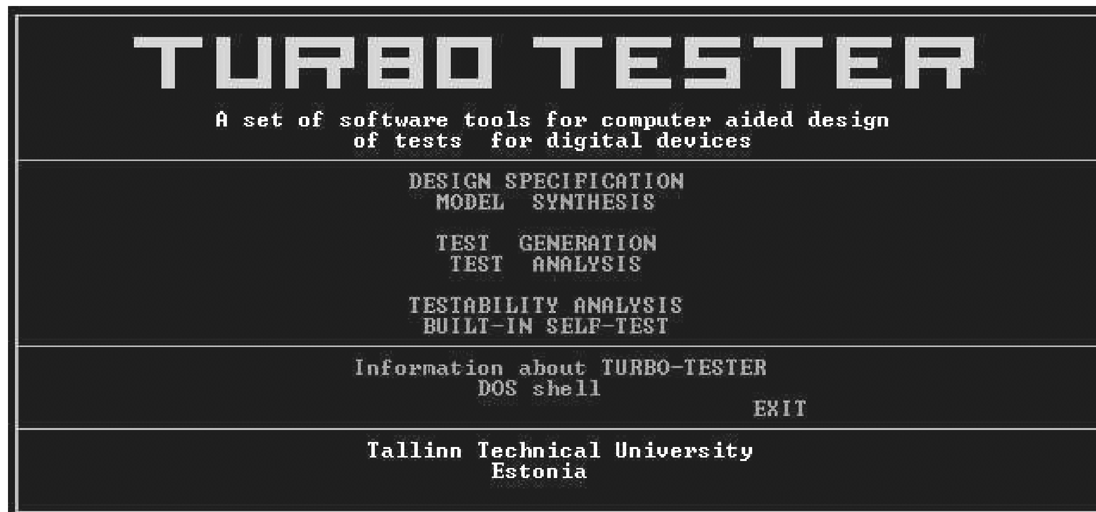

Figure 3 Turbo Tester 1.0 User Interface

The initial version did not contain much tools, just those most needed. The integration of tools was not a relevant topic either – TT was simply a collections of tools. In following years TT got more tools and their algorithms were improved. TT become guite popular and was even integrated into small microelectronics CAD system "DixiCAD"[2].

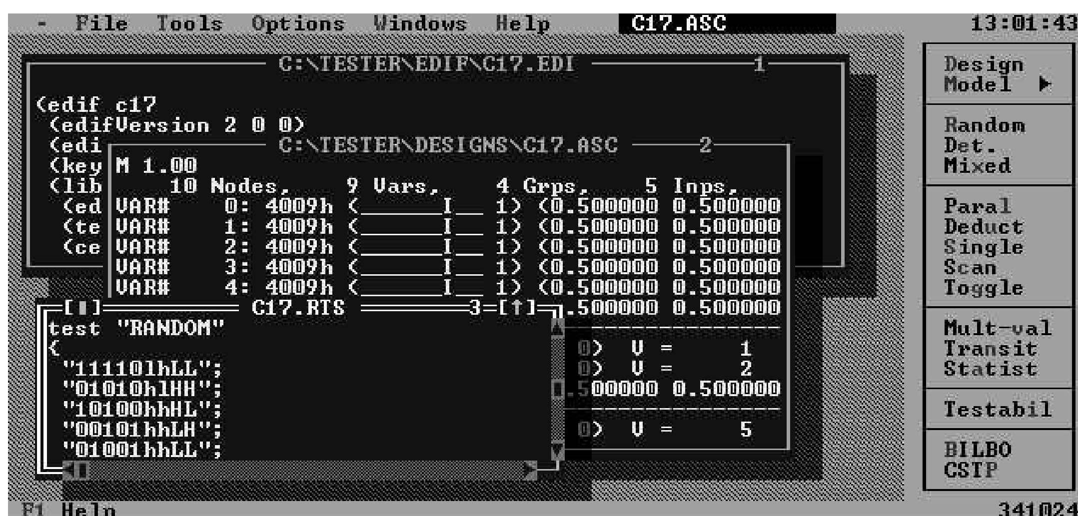

Figure 4 Turbo Tester 2.0 User Interface

In the 1993, when I joined the TT group, it had a quite bulky user interface. We started with development of a new user interface that was easier to use. It had a version number 2.0. Two years later, in 1995, anew, redesigned user interface emerged, to to make it

[2] DixiCAD is an educational CAD-system for microelectronics design and IC production. DixiCAD is developed by DIGSIM DATA AB, Sweden.

more intuitive and even easier to use. In this version (v 2.5) the customization features were added (see Figure 4).

Figure 5 Turbo Tester 2.5 User Interface

Turbo Tester v2.5 consists of following tools:

- **Automated Test Pattern Generators**: random, deterministic, mixed
- **Fault simulation for combinational circuits**: parallel critical path tracing, deductive fault analysis, single/multiple fault simulation, toggle test analysis.
- **Dynamic fault analysis**: multi-valued simulation (hazard analysis), fault cover calculation for dynamic testing
- **Test set optimization**
- **Testability analysis**: calculation of controllability and observability measures
- **Built-In Self Test**: Built-In Logic Block Observer (BILBO), Circular Self-Test Path (CSTP), Store-and-generate

Turbo Tester can work on Gate lavel as well as on macro level, it can model stuck-at 0(1) as well as delay faults. TT has interfaces to most of the commertially available VLSI design tools, including Synopsys, Cadence, Mentor Graphics, Viewlogic, Compass, OrCAD, ASYL+, DixiCAD, SOLO1400.

TT runs on MS DOS environment and several tools are available on UNIX workstations also. It is lightweight and easy to learn.

On the basis of Turbo Tester software an advanced laboratory course has been developed whose aim is to teach and train students to integrate design and test, and to give them knowledge on how to create testable designs with self-testing capabilities and how to obtain test patterns of higher quality [IVA94]. The course has been introduced into the curricula of Tallinn Technical University and has gained a great popularity among students. The course has received good credits from the students of Michigan State University, the software has been used in laboratory training in universities in Finland and Sweden, and it has been recommended for teaching design for testability in universities of United Kingdom.

2.2.1 Alternative Graphs

All Turbo Tester tools are based on the same common mathematical units – Alternative Graphs (AG). Alternative Graphs are based on a graph theory and are defined as follows:

Definition 2.1 In general case, an AG that represents function $y = F(X)$ is a directed, non-cyclic graph $G_y = (M, \Gamma, X)$ with set of nodes M, single root node $m_0 \in M$ and relation Γ in M, where $\Gamma(m) \subset M$ denotes the set of successor nodes of m. Non-terminal nodes m for $\Gamma(m) \neq \varnothing$ have variables $x_i \in X$ as labels. Terminal nodes for $\Gamma(m) = \varnothing$ have variables x_i, functional sub-expressions of $F(X)$, or constants as labels. Let $x(m)$ be the label of node m. In graph G_y, for all non-terminal nodes m for which $\Gamma(m) \neq \varnothing$, a one-to-one correspondence exists between the values of label variable $x(m)$ and the successors, $m_k \in \Gamma(m)$ of m.

AGs serve as a mathematical basis for solving a wide spectrum of test design tasks, resulting in a uniform model and a restricted set of standardized procedures (horizontal universality). They also allow a uniform approach to design at different system levels (vertical universality).

Since AG format is common for all tools found in TT, it also serves as a common design database, as described in Chapter 2.1.2, thus elliminating the need for another data repository format.

2.3 Testing software on the market

There is a number of software products available, and we give brief overview describing their features and drawbacks in educational use.

There are two types of test generation software:

1. Native Test Generation and simulation packages
2. Test Generation is a part of the package (one of its functions)

For teaching digital test both types could be used, but only the first one provides enough flexibility and thoroughness. Type 2 testing tools are part of some larger system and are tuned for the specifics of the host environment.

Name	**Type**	**Yearly Fee (EEK)**[3]	**Licence Price (EEK)**[4]	**Max. Design Size (Gates)**	**Installation Size**
System HILO	**1**	**8500**	**3454**	**1 000 000**	**100 MB**
SUNRISE	**1**	**n/a**	**n/a**	**n/a**	**n/a**
SYNOPSYS	**2**	**25900**	**26000**	**Limited by available**	**300 MB –**

[3] Large systems require a yearly maintenance fee to keep up with support.
[4] Prices are taken from Europractice list for educational use. They mark the price per single workplace (license). Real, commercial prices are 10 – 1000 times higher that of educational ones (Synopsys for instance 1,5 – 3 Million EEK)

				virtual memory[5]	**1GB**
CADENCE	**2**	**10200**	**17270**	**Limited by available virtual memory**	**1GB**
logicBIST	**1**	**n/a**	**n/a**	**n/a**	**n/a**
Mentor Graphics	**2**	**17270**	**25905**	**n/a**	**n/a**

Table 1 Commertially available test generation tools

As shown in Table 1, only System HILO and Sunrise would be the choices for full-scale digital test education. For more information on each of the tools, see following sub-sections. logicBIST is also Type 1 tool, having award-winning tools, but it provides only BIST methods.

The prices shown in the table demonstrate vividly that cost per seat is quite high. If we would like to teach a course on digital test, we would need approximately 20 workplaces. Simple arithmetic shows that this yields cost for a class from 69080 EEK for System HILO to 520000 EEK for Synopsys. Please note that prices for Sunrise are not known at the time of writing this document and so is the fact whether it has educational licenses, since it was recently acquired by Synopsys.

Price is not the only factor that should be considered. Usually large packages come on multiple CDs, and when installed they have a footprint of several gigabytes. Not every institution is ready to use and maintain such systems. When systems are running they tend to use a lot of machine’s resources, thus limiting the effective number of users that a machine can handle. The practice shows that to run Cadence package simultaneously for 20 users, it would require a server with minimum of 1.2 GB of RAM. Such an intriguing amount of needed resources comes from the fact that systems of Type 2 have many other tools besides ones needed for testing purposes. As these frameworks are tightly integrated its practically impossible to take out individual tools from the host environment to work with.

Table also shows that each of these packages needs a yearly maintenance fee to have technical support from the vendor.

2.3.1 System HILO

System HILO is a suite of design analysis, verification and test generation tools that form a core simulation resource for PCB, ASIC and system designers working within most popular frameworks.

System HILO consists of a set of integrated modular programs, each module executes from a menu-driven user interface.

[5] Design sizes that can be tested using Synopsys Test Compiler are limited to available virtual memory. The following equation applies to memory requirements: 16,3MB + (5,9MB * number of Kgates)). It has been verified that on the system with 1GB of swap space it can test combinational circuits with 700Kgates, and sequential circuits with 200Kgates.

HILO is one of two native testing environments in our overview. It has a number of tools, pros and cons of which are outlined below.

HILO has following components:

- Faultless simulation
- Delay simulation
- Stuck-at fault simulation
- ATPG for combinational circuits, scan
- Interactive test generator for sequential circuits
- VHDL simualtion and debugging
- High-level simulation

OS Support:
Sun SPARC, HP9000/7xx
Advantages:
HILO is cheaper to purchase and maintain than other systems in our overview.
Disadvantages:
Although HILO has many tools, it still lacks automatic test pattern generator for both combinational and sequential circuits. It has support for VHDL designs with a lot of restrictions. One common problem that is present in most of packages is the need for separate component libraries for each technology.

2.3.2 SUNRISE

Sunrise is a specialized toolset for test development for digital integrated circuits. The solution includes scan loginc insertion, IEEE 1149.1 boundary scan support, testability analysis, design rule checking, Automatic Test Pattern Generator (ATPG), fault simulation and test vector post-processing.

Sunrise consists of tightly integrated set of tools including:

- **START** - Sunrise's Testability Analysis and Rule Checking Tool
- **TestGen** – ATPG for full and partial-scan methodologies
- **FaultSim** – fault simulatior for grading existing functional test vectors
- **SHERLOC** – identifies and isolates circuit defects following tester mismatch
- **Interfaces** – netlist and test vector interfaces to popular EDA platforms and ASIC vendors
- **Library Builder** – creates custom macrocell libraries for use with Sunrise tools

Only Sunrise is powerful enough to provide all necessary functions needed for teaching of digital test. Sunrise is practically de-facto standard in the area of digital test.

OS Support:
Sun SPARC
Advantages:
Most comprehensive set of testing tools for both scan and partial-scan designs.
Disadvantages:
Sunrise is quite expensive and also requires separate component libraries.

2.3.3 SYNOPSYS

The Synopsys package will allow the designer to explore implementation options at the behavioral level to determine the optimal architecture before committing to a specific RTL implementation. Designing at the behavioral level inherintly reduces design complexity, thereby reducing specification time and simulation runtimes while improving design quality.

It consists of following items:

- Design Compiler Expert
- Design Analyzer
- FPGA Compiler
- Test Compiler
- HDL Compilers
- Library Compiler
- Simulation tools
- Behavioral Compiler
- Floor Plan Manager

Synopsys is a great package for IC design synthesis from high-level descriptions. It allows for technology mapping, simulation and other features vital to IC development. Test Compiler component allows to insert scan paths, it includes ATPG for combinational and scan logic, plus it features fault simulator.

OS Support:
Sun SPARC, HP9000/7xx

Advantages:
Integrated testing component makes test generation and built-in self-test circuit insertion an easy and straightforward task, due to fact that all design infromation is at hand all the time. Moreover, testability measure could be one of the factors in evaluating different layout and routing alternatives. Synopsys' Test Compiler is also very comfortable to use.

Disadvantages:
Synopsys may be ideal for IC designers, but it's too large and not directly applicable to teaching of digital test. It is also really expensive, too expensive for teaching digital test only.

2.3.4 CADENCE

Cadence is a huge set of tools for design entry, simulation, synthesis, IC design and more for both digital and analogue circuits. It consists of following packages:

- **ASIC Front End Package**. It provides core set of Cadence design entry, simulation and synthesis tools for digital circuits.
- **IC Package**. IC Package provides high-performance tools for each step of the IC design process from architectural definition through detailed structural implementation.

- **Systems Package**. This package supports design capture, simulation, implementation in PCB and MCM and routes to third party implementation tools for FPGAs.
- **Alta Package** is a high-level design environment for communications, networking and multimedia applications.
- **Dracula Package**. Dracula III is a complete system for verifying ASICs.

All packages except for Alta, support test generation tools. The ASIC Front End is the cheapest (licence costs 5 times less than that of Dracula's).

Cadence test generation tools include:

- Verifault-XL fault simulator
- Test synthesizer

OS Support:
Sun SPARC, HP9000/7xx
Advantages:
Advancages, described for Synopsys also apply here.
Disadvantages:
See for details Synopsys section.

2.3.5 LogicBIST

logicBIST is a product of LogicVision and is aimed at built-in self-test generation methodologies. It provides a complete, automated embedded ATE solution for at-speed testing of complex, high-speed, multi-frequency, multi-clock logic. logicBIST automatically generates embedded ATE core in form of synthesizable RTL code (VHDL or Verilog) These soft cores provide the neccesary hardware functionality for generating test patterns on-chip, delivering these patterns at-speed to the logic under test, and collecting the results, as well as providing IEEE 1149.1 TAP and boundary scan implementation.

OS Support:
N/A
Advantages:
This award-winning software has very sophisticated tools for BIST generation.
Disadvantages:
Since this set of tools is aimed at BIST methods, it does not include any other methods for test generation thus being incomplete in terms of educational needs.

2.3.6 Mentor Graphics

Mentor Graphics package is available in 3 different design flows:

- IC-Statcio Flow
- PCB/MCM Flow
- Interconnect Synthesis Flow
- And additionally Leonardo for PC or workstation.

Common to all these flows are: schematic entry and digital simulation, system design with VHDL/Verilog, digital, analog and mixed signal simulation plus the synthesis tool. It also includes DFT and fault analysis software, as well as print servers. Mentor Graphics has tool called FlexTest for Automatic Test Pattern generation for sequential circuits on gate level.

OS Support:

Sun SPARC, HP7xx, Windows NT(some tools)

Advantages:

Same as for Synopsys and Cadence.

Disadvantages:

Most of the same apply as for Synopsys and Cadence. In addition Mentor Graphics does not support built-in self-test features.

3 Designing new CAD framework

In this chapter, we will look at the requirements set to educational software tools and CAD frameworks, as well as directions in which the system could be extended.

3.1 Motivation

The most important point in the motivation is the lack of CAD systems enabling teaching of digital test. Turbo Tester was brought to life to fill that gap. Since its first incarnation, many things have changed, but one aspect has remained unchanged – the programming model of TT tools.

Starting with re-implementation of all TT tools based on new architecture, a need for new user interface emerged. Latest version of TT UI was character-mode program that mimicked multitasked environments in very childish form. These days most operating systems provide graphical environments with many new features not available before. Some of these include visualization of data, visual-programming interfaces, multimedia applications, and virtual reality - to just name a few.

3.2 Requirements for educational tools

In order to use some sort of software for educational purposes, it must confirm to certain requirements:

- Easy and intuitive to use user interface
- Very flexible architecture in terms of configurability and extensibility (open and dynamic)
- Report generator (not required but useful)
- Comprehensive tutorials and on-line help
- Smart CAD framework minimizing the risk of error and preventing the tools being executed in wrong sequence.
- Programmability options to provide means for batch works in experiments.

3.3 Conformance with CAD framework design principles

Let us first see how current Turbo Tester environment confirms to roles outlined in Chapter 2.1:

- **Role 1: Design Database**. As already mentioned earlier, all TT tools share common design format – AG format. This is a simple text file, which can easily be read and written.
- **Role 2: Design Data Manager**. This role is not available in the current implementation of TT. Since TT is a small package and is aimed mainly at education, it probably will never be implemented.

- **Role 3: Design Process Manager**. This is the area where TT can be extended. It would be very useful to have a design flow management to suggest alternative design flows to current task and guide end-user through steps of the flowchart. Setting parameters for each tool could also be simplified.

3.4 Design implementations

The easiest solution to a CAD framework that comes in mind is the ordinary single-user application, which runs on a local machine. In Chapter 4, we will develop such system with rich visual interface. Larger systems allow multiple users to use its tools by installing software centrally on a server and having license management to enable (and limit) the amount of concurrent users (according to the license agreement with the vendor). The same methodology could be used in smaller systems as well.

One of the areas where the functionality of a CAD framework could be extended is network computing, the hottest topic in these days. The simplest form of network computing is to provide a form-based interface to CAD tools via WWW. Network computing also diminishes the burden around portability issues. One can find out more about this in Chapter 5.

4 Single-user graphical environment

In this chapter, we will develop a single-user environment with a rich visual interface and simple Design Process Manager.

4.1 Motivation

As an outcome of this work, we will have a complete working CAD framework for teaching digital test and design for testability. Current implementation is single-user version, but considering its small size, this hardly becomes any obstacle. To overcome this limitation and add even new dimensions to CAD frameworks, we design an internet-ready system, in Chapter 5.

Current implementation has following features:

- Hierarchical architecture
- Graphical User Interface
- Simple Design Process Manager
- Openness, allowing new extensions
- Dynamic by means of configurability
- Programmability options
- Inter-process communications (automation)
- Integrated GUI widgets (waveform viewer, AG editor, test pattern editor)

The framework is written in Delphi 3.0 Professional and is available for Windows 95/98/NT operating systems.

4.2 System Requirements

These are the requirements needed to install Turbo Tester 3.0:

- Windows 95, Windows 98, or Windows NT
- 16 MB RAM
- 20 MB HD

4.3 Hierarchical architecture

On the lowest level of each CAD framework are CAD tools that carry out the actual work (see Figure 6) Each tool can be executed from the command prompt of the underlying operating system. Parameters can be passed via command-line, but other alternatives do exist.

The next level is formed by command a shell, which automates some parts of tool execution (and management), and adds programmability options (described in Chapter 4.7).

On the highest level is a graphical user environment which completely hides the complexity of any underlying levels, plus adds simple design process management option (described in Chapter 4.5)

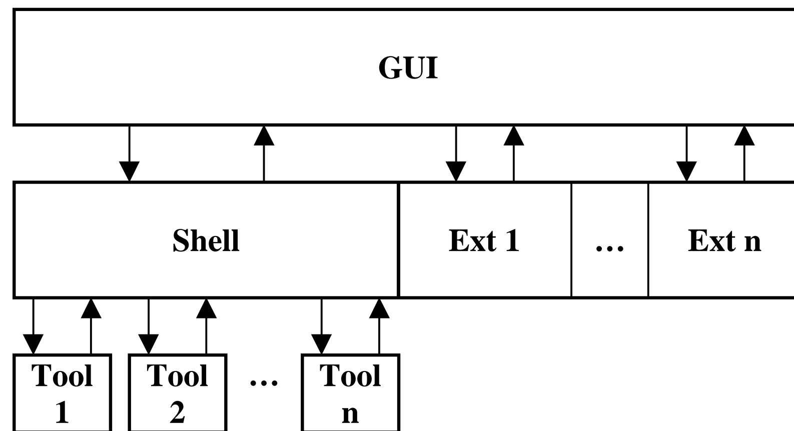

Figure 6 Hierarchical architecture of a CAD framework

The Figure 6 clearly shows that GUI has a full control over shell and any plug-in extensions it hosts. Shell in turn has a full power over CAD tools. In order for GUI to communicate with a tool, it has to marshal it through shell, which mediates messages sent.

4.3.1 Requirements for CAD tools

A lot of work has been done to investigate easy integration of existing tools into a new framework. There are only four requirements that apply:

1) A tool must have no graphical interface elements, i.e. should be command-line a tool.

