Porting MATT from MatrixX to MATLAB

Jing Tao
The University of Montana

Follow this and additional works at: https://scholarworks.umt.edu/etd

Let us know how access to this document benefits you.

Recommended Citation
Tao, Jing, "Porting MATT from MatrixX to MATLAB" (2002). Graduate Student Theses, Dissertations, & Professional Papers. 5129.
https://scholarworks.umt.edu/etd/5129

This Thesis is brought to you for free and open access by the Graduate School at ScholarWorks at University of Montana. It has been accepted for inclusion in Graduate Student Theses, Dissertations, & Professional Papers by an authorized administrator of ScholarWorks at University of Montana. For more information, please contact scholarworks@mso.umt.edu.
The University of Montana

Permission is granted by the author to reproduce this material in its entirety, provided that this material is used for scholarly purposes and is properly cited in published works and reports.

**Please check "Yes" or "No" and provide signature**

Yes, I grant permission  

No, I do not grant permission

Author's Signature: ____________________________

Date: 4/16/2022

Any copying for commercial purposes or financial gain may be undertaken only with the author's explicit consent.
Porting MATT from MatrixX to MATLAB

by

Jing Tao

B.Eng. Nanjing University of Chemical Technology, Nanjing, China, 1992

Presented in partial fulfillment of the requirements

for the degree of

Master of Computer Science

Computer Science Department

The University of Montana

May 2002

Approved by:

Chairperson

Dean, Graduate School

Date

5-7-02
Porting MATT from MatrixX to MATLAB

Director: Joel E. Henry

MatrixX/MATLAB Automated Test Tool (MATT) 2.0, a tool used for real-time systems, can generate tests and run simulations on real-time software models specified in MATLAB. MATT can execute millions of tests and automatically detect defects.

In this project, MATT 2.0 was developed for both Windows and Sun Solaris platforms. Beta Version for Windows was released and Beta Version for Sun Solaris will be released soon. Compared to MATT 1.0, MATT 2.0 cooperates with MATLAB (rather than MatrixX), which creates real-time system models and runs simulation.

A software engineering approach was applied to this project. It includes requirement analysis, abstract and physical design, implementation and testing. Finally, commercial quality software products were created. According to users' feedback, the requirement of MATT 1.0 was revised to create requirement of MATT 2.0. The design and testing plan of MATT 2.0 is as same as MATT 1.0's. Windows version of MATT 2.0 was implemented by Microsoft Visual C++ and Sun Solaris version was implemented by C++ and QT.

A recommendation for future work of MATT 2.0 was given.
Acknowledgement

I wish to show my deepest gratitude to my advisor, Professor Joel E. Henry, for his guidance, support and friendship, who proved to be an excellent mentor, a wonderful collaborator with whom to share ideas, and an exceptional person in general.

Thanks are also expressed to the members of the dissertation committee: Professor Jerry D. Esmay and Professor George D. McRae for their patience with me and their concern with my project.

I wish to thank the entire Henry group, Scott Shield and Josh Stiff for their patience, help and discussion.

Finally, I express my highest gratitude to my parents, brothers and my wife, Jinsong, who have supported me through never-ending love, encouragement, generosity and friendship.
Chapter 1 Introduction

This project involves the MatrixX/MATLAB Automated Test Tool (MATT), a tool used to test real-time software via simulation. MATT performs the following functions\textsuperscript{1}:

- Automated test matrix generation based on both model and user information
- Automated model simulation within the MatrixX/MATLAB environment
- Capture and presentation of the simulation output
- Import/export of test matrices, test generation criteria, and output

Why MATT?

MATT attacks the challenging problems involved in testing real-time systems. Testing real-time systems involves hardware and software with significant risk of damage to hardware. In aeronautic and aerospace applications, the result of this damage is costly and can be dangerous. Moreover, in some cases, the hardware cannot be taken off-line in order to test and therefore presents special problems. In order to overcome these difficulties, a large amount of effective testing needs to be completed in advance to the real-time application of software to hardware\textsuperscript{2}.

Engineers often employ simulation prior to, or in conjunction with, hardware-software integration\textsuperscript{2}. This method can reduce the risk of hardware damage if done correctly. However, at the same time, other problems arise when simulation is utilized, such as generation and use of potentially huge data sets, time-consuming and error-prone output
analysis, and testing time constraints. For example, a real-time system which samples 100
input values at 100 millisecond update time, would require a simulation that supplies
1,000 input values per second. A modest one-hour simulation would require 3,600,000
values. If these values are floating point numbers, a 14.4-megabyte file is required for
this simulation. In addition, if the system produces 100 output values every 100
milliseconds, the output file would contain another 14.4 megabytes of data. Creating the
input file is overwhelming to a project team without the support of automated tools.
Furthermore, many real-time systems include the requirement to execute for months or
years without interruption. Testing of this length of time is not feasible within most
projects.

Without question, testing real-time systems presents a number of complex and time-
consuming problems. Automated testing tools are needed because they can improve
software quality, increase testing productivity, and enhance management insight into
process and product risk. MATT is this kind of tool.

Many real-time systems operate in a relatively simple fashion. The system samples
external devices (hardware), executes a single loop through a program, and then refreshes
the output values. Figure 1-1 is a typical architecture of real-time systems.

The external device in Figure 1-1 has 3 input variables and 3 output variables (Note, the
input and output in Figure 1-1 is for a real-time system). The real-time system that
controls the external device gets the input data from the device. So the real-time system
in this figure has 3 inputs. If we want to test the system, an input file, which is often referred as an input matrix, should be created. Each row in the matrix contains input values for a single input variable for all time steps. Each column contains all the input values need for execution of a single time step. Table 1-1 is an example of input matrix for the real-time system that has 9 time steps.

![Figure 1-1 Typical Real-Time System Architecture](image)

Figure 1-1 Typical Real-Time System Architecture[3]
Table 1-1 Simulation Input Matrix

<table>
<thead>
<tr>
<th></th>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
<th>Step 5</th>
<th>Step 6</th>
<th>Step 7</th>
<th>Step 8</th>
<th>Step 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input 1</td>
<td>1.0</td>
<td>1.1</td>
<td>1.2</td>
<td>1.3</td>
<td>1.4</td>
<td>1.5</td>
<td>1.6</td>
<td>1.7</td>
<td>1.8</td>
</tr>
<tr>
<td>Input 2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Input 3</td>
<td>5</td>
<td>4.9</td>
<td>4.8</td>
<td>4.7</td>
<td>4.6</td>
<td>4.5</td>
<td>4.4</td>
<td>4.3</td>
<td>4.2</td>
</tr>
</tbody>
</table>

After creating the input matrix, the simulation executes the input matrix as input. Then the output matrix is created and must be captured. Table 2-1 is an example of an output matrix. Each row in the matrix contains output values for a single output variable for all time steps. Each column contains all the output values needed for execution of a single time step.

Table 2-1 Simulation Output Matrix

<table>
<thead>
<tr>
<th></th>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
<th>Step 5</th>
<th>Step 6</th>
<th>Step 7</th>
<th>Step 8</th>
<th>Step 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output 1</td>
<td>11.3</td>
<td>11.0</td>
<td>10.7</td>
<td>10.4</td>
<td>10.1</td>
<td>9.8</td>
<td>9.5</td>
<td>9.2</td>
<td>8.9</td>
</tr>
<tr>
<td>Output 2</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
</tr>
<tr>
<td>Output 3</td>
<td>2.1</td>
<td>2.2</td>
<td>2.1</td>
<td>2.2</td>
<td>2.1</td>
<td>2.2</td>
<td>2.1</td>
<td>2.2</td>
<td>2.1</td>
</tr>
</tbody>
</table>

The difficult portions of this process are the following: quickly and efficiently generating an input matrix with meaningful test values, and capturing and analyzing the output.
matrix produced during simulation. An engineer must be able to identify a defect among
the possibly millions of output values quickly.

Engineers typically manually generated small test matrices (10 - 100 time steps) to test
for specific functionality. Larger test matrices (100 - 1000 time steps) were created using
random number generators or software programs with extremely limited capability (i.e. a
C program that created input values linearly between a minimum and maximum value).
These engineers expressed the desire to create much larger input matrices with series of
test values based on functions (i.e. randomly ascending values, oscillating values, etc.).
Equally important to engineers and managers was the ability to measure or estimate
reliability prior to hardware/software integration [2].

Previous work by Tian provided testers with test scripts that are used to construct test
cases[4]. Test scripts are based on Model Application Units to support test generation.
However, this approach requires additional automation tools to execute tests, collect and
analyze test results as well. Jeng et al. focused on automatic generation of domain test
data. Their work utilized a random function to generate thirty test cases, and then employ
simulation to execute these tests. Test results are averaged to assess the effectiveness of
the test cases in detecting defects[5].

In order to increase efficiency and performance, the MATT project team worked to
extend previous work in automated test case generation, simulation and analysis. The
target of this work is to specify, automate, and deploy measurable testing strategies into
the existing maintenance process. The automated testing strategies allow a tester to create test matrices, perform a simulation, capture output matrices, and analyze simulation output. In order to specify these strategies and develop an automated testing tool, research into test generation functions, development of a prototype system, evaluation of the prototype, and development of a commercial quality testing tool were developed[2].

**History of MATT**

This project originated from Dr. Joel Henry and a team at East Tennessee State University working for an Interchange Agreement entitled "Verification of MatrixX Models" for NASA (MatrixX is a product of Wind River Software) from January 2000 to December 2000[6]. They created a prototype and commercial quality version of the MatrixX Automated Test Tool (MATT 1.0), and specification and publication of a verification methodology based on simulation as well. The project focused on using simulation to test real-time systems prior to hardware-software integration. A simulation requires a series of values for each input variable and the collection of the series of output values produced by the simulation. The project specified a set of functions, referred to as test types that are used to create a series of input values for input variables. Strategies for selecting combinations of test types that form test cases were specified as well.

The features of MATT 1.0 are[7]:

- User selectable number of test steps
- Twenty-four test types available
- Test types selectable by individual input variable
- User selectable test limits and accuracy
- Exception reporting by output
- Summary results of tests performed
- Graphing capabilities
- Save and load testing data and parameters
- Both Unix and PC platform versions
- Distributed without cost

**Goal of our project:**

Our project will build on the existing MatrixX version of MATT 1.0 that automates test case generation, simulation, and analysis of results.

The primary focus of this project is to port MATT to MATLAB, a product of the Math Works Inc., the current industry leader in real-time system model generation on both the Windows and Sun Solaris platforms. Three years ago, MatrixX was the industry leader and widely used across NASA. However, NASA has recently shifted focus to MATLAB in many projects. MATLAB is widely used within academic and research settings as well\(^8\).
The new product for MATLAB, named MATT 2.0, must provide the same functionality and testing strategy of MATT 1.0 for MatrixX. In addition to the features of MATT 1.0, MATT 2.0 requires the following:

- New features which are unique in MATLAB will be added to MATT 2.0.
- Specification and documentation for MATT 2.0 will be created.
- MATT 2.0 will be developed as a commercial grade software tool.
Chapter 2 Approach and Solution of MATT 2.0

As described before, the goal of this project is to port MATT from MatrixX to MATLAB with a base set of source code, MATT 1.0, available. The software engineering approach utilized to develop our product is “staged”. This means that research results will be integrated into an existing product, which will then be stabilized before additional research results are added. This approach is widely used throughout industry to control requirements, reduce risk, and promptly eliminate defects. The approach also works well in academic settings where student research and development teams change every semester or every year[6].