   Tool should not interfere with the rest of user interface. UI redirects standard output and error streams to a specialized console window, and it would be annoying for the user when out-of-the-context dialog boxes pop up.

2) All output messages must be sent to operating system's `stdout`.

   Sending standard output stream to `stdout` allows standard way to flexibly redirect these messages to GUI widgets and operate upon them.

3) All error messages must be sent to operating system's `stderr`.

   Reason is the same as described above, but used to display error messages

4) Exit codes must comply to following values:

Exit Code	Description
0	Successful completion, no error.
1	Ctrl-C/Ctrl-Break pressed
2	Hardware failure
3	Out of memory

4	Out of resources (other than memory)
5	Missing configuration file
6	File not found
7	Path not found
8	Error in command-line processing
9	License not applicable
10	Help screen (usage). Displayed when user executes tool with -?, -h or –u switches
…	Any developer-defined meaning
255	Anti-debug stop (debug anti-tamper behavior)

Table 2 Predefined tool exit codes

Only the exit code 0 indicates successful completion. All other values indicate somewhat problem, severity of which depends on the meaning.

4.3.2 Parameter passing to tools

In order to specify what the tool should do, we must somehow pass information to it. The most common way is to pass parameters via command-line at the moment of tool execution, which is also the easiest way. Other alternatives would be using of environment variables, standard input (`stdin`) stream, or intermediate files.

4.3.2.1 Passing parameters via command-line

Let us consider the following sample command-line:

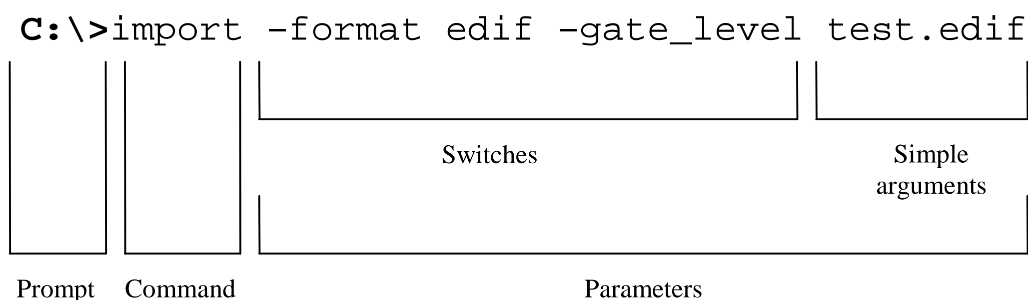

Figure 7 General command-line format

The general command-line format has three distinct parts:

- **Prompt** – The command-shell identifier of the location (context) user is currently in. It may display path to currently active directory, name of an application or something else. Its presence is not mandatory; it only carries an informational meaning.
- **Command** – The name of shell's internal command or external executable file. In UNIX systems, executable file is identified by the *'executable'* flag provided by

operating system's file system. In Windows, including DOS, executable files are identified by their extension (.EXE, .COM, .BAT). When executing a tool, this extension may be omitted from the command-line, thus having the same look and feel as on UNIX-based systems.

- **Parameters (arguments)** – they are used to specify what the application should do, and how. Typically there are two kinds of command-line parameters:

  - **Simple, positional parameters**

    Simple are the parameters that appear on the command-line by their own, and whose position (relative to others) is relevant. The position defines its meaning. Simple parameters are most common in small applications, and are often used to specify file names. In our case, the `test.edif` is a representative of this type of parameter.

  - **Switches**

    Switches modify the behavior of the tool. They are so-called "named parameters", whose position is NOT relevant. Switches are usually prepended with '`-`', '`+`', '`/`', or '`\`' character, and if present, define behavior, different from default.

    In our sample command line, there are two switches: `-format edif` and `-gate_level`. Assuming, that command `import` tries to read in a design, format of which is alien to it, a conversion has to be made. First switch specifies the current format of the design, and second details the output.

    Switches can carry information in several ways, form of which depend on the context. They may or may not have an additional argument attached to it (as in case of '`-format edif`'), specifying an alternative from a list of options.

    It is recommended that switches have descriptive names (`-format edif`), instead of cryptic ones (i.e. `-fe`, `-f e`), to improve readability and lessen the risk of misspelling.

There is a drawback to this approach: the number of characters that can be passed via command-line is in most operating systems limited to 127, thus rendering it impossible to pass large amounts of information. Developers should be careful when designing names and types of parameters.

The good news is that application can easily determine the number and contents of each argument, whereas in case of some other approaches it gets quite difficult.

4.3.2.2 Passing parameters via environment variables

Most operating systems provide a pool of information that can be used to hold common data. This pool is called *environment*. Its size on UNIX-based and Windows-based systems is dynamic and is limited only to the amount of available virtual memory; on DOS-based systems it is set a-priori, and is in range from few hundred bytes to several kilobytes.

Technically speaking, the environment is a set of key-value pairs, with no grouping options i.e. all applications share the same pool:

```
SHELL=/bin/tcsh
TERM=vt100
HOSTTYPE=sun4
VENDOR=sun
OSTYPE=solaris
MACHTYPE=sparc
SHLVL=1
HOST=pitsa
REMOTEHOST=warp9.pld.ttu.ee
WINDOWMANAGER=/usr/openwin/bin/olvwm
```

Figure 8 Sample of environment variables

As already mentioned, all applications share the same pool of data, thus any of these application can set, reset and change any of these variables, providing universal means for inter-process communication. There is a problem of security when handling environment variables. As any application has an access to any variable, nothing prevents from modifying wrong variables, unless standard interpretation is defined. It may very well happen that two different applications are using the same name for a variable, but have a totally different interpretation of what it means (called *namespace collision*). There is no way to figure out what application set what variables, as there is no way to prevent tampering of values of variables by some other application.

As a conclusion, we can say that, environment variables let us transfer larger amounts of data, but applications have to know under what name and to be prepared for unexpected values.

4.3.2.3 Passing parameters via standard input stream

Most operating systems provide such general-purpose data streams as *stdin*, *stdout* and *stderr*. STDIN is a standard input to read data from, which usually is connected to keyboard. STDOUT is standard output stream, which is usually sent to console. STDERR is a standard stream to output error messages that are also normally sent to console. Each application has means to redirect all or some of these to some other device or file. Following command, for example, concatenates output stream of one application to the input of the other via the pipe ('|'): '`ps -aux | grep john`'. Pipes in software act analogous to their real-life counterparts – what is put in at one end, comes out from the another.

There is no limit on how much data can be sent to an application via STDIN, but often it is used in redirection commands of the operating systems, thus making it inappropriate for most cases.

4.3.2.4 Passing parameters using intermediate files

There is one old and proven method for data transfer – a file. It allows large amounts of data to be transferred from one place to another. Both sender and receiver must know the format of the file and they must have a common location (with read-write permission) to place the files.

One simple platform independent file format is INI file. It has similar structure to environment variables, but allows them to be grouped into sections:

```
[General]
FirstName=Nipi
LastName=Tiri

[Contact]
Phone= 12-345-678
Email=nipi@kodu.xx
Fax= 23-456-789
```

Figure 9 Sample INI file

In addition, each application can create their own INI file, thus efficiently removing the problem of unintended changing of values by other applications.

The approach of using files is slow because the need for file creation, opening and closing. Each of these operations takes time, which is often what we just do not have.

The suggestion is that files be used for holding design and other common data, not for passing parameters to tools.

4.3.3 Alternative ways to integrate tools and transfer information

So far, we have looked at the ways to integrate CAD tools that are in separate, external, executable files. While it gives us easiest transition from old system, it has significant disadvantage in terms of speed. When calling external application, a new process has to be created wasting time and resources. What's the alternative, one may ask. The answer lay in dynamic libraries, modules that can be loaded dynamically as required and unloaded from memory when no one needs them. Luckily, both UNIX (.so) and Windows (.dll) systems support them.

Dynamic link libraries have a set of interface routines that are available to outer world. Application that loads such library dynamically can call any of these routines. This approach makes it easy to transfer data back and forth. There exist two types of DLL loading a) automatic, at the moment of program startup b) manual, at any time during program execution. Automatic loading takes place when the library is explicitly defined in the source file of the program using it:

```
procedure Test(S: PChar; X: Integer); stdcall; external 'usrlib.dll'
index 1;
```

Each DLL maintains a table of exported routines. Application using this library may use any of the routines by specifying the library and the index or name of the routine. In case if manual loading, program defines the DLL at run-time using `LoadLibrary` function. If successful, the address of the routine needs to be fetched using the `GetProcAddress` function. This is the way of extending capabilities of an application using plug-in modules. Note that in case of manual loading, one has to know the name of the routine. According to this note, each plug-in has to comply with some sort of standard notion – set of required functions, names that the host application knows and can call. Following code sample illustrates the manual loading of a dynamic library:

```
type
  TTestProc = procedure (S: PChar; X: Integer); stdcall;
var
```

```
  hInst: Integer;
  Test: TTestProc;
begin
  hInst := LoadLibrary('usrlib.dll');
  if hInst > 32 then     // hInst <= 32 indicates error
  begin
    @Test := GetProcAddress(hInst, 'Test');
  end;
end.
```

In both cases, we can call the function Test as follows:

```
Test('Welcome to DLL', 10);
```

There's number of technologies that could be used in data exchange:

- Sockets
- Pipes (both named and unnamed)
- Remote Procedure Calling (RPC), Java Remote Method Invoking (RMI)
- CORBA
- Mailslots
- Clipboard
- System messages
- Memory-mapped files
- Dynamic Data Exchange
- Object Linking and Embedding
- COM, DCOM

Some of these technologies are available on both UNIX and Windows systems (Sockets, Pipes, RPC, RMI and CORBA), but others are the playgrounds of Windows. First four can be used to form a link between computers; the rest of functions can be used within one machine one (with an exception of DCOM, which is also a distributed version of COM).

We will not go into more details here, unless absolutely necessary, since it is all too technical.

The implementation we are developing throughout this paper uses the old-fashion tool execution, but it may change in the future to the benefit of dynamic libraries.

4.4 Graphical User Interface

A Graphical User Interface (GUI) is an essential part of a CAD framework. It defines the overall appearance of entire system. Everything the user does goes through it and all that shell or tool has to say, come through it. The visual effect and ease of use depend heavily on well-designed user interface. The Figure 10 shows the outlook of the current implementation of TT 3.0.

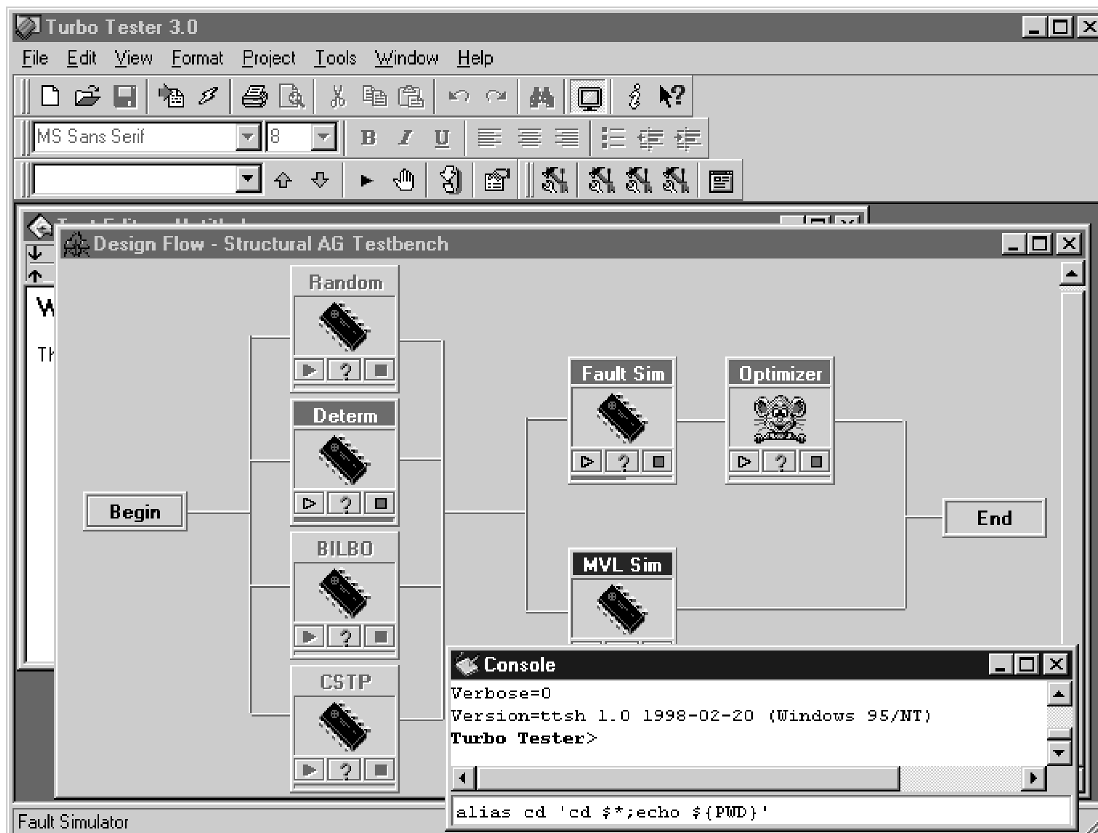

Figure 10 Turbo Tester 3.0 GUI

The GUI we are designing is planned to have following features:

- Simple Design Process Manager – described in Chapter 4.5
- Command shell console; Command shell for invoking CAD tools, routing internal commands and enabling programmability of the entire system.
- Small RTF Text Editor for lab work report generation
- Customizable toolbars and menus
- GUI extensibility options
- Easy CAD tool integration
- Inter-process communication options for external process automation

In current implementation, not all parts are so highly customizable as they would be when the project is finalized.

The GUI uses configuration files to keep the information about settings and installed components. The main configuration file is TESTER.INI, format of which is described in Appendix A. The other three types are for tools and design process manager (for more details see Chapter 4.6.2).

4.4.1 Command shell console

Turbo Tester 3.0 has a console for executing shell commands manually and receiving output of tools for viewing. As shell executes tools or other command-line programs, their output is captured and redirected to TT console. It has history list of recent

commands, the size of which is controlled by environment variable `HistSize`. Items from the history list can be selected using up and down arrow keys. The size of console's input buffer is currently determined by variable `MaxConsoleBuffer`.

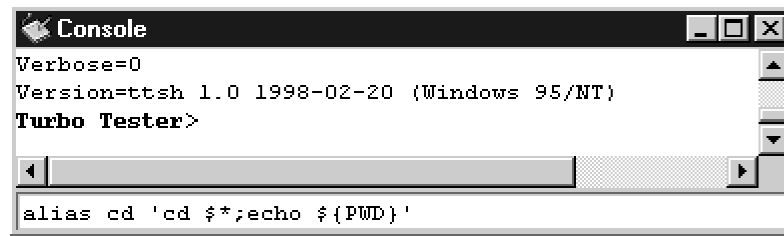

Figure 11 TT 3.0 Console

Current implementation of console does not allow for interactive tools, i.e. it cannot be used to answer questions asked by a tool.

4.4.2 Text editor

Turbo Tester has a small RTF[6] text editor. It may be used for lab report composition and generation. RTF format also a support for external objects, such as spreadsheets, charts, etc that can be inserted into a text.

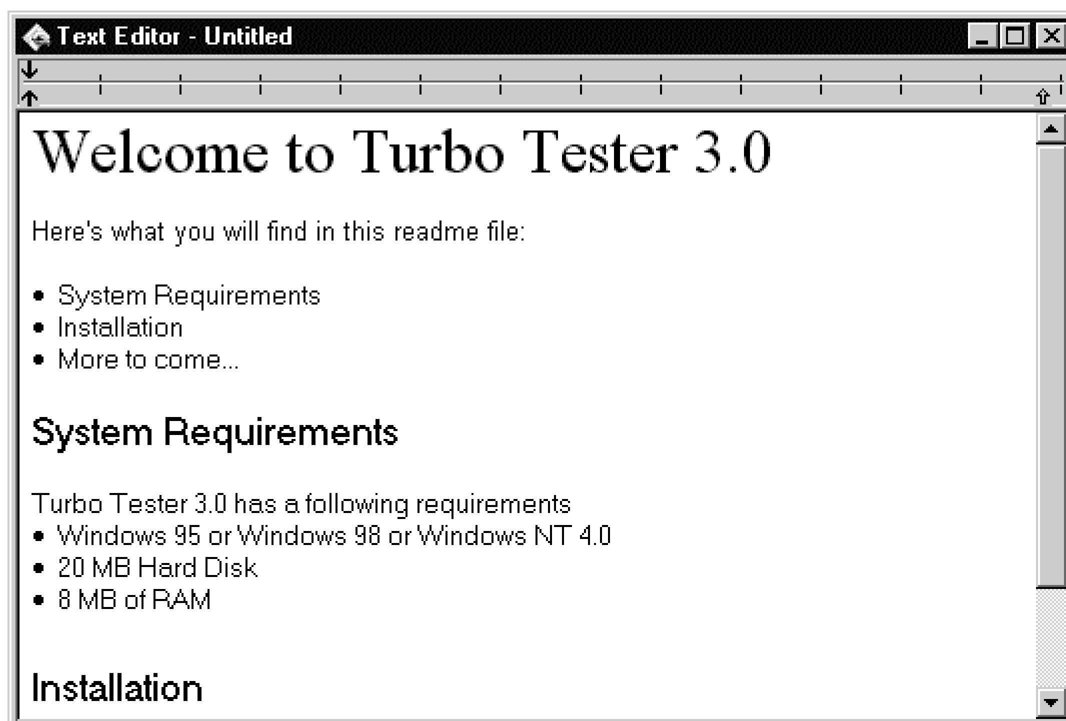

Figure 12 TT 3.0 Text Editor

Current implementation does not have many features, but it is expected to change in future releases. In addition, when TT GUI plug-in API[7] is finalized, any kind of text editing component can be attached to Turbo Tester.

---

[6] RTF – Rich Text Format is a document format designed by Microsoft to enable easy portability from one platform to another.
[7] API – Application Programming Interface. API is a set of functions that the host application exports in order to be able to extend its functionality.

4.4.3 Customizable toolbars and menus

For a convenience, everyone can rearrange toolbars in TT 3.0 environment. Currently it is not possible to change the items of toolbars or menus, nor can one create new ones. This is supposed to be added in future releases.

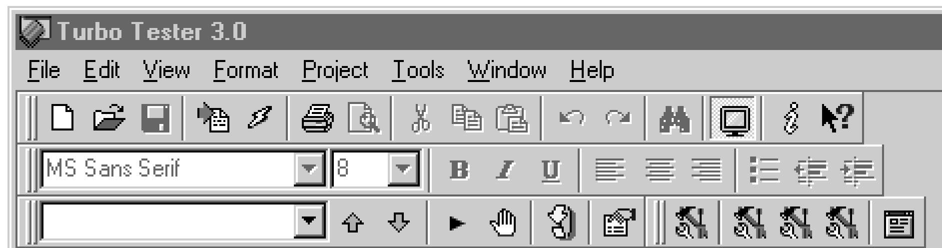

Figure 13 TT 3.0 Menu and Toolbars

Tools menu and respective toolbar (in the lower right corner) is composed at startup according to settings in the configuration file TESTER.INI, thus providing us convenient way to extend TT functionality and include references to external files or commands.

4.5 Simple Design Process Manager

The Design Process Manager (DPM) is responsible for invoking and controlling child processes in the order which is defined by the design flow rules. These rules prevent end-user from invoking CAD tools in wrong order.

DPM has a nice graphical view of the actual design flow (see Figure 14). It uses “boxes” to identify tools used in the flow and “wires” to interconnect them. Initially, no tool is selected and all have default values for their properties. It is the responsibility of a designer to select desired tools (those that cannot be selected are grayed) and set their properties. When preparations are made, the flow can be executed. As task finishes, its progress bar is updated to reflect the status. If task finished successfully, the color of status bar is green; if fails, it has a red pigment. Wires connecting the boxes show the active path and its progress.

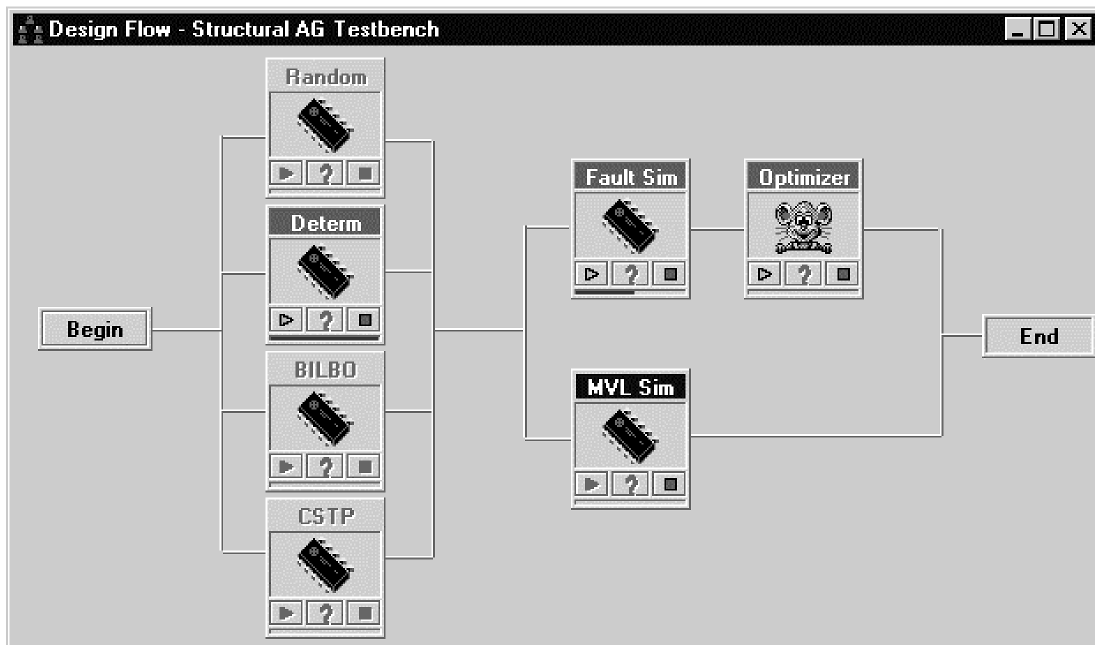

Figure 14 Simple Design Process Manager

For more visual attractiveness, tools can have animation on their boxes when run; moreover, each state of the tool can have different image associated with it.