The project plan is:

1) Pre-Beta Windows Version of MATT 2.0

In this version, MATT 2.0 has the same user interface and functionality as MATT 1.0 but will be run in MATLAB environment rather than MatrixX environment. Specification and Documentation will be revised as well. Some limited defects are allowed to exist in this version but are targeted for correction in the future.

2) Beta Windows Version of MATT 2.0

According to feedback from the demonstration of Pre-Beta Windows version of MATT 2.0 and further research, the product will be revised and Beta Windows version of MATT 2.0 for MATLAB will then be released.

3) Pre-Beta Sun Solaris Version of MATT 2.0
New classes from Beta Windows version of MATT 2.0 used for communication with MATLAB will be imported to MATT 2.0 for the Sun Solaris platform. In this Pre-Beta version, MATT 2.0 will have the same user interface and same functionality as MATT 1.0. However, MATT 2.0 will be run in MATLAB environment instead of MatrixX environment. Specification and Documentation will be revised as well. A limited number of defects are allowed to remain in this version.

4) Beta Sun Solaris Version of MATT 2.0

Pre-Beta Sun Solaris version of MATT 2.0 will be revised and a Beta Sun Solaris Version of MATT 2.0 will have same user interface and functionality as Beta Windows version. Pre-Beta defects will be corrected.

5) Windows and Sun Solaris Release of MATT 2.0

After thorough testing and debugging, Windows and Sun Solaris Releases of MATT 2.0 will be created.

The following chapters will describe how the product was developed through requirements, design, implementation, testing and installation.
Chapter 3 Requirement for MATT 2.0

The requirement for the Pre-Beta Windows version of MATT 2.0 is the same as Windows version MATT 1.0, which was developed by the team at East Tennessee State University. This requirement must be maintained. The main requirement points are:

1) User can select a top system or subsystem to test.
2) User can specify test steps and time interval between two test steps.
3) User can specify a test type for each input variable. These test types include critical point testing, boundary value testing, linear value testing, random value testing and sinusoidal testing.
4) User can specify range and accuracy for each input variable.
5) User can specify exception and value limit for each output result. These exceptions include “outside limits”, “above maximum”, “below the minimum”, “inside the limits” and “no report’, which means “don’t care”.
6) User can create test cases and run simulation based on the test cases.
7) User can import and export test cases and simulation results.
8) User can graph test cases and simulation results.

After review by a set of users, the Pre-Beta Windows version of MATT 2.0, required the addition of several new requirements. These requirements for Beta Windows version of MATT 2.0 include:
1) Five dialog boxes - “Simulation Start Time”, “Simulation Stop Time”, “Total Simulation Time”, “Test Step Size” and “Total Test Steps” should replace the two original boxes named “Test Steps” and “Time Interval”.

2) The unit for “Simulation Start Time”, “Simulation Stop Time” and “Total Simulation Time” should change from millisecond to second.

3) Test cases and simulation result matrices should be able to be imported/exported from/to MATLAB data files.

The requirements of Pre-Beta Sun Solaris version of MATT 2.0 are almost the same as Pre-Beta Windows version of MATT 2.0 except that Sun Solaris version does not require the functionality named Open Excel (the ability to open an Excel spreadsheet populated with input and output data).
Chapter 4 Design for MATT 2.0

The team in East Tennessee State University finished the abstract and physical design of MATT 1.0 with Rational Rose. In their design, only one class (implemented as one header file and one source file) handles the connection to MatrixX. This class uses MatrixX functions to read model information, run simulation and graph input and output values. The structure of the design is shown in Figure 4-1. From this figure it is obvious that the design of MATT 1.0 makes it easy to port MATT from MatrixX to MATLAB. By replacing the old class that handles the connection to MatrixX of MATT 1.0 with a new class that handles the connection to MATLAB, the new functionality of MATT 2.0 is fulfilled (Figure 4-2).

![Diagram](image)

Figure 4-1 Scheme of the Design of MATT 1.0
Moreover, the design makes it simple to port MATT to different operating systems because the operating system independent classes can be replaced when porting to a different platform. Simply put, the high quality initial design made our work easier.

Figure 4-2 Scheme of the Design of MATT 2.0
Chapter 5 Implementation of MATT 2.0

This chapter contains a description of how the code for MATT 1.0 was revised according to the new requirements and design changes for MATT 2.0.

5.1 Programming Guide

During the implementation phase, we followed the Programming Guild of MATT 1.0 that was created by the team in East Tennessee State University so that the project design remains consistent and the code remains readable and maintainable. The main rules are[10]:

1) General Principles - Consistency and Readability

Developers should maintain consistency across the project base that makes code readable and modification easy. Readability means the code produced should be fairly self-documenting. Modifiability means the code is easy to change and extend.

2) File naming

File naming is the responsibility of the developer. File names should be descriptive and file extensions clearly indicating the type of file. All the platforms used in the project support long file names and are not restricted to the format of DOS. Enough information should be provided in file name so that the general function of the file can be recognized.
The following guidelines for file name should be followed:

- Avoid Use of Spaces.
- Avoid Symbols and Special Characters.
- The leading characters for MATT source files will be “mt_” followed by a descriptive name.
- All files should be in lowercase letters because UNIX platform is case sensitive. For example, Matt.cpp, matt.cpp and MATT.CPP are all considered different files under UNIX.

3) Variable Declaration

As a general rule, variable identifiers should be nouns to describe the data that they refer to in memory. Function names should generally consist of a verb and noun combination such as LoadMatrix

Variable Modifiers:

Variables should be prefixed to indicate the type of data represented by the object. Table 5-1, 5-2 and 5-3 list data type modifiers, scope identifier modifiers and control type modifiers.
### Table 5-1 Data Type Modifiers

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Modifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>a</td>
</tr>
<tr>
<td>Boolean</td>
<td>b</td>
</tr>
<tr>
<td>Double</td>
<td>d</td>
</tr>
<tr>
<td>Integer</td>
<td>i</td>
</tr>
<tr>
<td>Long</td>
<td>l</td>
</tr>
<tr>
<td>Pointer</td>
<td>p</td>
</tr>
<tr>
<td>String</td>
<td>s</td>
</tr>
<tr>
<td>Unsigned</td>
<td>u</td>
</tr>
<tr>
<td>Zero Terminated</td>
<td>z</td>
</tr>
<tr>
<td>Zinc</td>
<td>zaf</td>
</tr>
</tbody>
</table>

### Table 5-2 Scope Identifier Modifies

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Modifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class member</td>
<td>m</td>
</tr>
<tr>
<td>Static</td>
<td>s</td>
</tr>
<tr>
<td>Global</td>
<td>g</td>
</tr>
<tr>
<td>Local</td>
<td>l</td>
</tr>
</tbody>
</table>
### Table 5-3 Control Type Modifiers

<table>
<thead>
<tr>
<th>Tag</th>
<th>Control Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>btn</td>
<td>Button</td>
</tr>
<tr>
<td>cbo</td>
<td>Combo Box</td>
</tr>
<tr>
<td>chk</td>
<td>Check Box</td>
</tr>
<tr>
<td>cmd</td>
<td>Command Button</td>
</tr>
<tr>
<td>dir</td>
<td>Directory List Box</td>
</tr>
<tr>
<td>grp</td>
<td>Group (panel for grouping related interface objects)</td>
</tr>
<tr>
<td>img</td>
<td>Image</td>
</tr>
<tr>
<td>nbk</td>
<td>Notebook (tabbed dialogue)</td>
</tr>
<tr>
<td>opt</td>
<td>Option Button</td>
</tr>
<tr>
<td>pge</td>
<td>Notebook Page (tab)</td>
</tr>
<tr>
<td>pmt</td>
<td>Prompt</td>
</tr>
<tr>
<td>str</td>
<td>Input field</td>
</tr>
<tr>
<td>txt</td>
<td>Text Box</td>
</tr>
<tr>
<td>win</td>
<td>Window</td>
</tr>
</tbody>
</table>

**Class Objects:**

Prefix each class object with “mt_” and an uppercase C. For example, mt_Csuperblock.

Member attributes should be prefixed with “m_”, too.
4) Source Code Comments

Project source code should be well commented. The goal is not to write a complete novel, but to comment the code in order to provide a basis for understanding the code.

Comments in the file header will include the following:

- Author
- Description
- Modification History - most configuration management tools can insert this information

Comments in the function header will include the following:

- Author
- Description
- Inputs
- Outputs
- Return Type
- Caller Function Name
- Calling Function Name

For this project, Beta Version of both Windows and Sun Solaris of MATT 2.0 are released. So I will address them separately.
5.2 Windows Version

As mentioned before, the main requirement of MATT 2.0 is to use MATLAB to replace MatrixX, with same user interface and functionalities intact. The following steps were carried out.

1) Choose programming platform and language

MATT 1.0 for Windows Version was developed in Visual C++ 6.0 environment. Visual C++ is a product of Microsoft and has a powerful ability to create Windows-based user interface application software. Software created in Visual C++ can easily be ported to many versions of the Windows operating system. Visual C++ is a good Integrated Development Environment because developer can write, compile and link code with it easily. Moreover, Visual C++ has strong support for help functionality that is needed by every commercial software product. For these reasons, Visual C++ was used as the primary development environment.

2) Recompile the code of MATT 1.0

Todd Konesky et al recompiled the code of MATT 1.0 and ran MATT 1.0 in the environment of MatrixX.

3) Implement the function to read MATLAB model’s information and system or subsystem’s input variables information
Todd Konesky et al rewrote the class named `mt_CXmathLink` of MATT 1.0 which connected MATT to MatrixX. The name of the class name was changed to `mt_CmatlabLink`. First they implemented the functions needed by MATT 2.0 to read MATLAB model’s information, such as the name of the model, the number of subsystems in the model, the name of each subsystem, the model creation date, and how many input and output variables in the model or a subsystem. Then they implemented the function to read the input variables’ information. For any given model and all its subsystems, this function can read the detailed information about the inputs. The information includes name, data type and value range of every input variable.

4) Implement the function to read MATLAB system or subsystem’s output variable information

When we implemented the function to read MATLAB system or subsystem’s output variable information, a problem arose. We could NOT utilize the same method, by which MATT read the system or subsystem’s input variables’ data type, to read output variables’ data type. After thoroughly reviewing the MATLAB documentation, the development team contacted the Help Engineers in the Math Works and found the model must be compiled first. This solved the problem quickly.

5) Implement the function to run simulation

Running a simulation is one of the most important functions in MATT. This function does the following three steps. First, the external input array, which was created by
MATT according to the user’s specification, was sent to MATLAB. Then, the model is simulated using the input array as input values. Finally, the simulation produces output that is captured in the output array for every output variable.

In order to implement this function, we needed to call some functions in the MATLAB engine library. The MATLAB engine library provides a set of functions to start and end the simulation process, send data to and from MATLAB, and send commands processed in MATLAB. Programs using MATLAB engine functions are written in either C or Fortran, to which our C++ code is compatible. So through this library, MATT 2.0 can communicate with MATLAB conveniently and employ MATLAB to run a simulation successfully by sending a set of commands to MATLAB engine.

MATLAB doesn’t run subsystem simulation directly while MatrixX does. In order to run simulation on subsystem, a new class named mt_CsysWrapper was created by Josh Stiff. This class can copy a subsystem to a blank new top-level system and then the new top system will contain all information of the subsystem. So running simulation on the new top-level system is equivalent to running simulation on the subsystem.