As described above, a tool in design flow is represented by a box. More detailed view is shown in Figure 15.

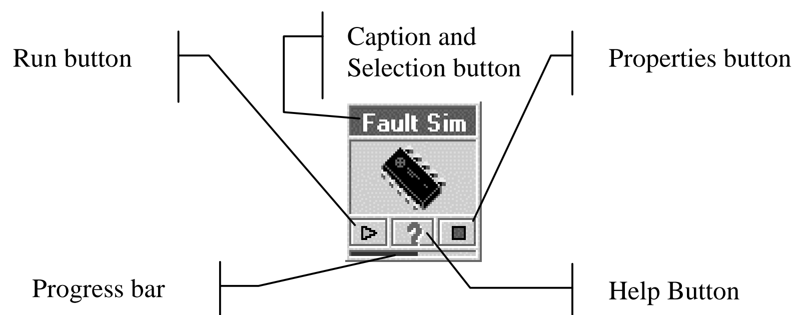

Figure 15 Design Process Manager's Tool Icon

Each box is responsible for executing a tool, associated with it, through shell, wait for a tool to complete and change its state according to result. Each box can have following states:

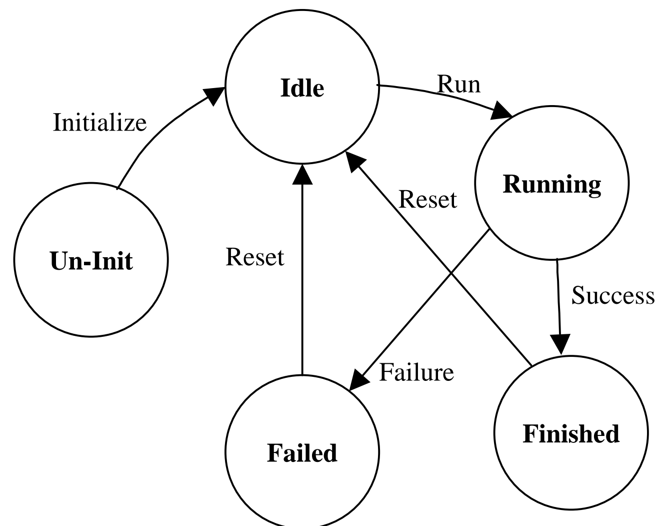

Figure 16 State chart of Design Process Manager's Tool

Each state has two associated parameters that define box's appearance: color and image. In addition, run state may have animation instead of static image, thus improving the perception of what is going on. Currently, each box can represent only real tool. In the future, they could support so-called "soft-boxes" or virtual tools, incorporating design sub-flows, thus making possible hierarchical representation of the design flow.

Lets now look at the DPM toolbar depicted on Figure 17.

Figure 17 Design Process Manager Toolbar

The leftmost combobox represents the current level in the design manager's hierarchy. The up and down arrows let user to ascend and descend in the hierarchy tree. Following two icons run and stop the project respectively. Next button resets design flow project into initial (idle) state. Last button allows setting properties for entire project.

4.6 Configurability and extendibility options

Turbo Tester 3.0 has several ways to customize and extend its behavior. Herein we will discuss the GUI extensibility options in Chapter 4.6.1 and easy CAD-tool integration in Chapter 4.6.2.

On the side of customizability, we already mentioned menus and toolbars in Chapter 4.4.3. In addition, toolbars can be dragged to any of the four sides of the application window, or be left floating above other windows (similar to the behavior found in MS Office 97 applications). Each component in TT remembers its last settings and restores them on next startup.

4.6.1 GUI extensibility options

A planned feature of the new TT environment is the possibility to extend its functionality using plug-ins. A plug-in is a dynamically loadable library (DLL) that has

a predefined set of functions to interface with a host application. This set of functions is called plug-in API (Application Programming Interface).

The most difficult part in the process of plug-in API design is the question of interface implementation. There are several requirements to that apply:

- API should not be limited to one design-tool or language, thus using only a set of typical primitives that are available on most target languages, like Delphi (Pascal), C, and C++. The API should be selected carefully having the usability issues in mind. To find out more about plug-in development, refer to Chapter 4.6.1.1.
- Must be robust enough to prevent malfunctioning extension from crashing the environment.
- Should be easy to develop and use.

The first two requirements are the most important ones. No one likes application that crashes when somewhere deep inside an extension some kind of problem arises. Carefully designed API contributes to ease of development and use as well as stability of the system in overall.

Turbo Tester uses plug-in extensions to enrich the functionality of the UI. Although, currently both Design Process Manager and Text Editor are built into the core of TT environment, they could easily be implemented as plug-ins, enabling easy modification and customization. New components as waveform viewer, test pattern editor, etc could be added later.

Plug-ins give us yet another benefit – extension can be developed separately from the rest of the system. Any member of the development team can create a plug-in with any tool he/she pleases, at any time, and then just plug it into the main environment.

4.6.1.1 Plug-in API development

Plug-ins are based on dynamically loadable libraries with clearly defined set of interface functions (API). There are several ways to interface host and plug-in:

- Set of functions – API includes several functions that can be called by host application, and some call-back functions that let plug-in to send back commands to a host. The only requirement is that functions use data types and calling conventions, which are compatible between languages that should be supported.
- Interface functions and shared objects – API provides some interface functions that create and initialize objects, which will be used later (factory functions). There must exist an interface definition of the object (in technical terms a class that has no actual implementation, which in C++, is called “pure virtual”). This method can be used only with languages that can pass compatible object pointers (for example, Delphi and Borland C++) and is also called “mapping of virtual method tables of an object”.
- Common object frameworks or models. In Windows’ world, there is a standard way of communicating between objects – the Common Object Model (COM). It enables automation between two applications or between client and server.

The first and last one are the candidates for plug-in implementation that give us language-independence. Plain functions contribute to greater speeds and simplicity when the interface is compact. COM interface, on the other hand, can connect

components written in <u>any</u> language that support COM interfaces. COM servers can reside either in DLL or in regular executable (EXE). Drawback of COM is the degradation in speed.

Here we will look at one small example of COM interface usage. Lets suppose we have a Delphi interface that exports following methods:

```
ITest = interface(IDispach)
  ['{1746E520-E2D4-11CF-BD2F-0020AF0E5B81}']
  function GetValue: Integer; safecall;
  procedure SetValue(Value: Integer); safecall;
  function GetName: WideString; safecall;
  procedure SetName(const Value: WideString); safecall;
  procedure Prompt(const Text: WideString); safecall;
  procedure VarTest(var v1, v2, v3: Variant); safecall;
  property Value: Integer read GetValue write SetValue;
  property Name: WideString read GetName write SetName;
end;
```

Assume further that the DLL that contains this interface is referenced in the registry in association with CLSID[8] that has a ProgID called "TestLib.Test".

Here is how we can retrieve the interface and call the Prompt method:

```
procedure Form1.Button1Click(Sender: Tobject);
var
  V, V1, V2, V3: Variant;
begin
  V := CreateOleObject('TestLib.Test');
  V1 := 5;
  V2 := 'Sam';
  V3 := V;
  V.VarTest(V1, V2, V3);
end;
```

In conjunction with advanced shell, which supports OLE interfaces, any external application supporting OLE Automation can be used, including MS Excel, MS Word, etc.

Now all that remains to do is to select the appropriate method (set of functions or COM interface) and design a meaningful plug-in API.

4.6.2 Easy CAD tool integration

There are two types of tools in TT: a) ones used for importing external design files, and b) tools used to perform testing tasks. Each tool has a small configuration file describing its settings and format of parameters. The formats of import and regular tools are described in Appendix D, and Appendix B, respectively.

In order to install a new tool, one has to copy the tool and its configuration file to TT installation directory, and add the reference to TESTER.INI file.

In order to modify settings for specific tool (to add a new command-line switch, for instance), edit the respective configuration file.

Here we could use TT Tool API for simplifying tool integration. This is left for future.

[8] CLSID – Class ID is the identifier used in COM to look up appropriate interface.

4.7 Shell and programmability options

As described in Chapter 4.3, TT has hierarchical structure: GUI-Shell-Tool. We have discussed both GUI and tools, but so far, we haven't touched the topic of shell. It is actually the most important part of the design in the sense that it mediates both commands sent inside the GUI and commands sent from the GUI to a tool.

So, why do we need yet another layer of abstraction and a central place to route all messages through? The answer is simple – to provide the easiest transition from old system to a new one adding the power of programmability.

Current implementation of shell does not have any internal programming language yet, but is supposed to get it in near future. We have not decided yet which language to use. It could be any language available through ActiveScript[9] technology or we might develop one of our own.

Despite of the missing programming language, it still has many important features. Firstly, it allows for definition of commands and functions. It is illustrated in the following code sample:

```
function TForm1.FnOpenEditor(const Argc: Integer; Argv: Variant):
Integer;
var
  S: String;
  I: Integer;
begin
  S := '';
  if (Argc > 1) and VarIsArray(Argv) then
  begin
    for I := VarArrayLowBound(Argv, 1) + 1 to
             VarArrayHighBound(Argv, 1) do
      S := S + Argv[I] + ', ';
    S := Trim(S);
    if S[Length(S)] = ',' then
      SetLength(S, Length(S) - 1);
    S := Trim(S);
  end;
  AppShell1.WriteLine(
     Format('Opening editor window with params (%s)', [S]));
  MessageDlg(Format('Opening editor window with params (%s)', [S]),
     mtInformation, [mbOK], 0);
  Result := 0;
end;

function Tform1.CmdPWD(const Cmd, CmdLine: String): Integer;
begin
  WriteLine(GetCurrentDir);
  Result := ecSuccess;
end;

procedure TForm1.AppShell1Initialize(Sender: TObject);
begin
  AppShell1.AddCommand('pwd', 'Displays the current directory',
    CmdPWD);
  AppShell1.AddFunc('OpenEditor',
    'OpenEditor(FileName: String; Type: Integer)',
```

[9] ActiveScript is a technology based on ActiveX that allows any scripting language to be used that confirms to the standard. There are currently two official scripting languages available: VBScript and JScript (ECMAScript). There are also some third-party scripting languages available, like Perl.

```
    'Opens new editor Window', FnOpenEditor,
    [varString, varInteger]);
end;
```

As it can be seen the definition of new command and new functions is quite similar – both declare the name, function to be called, and provide short description of what it is. Functions, in addition, define the usage string and the types of required arguments.

One more difference that can be noticed is in the arguments that will be passed to command- and function implementations. Command gets the name and list of arguments in form of strings; function on the other hand receives the number of parameters and its contents. Furthermore, function implementation must check whether the variable of type Variant is actually an array of Variants. The argument count and type matching is checked by the shell before the implementation function is called.

Functions can be called using two different forms:

```
OpenEditor('C:\TESTER\README.RTF', edRTF)
```

or

```
OpenEditor 'C:\TESTER\README.RTF'  edRTF
```

The latter one, as you may notice, is similar to the orthodox command calling. You're right, TT shell lets us override commands (whether they are internal or external) using internal functions. We can omit both parenthesizes and commas separating arguments and we are done. In fact, when TT shell meets a command, it first looks up the table of internal functions. If it does not find one, the table of commands is searched. If still not found, it searches directories specified in the shell's PATH environment variable to find an executable with that name and if found executes it.

4.8 Inter-process communication and automation

As Turbo Tester allows end-user to extend its functionality via plug-ins, it also provides an interface for other programs. TT uses OLE Automation to enable inter-process communication and a means for external application to operate upon it.

Here is a small sample of hypothetical use of TT via OLE Automation:

```
program TT_Test;
var
  TT: Variant;
  Edif: Variant;
  Coverage: Double;
begin
  TT := CreateOleObject('TurboTester.Application');
  // TT.Visible := False;
  if TT.Formats.Exists('EDIF') then
  begin
    TT.NewProject;
    EDIF := TT.Formats.GetFormat('EDIF');
    EDIF.Library := 'ES2020';
    EDIF.ClockSignal := 'CLK';
    if EDIF.ImportDesign('C:\TESTER\DESIGNS\c1971.edif') then
    begin
      if TT.Tools.GetTool('ATPG.Random').Run = 0 then
      begin
        Coverage := TT.Tools.GetTool('ATPG.Random').Coverage;
```

```
        Writeln('Test coverage is: ', Coverage);
      end;
    end
    else
      Writeln('Error importing design');
    TT.CloseProject;
  end;
end.
```

Note that the actual implementation of the interface has not yet been defined, and is supposed to complete in the future.

4.9 Discussion

As it can be seen, Turbo Tester's Windows GUI has many features that comply with CAD framework requirements:

- Easy to use interface
- Simple Design Process Management
- High configurability and extensibility (open and dynamic)

It also has small footprint in terms of requirements to disk space and main memory, thus being suitable for teaching large number of students.

Turbo Tester also provides a comprehensive set of CAD tools for teaching digital test, starting from gate-level to up to high-level design descriptions; including methods for built-in self-test generation and design for testability – those that many commercial frameworks are missing.

5 Internet-based multi-user environment

As an outcome of this work, we will have a working network-oriented CAD framework for teaching digital test, and design for testability that can be accessed from anywhere in the world.

5.1 Motivation

Until now, we have looked only at one solution - a single-user application. Such an implementation has several advantages, but also many disadvantages:

Advantages:

Having a program for one distinct platform allows utilization of every feature of underlying OS most efficiently. The speed and visual richness of native application clearly outperforms any other solution.

Disadvantages:

We are tightly tied to one platform and if we wanted to port our system to other operating systems, it would mean a total rewrite of the entire package. It would also mean that we have to support and maintain each version separately. In other words, the package is not portable. Another drawback is the lack of remote usability – we have to install the package onto every computer we are planning to use. There is no way to install the application centrally on a server and access it from multiple locations. Well, to be honest, there is a way to enable it, but it is restricted to use within of one distinct LAN only. The solution is based on a fact that servers can export (share) disks and directories for public use. Users have to map server's shared resource with an installed package and gain access to it. As already mentioned, it is not possible to use it globally because of security reasons, and no system administrator is willing to take risks and give world-wide access to its file systems.

One solution to the problem of portability is the use of programming tools that allow creating of portable applications. There exist two types of such products:

a) Tools that allow porting of MFC code to UNIX systems. Representatives of this group are Wind/U from Bristol Technology, and MainWin XDE 3.0 from MainSoft. The prices for such systems are extremely high, starting from $12000 per license in case of MainWin, for instance.

b) Tools that use higher level of abstraction to achieve portability. The representative of this group is ZAF (Zinc Application Framework) from Zinc. The price for one license varies from $400 to $5000 depending on modules used. Good news is that Zinc gives personal development license away for free.

Tools like these can be successfully used for building multi-platform applications, but sometimes they do not have enough power for building network-computing projects. System like MainWin claims that it does not require any modifications of the source code in order to port it to UNIX systems. As time passes by, these systems will evolve too, giving more and more features, but price is probably going to remain at high levels for a long time, thus making it inaccessible for creating educational, free-of-charge applications.

With the advent of Java programming language and rapid evolution of Web, we now have perfect infrastructure for adopting distributed computing at large. Java language is especially designed for building network-enabled applications, providing such important features as easy network access, remote procedure calling, and high level of security, simple language constructs and strong typing. On the other hand, Web has gained tremendous popularity in the recent years and has become an integral part of our everyday business life. Most currently available web browsers support feature-rich HTML, including embeddable objects and Java applets, which makes it a perfect platform for remote information delivery and processing. Both WWW and Java are expected to enter domestic areas soon. Developments, such as WebTV, will give access to Internet through ordinary TV using simple add-on box. This add-on uses Java applications for accessing the Web. Java will also be one of the operating environments for mobile, distributed and embedded systems.

The latest buzzwords are network computing and thin clients, and this is where we going to extend the functionality of our Turbo Tester.

5.2 History of network computing

In the early days, there were applications running on mainframes. Users had to connect their thumb terminals to mainframe server in order to access data that was kept there. Programs were running on server machine; users had only output displayed on their terminals. On the other hand, mainframes were easy to manage because of centrally installed software – whenever changes were made, they instantly became available to all users.

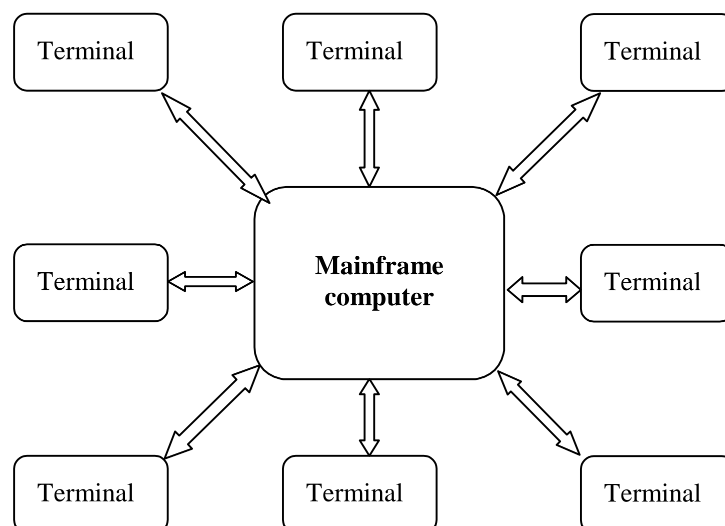

Figure 18 Mainframe-based distributed computing

About fifteen years later, with the emergence of PCs and the growth of computing power, the next step in distributed computing has been made - the child was named “client-server architecture”. In this approach, the thumb terminal is replaced by “smart” client, moving computational power off from server to desktop. The cost of development and maintenance increased because the complexity moved to desktop. Due to the different and disperse nature of PC environments, control and the ability to respond to change has been significantly impaired.

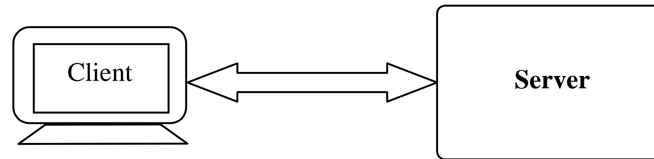

Figure 19 Client-Server architecture

We are now evolving to the new era of network computing, which leverages the simplicity of thin[10], inexpensive, Web-based clients, and re-establishes cost-effectiveness and control. The resulting server-centric architecture allows organizations to shift expensive operational and management emphasis away from their fragmented, resource-intensive, desktop infrastructures and focus instead on creating high-impact, flexible applications. Network computing uses 3-tier architecture, which can easily be extended to n-tier. In typical, 2-tier or client-server approach, there were two sides – client and server. In 3-tier architecture (see Figure 20), there is an additional "mediator" layer between the client and server. In this approach, server breaks into two separate parts – a) an application server, and b) main server, typically a database server.

Figure 20 3-tier network computing

3-tier architecture removes complexity from the desktop and keeps it apart from the main server. Client can now be implemented as thin web-based application running inside web browser. With the appearance of Java applications and -applets, thin clients got a whole new meaning: user can open up a web page, which includes Java applets; web browser then downloads all necessary software and executes it. This way, the need for keeping separate copy for each user, vanishes. We only have to maintain one copy that can be handed out by application server and loaded into user's machine dynamically, on a request.

Years	Architecture	Access	Upgrades	Cost
1965-1985	Mainframe	Low	Rarely changed	Moderate
1990-1996	Client-Server	Good	Hard to change	High
1997-2xxx	Network Computing	Simple	Quick to change	Low

Table 3 Evolution of network computing architectures

Moreover, today's network computing architecture (NCA) is not limited to only one application server. There can be a multitude of servers, each tuned to serve different

[10] Thin client is a client machine that does not have applications stored in it and typically, there is no hard disk either. Instead, everything is downloaded from application server, except for operating system, which resides in ROM.

aspect of business logic, such as survey, reports, customers, etc. You can see similar distribution in the Web – clicking on a link may take you to whole new document on a whole new server. In other words, everything is so tightly integrated that you cannot even make a difference.

5.3 Possible solutions

Our goal is to create a CAD system, in terms of network computing, which is accessible through Internet, using standard Web-browser. Two possible solutions come in mind:

- HTML form-based system, allowing parameter entry for CAD tools
- Interactive Java applet, providing nearly the same functionality as native application developed in Chapter 4

The first solution is useful because of its inherent simplicity. The only problem we will face is the inconsistency of HTML layout presentation among Web-browsers. Despite of this deficiency, we can still create good enough interface with decent Design Process Manager.

The second solution gives us practically same powerful environment, as did the single-user version. The advantage is its distributed, network-oriented nature and ability to use service simultaneously by any number of users. Java applets are also rendered same way on every platform.

Within this thesis, we are going to develop one solution, using HTML-form-based implementation, in Chapter 5.5. In Chapter 5.6, we will look at the possible solutions for Java-based system.

Now, let's peak at one possible use of distributed computing in CAD system – the Virtual Laboratory project.

5.4 Virtual Laboratory

The Virtual Laboratory project (VILAB) was brought into life by a need for cooperative research facilitating immediate exchange of information, sharing of software tools developed by the partners, enabling joint work on research projects and practical designs, providing access to libraries, benchmarks, design examples etc, and serving as a source of information not only for partners, but also to all other interested persons and institutions, including national industries with a special emphasis on SMEs[VILAB 98].

The idea is to keep CAD tools at the hands of their developers and provide consistent interface to users of these tools. The tool-exchange architecture of VILAB could be depicted as follows:

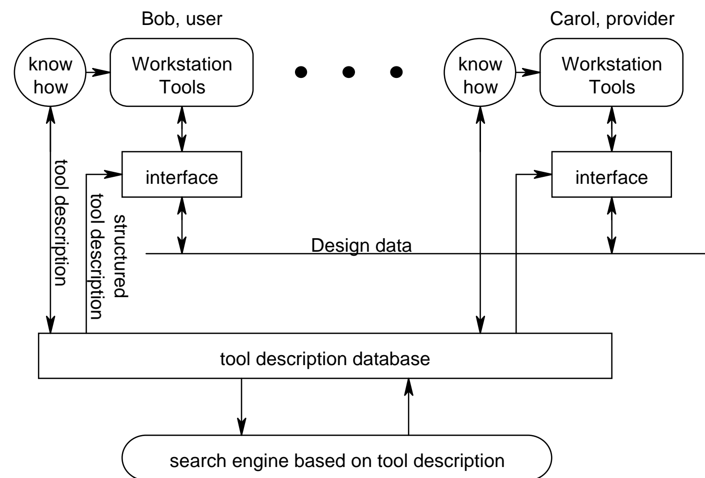

Figure 21 The Architecture of Virtual Laboratory

The architecture consists of three distinct parts:

1. **Service provider** is a person or institution exporting services (a tool or a set of tools) for common use through its web site. The service must be registered in the central database in order to notify the community about its availability. Provider is responsible for guaranteeing the availability and usability of the service. Maintenance of such system is elementary and straightforward since it is installed locally.

2. **Central server** maintains a database of registered tools of VILAB. It is necessary to have one central location where users can look up the service they are interested in, using a general-purpose search engine.

   If restrictions apply to users accessing the database and services, there could additionally be a database of registered users as well, removing the need for re-registering at each provider's site repeatedly.

   Centralized server is also a perfect place for keeping common pieces of software and other intellectual properties developed for or by VILAB partners (Java applets, scripts, design, libraries etc).

3. **User** is a person or machine, using services. If the user is a human being, we need to provide him with nice consistent graphical interface to services. The question of consistency is of great importance, because the whole idea of VILAB is to provide easy access to any CAD tool, and consistent interface is a key. The interface can be interactive visual or form-based. If, on the other hand, the user is a machine or program, we only need to provide standard interface definition for calling tools remotely using remote automation technologies, such as CORBA, DCOM or RPC. The automatic remote usage of tools is quite tricky and will be described in chapters 5.4.2.2 and 5.6.

5.4.1 Levels of intricacy

From the points mentioned above, we can draw several levels of abstraction, depending on complexity, feature richness and thoroughness of implementation. One other thing deserves mentioning – in all these levels, we use Web server as mediating application server.

5.4.1.1 Level One

On the first, and simplest level, only HTML Forms are used. User opens his/her Web browser, connects to provider's Web site and selects VILAB's page for a tool he/she wishes to use. On the provider's side, a Web-server extension (CGI, WinCGI, FastCGI, DGI, TGI, ISAPI, NSAPI, WSAPI, Java Servlet, etc) is used to provide interface to a tool. The extension is responsible for converting HTML-form parameters into arguments that can be passed to specified tool through command line. When tool finishes its job, results are sent back to the user.

Advantages:

The predilection of this approach is the ease of creating HTML pages. It also gives maximum freedom in designing interfaces. On the other hand, web server extensions can be written in any language, including scripting languages and they do not have to be very complex to get things running.

Disadvantages:

Each provider has to create it's own HTML pages for each tool and handle all possible errors that might occur. Not all browsers format HTML Forms similarly and in order to make web pages a little more interactive, client-side scripting would be necessary. Unfortunately, not all browsers support scripting. In addition, there is a problem that accompanies many CAD tools – they take long time to complete. This becomes an issue with Web servers – the time the CGI extension is allowed to run is quite limited. Although, many servers allow configuration of the value, it is often hard to guess, since it may vary in gigantic ranges.

5.4.1.2 Level Two

The second level is an intermediate between levels One and Three, and might not be of much interest. In general case, it is same, as Level 1, with an exception that on the user's side, we use a Java applet instead of HTML forms.

Advantages:

User gets well-formatted entry-based forms that look the same on every Web-browser, which supports Java Applets. Alternatively, one could use special Java applet viewers that are distributed with Java Development Kit and several Java-programming interfaces. Using applets allows embedding VILAB applications inside any HTML page, not just into specially designed one.

Disadvantages:

It requires more programming and knowledge of Java programming language. Interface between Java applet and web server extension expecting HTML-form query strings can be tricky.

5.4.1.3 Level Three

This is the most advanced level that demands the advanced usage of Java applets. In my vision, there could be only one applet, which is used to interface all available tools. This requires a format, which can be used to describe a service. Moreover, it should be as little restricting as possible. The applet reads in description file and renders forms based on that portrait.

Advantages:

Common interface to all tools. Whenever it is updated, all users get the new version from the central server. Tool developers have only to provide a file describing their tool(s).

Disadvantages:

It requires a tool description format and parser that can dynamically produce forms and adapt to various situations. There is a vast number of different tools of different categories, and it may become a bit difficult to find such a common denominator for a format that would satisfy needs of all parties involved.

5.4.2 User-Tool interface

All three levels described use the same building blocks – web browser, web server, and server extension. The level 3 solution could alternatively use dedicated RPC, CORBA or DCOM server for advanced communication. Since tool itself is never exposed to world, the actual implementation of a tool and protocol used in communication between tool and interface, are not significant. The following figure shows the user's interaction with provider's tool interface:

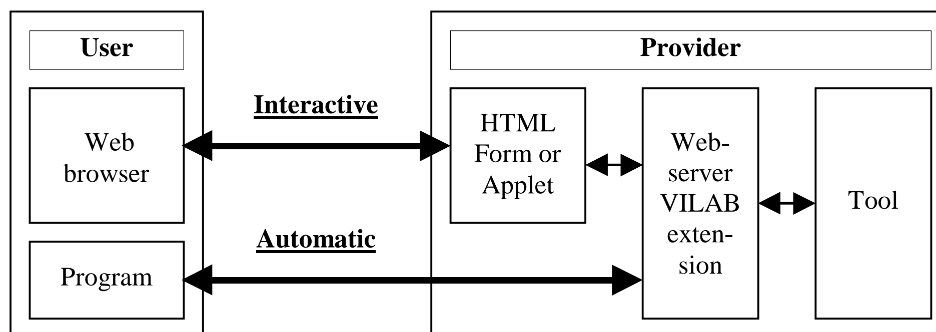

Figure 22 User-Tool interaction

As it can be seen from the Figure 22, there are two ways to execute a CAD tool: interactive and automatic. We already discussed these two approaches briefly and let's go now into details.

5.4.2.1 Manual (interactive) usage

Interactive interface consists of HTML page or Java applet (depending on the level of abstraction used) which is downloaded from provider's site. User then fills form entries and sends it back to provider's VILAB server-extension, which in turn executes specified tool and returns results as they become available.

Interacting is defined as follows: two or more people or things acting, or having some effect, on each other.

From this definition, we deduce that interactive usage means acting on the interface from both sides. It is natural to form-based systems where nothing happens before user fills out forms and submits them. After processing form query, system sends a reply. This "request-reply" cycle can easily be used in Java-based interface as well. Moreover, applets allow automation of several trivial tasks, so that user has to give only guidelines how to proceed. Following this trail, we could create networked application that performs most of the same tasks, as did the single-user version, only from a distance.

5.4.2.2 Automatic usage

Several modern design tools have internal scripting language that allows calling of operation system commands and other programs. This way we can create external batch files that do the job for us. In order to use VILAB services from any batch file, we need a tool that is able to access remote networks. There are many languages that make it easy, such as Perl, Python, and Java – to name just few. Secondly, it must be able to form parameters that can be passed to VILAB service interface. Finally, it must be able to retrieve any information returned.

Considering points outlined, there are two possible approaches:

a) Client program must pretend as if it is a web browser, form a query-string, and send it to web server. The program must also be familiar with the format in which results are returned. In order to leverage the burden a bit, specially designed server extensions could be used, which interface is well know.

b) Client script must be able to use protocols for distributed computing, such as RPC, DCOM and CORBA, in order to invoke functions residing in provider's application server (in most cases a separately designed server daemons).

We will return to this topic in Chapter 5.6.

5.4.3 Web-server extension

Web servers allow execution of external applications for processing requests they cannot resolve themselves. Historically it was meant only to process HTML forms using CGI-compliant applications. Nowadays there are many protocols beside CGI that are designed for advanced processing of information, such as WinCGI, FastCGI, DGI, TGI, ISAPI, NSAPI, WSAPI, Java Servlets, etc. All xxCGI implementations use external processes that are spawned on each request, which makes them pretty slow. On the other hand, all xxAPI methods and Java Servlets are in-process solutions, which are invoked on a first request, and stay up as long as needed. The later approach yields far more excellent speeds, thus clearly outperforming CGI-based solutions. For more detailed discussion refer to references on HTTP protocol and HTML Forms.

In trivial case, a PERL script as a CGI module can be used. The core of the parser could be publicly available on the server. Parameters sent to extension by browser are coded as *key=value* pairs. The extension's duty is to decode them, and convert to form understood by tool. Some of the parameters could be standardized, such as *Tool*, *DesignName*, *SessionID*, etc.

After all preparatory work has been done, the sacred process of tool rocking can be initiated. During the execution user should be able to send STOP signal indicating, that tool should be stopped immediately (halted). Unfortunately, the process of tool running is so divine that it cannot be intervened, even if desperately needed – all due to the stateless nature of HTTP protocol. When tool finishes successfully, results will be returned. In case of some error, error message will be returned. At this point, I do not specify how exactly returned data will be displayed to user. It depends on level of intricacy used.

5.4.4 Tool registration with central server

When the development cycle is over and provider wishes to announce service for public use, it has to be registered in central server. However, before we can do that, some additional steps must be made:

1. Developer should thoroughly test its newly created interface to verify it against programming errors and other factors not foreseen before.
2. If the Level 3 of abstraction is used, a service description file must be created. In this file, the behavior of service interface pages and parameters to tools are described. For more information, refer to Chapter 5.4.5.

The registration of the service may also follow two different trails:

a) User goes to service registration web page, which resides on central server, and fills out all fields required for identifying and describing the service.

b) User goes to service registration page and points out the location of service description file.

In either case, the server retrieves information, enters it to database and confirms service provider of successful update.

5.4.5 Service Description File

It could be good idea to have a format for describing tools from a variety of vendors. This would allow registration of a service with a single operation instead of having to enter information for each tool separately. Herein, I propose Service Description Format (SDF), which is based on XML standard. The XML was designed by W3C for structured content delivery over Internet, allowing custom tag-based formats (similar to HTML) to be developed, with semantics applied to it. The best-known representative of XML is probably CDF (Channel Description Format), designed by Microsoft for describing server-push web pages, called *channels*.

The SDF format must be flexible enough to allow description of various services (both manual and automatic) and tools inside them. The problem is quite serious, considering the fact that there exists a multitude of different and sovereign design tools with varying needs, and describing them all in orderly fashion could become a real headache.

The SDF format has hierarchical structure:

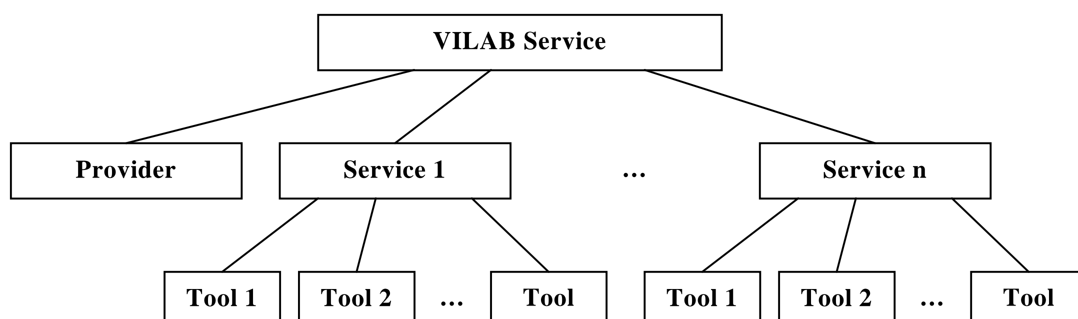

Figure 23 SDF Format's hierarchy tree

The SDF file consists of two main parts: Provider information and Service information. There can be only one sections describing provider, but unlimited number of service sections. Each service section can contain any number of tools.

In the provider section, following information should be contained:

a) Name of the provider, including its acronym or abbreviation
b) Identifier assigned by VILAB, if any
c) Institution, department or company
d) Contact information: e-mail, phone, fax, Web page
e) Local VILAB service intro page address

Each service should include following information:

a) Name of the service and its version number
b) Short and long descriptions of the service
c) Category into which it falls, and any keywords that are related to it
d) Support e-mail address
e) Web-page address of the service
f) List of external formats it supports
g) Tools that are available

The detailed information of the SDF format is in the process of development and will be available later.

5.4.6 Central Server

Central Server is needed for several tasks:

a) Keep service database
b) Keep user database
c) Have the list of sample designs
d) Hold online courses
e) Be a source of information about latest news in VILAB area
f) Allow searching of services, providers and possibly users (in reasonable measures of course)
g) Registration of services and users (if necessary)

Central server is a web server providing VILAB-related information. Besides sample designs, libraries and so forth, it includes powerful search engine allowing searching for services, providers, etc by keywords such as category, supported design formats, service providers, and so on. Search engine will fetch related material from database and presents it to user.

The steps which user could follow would be following:

1. User connects to central server to find out what services are available. He/she then opens search page and enters keywords to specify his/her interest.

2. From a list of services found, user selects a link that most closely matches query, and specified service is brought up from the provider's web site.
3. At the opening of VILAB's interface, user is prompted for login (in case authentication is required). During login, user is looked up from central user database to see if he/she has a right to access selected services. If everything is OK, important parameters (it is useful to keep login information through one session to avoid need for re-logging, in case another VILAB site is contacted soon) are memorized and next page is shown. To keep logging information on user's machine HTTP 'cookies' could be used.
4. User specifies design and selects appropriate input format. If tool is not able to read design directly, it may be necessary to import it into native format used by a tool. At the provider's side temporary directory is created for current session, which will be removed at the end of session (or after some timeout if connection is abruptly terminated).
5. User selects tool, sets it's parameters and executes it.
6. On a successful finish, user will be informed with brief set of results. If more information is required, entire resulting file can be opened by pressing the "View Details" button.
7. Any of these steps may be repeated.
8. User disconnects from provider's site.

5.4.7 Security

The problems could arise from the design data secrecy and the possible need to pay for a service. There might also be need to authenticate providers, tools and even users. After all, how do we guarantee that no-one malevolent can dumper the design file while being transferred over Internet?

Luckily, these problems can be solved quite efficiently with cryptography and e-cash. Most modern web-servers allow for encrypted connections using Secure HTTP (HTTPS) or some other similar protocol.

Of equal importance is the problem of versioning of tools and services. Using parameters for different version of tool could result in misinterpretation of the meaning of the parameter. There should be some version-checking features built right into services, extensions and SDFs.

5.4.8 Conclusion

VILAB is just one way to utilize tools in global network for both educational and commercial use. As the VILAB project is on the move, many questions remain open and will find their answers in the process of active research work.

5.5 The form-based interface

This chapter concentrates at the simplest solution – Level 1 interface. In this implementation, we use HTML pages with forms for client side, and Python scripts for

building server extension. The advantage is the inherent portability of both HTML and Python code.

5.5.1 System Requirements

Server

- 20 MB of Hard Disk space + additional space for designs in the future
- 16 MB of RAM
- Python interpreter installed
- Any Web server supporting CGI programs

Client

- Computer with Internet connection
- Web browser supporting JavaScript and CSS1 style sheets.

5.5.2 The architecture

We use Python scripts for generating HTML pages dynamically, depending on parameters passed. In following, we will look at both sides of the story in more detail.

5.5.2.1 The client side

Let us first look on how client sees the process of remote usage of VILAB tools. We skip the central server part for now and do not require user to log in, but these could be added easily.

First, an introductory page is displayed. Here, a selection of available services is displayed. At present, only one service is available.

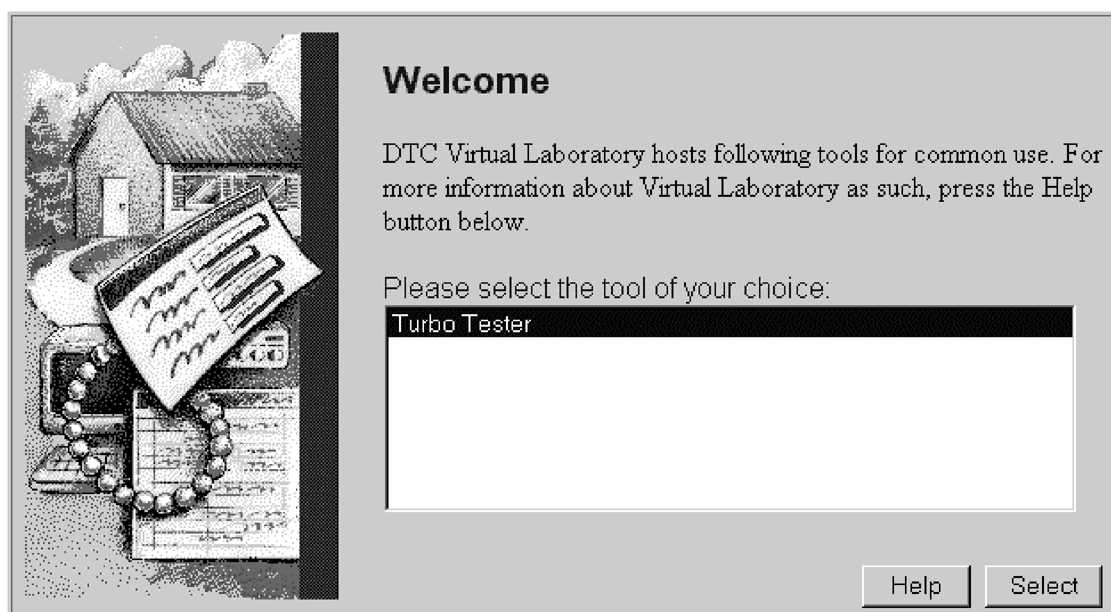

Figure 24 VILAB Welcome page

Choosing service and pressing “Select” button opens next page, asking for design, and the format in which it is saved. By selecting a format, the list below changes showing available sub-formats.

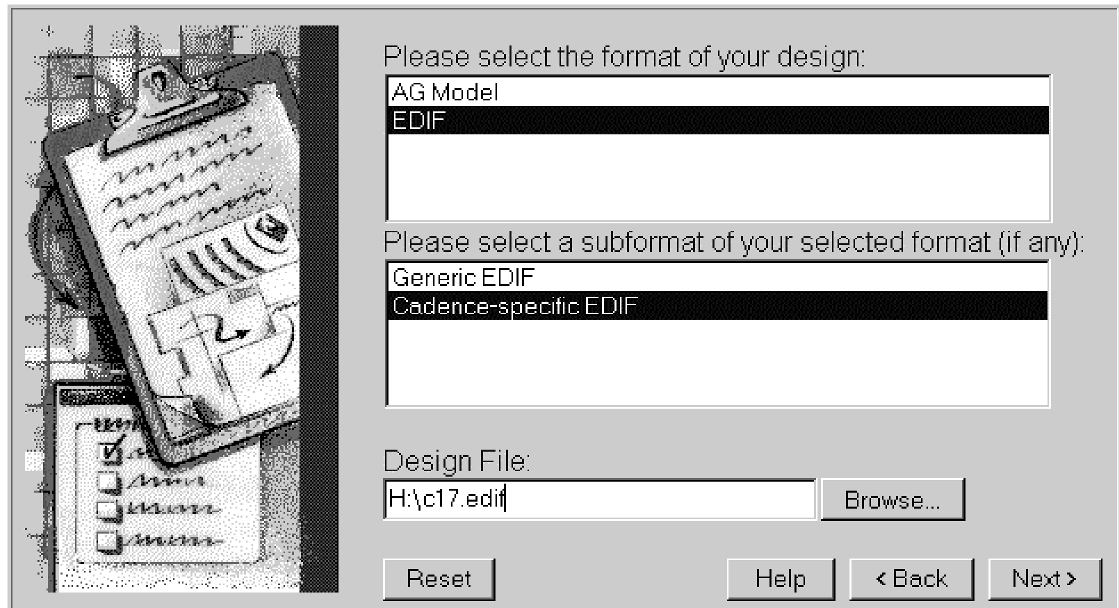

Figure 25 Turbo Tester VILAB front page

Pressing “Next” button starts import process, which may take a while to complete. Once finished, summary is displayed.