6) Implement the function to graph

Graphing is a function in MATT. Through graphing the user can plot and analyze their data conveniently.
In this function, a user can easily select one or more input and/or output variable vector(s) to be displayed in a two-dimension graph. The user can specify the limits for both X-axis and Y-axis. User can plot data with a logarithmic scale for X-axis, Y-axis, or both. Moreover, user can edit graph labels, such as graph title and axes labels. The grid can also be turned on or off.

MATT 1.0 implemented graph function by calling MatrixX API, which is obviously MatrixX dependently. MATT 2.0 will utilize MATLAB graph commands. MATLAB has extensive plotting capabilities and user can plot data by typing a set of commands in MATLAB command window. For MATT 2.0 users, they can select plotting data and graph option in a dialog box. MATT 2.0 will transfer these options using a set of MATLAB commands. Then MATLAB will plot the data automatically.

7) Change MATT from debug build version to release (non-debug) build version

In a Visual C++ project, both a Win32 Debug build and a Win32 Release build version are automatically created. User can select either option to build product. Table 5-4 contains general information about both of them.

By default, a release build uses optimizations. When optimizations are used to create a release build, the compiler will not produce symbolic debugging information. The absence of symbolic debugging information, along with the fact that code is not
generated for TRACE and ASSERT calls, means that the size of the executable file is reduced and will therefore be faster\textsuperscript{111}.

Table 5-4 Win32 Debug Build versus Release Build\textsuperscript{111}

<table>
<thead>
<tr>
<th>Version</th>
<th>Default Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win32 Debug</td>
<td>Full symbolic debugging information in Microsoft format</td>
</tr>
<tr>
<td></td>
<td>No optimization (optimization generally makes debugging more difficult)</td>
</tr>
<tr>
<td>Win32 Release</td>
<td>No symbolic debugging information</td>
</tr>
<tr>
<td></td>
<td>Optimized for maximum speed</td>
</tr>
</tbody>
</table>

In general, a developer will first build and test his project with a debug build. After testing and debugging, developer will switch to release build.

The following paragraphs describe the primary differences between a debug and a release build. There are other differences, but these are the primary differences that will cause an application to fail in a release build when it works in a debug build\textsuperscript{111}.

Heap Layout: Heap layout will be the cause of about ninety percent of the apparent problems when an application works in debug, but not in release build.

When developers build a project for debug, they are using the debug memory allocator. This means that all memory allocations have guard bytes placed around them. These
guard bytes are placed there to detect a memory overwrite. Because heap layout is different between release and debug build, a memory overwrite might not create any problems in a debug build, but may have catastrophic effects in a release build.

Compilation: Many of the Microsoft Foundation Classes (MFC) macros and much of the MFC implementation change when you build for release.

Pointer Support: The lack of debugging information removes the padding from the application. In a release build, stray pointers have a greater chance of pointing to uninitialized memory instead of pointing to debug information.

Optimizations: Depending on the nature of certain segments of code, the optimizing compiler might generate different executable code than that tested in debug mode. This is the least likely cause of release build problems, but it does arise on occasion.

At first, MATT 2.0 was built in debug option. Before releasing Pre-Beta version, we decided to switch it from debug build to release build. After configuring the settings for release build, the code was recompiled and linked successfully. However, when MATT 2.0 was tested, some memory problems arose. We followed the instruction from Help File and figured out that they were Heap Layout problems. Finally, a release build version was created successfully. The size of executable file for debug build was 1,500 kilo bytes, but is only 292 kilo bytes in release build.

8) After Pre-Beta Windows Version MATT 2.0 Demonstration
As mentioned previously, after Pre-Beta Windows version MATT 2.0 demonstration, users changed some requirements:

1) Five dialog boxes - “Simulation Start Time”, “Simulation Stop Time”, “Total Simulation Time”, “Test Step Size” and “Total Test Steps” should replace the original two boxes named “Test Steps” and “Time Interval”.

2) The unit for “Simulation Start Time”, “Simulation Stop Time” and “Total Simulation Time” should change from millisecond to second.

3) Test cases and simulation result matrices should be able to be imported/exported from/to MATLAB data files.

According to these requirements, the interface of MATT was changed and new code was created and tested.

9) Summary of changed files

Like other large projects, if one file was changed, the related files should be changed correspondingly; otherwise, some incompatibility problems can arise. Table 5-5 is the list of files that were revised in this project. Twenty-seven out of fifty nine files were changed. The number of lines of the code revised or created is difficult to count. Initial estimates of changed and new code total approximately 3,000 lines.
Table 5-5 Files revised in Windows Version of MATT 2.0

<table>
<thead>
<tr>
<th>Header File</th>
<th>Source File</th>
<th>Resource File</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matt.h</td>
<td>Matt.cpp</td>
<td>MATT.ico</td>
<td>Windows</td>
</tr>
<tr>
<td></td>
<td>mt_cgraphdialog.cpp</td>
<td>MATT.rc</td>
<td>Windows</td>
</tr>
<tr>
<td>mt_cinputdialog.h</td>
<td>mt_cinputdialog.cpp</td>
<td></td>
<td>Windows</td>
</tr>
<tr>
<td>mt_cinputmatrix.h</td>
<td>mt_cinputmatrix.cpp</td>
<td></td>
<td>Windows</td>
</tr>
<tr>
<td>mt_cmaindialog.h</td>
<td>mt_cmaindialog.cpp</td>
<td></td>
<td>Windows</td>
</tr>
<tr>
<td>mt_cmatlablink.h</td>
<td>mt_cmatlablink.cpp</td>
<td></td>
<td>Independent</td>
</tr>
<tr>
<td></td>
<td>mt_cmatrix.cpp</td>
<td></td>
<td>Independent</td>
</tr>
<tr>
<td></td>
<td>mt_coutputdialog.cpp</td>
<td></td>
<td>Independent</td>
</tr>
<tr>
<td></td>
<td>mt_cresultsdialog.cpp</td>
<td></td>
<td>Independent</td>
</tr>
<tr>
<td>mt_csuperblock.h</td>
<td>mt_csuperblock.cpp</td>
<td></td>
<td>Independent</td>
</tr>
<tr>
<td>mt_csystwrapper.h*</td>
<td>mt_csystwrapper.cpp*</td>
<td></td>
<td>Independent</td>
</tr>
<tr>
<td>mt_ctestcase.h</td>
<td>mt_ctestcase.cpp</td>
<td></td>
<td>Independent</td>
</tr>
<tr>
<td>mt_ctestinput.h</td>
<td>mt_ctestinput.cpp</td>
<td></td>
<td>Independent</td>
</tr>
<tr>
<td></td>
<td>mt_ctestoutput.cpp</td>
<td></td>
<td>Independent</td>
</tr>
<tr>
<td>mt_ctestscript.h</td>
<td>mt_ctestscript.cpp</td>
<td></td>
<td>Independent</td>
</tr>
</tbody>
</table>

Note: * Josh Stiff created
5.3 Sun Solaris Version

MATLAB has both Windows and Unix versions. So MATT should have both as well. After the Beta Windows version release, we focused on developing Sun Solaris version of MATT 2.0.

MATT 1.0 Sun Solaris version was developed with C++ programming language. The graphic user interface (GUI) was supported by QT library. The team in East Tennessee State University didn’t use any IDE to develop the product. Instead, they utilized vi to write hard code to set up a GUI, and Makefile to compile source files and link object files.

The following recorded the steps to develop MATT 2.0.

1) Choose IDE

As described above, the team developing MATT 1.0 didn’t use any IDE. However, IDE is very convenient to developers. We planned to select KDevelop as the IDE to develop MATT 2.0.

There are numerous advantages of KDevelop:

1) KDevelop provides developer an easy way to create GUI interfaces with the built-in dialog editor. Developer can let KDevelop generate the dialog source code and get full control of the dialog functionality.
2) Full-featured editor with syntax highlighting (KWrite).

3) It is free.

Though KDevelop has lots of good features, we failed to set it up in our machine after several trials of installation. We had to use text editor and Makefile to implement our project.

2) Download the Sun Solaris code of MATT 1.0

The first step is to download the source code from the web site for MATT 1.0 to our Sun Solaris machine.

3) Recompile the source code and link the object files

Following the installation documentation, the environment variables for MATT 1.0 was set up. Then we recompiled the source code and linked the object files successfully. MATT 1.0 couldn’t be run on our machine because there are some license problems in MatrixX. Until recently it was not possible to run Sun Solaris version of MATT 1.0 for MatrixX, which prevented comparison of MATT 2.0 to MATT 1.0.

4) Import new class, MT_CMatlabLink, to the project

In order to develop MATT 2.0, we should replace the old class named MT_CxmathLink with new class, MT_CMatlabLink. The MT_CxmathLink class communicates with
MatrixX, while MT_CMatlabLink class, which was created in Windows Version of MATT 2.0, communicates with MATLAB.

We need to rewrite the mattlnx.cpp, as well. This file initializes MATT 2.0.

Windows and Sun Solaris have different compilers: Microsoft Visual C/C++ Version 6.0 and g++ compiler, which are quite different. Even though MT_CMatlabLink class was developed as platform independent class, the code in MT_CMatlabLink class, developed in Visual C++, is not completely compatible in g++. So after compiling the code, we needed to make some changes to the code due to compiler issues.

5) Revise Makefile

The file name of class MT_CMatlabLink is different from the file name of class MT_CxmathLink. In order to compile MT_CMatlabLink class, Makefile needs to be revised.

Makefile in this project has a hierarchic structure. The scheme of this hierarchy is shown in Figure 5-1. The Makefile system is complicated and not easy to understand. However, after analysis and testing the team was able to revise the Makefile.q.in in Unix/src subdirectory.
6) Reset environment variables for MATT 2.0

In order to call the library of MATLAB, we need to set environment variables for the system. These environment variables include MATT installation directory, compiler, MATLAB installation directory, MATLAB architecture and QT installation directory. Appendix D and E will give detailed information about the settings.

7) Compile the code and fix problems

After revising Makefile and resetting environment variables, MATT 2.0 was recompiled and linked to form an executable. As anticipated, some defects arose in MT_CMatlabLink class because of the difference between the two compilers. The defects were corrected one by one.
Sun Solaris version of MATT 2.0 now has the same functionality to the Pre-Beta Windows version.

8) Change the user interface and make it the same as Windows version

According to the new requirement, the user interface was changed and new functions were added to Sun Solaris version (All changed common files were ported to the Sun Solaris version from the Windows version).

9) Import the new class MT_CsysWrapper to the project

Until now, MATT can only create test cases and run a simulation in a top-level system. In order to work on a subsystem, we should import class MT_CsysWrapper too. After revising the Makefile and fixing some errors caused by importing the new class, the Beta Sun Solaris version of MATT 2.0 was created.

10) Summary of changed files

Table 5-6 is a list of changed files. The number of changed files is 14 and the total number of files in this project is 55.
## Table 5-6 Files revised in Sun Solaris Version of MATT 2.0

<table>
<thead>
<tr>
<th>Header File</th>
<th>Source File</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>mt_csuperblock.h</td>
<td>mt_csuperblock.cpp</td>
<td>Independent</td>
</tr>
<tr>
<td>mt_cmatlablink.h</td>
<td>mt_cmatlablink.cpp</td>
<td>Independent</td>
</tr>
<tr>
<td>mt_csystwrapper.h *</td>
<td>mt_csystwrapper.cpp *</td>
<td>Independent</td>
</tr>
<tr>
<td>mattlnx.h</td>
<td>mattlnx.cpp</td>
<td>Unix</td>
</tr>
<tr>
<td>mt_cwinmatt.h</td>
<td>mt_cwinmatt.cpp</td>
<td>Unix</td>
</tr>
<tr>
<td>mt_cmatttabs.h</td>
<td>mt_cmatttabs.cpp</td>
<td>Unix</td>
</tr>
<tr>
<td></td>
<td>mt_clistview.cpp</td>
<td>Unix</td>
</tr>
<tr>
<td></td>
<td>mt_cgraphdialog.cpp</td>
<td>Unix</td>
</tr>
</tbody>
</table>

Note: * Josh Stiff created
Chapter 6 Testing and Debugging of MATT 2.0

Testing and debugging are critical activities in software quality improvement. Only after extensive testing and debugging, can high quality and reliability be achieved. The recommended distribution of effort is 40%, 20% and 40% for design, coding and testing respectively\[13\].