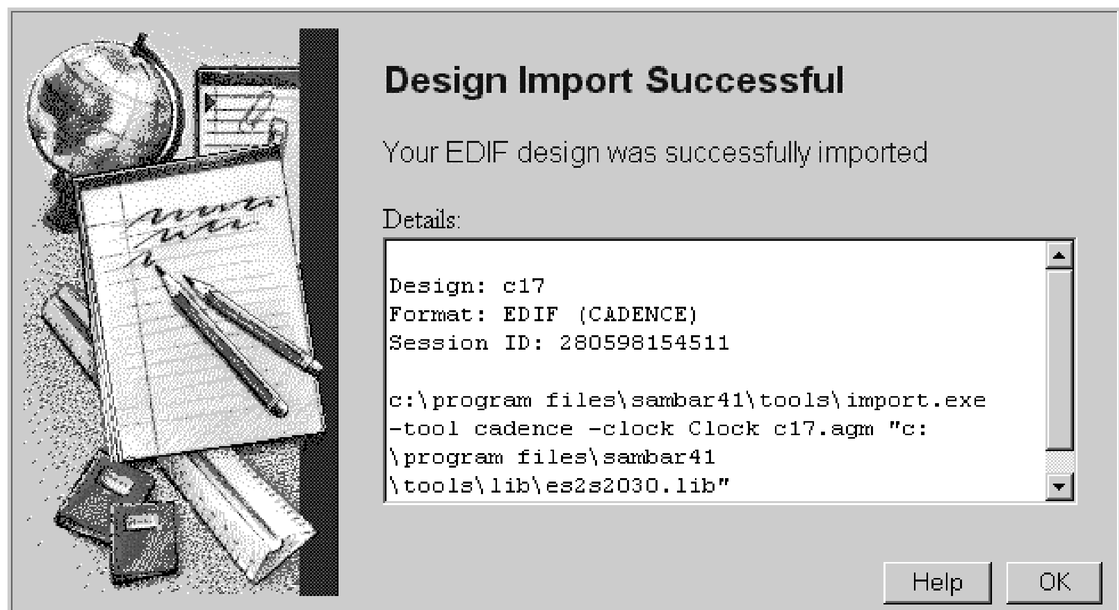

Figure 26 Page displayed on successful design import

Agreeing with the summary, a list of available tools is presented:

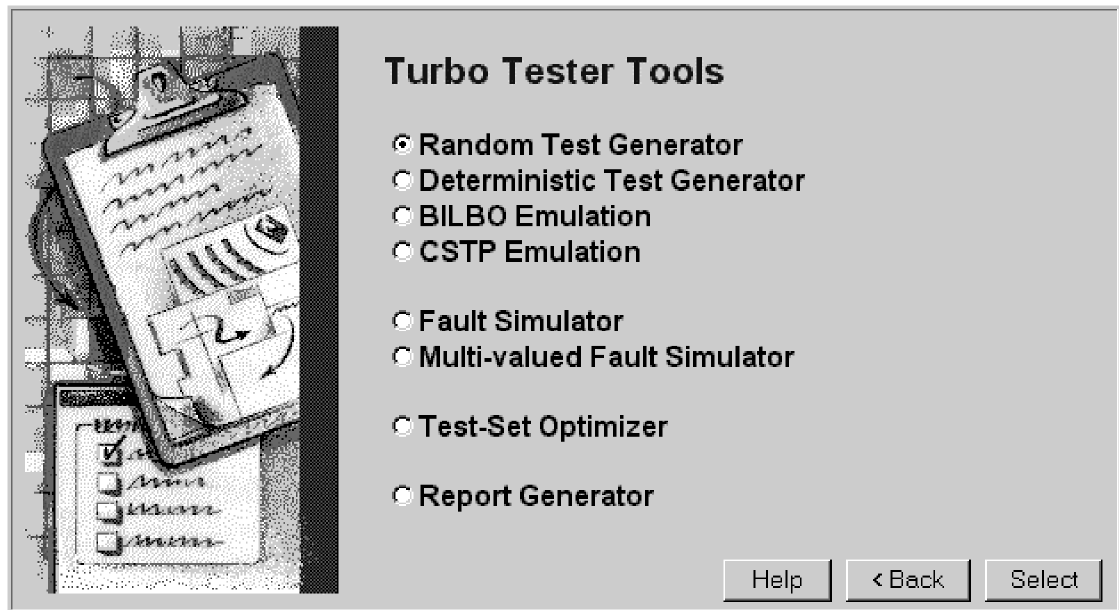

Figure 27 Selection of VILAB tools

When rendering tool selection page, we could use Design Process Manager to enable only those tools that can be executed at current step (Fault simulation can only be executed if test patterns are generated, for instance).

Selecting one of the tools opens new page, which allows setting parameters to that tool. Here we could exploit DPM once again, setting parameters to values used last time.

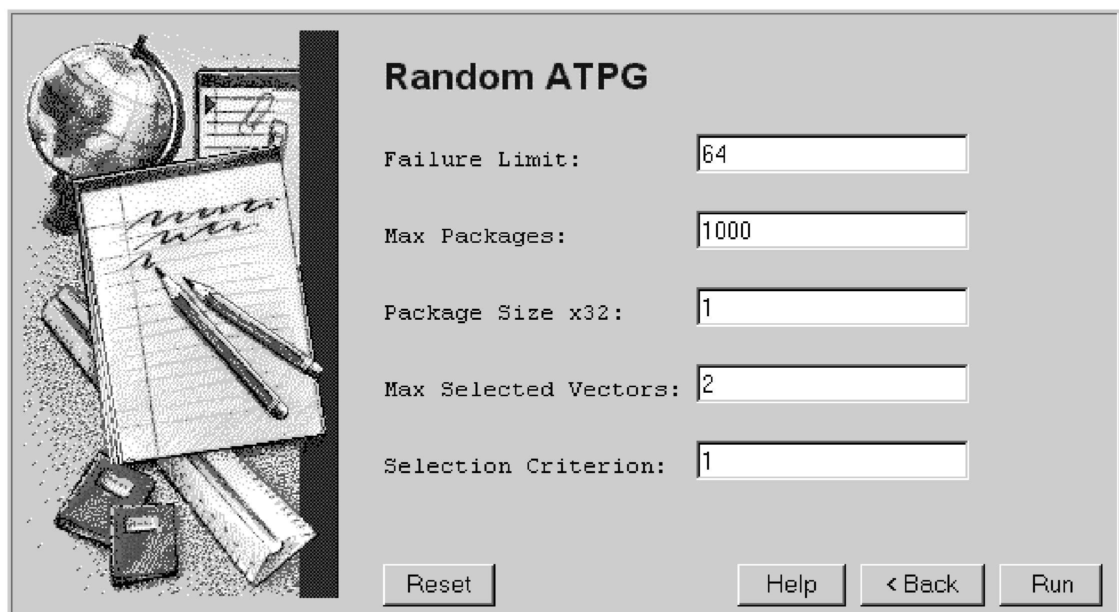

Figure 28 Setting properties for a VILAB tool

Now we are ready to run the tool with specified parameters. It may take long time before results are given, thus it would be nice to be able to see the progress of execution and possibly interrupt the process if too much time has passed. Unfortunately, it is not as simple to implement as it would seem at first. Due to the stateless nature of HTTP protocol, we have to use some kind of identifier to differentiate multiple users accessing the service simultaneously. For stopping a tool, we would have to know some internal

information about a tool actually running, typically a *Process ID*. This would mean that we would have to give out information about what is going on inside our server's memory, and this is not what we want. Giving out progress information is also a tricky task, because most Web servers tend to buffer output of the CGI and not send it to user until tool has finished. For more detailed discussion see the server's side of the story.

When the tool finally stops, results are shown:

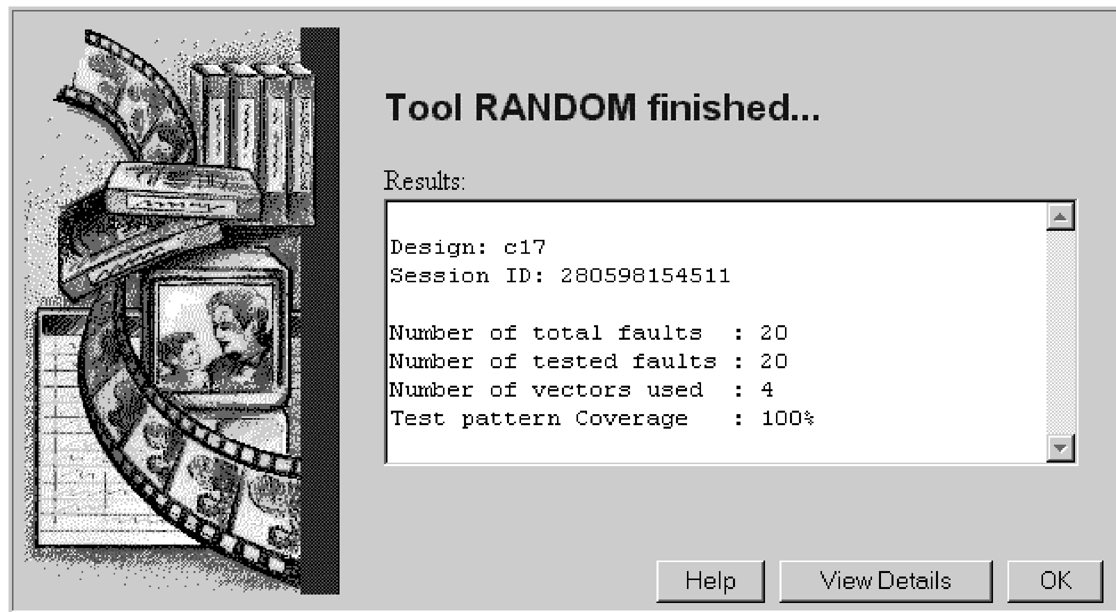

Figure 29 Results shown when tool has finished

In case of error, an error message is displayed instead and depending on severity of error, a retry might be allowed. When eventually everything is fine, we are presented with the VILAB tool selection again, and the process recurs.

5.5.2.2 The server side

Whenever user selects a VILAB site, an introductory page is displayed displaying all available services. This selection can be in static HTML page, or be generated on the fly by a server extension. We have chosen to follow the latter solution, in which one could fetch service information from local dedicated VILAB database. Whether it is static page or a reference to server extension, there is no parameters passed to it (in our case, it's Python script file, named *vl.py*). The only purpose of this page is to redirect user to specified service's "home" page.

Each service should ask for design file and format in which the file is saved. This is the most complex step in entire process – we have to transfer the design file to server, generate a *SessionID* for the rest of the session, create temporary directory using generated *SessionID*, and save the design file in it. The session ID is generated using current date and time (total of 12 decimal numbers), which should be enough for most cases, distinguishing active users. This coding works if it is guaranteed that no two users will start using VILAB service at the same moment, namely second.

Extension then generates new HTML page inserting following hidden form parameters: `DesignFile`, `SessionID`, `Format` and `SubFormat` (if defined). If format of the design defined, was different from native format used by CAD tools, importing is

required. The newly generated page's content depends on whether import format requires additional parameter to be set or not. All this work is done using script file called *vltt.py*.

If not, a list of available VILAB tools is presented (using file *vlttrun.py*), also having additional parameters defined (*Design* and *SessionID*). When user selects one of the tools from the list, new parameter is set: `GetTool`. The presence of this parameter informs the script which property page it should display when generating next HTML page. On the tool's property page, three hidden parameter fields exist: *Design*, *SessionID* and *Tool* in addition to property parameters specifying tool's behavior. Submitting this page, the actual CAD tool will be invoked with specified parameters.

It would be nice to display new HTML page indicating the start of the process and provide means for stopping the tool. It would also be good idea to let the user know how long he/she has to wait. Unfortunately, neither of these things can be solved using current implementation. The problem lies deep inside HTTP protocol and web server implementations.

By the nature, HTTP is stateless, with no history about who and when accessed it. This is natural in case of simple HTML page delivery, but is not directly suitable for session-oriented systems. There exists a solution, which uses HTTP cookies to keep session information directly on user's machine. Some users disable this feature, being afraid that it is a threat to security. We instead use a method that sets hidden fields inside each HTML page clearly specifying current step.

Stopping a running process is also not so simple – we need to know the very specific information about process – its *ID,* in order to access it from outside (we cannot send any other signals to CAD tools to stop them). This would mean that we give out information about programs running on the system, which is a direct and serious security breach for any system.

Sending status and progress information to user is also difficult, since most web servers tend to buffer the output from its extensions, and send it out only when extension has finished. Some servers let administrator to disable buffering, but most do not. In general, it is not easy to achieve unbuffered data stream to client using CGI-type extensions. It could be possible using in-process extensions, such as ISAPI, NSPAI, WSAPI, and Java Servlets. These provide greater control over server's demeanor.

So, after the CAD tool has finished, brief results are displayed. Pressing "View Details" button opens new browser window in which complete result file is displayed. In case of error, a reason message is displayed. If the severity of error is not too high, process could be retried. If the error persists, service provider could be contacted to find out the cure to the problem. If, on the other hand, retry was successful, we can continue using the service.

At this point, a list of tools is displayed again, and the cycle starts over.

There is one question, which remains open: when do we remove the directory created at the beginning of session? Since we do not have session-oriented connection, we do not know when the session is over. If we leave it on the server for a long time, it would waist the valuable hard disk space. If we remove it too soon, the user cannot finish his/her task. What is the Golden Gate for this problem? Honestly, I do not know. Right now, there runs a separate task on the server, looking at the age of directories in VLTEMP directory and removing those, older than 2 days.

5.5.3 Conclusion

As a conclusion, we can say that from the user's side, everything looks fine and is easy-to-use. To gain such simplicity, lot of hard work has to be done on the server side. Although most required features are present in this implementation, it still leaves several things to be desired, such as possibility to stop running process and see progress of calculation.

5.6 The interactive client

We have just seen how HTML-forms and HTTP protocol could be used for implementing the interface to CAD tools. In this section we shall look at more advanced forms of distributed computing. Wouldn't it be nice to have all the features of the native framework, described in Chapter 4, and still be able to use the benefits of remote usage.

Although HTML has many useful features added lately, like style sheets, dynamic behavior through DOM[11] and scripting, it still does not provide us all the required functionality across available web browsers. Moreover, it is impossible to get two-way session-oriented communication between client and server using HTTP protocol. What HTML does have, is the ability to include active components, such as plug-ins, Java applets, ActiveX controls, etc. Using these components, we can extend browser's capability to far higher levels, and most important, they look and behave same way on all supported browsers.

In the following, we will concentrate on Java applets, since ActiveX controls are not available on non-Windows platforms (at least for now), and the only know browser supporting them is Microsoft's Internet Explorer. Java applets have basically the same capabilities as Java applications, except for those that might rise security hazards. The things that applet can not do, include:

- An applet can not touch the local disk
- It cannot have menus
- Dialog boxes are "untrusted"
- It cannot access network addresses, but the one it was downloaded from
- It cannot read user machine's system properties
- It cannot load libraries or define native methods
- It cannot start any program on the host that's executing it

Most of these restrictions can be removed when using signed applets and customized security manager.

By nature, Java is a network-oriented language, giving us high-level access to networked objects using Remote Method Invocation (RMI) or Common Object Request Broker Architecture (CORBA) interface. Both of these allow calling methods of remote objects (or classes), the only difference is that RMI can be used between Java programs, while CORBA-compliant applications may be written in any language. Using Java gives us one more benefit – it is easy to tweak the server part to allow dynamic load balancing using dynamic distribution [ROU98].

[11] DOM – Document Object Model. A formalized model of hypertext document objects to allow manipulation of these elements from scripts and external programs.

We could use HTTP protocol for delivering applet to user's machine, which then starts talking to server in the language of RMI or CORBA. Both of them define a virtual interface which can be accessed from outside. We only need to call methods included in these interfaces without having to worry about any low-level network-related programming issues.

Now that we know how to initiate and do the content negotiation between client and server, we can design interfaces that both sides export (for two-way communication we need to have interface on both sides). This will enable us to call clients from server or vice-versa asynchronously, which could be used in creating educational student applications that can be controlled through teachers UI (for giving collective guidelines, for instance).

Java applets can have any kind of UI widgets, which makes it easy to create virtual CAD laboratory with all the features that native applications have. We can now create GUI that incorporates Design Process Management and other important things on the applet. The only difference from native program is that design files are sent to and tools are executed by the server.

Discussed architecture gives us one more good benefit – collaborative teamwork.

5.7 Discussion

Network-oriented applications are going to be one of the major development areas during next several years. The network computing is getting more and more ground in many areas, especially in computing-intensive applications.

The versions that have been discussed in this chapter illustrate the nature of network-based computing. They also points out reefs, onto which are easy to sail. We have seen that the ability to use applications remotely significantly increases their usefulness and productivity.

The distributed, multimedia-enabled world of computing is entering our neighborhood very soon, so it is important to be prepared for accepting the challenge.

6 Conclusions

In this chapter we will conclude our work and see what can be done in the future.

6.1 Summary

In this thesis we have developed two versions of interactive, open and dynamic user interfaces to CAD frameworks that can be used in education. One of them is fixed to one platform and only one user can access it at a time. On the other hand, it gives better performance, and better design process management, as well as richer design environment.

The network-enabled multi-user version allows collaboarted projects to be carried out and is easily accessible from any point in the world having Internet connection.

6.2 Future work

Both projects are still on going and have many things that could be enhanced.

In the single-user version the following topics are being developed: GUI plug-in API, automation interface, full Design Process Manager with hierarchical design flow display, scripting language, extended customizability.

The network-oriented version could be extended to use Level 3 architecture with dedicated VILAB application server. This way we are free to create networked version of TT CAD framework. Moreover, Java language allows creating system that distributes computing modules of the server among workstations that have small load [ROU98], thus solving the question of scalability.

6.3 Acknowledgements

I would like to thank everyone involved in the design and development of the Turbo Tester software, in particular Prof. Raimund Ubar, Jaan Raik, Gert Jervan, Antti Markus, Eero Ivask, Marina Brik.

I also would like to express my gratitude to those previously worked on the project, including Andres Kaasik, Ahto Buldas, Viljar Tulit, Villem Alango, Märt Saarepera and Kalle Pungas. Special thanks go to Bengt Magnhagen from University of Linköping for numerous suggestions for enhancements.

Bibliography

[BRO96] Marc H. Brown, Marc A. Najork, “Distributed Active Objects”, Fifth International World Wide Web Conference, Paris, 1996

[COO96] Alan Cooper, “Goal-Directed Software Design”, Dr.Dobbs Journal, September 1996, pp. 16-22

[IVA94] Eero Ivask, “Laboratoorsete tööde tsükkel aines ‘Digitaalseadmete diagnostika’”, Diplomitöö DT.2201-1.89-1652, TTÜ

[JER98] G.Jervan, A.Markus, P.Paomets, J.Raik, R.Ubar, “Turbo Tester: A CAD system for teaching digital test”, Microelectronics Education, Kluver Academic Publishers, 1998, pp. 287-290

[MOR98] Bryan Morgan, “Building Distributed Applications with Java and CORBA”, Dr. Dobbs Journal, April 1998, pp. 94-99.

[ROU98] Philip Rouselle, “Dynamic Distributed Systems in Java”, Dr. Dobbs Journal, April 1998, pp. 88-92.

[SCH96] Troy A. Schauls, “Writing User Definable GUIs”, Dr. Dobbs Journal, September 1996, pp. 36-39

[WOL94] Pieter van der Wolf, “CAD Frameworks. Principles and architecture”, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1994.