The team in East Tennessee State University created a test plan document for MATT 1.0\[14\]. During development of MATT 2.0, we followed the testing plan of MATT 1.0. The purpose of the testing plan is to discover and correct defects in MATT as efficiently as possible. The main points are\[14\]:

6.1 Unit Testing

Unit testing examines the lowest level software routine to ensure the source code is as error free as possible and that these routines provide functionality as defined in the design specification. Without doubt, it is very important that every developer assume responsible for testing their routines and make sure that design requirements are met. Developers may use the debugger or they may print their source code in order to follow the logic and make sure that the code is performing correctly.
After unit testing, many logic errors will be discovered and corrected. The earlier the errors are corrected, the lower the cost to repair errors.

1) Isolation or Module Testing
Isolation testing means that each class or function should be tested in isolation so that its' capabilities can be verified as a standalone unit. Each module’s functionality should be checked against design specifications. If there are errors, the developer should discover and correct them in isolation before integrating the unit into the system. Each MATT developer will be responsible for testing any class, program, or form that they created.

2) Specification Testing
Specification testing, also called functional testing or “black box” testing, verifies that the system provides functionality as documented in the software specification. “Black box” testing focuses on the functionality of each module (what the user expects) but not with the logic inside.

6.2 Integration Testing
After unit testing, integration testing is next. Integration testing involves making sure that the individual modules perform correctly when executed with other modules in the system. All developers whose code must work together should perform this testing. Integration testing can always find some new errors that weren’t detected in unit testing.
Developers also need to re-test their own individual units after correcting defects found during integration testing.

1) Inheritance Hierarchy

Classes may be tested in isolation but inheritance tests are needed to verify that child objects have been instantiated with the correct attributes and capabilities as their parent class.

2) Regression Testing

Regression Testing involves conducting some or all of the tests that were performed previously as each new module is added and tested. Regression testing ensures that no new defects are introduced to the system as the software is updated.

6.3 System Testing

System testing is performed to verify the end-to-end functionality of a system. It verifies that the requirements have been implemented and that they have been implemented to an acceptable quality level\textsuperscript{[15]}.

During System Testing, all hardware and software components of MATT will be tested to ensure that the system satisfies all requirements and meets specifications. Not only software, but also documentation will be reviewed. Other issues that will be addressed during System Testing include validation of security and reliability of the system.
1) Build Testing

Build testing was performed by the compiler and linker to verify the syntactic correctness of the system, but not the logic. The developer in charge of build testing should have an overall knowledge of MATT and its components.

2) Volume and Stress Testing

Volume testing will be performed on MATT system to ensure that it can manipulate large amount of data. This testing will be especially critical when reviewing MATT functions that deal with large matrices. Similarly, stress testing will also be performed to validate system boundaries and unexpected inputs from the user.

3) Performance Testing

MATT will be tested at various times throughout the software development procedure to verify that forms, reports, simulations and so forth can be run in a reasonable amount of time. Modules that are often reused in the system may be analyzed to see if they could be recoded to execute faster.

6.4 Software Problem Reporting and Tracking

Software problems are documented as defects in the product. It is very important to document and report all defects. Those responsible for QA (Quality Assurance) track the defects and their corresponding corrections.
Defects include instances when the software actually crashes or displays system error messages. Another serious problem, which is often hard to discover, is the generation of erroneous data. On the other hand, defects can also include functionality that does not meet the specified requirements.

As defects are discovered during testing, each tester should record defect information, including:

- Defect number
- Name of person reporting problem
- Functional area (or Subsystem)
- Date reported
- Description of problem
- Problem severity (1 – Fatal Error, 2 – Does not match specification, 3 – Annoying behavior, 4 – Cosmetic)
- Steps taken to cause problem
- Priority of problem fix/change (May different from Problem Severity)
- Resolution type (Fixed, Irreproducible, As Designed, Need More Info, Disagree with Suggestion, Cannot be fixed)
- Date corrected
- Effort taken to correct
- Person who corrected the defect
Other notes

During group meetings, testers provide defect information to all team members so the project team can judge the status of the project and product quality.
Chapter 7 Installation and Deployment of MATT 2.0

Installation and deployment is a critical step for our product. After this step, users can utilize our software to support testing and debugging of real-time system models. A professional installation is absolutely necessary for the full utilization of MATT by users. Because the installation of Windows and Sun Solaris versions is very different, they will be addressed separately.

7.1 Windows Version

The Windows installation package was created using InstallShield for Microsoft Visual C++ 6 (Build -130).

InstallShield for Microsoft Visual C++ 6 is a software installation development product using world-renowned InstallShield technology in an easy-to-use Installation Development Environment (IDE). InstallShield for Microsoft Visual C++ 6 gives users the features and functionality of author, lay out, build and test of installation setups with point-and-click ease[16].

Moreover, InstallShield for Microsoft Visual C++ 6 includes InstallShield Visual C++ Wizard. With this wizard developers can quickly create a setup file that installs the application, properly handles file dependencies and has uninstall functionality.
InstallShield for Microsoft Visual C++ 6 will continue to be used for MATT installation development.