List of publications

Articles

- R.Ubar, A.Buldas, P.Paomets, J.Raik, V.Tulit. **A PC-based CAD System for Training Digital Test**. *Proc. of the V EUROCHIP Workshop on VLSI Design Training*. pp. 152-157, Dresden, Germany, Oct. 1994.
- R.Ubar, E.Ivask, P.Paomets, J.Raik. **A CAD System for Teaching Digital Test**. *Proc. of the 4-th Baltic Conference*. pp. 369-372, Tallinn, Estonia, Oct. 1994.
- P.Paomets, J.Raik, R.Ubar. **A CAD System for ASIC Test and Design**. *Exhibition 'Search for Partners' at the Special Session 'European Cooperation in Science, Technology and Education. Workshop on Sampling Theory and Applications.* Riga, Latvia, Sept. 1995.
- R.Ubar, J.Raik, P.Paomets, E.Ivask, G.Jervan, A.Markus. **Low-Cost CAD System for Teaching Digital Test**. *Proc. of the 1st European Workshop on Microelectronics Education.* p. 48, Villard de Lans, France, Feb. 5-6, 1996.
- R.Ubar, J.Raik, P.Paomets, E.Ivask, G.Jervan, A.Markus. **Low-Cost CAD System for Teaching Digital Test**. *Microelectronics Education. World Scientific Publishing Co. Pte. Ltd.* pp. 185-188, Grenoble, France, Feb. 1996.
- G.Jervan, A.Markus, P.Paomets, J.Raik, R.Ubar. **Teaching Test and Design with Turbo Tester Software**. *Proc. of the 3rd Advanced Training Course: Mixed Design of Integrated Circuits and Systems MIXDES'96*. pp. 589-594, Lodz, Poland, May 30 - June 1, 1996.
- J.Raik, P.Paomets. **Test Synthesis from Register-Transfer Level Descriptions**. *Proc. of the 5-th Baltic Electronics Conference*. pp. 311-314, Tallinn, Estonia, Oct. 1996.
- G.Jervan, A.Markus, P.Paomets, J.Raik, R.Ubar. **CAD Software for Digital Test and Diagnostics**. *Proc. of the Conference on Design and Diagnostics of Electronic Circuits and Systems '97.* Ostrava, Czech Republic, May 12-14, 1997.
- G.Jervan, A.Markus, P.Paomets, J.Raik, R.Ubar. **A Set of Tools for Estimating Quality of Built-In Self-Test in Digital Circuits**. *Proc. of the International Symposium on Signals Circuits and Systems.* pp. 362-365, Iasi, Romania, Oct. 2-3, 1997.
- G.Jervan, A.Markus, P.Paomets, J.Raik, R.Ubar, "**Turbo Tester: A CAD system for teaching digital test**", *Microelectronics Education*, pp. 287-290, Kluver Academic Publishers, 1998

Reports

- J.Raik, P.Paomets. **Definition of Benchmarks**. *Technical Report FUTEG 8/95*, Aug. 1995.

- J.Krupnova, J.Raik. **High Level AG-model Synthesis from VHDL**. *Technical Report FUTEG 10/95*, Aug. 1995.
- R.Ubar, J.Krupnova, M.Brik, P.Paomets, J.Raik, E.Ivask, G.Jervan, A.Markus. **Hierarchical Test Generation System Based on Alternative Graphs**. *ATSEC Report*, Oct. 1995.

Presentations

- J.Raik, P.Paomets, G.Jervan, A.Markus. **Test and Diagnostics Software for Digital Circuits**. Competition for student works, Riga, Oct. 31, 1996.

Avatud, dünaamiline kasutajaliides CAD-süsteemile “Turbo Tester”

Koostaja: Priidu Paomets

Resümee

Viimastel aastakümnetel on CAD-süsteemide populaarsus oluliselt kasvanud. Seda peamiselt pidevalt suureneva CAD-vahendite hulgaga, millede endale selgekstegemine on äärmiselt raske. Selleks, et orienteeruda säärases töövahendite baabelis, on tarvis süsteeme, mis oleksid võimelised keerukuse kasutaja eest ära peitma ning talle ka muul moel abiks olema.

CAD-süsteemide oluliseks komponendiks on ka disainiprotsessi sammude määramine ning läbi selle kasutaja toetamine.

Mis puutub CAD-süsteemide digitaaldiagnostika alastesse võimalustesse, siis on need senini veel suhteliselt piiratud. Neid pole kuigi lihtne kasutada vastava eriala laialdaseks õpetamiseks nende piiratud algoritmide valiku, suure installatsiooni ning suurte nõudmiste tõttu riistvarale ja mälule.

Antud töös on välja pakutud kaks lahendust digitaaldignostika õpetamiseks mõeldud tarkvarale. Esimene on Windows operatsioonisüsteemis töötav CAD-süsteem, mis võimaldab kasutada erinevaid meetodeid eksperimentides. Ta on avatud tänu oma lihtsale laiendatavusele pistikmoodulite (plug-in) abil ning dünaamiline tänu oma suurele ümberkonfigureermise võimalusele. Ta omab ka disainiprotsessi manageri, mis näitab graafiliselt andmete võimaliku liikumise teid ning teda on lihtne ümber kohandada vastavalt läbiviidatavte kursuste vajadustele.

Teine variant kasutab ära võimalust kasutada CAD-vahendeid läbi Interneti kasutades Webi-brauserit. Antud lahendus on üks võimalikest ning ka teised alternatiivid on ülevaatuse all. Toodud töö üheks väljundiks virtuaalse laboratooriumi projekt VILAB, mis on ellu toodud õppevahendite kasutamiseks Interneti kaudu, kasutades jaotatud arvutamise (*distributed computing’u)* võimalusi.

Juhendaja: Prof. Raimund Ubar

Appendix A. TT3 Configuration File (TESTER.INI)

```
[General]
Version = 3.0a
Build = 0001
RootDir = 'C:\Turbo Tester'
BinDir = '${RootDir}\Bin'
HelpDir = '${RootDir}\Help'
LibDir = ${RootDir}\Lib'
MediaDir = '${RootDir}\Media'
SetupDir = '${RootDir}\Setup'
StartupScript = '${SetupDir}\init_tt.bsh'
HelpFile = 'C:\Borland\Delphi 3\HELP\Delphi3.hlp'
;'${HelpDir}\tester.hlp'
;Install =
;Run =
;RunOnce =
DefaultToolGlyph = '${MediaDir}\ToolBtn.bmp'
Shell = 'c:\command.com'
DefaultDesignMode = 'STRUCTURAL'

[NewItems]
Count = 4
1 = NewItems.TextFile
2 = NewItems.RTFFile
3 = NewItems.Script
4 = NewItems.Report

[NewItems.TextFile]
Caption = 'Text File'
Description = 'Creates new plain-text file'
Icon = '${MediaDir}\TextEditL.ico'
Command = 'NewEditor(PlainText)'
Page = 'New'
IsDefault = 0

[NewItems.RTFFile]
Caption = 'Rich Text File'
Description = 'Creates new RTF file'
Icon = '${MediaDir}\New Text Doc.ico'
Command = 'NewEditor(RTF)'
Page = 'New'
IsDefault = 1

[NewItems.Script]
Caption = 'Script File'
Description = 'Creates new Turbo Tester Script file'
Icon = '${MediaDir}\Hex1.ico'
Command = 'NewScript()'
Page = 'New'
IsDefault = 0

[NewItems.Report]
Caption = 'Report'
Description = 'Creates new Report'
Icon = '${MediaDir}\Report.ico'
Command = 'NewReport()'
Page = 'New'
IsDefault = 0

[OpenItems]
Count = 7
1 = OpenItems.AGM
2 = OpenItems.Pattern
3 = OpenItems.Script
4 = OpenItems.Report
5 = OpenItems.RTF
6 = OpenItems.Text
7 = OpenItems.All
```

```
[OpenItems.AGM]
Caption =
Filter = 'AG Model (*.agm)|*.agm'
Command = 'OpenDesign("${FileName}")'
Descr = 'Opens Turbo Tester Design Model'

[OpenItems.Pattern]
Caption =
Filter = 'Test Patterns
(*.tst;*.cts;*.bil;*.flt;*.mvl;*.pak)|*.tst;*.cts;*.bil;*.flt;*.mvl;*.pak'
Command = 'OpenEditor("${FileName}", edPlainText)'
Descr = 'Opens Tester Patterns'

[OpenItems.Script]
Caption =
Filter = 'Script Files (*.bsh)|*.bsh'
Command = 'OpenScript("${FileName}")'
Descr = 'Opens Turbo Tester Script File'

[OpenItems.Report]
Caption =
Filter = 'Report Files (*.rep)|*.rep'
Command = 'OpenEditor("${FileName}", edPlainText)'
Descr = 'Opens Turbo Tester Report'

[OpenItems.RTF]
Caption =
Filter = 'Rich Text Files (*.rtf)|*.rtf'
Command = 'OpenEditor("${FileName}", edRTF)'
Descr = 'Opens Rich Text Format File for editing'

[OpenItems.Text]
Caption =
Filter = 'Text Files (*.txt)|*.txt'
Command = 'OpenEditor("${FileName}", edPlainText)'
Descr = 'Opens Text File for editing'

[OpenItems.All]
Caption =
Filter = 'All Files (*.*)|*.*'
Command = 'OpenAsk("${FileName}")'
Descr = 'Allows you to open any file and then asks for type'

[RunItems]
Count = 1
1 = RunItems.Script

[RunItems.Script]
Caption =
Filter = 'Script Files (*.bsh)|*.bsh'
Command = 'RunScript("${FileName}")'
Descr =

[ImportItems]
Count = 1
1 = '${SetupDir}\imp_edif.ini'
;VHDL = '${SetupDir}\imp_vhdl.ini'

[Components]
Count = 7
1 = '${SetupDir}\random.ini'
2 = '${SetupDir}\determ.ini'
3 = '${SetupDir}\bilbo.ini'
4 = '${SetupDir}\cstp.ini'
5 = '${SetupDir}\faultsim.ini'
6 = '${SetupDir}\mvlsim.ini'
7 = '${SetupDir}\optimize.ini'

[Components.DefaultIcons]
Default = '${MediaDir}\normal.ico'
Normal = '${MediaDir}\normal.ico'
Running = '${MediaDir}\run.ico'
AnimatedRunning = '${MediaDir}\run.ani'
Selected = '${MediaDir}\select.ico'
Stopped = '${MediaDir}\stop.ico'
Failed = '${MediaDir}\fail.ico'
Finished = '${MediaDir}\finish.ico'
```

```
[Components.DefaultButtonGlyphs]
Run = '${MediaDir}\btn_run.bmp'
Stop = '${MediaDir}\btn_stop.bmp'
Help = '${MediaDir}\btn_help.bmp'
Properties = '${MediaDir}\btn_props.bmp'

[Components.DefaultButtonHints]
Run = 'Run'
Stop = 'Stop'
Help = 'Help'
Properties = 'Properties'

[Pages]
Count = 2
1 = '${SetupDir}\page_blank.ini'
2 = '${SetupDir}\page_sag.ini'

[Tools]
Count = 5
1 = Tools.Report
2 = <SEPARATOR>
3 = Tools.EditModel
4 = Tools.EditPat
5 = Tools.Waveforms

[Tools.Report]
Caption = 'Generate Report'
Description = ''
Hint = 'Report|Generate Report'
Command = 'RunDLL("${BinDir}\report.dll")'
Glyph = '${MediaDir}\RepBtn.bmp'

[Tools.EditModel]
Caption = 'Edit Model'
Description = ''
Hint = 'Model|Edit Design Model'
Command = 'RunApp("${BinDir}\EdModel.exe", "${DesignFile}", swShowNormal)'
Glyph = ''

[Tools.EditPat]
Caption = 'Edit Test Patterns'
Description = ''
Hint = 'Patterns|Edit Test Patterns'
Command = 'RunApp("${BinDir}\EdPat.exe", "${DesignFile}", swShowNormal)'
Glyph = ''

[Tools.Waveforms]
Caption = 'Waveform Display'
Description = ''
Hint = 'Waveforms|View Waveforms'
Command = 'RunApp("${BinDir}\WaveFrom.exe", "${DesignFile}", swShowNormal)'
Glyph = ''

[WebPages]
Count = 7
1 = '&Project News|http://www.pld.ttu.ee/project.html'
2 = <SEPARATOR>
3 = '&Turbo Tester Home Page|http://www.pld.ttu.ee/tt/'
4 = '&Best of the Web|http://www.pld.ttu.ee/links.html'
5 = '&Search the Web in Estonia|http://www.ee/www'
6 = <SEPARATOR>
7 = '&Design and Test Center Home Page|http://www.pld.ttu.ee'
```

Appendix B. TT3 Tool Description File

```
[General]
Left = 0
Top = 0
Width = 69
Height = 80
Caption = 'Random'
Name = Random
Title = 'Random Test Pattern Generator'
Hint = 'Random Test Pattern Gererator'
HelpContext = 10000
HelpFile = ''
Script = '${SetupDir}\mixed.bsh'
Command = 'call mixed ${switches} ${design}'
Input = Model
Output = TestPatterns
OutExt = 'TST'
Hierarchical = False
HierarchyPage = '${Page(Random_sub1)}'

[Icons]
Default = '${MediaDir}\normal.ico'
Normal = '${MediaDir}\normal.ico'
Running = '${MediaDir}\run.ico'
AnimatedRunning = '${MediaDir}\run.ani'
Selected = '${MediaDir}\select.ico'
Paused = '${MediaDir}\pause.ico'
Stopped = '${MediaDir}\stop.ico'
Failed = '${MediaDir}\fail.ico'
Finished = '${MediaDir}\finish.ico'

[ButtonImages]
Run = '${MediaDir}\btn_run.bmp'
Stop = '${MediaDir}\btn_stop.bmp'
Help = '${MediaDir}\btn_help.bmp'
Properties = '${MediaDir}\btn_props.bmp'

[ButtonHints]
Run = 'Run ${Title}'
Stop = 'Stop'
Help = 'Help'
Properties = 'Properties'

[PropertyPages]
; Define Page Names here, like follows
; 1 = 'General'
; 2 = 'Advanced'

[Switches]
; These Switches will be inserted into command-line in the order they are defined
failure_limit = [switches.failure_limit]
packages = [switches.packages]
pack_size = [switches.pack_size]
select_max = [switches.select_max]
criterion = [switches.criterion]

[switches.failure_limit]
Caption = 'Failure limit:'
Key = '-failure_limit ${key_value}'
Hint = ''
HelpContext = 10001
GroupIdx = 0
JoinIdx = 0
ExclIdx = 0
Type = int_dec
PropertyPage = 0
DefaultValue = 64
Validate = 'range: x>=0;x<100'
Visible = True
```

```
TabStop = 1
Tag = 0
Required = False

[switches.packages]
Caption = 'Max packages:'
Key = '-packages ${key_value}'
Hint = ''
HelpContext = 10002
GroupIdx = 0
JoinIdx = 0
ExclIdx = 0
Type = int_dec
PropertyPage = 0
DefaultValue = 1000
Validate = 'range: x>=1;x<10000'
Visible = True
TabStop = 1
Tag = 0
Required = False

[switches.pack_size]
Caption = 'Package size x 32:'
Key = '-pack_size ${key_value}'
Hint = ''
HelpContext = 10003
GroupIdx = 0
JoinIdx = 0
ExclIdx = 0
Type = int_dec
PropertyPage = 0
DefaultValue = 1
Validate = 'range: x>=1;x<100'
Visible = True
TabStop = 1
Tag = 0
Required = False

[switches.select_max]
Caption = 'Max selected vectors:'
Key = '-select_max ${key_value}'
Hint = ''
HelpContext = 10004
GroupIdx = 0
JoinIdx = 0
ExclIdx = 0
Type = int_dec
PropertyPage = 0
DefaultValue = 2
Validate = 'range: x>=1;x<100'
Visible = True
TabStop = 1
Tag = 0
Required = False

[switches.criterion]
Caption = 'Selection criterion:'
Key = '-criterion ${key_value}'
Hint = ''
HelpContext = 10005
GroupIdx = 0
JoinIdx = 0
ExclIdx = 0
Type = int_dec
PropertyPage = 0
DefaultValue = 1
Validate = 'range: x>=0;x<100'
Visible = True
TabStop = 1
Tag = 0
Required = False
```

Appendix C. TT3 Design Flow Description File

```
[General]
Caption = 'Structural AG Testbench'
Name = page_SAG
Input = Model
Output = None

[Components]
Begin = [Components.Begin]
Random = [Components.Random]
Determ = [Components.Determ]
BILBO = [Components.BILBO]
CSTP = [Components.CSTP]
FaultSim = [Components.FaultSim]
MVLSim = [Components.MVLSim]
Optimizer = [Components.Optimizer]
End = [Components.End]

[Components.Begin]
Base = Terminals.Begin
Left =
Top =
Tag = 1
GroupIdx = 0

[Components.Random]
Base = Tools.Random
Left =
Top =
Tag = 2
GroupIdx = 1

[Components.Determ]
Base = Tools.Determ
Left =
Top =
Tag = 3
GroupIdx = 1

[Components.BILBO]
Base = Tools.Bilbo
Left =
Top =
Tag = 4
GroupIdx = 1

[Components.CSTP]
Base = Tools.Cstp
Left =
Top =
Tag = 5
GroupIdx = 1

[Components.FaultSim]
Base = Tools.FaultSim
Left =
Top =
Tag = 6
GroupIdx = 2

[Components.MVLSim]
Base = Tools.MVLSim
Left =
Top =
Tag = 7
GroupIdx = 2

[Components.Optimizer]
Base = Tools.Optimizer
```

```
Left =
Top =
Tag = 8
GroupIdx = 0

[Components.End]
Base = Terminals.End
Left =
Top =
Tag = 9
GroupIdx = 0

[Joins]
; Any joins between components that need to act identically

[Exclusions]
Bilbo = Optimizer
Cstp = Optimizer

[Connections]
1 = Begin, Random.In, Determ.In, Bilbo.In, Cstp.In
2 = Random.Out, Determ.Out, Bilbo.Out, Cstp.Out, FaultSim.In, MVLSim.In
3 = FaultSim.Out, Optimizer.In
4 = MVLSim.Out, Optimizer.Out, End
```

Appendix D. TT3 Importer Description File

```
[General]
Name = ImpEdif
Caption = 'EDIF Import'
Filter = 'EDIF Files (*.edn;*.edi;*.edif)|*.edn;*.edi;*.edif'
Script =
Command = 'import ${ImpEdif.SubFormat.Key} ${ImpEdif.ClockSignal} ${ImpEdif.GNDName}
${ImpEdif.VDDName} ${DesignFile} ${ImpEdif.LibraryFile.Text}'
Description = 'This filter allows to import Electronic Design Interchange Format
(EDIF) files'
HelpFile = ''
HelpContext = 10

[SubFormats]
Count = 2
1 = SubFormats.Generic
2 = SubFormats.Cadence

[SubFormats.Generic]
Caption = 'Generic'
Key = ''
Description = ''

[SubFormats.Cadence]
Caption = 'Cadence'
Key = '-tool Cadence'
Description = ''

[Pages]
; Full ImportWizard Notebook dimensions: L = 5; T = 5; H = 281; W = 463
; Image size on each page: L = 0; T = 0; H = 281; W = 163
Count = 2
1 = Pages.Details
2 = Pages.Advanced

[Pages.Details]
Caption = 'General Settings'
Name = 'ModelInf'
Bitmap = '${MediaDir}\WizardImage.bmp'

[Pages.Advanced]
Caption = 'Advanced Settings'
Name = 'Advanced'
Bitmap = '${MediaDir}\WizardImage.bmp'

[Controls]
Count =
1 = Controls.1
2 = Controls.2
3 = Controls.3
4 = Controls.4

[Controls.1]
Type = StaticText
Caption = 'Library File: '
Left =
Top =
Width =
Height =
Page = 1
HelpContext = 0
Hint = ''
Name = ''

[Controls.2]
Type = EditFile
Left =
Top =
Width =
```

```
Height =
Page = 1
HelpContext = 0
Hint = ''
Name = 'LibraryFile'
DefValue = ''
Tag = 0
Validate = ''
Required = True
Filter = 'Library Files (*.lib)|*.lib|All Files (*.*)|*.*'
InitialDir = '${LibDir}'
Key = ''

[Controls.3]
Type = RadioGroup
Caption = ' Compaction '
Left =
Top =
Width =
Height =
Page = 1
HelpContext = 0
Hint = ''
Name = 'LibraryFile'
DefValue = ''
Tag = 0
Validate = ''
Required = True
Key = '${group_key()}'
Items = Controls.3.Items

[Controls.3.Items]
0 = 'Macro|'
1 = 'Gate|-gate_level'

[Controls.4]
Type = RadioGroup
Caption = ' Circuit Type '
Left =
Top =
Width =
Height =
Page = 1
HelpContext = 0
Hint = ''
Name = 'LibraryFile'
DefValue = ''
Tag = 0
Validate = ''
Required = True
Key = '${group_key()}'
Items = Controls.4.Items

[Controls.4.Items]
0 = 'Combinational|'
1 = 'Sequential|'
```

Appendix E. Sample of the SDF Format

Below is one hypothetical version of the SDF file (note, that it is not an actual proposal, rather it is just a vision):

```
<?XML VERSION="1.0" ENCODING="UTF-8" RMD="NONE"?>
<!DOCTYPE VILAB_SERVICE SYSTEM "VILABSVC.DTD">
<VILAB_SERVICE>
  <PROVIDER>
     <ID>DTC</ID>
     <NAME HREF="http://www.pld.ttu.ee/">Design and Test Center</NAME>
     <INSTITUTION>Tallinn Technical University</INSTITUTION>
     <EMAIL>vl@pld.ttu.ee</EMAIL>
     <PHONE>+(372) 6 202 253</PHONE>
     <FAX>+(372) 6 202 253</FAX>
     <VILAB HREF="http://www.pld.ttu.ee/vilab.html">DTC VILAB Page</VILAB>
  </PROVIDER>
  <SERVICE HREF="http://www.pld.ttu.ee/vl/tt/">
    <TITLE>Turbo Tester</TITLE>
    <VERSION>3.0</VERSION>
    <DESCRIPTION>
      Turbo Tester VL Edition is a set of ATPG tools...
    </DESCRIPTION>
    <SUPPORT TYPE="e-mail">tt@pld.ttu.ee</SUPPORT>
    <CATEGORY>Test Generators</CATEGORY>

    <FORMATS>
      <FORMAT TYPE="naitive" ID="AG">AG Format</FORMAT>
      <FORMAT TYPE="import" ID="EDIF" LANG="EDIF 2.0.0">EDIF Format</FORMAT>
      <SUBFORMAT FOR="EDIF" ID="GenEDIF">Generic EDIF</SUBFORMAT>
      <SUBFORMAT FOR="EDIF" ID="CadEDIF">Cadence-specific EDIF</SUBFORMAT>
      <FORMAT_ALIAS FOR="EDIF" LANG="EDIF 3.0.0"/>
    </FORMATS>

    <TOOL ID="RND" HREF="http://www.pld.ttu.ee/cgi-bin/vltt.pl">Random ATPG</TOOL>
    <TOOL ID="DETERM" HREF="http://www.pld.ttu.ee/cgi-bin/vltt.pl">Deterministic
ATPG</TOOL>
    <TOOL ID="BILBO" HREF="http://www.pld.ttu.ee/cgi-bin/vltt.pl">BILBO
Emulation</TOOL>
    <TOOL ID="CSTP" HREF="http://www.pld.ttu.ee/cgi-bin/vltt.pl">CSTP
Emulation</TOOL>
    <TOOL ID="FSIM" HREF="http://www.pld.ttu.ee/cgi-bin/vltt.pl">Fault
Simulator</TOOL>
    <TOOL ID="MVSIM" HREF="http://www.pld.ttu.ee/cgi-bin/vltt.pl">Multi-valued
Simulator</TOOL>
    <TOOL ID="OPTIM" HREF="http://www.pld.ttu.ee/cgi-bin/vltt.pl">Test-set
Optimizer</TOOL>
    <TOOL ID="REPORT" HREF="http://www.pld.ttu.ee/cgi-bin/vltt.pl">Report Generator
</TOOL>
</SERVICE>
</VILAB_SERVICE>
```

Appendix F. VILAB Python Script Files

VL.PY

```
#!c:\program files\sambar41\python\python.exe -u

import sys
sys.path.insert(0, "c:\\program files\\sambar41\\docs\\")
import traceback
import cgi
import tttools

sys.stderr = sys.stdout

vltools = {'Turbo Tester': '/vl/tt/vltt.py',
        #   'Sambar Web Server': vl_root
          }

vltext = """DTC Virtual Laboratory hosts following tools for common use.
      For more information about Virtual Laboratory as such, press the Help
      button below.
"""

form = cgi.FieldStorage()

# -----------------  Begin a VL page ------------------------
if form.has_key("Service"):
  if vltools.has_key(form["Service"].value):
    tttools.print_html_location(href=vltools[form["Service"].value])
else:
# ---------------------- Build Service Selections Page --------------------------
  st = ""
  idx = 0;
  for key in vltools.keys():
    st2 = "      var option%d = new Option(\"%s\", \"%s\");\n" % (idx, key, key)
    st = st + st2
    st2 = "      document.serviceForm.Service.options[%d] = option%d;\n" % (idx, idx)
    st = st + st2
    idx = idx + 1
  if st != "":
    st = st + "      document.serviceForm.Service.options[0].selected = true;\n"



  tttools.print_html_header()
  tttools.print_html_head(head_info="  <SCRIPT LANGUAGE=\"JavaScript\">\n" \
    "    function setupSelectBoxes() {\n" + st + "    }\n  </SCRIPT>\n")
  tttools.start_vl_body(form_name="serviceForm", onload_call="setupSelectBoxes();",
    form_method="POST", form_action="/vl/vl.py")
  try:
    try:
      print "     ",
      tttools.print_vl_h1("Welcome")

      print "      <P>" + vltext + "      </P>\n"

      print "     ",
      tttools.print_vl_h2("Please select the tool of your choice:")
      print "      <SELECT NAME=\"Service\" SIZE=7>"
      print "        <OPTION VALUE=\"\">-- Placeholder for VL Services --------------
-------------------------------"
      print "      </SELECT>"
    except:
      traceback.print_exc()
  finally:
    tttools.print_vl_buttons(
      buttons = (
        ('Help', '  Help  ', 'openHelp(\'vl_help.html\')'),
        ('Next', ' Select ', 'document.serviceForm.submit();')
```

```
      )
    )
    tttools.end_vl_body()
# -----------------------  End Service Selection Page ---------------------------
```

VLTT.PY

```
#!c:\program files\sambar41\python\python.exe -u

import sys
sys.path.insert(0, "c:\\program files\\sambar41\\docs\\")
import traceback
import cgi
import tttools

sys.stderr = sys.stdout

form = cgi.FieldStorage()

# -----------------  Begin a VL page ------------------------
if form.has_key("Format"):
  if form.has_key("DesignFile"):
    if form["DesignFile"].file and form["DesignFile"].value != "":
      # Now we have got Formats and DesignFile from the user
      # So, it's time to create a temporary session folder holding user's data files

      designFileItem = form["DesignFile"]
      try:
        sessionID, designName, designFile = tttools.create_vl_session(designFileItem)
        if form["Format"].value == "AG":
          tttools.print_html_header()
          tttools.print_html_head()
          tttools.start_vl_body(form_name="agokForm", form_method="POST",
            form_action="/vl/tt/vlttrun.py", image="/pics/school.gif")
          tttools.print_vl_h1("Ready to continue")
          print "<BR>"
          tttools.print_vl_h2("Design uploaded successfully")
          print "<BR>"
          print "Details:<BR><TEXTAREA COLS=46 ROWS=9 READONLY>\n"
          print "Design: %s" % designName
          print "Session ID: %s" % sessionID
          print "Design Format: AG",
          print "</TEXTAREA>\n"
          print "<INPUT TYPE=\"HIDDEN\" NAME=\"Design\" VALUE=\"%s\">" % designName
          print "<INPUT TYPE=\"HIDDEN\" NAME=\"SessionID\" VALUE=\"%s\">" % sessionID
          tttools.print_vl_buttons(
            buttons = (
              ('Help', '  Help  ', 'openHelp(\'vl_help.html\');'),
              ('Next', '   OK   ', 'document.agokForm.submit();')
            )
          )
          tttools.end_vl_body()
        elif form["Format"].value == "EDIF":
          # EDIF Properties
          sc = """<SCRIPT LANGUAGE="JavaScript">
  <!--
      function checkCT()
      {
        if (document.edifForm.Compaction[0].checked) {
//          document.edifForm.ClockName.visible = false;
        }
        else if (document.edifForm.Compaction[0].checked) {
//          document.edifForm.ClockName.visible = true;
        }
      }
  // -->
  </SCRIPT>
"""
          tttools.print_html_header()
          tttools.print_html_head(head_info=sc)
          tttools.start_vl_body(form_name="edifForm", form_method="POST",
            form_action="/vl/tt/vltt.py", form_encoding="multipart/form-data",
            image="/pics/resource.gif")
          tttools.print_vl_h2("Library File:")
```

```
          print "<INPUT TYPE=\"radio\" NAME=\"LibType\" ID=LTS VALUE=\"Std\"
CHECKED><LABEL FOR=LTS>Standard Library:</LABEL>"
          print "<SELECT NAME=\"StdLib\" SIZE=1>"
          print "  <OPTION VALUE=\"ALTERA_P\">ALTERA_P</OPTION>"
          print "  <OPTION VALUE=\"BENCH\">BENCH</OPTION>"
          print "  <OPTION VALUE=\"ECPD07\">ECPD07</OPTION>"
          print "  <OPTION VALUE=\"ES2S2030\">ES2S2030</OPTION>"
          print "  <OPTION VALUE=\"SOLO\">SOLO</OPTION>"
          print "  <OPTION VALUE=\"VSC370\">VSC370</OPTION>"
          print "</SELECT>"
          print "<BR>"
          print "<INPUT TYPE=\"radio\" NAME=\"LibType\" ID=LTC
Value=\"Custom\"><LABEL FOR=LTC>Custom File:</LABEL>"
          print "   <INPUT TYPE=\"file\" NAME=\"LibFile\" SIZE=17 ACCEPT=\"text/*\">"
          print "<BR><BR>"
          print "<TABLE BORDER=0 WIDTH=\"100%\" CELLPADDING=0 CELLSPACING=0>"
          print "<TR>"
          print "  <TD WIDTH=\"50%\">",
          tttools.print_vl_h2("Compaction:")
          print "    <INPUT TYPE=\"radio\" NAME=\"Compaction\" ID=CM VALUE=\"MACRO\"
CHECKED><LABEL FOR=CM>Macro</LABEL><BR>"
          print "    <INPUT TYPE=\"radio\" NAME=\"Compaction\" ID=CG
VALUE=\"GATE\"><LABEL FOR=CG>Gate</LABEL><BR>"
          print "  </TD>"
          print "  <TD>",
          tttools.print_vl_h2("Circuit Type:")
          print "    <INPUT TYPE=\"radio\" NAME=\"CircuitType\" ID=CTC
VALUE=\"COMBINATIONAL\" CHECKED onClick=\"checkCT();\"><LABEL
FOR=CTC>Combinational</LABEL><BR>"
          print "    <INPUT TYPE=\"radio\" NAME=\"CircuitType\" ID=CTS
VALUE=\"SEQUENTIAL\" onClick=\"checkCT();\"><LABEL FOR=CTS>Sequential</LABEL><BR>"
          print "  </TD>"
          print "</TR>"
          print "</TABLE>"
          print "<BR>"
          print "<!--INPUT TYPE=\"checkbox\" ID=GAG NAME=\"GenAG\"><LABEL
FOR=GAG>General AG Format</LABEL><BR>"
          print "<BR-->"
          tttools.print_vl_h2("Net names:")
          print "   <LABEL FOR=CNN>     Clock net
name:</LABEL>"
          print "   <INPUT TYPE=\"text\" NAME=\"ClockName\" ID=CNN SIZE=28
VALUE=\"Clock\"><BR>"
          print "  <INPUT TYPE=\"checkbox\" ID=IGND NAME=\"IsGND\"><LABEL
FOR=IGND>GND net name: </LABEL>"
          print "    <INPUT TYPE=\"text\" NAME=\"GNDName\" SIZE=28
VALUE=\"GND\"><BR>"
          print "  <INPUT TYPE=\"checkbox\" ID=IVDD NAME=\"IsVDD\"><LABEL
FOR=IVDD>GND net name: </LABEL>"
          print "    <INPUT TYPE=\"text\" NAME=\"VDDName\" SIZE=28 VALUE=\"VDD\">"
          print "<INPUT TYPE=\"HIDDEN\" NAME=\"Design\" VALUE=\"%s\">" % designName
          print "<INPUT TYPE=\"HIDDEN\" NAME=\"DesignBin\" VALUE=\"%s\">" %
designFile
          print "<INPUT TYPE=\"HIDDEN\" NAME=\"SessionID\" VALUE=\"%s\">" % sessionID
          print "<INPUT TYPE=\"HIDDEN\" NAME=\"Format\" VALUE=\"EDIF\">"
          if form.has_key("SubFormat") and form["SubFormat"].value != "":
            print "<INPUT TYPE=\"HIDDEN\" NAME=\"SubFormat\" VALUE=\"%s\">" %
form["SubFormat"].value
          tttools.print_vl_buttons(reset_action="document.edifForm.reset();",
            buttons = (
              ('Help', '  Help  ', 'openHelp(\'vl_help.html\')'),
              ('Back', ' < Back ', 'gotoPage(\'/vl/tt/vltt.py\')'),
              ('Next', ' Import ', 'document.edifForm.submit();')
            )
          )
          tttools.end_vl_body()
      except:
        # here may come all kind of errors produced by dir & file creation
        pass
    else:
      tttools.print_error(title="Error Filling Form", subtitle="Null Design File",
        message="You have filled all fields, but the Design File you submitted " \
          "was not a valid filename or was of zero length!",
        ok_action="gotoPage('/vl/tt/vltt.py\')",
help_action="openHelp('vl_help.html')")
  else:
    if form.has_key("SessionID") and form.has_key("Design"):
```

```
      # Import Format. First Setup executable and parameters,
      executable, params = "", ""
      format, subformat = "", ""
      design = form["Design"].value
      sessionID = form["SessionID"].value
      if form.has_key("Format") and form["Format"].value != "":
        format = form["Format"].value
      if form.has_key("SubFormat") and form["SubFormat"].value != "":
        subformat = form["SubFormat"].value

      if format == "EDIF":
        executable = tttools.vl_tool_dir + tttools.vl_import_edif
        if subformat == "CADENCE":
          params = "-tool cadence"
        if form.has_key("Compaction") and form["Compaction"].value == "GATE":
          params = params + " -gate_level"
        if form.has_key("CircuitType") and form["CircuitType"].value == "SEQUENTIAL":
          pass
        if form.has_key("GenAG") and form["GenAG"].value == "1":
          params = params + " -general"
        if form.has_key("ClockName"):
          params = params + " -clock " + form["ClockName"].value
        if form.has_key("IsGND") and form.has_key("GNDName"):
          params = params + " -gnd " + form["GNDName"].value
        if form.has_key("IsVDD") and form.has_key("VDDName"):
          params = params + " -vdd " + form["VDDName"].value

        if form.has_key("DesignBin"):
          params = params + " " + form["DesignBin"].value
        if form.has_key("LibType") and form.has_key("StdLib"):
          if form["StdLib"].value == "ALTERA_P":
            params = params + " \"" + tttools.vl_lib_dir + "altera_p.lib\""
          if form["StdLib"].value == "BENCH":
            params = params + " \"" + tttools.vl_lib_dir + "bench.lib\""
          if form["StdLib"].value == "ECPD07":
            params = params + " \"" + tttools.vl_lib_dir + "ecpd07.lib\""
          if form["StdLib"].value == "ES2S2030":
            params = params + " \"" + tttools.vl_lib_dir + "es2s2030.lib\""
          if form["StdLib"].value == "SOLO":
            params = params + " \"" + tttools.vl_lib_dir + "solo.lib\""
          if form["StdLib"].value == "VSC370":
            params = params + " \"" + tttools.vl_lib_dir + "vsc370.lib\""

      res = tttools.import_vl_format(design, sessionID, executable, params, format,
subformat)
      if res != "":     # Import was successful
        pass
    else:
      tttools.print_error(title="Error Filling Form", subtitle="Missing Fields",
        message="You need to provide Format, it\'s subformats, if available and,
Design File. " \
        "Please note also that you must provide non-zero design file!",
        ok_action="gotoPage('/vl/tt/vltt.py')",
help_action="openHelp('vl_help.html')")
else:
# ---------------------- Build Service Selections Page --------------------------
  st = """    <SCRIPT LANGUAGE="JavaScript">\n  <!-- Hide script text\n
      function clearSubFormats() {
        while (document.formatForm.SubFormat.options.length > 0) {
          document.formatForm.SubFormat.options[0] = null;
        }
      }

      function changeSubFormat(iIndex) {
        if (iIndex == 0) {      // It's AG format
          clearSubFormats();
        } else if (iIndex == 1) {  // It's EDIF format
          clearSubFormats();
          var option0 = new Option("Generic EDIF", "GENERIC");
          document.formatForm.SubFormat.options[0] = option0;
          var option1 = new Option("Cadence-specific EDIF", "CADENCE");
          document.formatForm.SubFormat.options[1] = option1;

          document.formatForm.SubFormat.options[0].selected = true;
        } /*else if (iIndex == 2) { // It's VHDL format
          clearSubFormats();
          var option10 = new Option("Generic VHDL", "GENERIC");
```

```
          document.formatForm.SubFormat.options[0] = option10;

          document.formatForm.SubFormat.options[0].selected = true;
        }*/
     }

       function validate()
       {
         var retVal = false;
         retVal = document.formatForm.DesignFile.value != "";
         return retVal;
       }

       function setupSelectBoxes()
       {
          var option0 = new Option("AG Model", "AG");
          document.formatForm.Format.options[0] = option0;
          var option1 = new Option("EDIF", "EDIF");
          document.formatForm.Format.options[1] = option1;
          //var option2 = new Option("VHDL", "VHDL");
          //document.formatForm.Format.options[2] = option2;

          document.formatForm.Format.options[0].selected = true;

          document.formatForm.SubFormat.options[0] = null;
       }

     function nextButtonClick()
     {
         if (validate() == true) {
           document.formatForm.submit();
         } else {
           alert("You must specify Design name before you can continue.");
         }
       }
     // -->
     </SCRIPT>
"""

  tttools.print_html_header()
  tttools.print_html_head(head_info=st)
  tttools.start_vl_body(form_name="formatForm", onload_call="setupSelectBoxes();",
    form_method="POST", form_action="/vl/tt/vltt.py",
    form_encoding="multipart/form-data", image="/pics/resource.gif")
  try:
    try:
      print "     ",
      tttools.print_vl_h2("Please select the format of your design:")
      print "     <SELECT NAME=\"Format\" SIZE=5
ONCHANGE=\"changeSubFormat(this.selectedIndex);\">"
      print "       <OPTION VALUE=\"\" >-- Placeholder for Design Formats -----------
--------------------------"
      print "     </SELECT>"
      print "     <BR>\n    ",
      tttools.print_vl_h2("Please select a subformat of your selected format (if
any):")
      print "      <SELECT NAME=\"SubFormat\" SIZE=5>"
      print "        <OPTION VALUE=\"\" >-- Placeholder for Design Subformats -------
-------------------------"
      print "      </SELECT>"
      print "      <BR><BR>"
      print "      ",
      tttools.print_vl_h2("Design File:")
      print "      <INPUT TYPE=\"file\" NAME=\"DesignFile\" SIZE=33
ACCEPT=\"text/*\">"
    except:
      traceback.print_exc()
  finally:
    tttools.print_vl_buttons(reset_action="document.formatForm.reset();",
      buttons = (
        ('Help', '  Help  ', 'openHelp(\'vl_help.html\')'),
        ('Back', ' < Back ', "gotoPage('/vl/vl.py')"),
        ('Next', ' Next > ', 'nextButtonClick();')
      )
    )
    tttools.end_vl_body()
# -----------------------  End Service Selection Page ---------------------------
```

VLTTRUN.PY

```
#!c:\program files\sambar41\python\python.exe -u

import sys
sys.path.insert(0, "c:\\program files\\sambar41\\docs\\")
import traceback
import cgi
import tttools

sys.stderr = sys.stdout

form = cgi.FieldStorage()

# -----------------  Begin a TT Tools page ------------------------
if form.has_key("Design") and form.has_key("SessionID"):
  Design = form["Design"].value
  SessionID = form["SessionID"].value
  if form.has_key("Tool"):       # Request to execute tool
    tttools.exec_tool(Design, SessionID, form["Tool"].value, form)
  elif form.has_key("GetTool"):  # It's a request to get tool's properties
    tttools.print_tool_props(Design, SessionID, form["GetTool"].value)
  else:                          # Request to display available tools
    tttools.print_html_header()
    tttools.print_html_head()
    tttools.start_vl_body(form_name="tttoolForm", form_method="POST",
      form_action="/vl/tt/vlttrun.py", image="/pics/resource.gif")
    tttools.print_vl_h1("Turbo Tester Tools")
    print "      <BR>"
    print "      <INPUT TYPE=\"radio\" NAME=\"GetTool\" CHECKED VALUE=\"RANDOM\"
ID=Rnd><LABEL FOR=\"RND\" CLASS=\"ToolItemName\">Random Test Generator</LABEL><BR>"
    print "      <INPUT TYPE=\"radio\" NAME=\"GetTool\" VALUE=\"DETERM\"
ID=Det><LABEL FOR=\"Det\" CLASS=\"ToolItemName\">Deterministic Test
Generator</LABEL><BR>"
    print "      <INPUT TYPE=\"radio\" NAME=\"GetTool\" VALUE=\"BILBO\"
ID=BILBO><LABEL FOR=\"BILBO\" CLASS=\"ToolItemName\">BILBO Emulation</LABEL><BR>"
    print "      <INPUT TYPE=\"radio\" NAME=\"GetTool\" VALUE=\"CSTP\" ID=CSTP><LABEL
FOR=\"CSTP\" CLASS=\"ToolItemName\">CSTP Emulation</LABEL><BR>"
    print "      <BR>"
    print "      <INPUT TYPE=\"radio\" NAME=\"GetTool\" VALUE=\"FA\" ID=FA><LABEL
FOR=\"FA\" CLASS=\"ToolItemName\">Fault Simulator</LABEL><BR>"
    print "      <INPUT TYPE=\"radio\" NAME=\"GetTool\" VALUE=\"MVL\" ID=MVL><LABEL
FOR=\"MVL\" CLASS=\"ToolItemName\">Multi-valued Fault Simulator</LABEL><BR>"
    print "      <BR>"
    print "      <INPUT TYPE=\"radio\" NAME=\"GetTool\" VALUE=\"OPT\" ID=Opt><LABEL
FOR=\"Opt\" CLASS=\"ToolItemName\">Test-Set Optimizer</LABEL><BR>"
    print "      <BR>"
    print "      <INPUT TYPE=\"radio\" NAME=\"GetTool\" VALUE=\"REPORT\"
ID=Rep><LABEL FOR=\"Rep\" CLASS=\"ToolItemName\">Report Generator</LABEL><BR>"
    print "      <INPUT TYPE=\"HIDDEN\" NAME=\"Design\" VALUE=\"%s\">" % Design
    print "      <INPUT TYPE=\"HIDDEN\" NAME=\"SessionID\" VALUE=\"%s\">" % SessionID
    tttools.print_vl_buttons(
      buttons = (
        ('Help', '  Help  ', 'openHelp(\'vl_help.html\');'),
        ('Back', ' < Back ', "gotoPage('%s/vl/tt/vltt.py\')"),
        ('Next', ' Select ', 'document.tttoolForm.submit();')
      )
    )
    tttools.end_vl_body()
else:                         # Oops, no Design nor SessionID defined
  tttools.print_error(title="Missing Form Data", subtitle="Missing \"Design\" and
\"SessionID\" Fileds",
    message="You have not specified your design\nand/or session ID!\n\n" \
            "Pressing OK takes you to the page\nwhere you can specify them",
    ok_action="gotoPage('/vl/tt/vltt.py');", help_action="openHelp('vl_help.html')")
# -----------------------  End TT Tool Selection Page ----------------------------
```

VLTOOLS.PY

```
#!c:\program files\sambar41\python\python.exe -u
```

```
import sys
import cgi
import os
import string
import traceback
import time
if sys.platform == "win32":
  import win32pipe

sys.stderr = sys.stdout

vl_root = "http://127.0.0.1"
default_web_page = vl_root + "/vl/"
#session_temp_dir = "c:\\progrm files\\sambar41\\session\\"
session_temp_dir = "c:\\vltemp\\"
vl_tool_dir = "c:\\program files\\sambar41\\tools\\"
vl_lib_dir = "c:\\program files\\sambar41\\tools\\lib\\"
do_vl_info_pages = 0

vl_import_edif = "import.exe"
vl_tool_random = "random.exe"
vl_tool_determ = "determ.exe"
vl_tool_bilbo = "bilbo.exe"
vl_tool_cstp = "cstp.exe"
vl_tool_fltan = "analyze.exe"
vl_tool_mvlan = "mvl.exe"
vl_tool_optim = "optimize.exe"
vl_tool_report = "report.exe"


def print_html_header(status="", boundary="", content_type = "text/html", headers):
  if status and status != "":
    print status
  if boundary and boundary != "":
    print boundary
  print "Content-type: %s" % content_type
  if headers and headers != ():
    for hdr in headers:
      print hdr
  print

def print_html_location(status="", boundary="", href, headers):
  if status and status != "":
    print status
  if boundary and boundary != "":
    print boundary
  print "Location: %s" % href
  if headers and headers != ():
    for hdr in headers:
      print hdr
  print

def start_html():
  print "<!DOCTYPE HTML PUBLIC -//W3C//DTD HTML 3.2//EN>"
  print "<HTML>"

def print_html_head(title = "Virtual Laboratory", head_info = "", start_html=1):
  if start_html:
    print "<!DOCTYPE HTML PUBLIC -//W3C//DTD HTML 3.2//EN>"
    print "<HTML>"
  print "<HEAD>"
  print "  <TITLE>Virtual Laboratory</TITLE>"
  print "  <LINK REL=STYLESHEET HREF=\"/styles/vl_styles.css\" TYPE=\"text/css\">"
  print "  <SCRIPT SRC=\"/scripts/vl_procs.js\"></SCRIPT>"
  if head_info != "":
    print head_info
  print "</HEAD>"

def start_vl_body(center=1, form_name="", form_action="", form_method="",
form_encoding="",
  image="/pics/houshold.gif", onload_call=""):
# allowed methods: application/x-www-form-urlencoded   multipart/form-data
  print "<BODY",
  if onload_call != "":
    print " onLoad='%s'" % onload_call,
  print ">"
```

```
  if center:
    print "  <CENTER>"
  print "  <FORM",
  if form_name != "":
    print " NAME=\"%s\"" % form_name,
  if form_action != "":
    print " ACTION=\"%s\"" % form_action,
  if form_method != "":
    print " METHOD=\"%s\"" % form_method,
  if form_encoding != "":
    print " ENCTYPE=\"%s\"" % form_encoding,
  print ">"
  print "  <TABLE BORDER=1 WIDTH=628 HEIGHT=344><TR><TD>"
  print "  <TABLE BORDER=0 WIDTH=100% HEIGHT=100% ID=Wizard2>"
  print "  <TR>"
  print "    <TD ROWSPAN=2 WIDTH=200><IMG SRC=\"%s\" HEIGHT=318 WIDTH=179></TD>" %
image
  print "    <TD CELLPADDING=15 COLSPAN=2 ALIGN=LEFT VALIGN=MIDDLE WIDTH=\"*\">"

def print_vl_buttons(reset_action="", buttons = []):    # default action:
resetForm();
  print "    </TD>"
  print "  </TR>"
  print "  <TR HEIGHT=25>"
  print "    <TD ALIGN=LEFT VALIGN=BOTTOM>"
  if reset_action != "":
    print "      <INPUT TYPE=\"button\" NAME=\"Reset\" VALUE=\" Reset \"
onClick=\"%s\">" % reset_action
  else:
    print "       "
  print "    </TD>"
  print "    <TD ALIGN=RIGHT VALIGN=BOTTOM>"
  if buttons != None and buttons != []:
    for btninfo in buttons:
      btn = btninfo[0]
      btn_title = btninfo[1]
      btn_href = btninfo[2]
      print "      <INPUT TYPE=\"button\" NAME=\"%s\" VALUE=\"%s\"
onClick=\"%s\"> \n" % (btn, btn_title, btn_href)
  else:
#    if os.environ.has_key("HTTP_REFERER"):
#      print "      <INPUT TYPE=\"button\" NAME=\"Ret\" VALUE=\"  Return  \" " \
#            "onClick=\"gotoPage('%s'); return true;\">" % os.environ("HTTP_REFERER")
#    else:
    print " "

def end_vl_body(center=1, end_html=1):
  print "    </TD>"
  print "  </TR>"
  print "  </TABLE>"
  print "  </TABLE>"
  print "  </FORM>"
  if center:
    print "  </CENTER>"
  print "</BODY>"
  if end_html:
    print "</HTML>"

def end_html():
  print "</HTML>"

def print_vl_h1(value=""):
  print "<DIV CLASS=\"TabHeader\">%s</DIV>" % value

def print_vl_h2(value=""):
  print "<DIV CLASS=\"TabItemHeader\">%s</DIV>" % value


def print_error(title="Error", subtitle="Syntax Error", message="",
                ok_action="", help_action=""):
  print_html_header()
  print_html_head()
  start_vl_body(form_name="esimene")
  try:
    try:
      print_vl_h1(title)
      print "<BR>"
```

```
      print_vl_h2(subtitle)
      print "<BR>"
      print "Detailed description of the error:<BR><TEXTAREA COLS=46 ROWS=9
READONLY>\n"
      try:
        print message
      except:
        traceback.print_exc()
    finally:
      print "</TEXTAREA>\n"
  finally:
    print_vl_buttons(
      buttons = (
       ('Help', '  Help  ', help_action),
       ('Next', '   OK   ', ok_action)
      )
    )
    end_vl_body()

def exec_command(command, params):
  pass


def create_vl_session(designFileItem):
  """Create a temporary session folder. Returns SessionID, DesignName"""
  design, ext, session_id, tail = "", "", "", ""
  try:
    session_id = time.strftime("%d%m%y%H%M%S", time.localtime(time.time()))
#    print "SessionID: ", session_id, "<BR>"
    session_dir = session_temp_dir + session_id + '\\'
#    print "SessionDir: ", session_dir, "<BR>"
    os.mkdir(session_dir)
    i = string.rfind(designFileItem.filename, '\\') + 1
    head, tail = designFileItem.filename[:i], designFileItem.filename[i:]
#    print "Filename: ", tail, "<BR>"
    i = string.rfind(tail, '.') + 1
    design, ext = tail[:i-1], tail[i:]
    design = string.lower(design)
#    print "Design: ", design, " Ext: ", ext, "<BR>"
    f = open(session_dir + tail, 'wb')
    try:
      f.write(designFileItem.value)
    finally:
      f.close()
    return session_id, design, tail
  except:
#    traceback.print_exc()
    return session_id, design, tail

def import_vl_format(Design, SessionID, Executable, Params="", Format="",
SubFormat=""):
  ret = ""
  anyerror = 0
  # 1. Print the page indicating that the work is in progress

  if do_vl_info_pages:
    print_html_header(status="HTTP/1.0 200 OK", content_type="multipart/x-mixed-
replace;boundary=TERM")
    print "--TERM"
    print_html_header()
    print_html_head()
    start_vl_body(image="/pics/workout.gif", form_method="POST", form_name="running")
    print_vl_h1('Importing design')
    print "<BR>"
    if Format and Format != "":
      st = "Importing " + Format + " design \"" + Design + "\""
    else:
      st = "Importing design \"%s\"" % Design
    print_vl_h2(st)
    print "<BR><BR>"
    print "<B>Please wait...</B>"
    print_vl_buttons()
    end_vl_body()

    print "--TERM"
    sys.stdout.flush()
    time.sleep(10)
```

```
  #  for I in range (0, 10000):
  #    pass

  # 2. Execute importing app
#  try:
#    fin, fout, ferr = win32pipe.popen3('dir /c c:\\')
#    try:
#      s = fout.readline()
#      while s != "":
#        ret = ret + s
#        s = fout.readline()
#
#      s = ferr.readline()
#      while s != "":
#        ret = ret + s
#        s = ferr.readline()
#
#      s = f.readlines()
#      if type(ret) is type([]):
#        for l in s:
#          ret = ret + l + "\n"
#      else:
#        ret = s
#      pass
#    finally:
#      fin.close()
#      fout.close()
#      ferr.close()
#    pass
#  except:
#    ret = "Some error"
#    anyerror = 1

  if anyerror:
    # 3. If there was an error, display appropriate page
    pass

  else:
    # 4. If no error, return results
    if ret == "":
      ret = "%s\n" % Executable
      ret = ret + "%s\n" % Params
    print_html_header()
    print_html_head()
    start_vl_body(form_name="edifokForm", form_method="POST",
                  form_action="/vl/tt/vlttrun.py",
                  image="/pics/school.gif")
    print_vl_h1("Design Import Successful")
    print "<BR>"
    print_vl_h2("Your EDIF design was successfully imported")
    print "<BR>"
    print "Details:<BR><TEXTAREA COLS=46 ROWS=9 READONLY>\n"
    print "Design: %s" % Design
    if Format != "":
      print "Format: %s" % Format,
      if SubFormat != "":
        print "(%s)" % SubFormat
      else:
        print ""
    print "Session ID: %s\n" % SessionID
    print ret
    print "</TEXTAREA>\n"
    print "<INPUT TYPE=\"HIDDEN\" NAME=\"Design\" VALUE=\"%s\">" % Design
    print "<INPUT TYPE=\"HIDDEN\" NAME=\"SessionID\" VALUE=\"%s\">" % SessionID
    print_vl_buttons(
      buttons = (
        ('Help', '  Help  ', 'openHelp(\'vl_help.html\')'),
        ('Next', '   OK   ', 'document.edifokForm.submit();')
      )
    )
    end_vl_body()
    if do_vl_info_pages:
      print "--TERM--"

  return ret
```

```
def exec_tool(Design, SessionID, ToolName, FormData):
  print_html_header()
  print_html_head()
  start_vl_body(form_name="tttoolForm", form_method="POST",
    form_action="/vl/tt/vlttrun.py", image="/pics/videos.gif")
  if ToolName == "RANDOM":
#    print_vl_h1("Random ATPG")
#    print "<BR>"
#    FLIM
#    PKG
#    PKGSIZE
#    SELVECT
#    SELCRIT
    pass
  elif ToolName == "DETERM":
#    print_vl_h1("Deterministic ATPG")
# FAULTSIM
# BACKTRACKS
# COMPACT
    pass
  elif ToolName == "BILBO":
#    print_vl_h1("BILBO Emulation")
# GPOLY
# GINIT
# APOLY
# AINIT
# COUNT
# LSB
    pass
  elif ToolName == "CSTP":
#    print_vl_h1("CSTP Emulation")
# POLY
# INIT
# COUNT
# LSB
    pass
  elif ToolName == "FA":
#    print_vl_h1("Fault Analyzer")
# FMODEL
    pass
  elif ToolName == "MVL":
#    print_vl_h1("Multi-valued Analyzer")
# ALPHA
    pass
  elif ToolName == "OPT":
#    print_vl_h1("Test Set Optimizer")
# FAULTCOVER
# UNIQUE
    pass
  elif ToolName == "REPORT":
#    print_vl_h1("Report Generator")
    pass
  print_vl_h1("Tool " + ToolName + " finished...")
  print "<BR>Results:<BR><TEXTAREA COLS=46 ROWS=9 READONLY>\n"
  print "Design: %s" % Design
  print "Session ID: %s\n" % SessionID
  print "Number of total faults  : 20"
  print "Number of tested faults : 20"
  print "Number of vectors used  : 4"
  print "Test pattern Coverage   : 100%"
  print "</TEXTAREA>\n"
  print "<INPUT TYPE=\"HIDDEN\" NAME=\"Design\" VALUE=\"%s\">" % Design
  print "<INPUT TYPE=\"HIDDEN\" NAME=\"SessionID\" VALUE=\"%s\">" % SessionID
  print_vl_buttons(
    buttons = (
      ('Help', '  Help  ', 'openHelp(\'vl_help.html\')'),
      ('View', ' View Details ', 'openHelp(\'vl_help.html\')'),
      ('Next', '   OK   ', 'document.tttoolForm.submit();')
    )
  )
  end_vl_body()

def print_tool_props(Design, SessionID, ToolName):
  print_html_header()
  print_html_head()
  start_vl_body(form_name="tttoolForm", form_method="POST",
    form_action="/vl/tt/vlttrun.py", image="/pics/school.gif")
```

```
  if ToolName == "RANDOM":
    print_vl_h1("Random ATPG")
    print "<BR>"
    print "<PRE><LABEL FOR=\"FLim\">Failure Limit:        </LABEL><INPUT
TYPE=\"text\" SIZE=\"20\" NAME=\"FLIM\" ID=\"FLim\" VALUE=\"64\"></PRE>"
    print "<PRE><LABEL FOR=\"Pkg\">Max Packages:         </LABEL><INPUT TYPE=\"text\"
SIZE=\"20\" NAME=\"PKG\" ID=\"Pkg\" VALUE=\"1000\"></PRE>"
    print "<PRE><LABEL FOR=\"PkgSize\">Package Size x32:     </LABEL><INPUT
TYPE=\"text\" SIZE=\"20\" NAME=\"PKGSIZE\" ID=\"PkgSize\" VALUE=\"1\"></PRE>"
    print "<PRE><LABEL FOR=\"SelVect\">Max Selected Vectors: </LABEL><INPUT
TYPE=\"text\" SIZE=\"20\" NAME=\"SELVECT\" ID=\"SelVect\" VALUE=\"2\"></PRE>"
    print "<PRE><LABEL FOR=\"SelCrit\">Selection Criterion:  </LABEL><INPUT
TYPE=\"text\" SIZE=\"20\" NAME=\"SELCRIT\" ID=\"SelCrit\" VALUE=\"1\"></PRE>"
  elif ToolName == "DETERM":
    print_vl_h1("Deterministic ATPG")
    print "<PRE><INPUT TYPE=\"checkbox\" NAME=\"FAULTSIM\" CHECKED ID=\"FaultSim\">
<LABEL FOR=\"FaultSim\">Simulate Each Vector</LABEL></PRE>"
    print "<PRE><LABEL FOR=\"BLim\">Backtrack Limit: </LABEL><INPUT TYPE=\"text\"
SIZE=\"20\" NAME=\"BACKTRACKS\" ID=\"BLim\" VALUE=\"10000\"></PRE>"
    print "<PRE><INPUT TYPE=\"checkbox\" NAME=\"COMPACT\" CHECKED ID=\"Comp\"> <LABEL
FOR=\"Comp\">Pattern Compaction</LABEL></PRE>"
    pass
  elif ToolName == "BILBO":
    print_vl_h1("BILBO Emulation")
    print "<BR>"
    print "<PRE><SPAN CLASS=\"TabItemHeader\">Generator:</SPAN>"
    print "  <LABEL FOR=\"GPoly\">Polynomial:           </LABEL><INPUT TYPE=\"text\"
SIZE=\"20\" NAME=\"GPOLY\" ID=\"GPoly\" VALUE=\"00000000\">"
    print "  <LABEL FOR=\"GInit\">Inital State:         </LABEL><INPUT TYPE=\"text\"
SIZE=\"20\" NAME=\"GINIT\" ID=\"GInit\" VALUE=\"00000000\"></PRE>"
    print "<PRE><SPAN CLASS=\"TabItemHeader\">Analyzer:</SPAN>"
    print "  <LABEL FOR=\"APoly\">Polynomial:           </LABEL><INPUT TYPE=\"text\"
SIZE=\"20\" NAME=\"APOLY\" ID=\"APoly\" VALUE=\"00000000\">"
    print "  <LABEL FOR=\"AInit\">Inital State:         </LABEL><INPUT TYPE=\"text\"
SIZE=\"20\" NAME=\"AINIT\" ID=\"AInit\" VALUE=\"00000000\"></PRE>"
    print "<PRE><LABEL FOR=\"Cnt\">Number of Clock Cycles: </LABEL><INPUT
TYPE=\"text\" SIZE=\"20\" NAME=\"COUNT\" ID=\"Cnt\" VALUE=\"1000\"></PRE>"
    print "<PRE><INPUT TYPE=\"checkbox\" NAME=\"LSB\" ID=\"LSB\"><LABEL
FOR=\"LSB\">POs at the Side of the LSB</LABEL></PRE>"
    pass
  elif ToolName == "CSTP":
    print_vl_h1("CSTP Emulation")
    print "<BR>"
    print "<PRE><SPAN CLASS=\"TabItemHeader\">LFSR:</SPAN>"
    print "  <LABEL FOR=\"Poly\">Polynomial:           </LABEL><INPUT TYPE=\"text\"
SIZE=\"20\" NAME=\"POLY\" ID=\"Poly\" VALUE=\"00000000\">"
    print "  <LABEL FOR=\"Init\">Inital State:         </LABEL><INPUT TYPE=\"text\"
SIZE=\"20\" NAME=\"INIT\" ID=\"Init\" VALUE=\"00000000\"></PRE>"
    print "<PRE><LABEL FOR=\"Cnt\">Number of Clock Cycles: </LABEL><INPUT
TYPE=\"text\" SIZE=\"20\" NAME=\"COUNT\" ID=\"Cnt\" VALUE=\"1000\"></PRE>"
    print "<PRE><INPUT TYPE=\"checkbox\" NAME=\"LSB\" ID=\"LSB\"><LABEL
FOR=\"LSB\">POs at the Side of the LSB</LABEL></PRE>"
    pass
  elif ToolName == "FA":
    print_vl_h1("Fault Analyzer")
    print "<BR>"
    print "  <DIV CLASS=\"TabItemHeader\">Fault Model:</DIV>"
    print "<PRE><INPUT TYPE=\"radio\" NAME=\"FMODEL\" VALUE=\"STUCK\" ID=\"StuckAt\"
CHECKED><LABEL FOR=\"StuckAt\">Stuck-at</LABEL>"
    print "<INPUT TYPE=\"radio\" NAME=\"FMODEL\" VALUE=\"DELAY\" ID=\"Delay\"><LABEL
FOR=\"Delay\">Delay</LABEL></PRE>"
    pass
  elif ToolName == "MVL":
    print_vl_h1("Multi-valued Analyzer")
    print "<BR>"
    print "<DIV CLASS=\"TabItemHeader\">Alphabeth:</DIV>"
    print "<PRE><INPUT TYPE=\"radio\" NAME=\"ALPHA\" VALUE=\"5\" ID=\"5val\"><LABEL
FOR=\"5val\">5 - valued</LABEL>"
    print "<INPUT TYPE=\"radio\" NAME=\"ALPHA\" VALUE=\"8\" ID=\"8val\"
CHECKED><LABEL FOR=\"8val\">8 - valued</LABEL></PRE>"
    pass
  elif ToolName == "OPT":
    print_vl_h1("Test Set Optimizer")
    print "<BR>"
    print "<PRE><LABEL FOR=\"Cover\">Required fault coverage: </LABEL><INPUT
TYPE=\"text\" SIZE=\"20\" NAME=\"FAULTCOVER\" ID=\"Cover\" VALUE=\"100.0\"> %</PRE>"
```

```
    print "<PRE><INPUT TYPE=\"checkbox\" NAME=\"UNIQUE\" CHECKED ID=\"Unique\"><LABEL
FOR=\"Unique\">Select vectors detecting unique faults first</LABEL></PRE>"
    pass
  elif ToolName == "REPORT":
    print_vl_h1("Report Generator")
    print "<BR>"
    print_vl_h2("Whoops, no options known to me:(")
    pass
  print "<INPUT TYPE=\"HIDDEN\" NAME=\"Design\" VALUE=\"%s\">" % Design
  print "<INPUT TYPE=\"HIDDEN\" NAME=\"SessionID\" VALUE=\"%s\">" % SessionID
  print "<INPUT TYPE=\"HIDDEN\" NAME=\"Tool\" VALUE=\"%s\">" % ToolName
  st = "/vl/tt/vlttrun.py?Design=%s&SessionID=%s" % (cgi.escape(Design),
cgi.escape(SessionID))
  st = "gotoPage(\'" + st + "\');"
  print_vl_buttons(reset_action="document.tttoolForm.reset();",
    buttons = (
      ('Help', '  Help  ', 'openHelp(\'vl_help.html\')'),
      ('Back', ' < Back ', st),
      ('Next', '   Run  ', 'document.tttoolForm.submit();')
    )
  )
  end_vl_body()
```