The specification for the Windows install package for MATT 2.0 is revised from the installation documentation that was created by the team in East Tennessee State University[17].

Setting the Visual Design

Application Information

Application Name: MATT
Application Executable: “path”MATT.exe
Company: The University of Montana
Default Destination Directory: <ProgramFilesDir>\MATT

Main Window

Main Title: Text – “MATLAB Automated Testing Tool”
Logo Bitmap: Created by Josh Stiff

Features

Automatic Uninstaller: Yes
Provide SMS Support: No
Target Platform: 32-Bit
Project Language: English

Specifying Components and Files

Groups and Files

Readme Files

Destination Directory: <INSTALLDIR>\Readme

File to install

readme.txt

Program Files

Destination Directory: <INSTALLDIR>

Files to install

MATT.dll
MATT.exe
min_sample_time.m
matt_getsim_time.m
matt_load.m
matt_load.m
matt_open_html.m
matt_save.m
matt_set_sim_time.m
MATTInputMatrixTool\MATTInputMatrix.exe
MATTInputMatrixTool\Pegrp32q.dll
Sprd\AutoMatt.exe

Help Files

Destination Directory: <INSTALLDIR>\Help

Files to install

*.html
*.gif

Example Files

Destination Directory: <INSTALLDIR>\Models

Files to install

fl4.mdl

License Files

Destination Directory: <INSTALLDIR>\License

Files to install

license.txt

Selecting User Interface Components

Welcome Bitmap: Yes
Welcome Message: Yes
Software License Agreement: Yes
Readme Information: Yes
User Information: No
Choose Destination Directory: Yes (User may choose where to install MATT)
Setup Type: No
Custom Setup: No
Select Program Folder: Yes
Start Copying Files: Yes
Progress Indicator: Yes
Billboards: No
Setup Complete: Yes

7.2 Sun Solaris Version

The installation of Sun Solaris version is very different from Windows version. Binary code cannot be delivered to users directly like the Windows version. In the Sun Solaris version, source code must be distributed to users along with operating system scripts that compile source code and link the object files to create an executable program. Then users can launch MATT 2.0 from MATLAB.

The main steps to install Sun Solaris Version of MATT 2.0 are the following:

1. Get Source File
The source file of MATT 2.0 is named “MATT2.tar.gz”. The user will have to place this file in the directory where they want to install MATT.

2. Uncompress Source File

Under the prompt of MATT installation directory, the user types “gunzip MATT2.tar.gz” and will get a file named “MATT2.tar”. Typing “tar –xvf MATT2.tar”, a subdirectory named Matlab will appear, which contains source files.

3. Set Environment Variables

The environment variables need to be set are MATT’s installation directory, compiler, MATLAB’s installation directory, MATLAB’s architecture, and QT’s installation directory. Appendix D and E provides detailed settings for these.

4. Configure MATT

Under the prompt of MATT installation directory, user types:

- “cd Matlab”
- “./build _matt.sh –configureonly –installdir $MATTDir”

These commands will configure the system in order to build MATT.

5. Build MATT

Type “make” and this command will compile source files and link object files.
6. Install MATT

Type "make install" and this command will create subdirectories – bin, cases, doc, help, matrices, model and script under the MATT installation directory and put relative files into these subdirectories.
Chapter 8 Summary and Future Work

Summary

In this project, we developed MATT 2.0 for both Windows and Sun Solaris. Beta version for Windows is released and Beta version for Sun Solaris will be released in the next six weeks. Compared to MATT 1.0, MATT 2.0 uses MATLAB rather than MatrixX to test real-time system models.

MATT 2.0 is an automated testing tool for real-time system models. Using simulation, it can generate and run tests on real-time system models specified in MATLAB. The automated nature of MATT allows tests containing millions of test values to be executed and defects to be automatically detected[6].

A software engineering approach was applied to this project. It includes requirement analysis, abstract and physical design, implementation and testing. Finally, commercial quality software products were created.

Future Work

In order to get a higher quality product, more effort should be put into the following fields.
1) Do more testing and debugging on both Windows Version and Sun Solaris Version of MATT 2.0

Additional MATLAB models are needed for testing MATT 2.0 that contain more features, such as trigger, enable, state-flow and so on. Based on these models, more testing of MATT 2.0 for both Windows and Sun Solaris version can be performed. After more testing and debugging, MATT 2.0 can be released.

2) Import Sun Solaris source code of MATT to an Integration Development Environment (IDE)

As mentioned before, Sun Solaris version of MATT 2.0 wasn’t developed in an IDE. If we import the source code to an IDE, developers can have more control over the graphical interface design and can compile and build product more conveniently. System administration support is needed to install KDevelop.

3) Some common files for both Windows version and Sun Solaris version need to be put back into Window version

We developed Windows version of MATT 2.0 first, and then developed Sun Solaris version. Some common files, which revised during the process of developing Windows version, were imported to Sun Solaris version. These files can be found in Table 5-5. During the process of developing Sun Solaris version, some common files, including mt_cmatlablink.h, mt_cmatlablink.cpp, mt_csyswrapper.h and mt_csyswrapper.cpp, were
revised. They need to be put back to Windows version and to be recompiled. We should keep them consistent in both Windows version and Sun Solaris version.

4) Implement the functionality that MATT can test source code generated by MATLAB automatically

MATLAB can generate source code in C or Ada language for simulation models. Using integrated Makefile based targeting support, it builds programs that can help speed up simulation or run on production target\sup{118}. This would provide MATT with the ability to test the automatically generated source code. There are some difficulties to implement this feature. For example, how to integrate the external input matrix to source code? More research is needed on this subject.
Appendix A

Proposal of Porting MATT from Matrix to MATLAB

Jing Tao

Computer Science Department

The University of Montana

12/16/2001

Background

- Describe MATT

MATT (MatrixX/MATLAB Automated Test Tool) is a tool used to test real-time software via simulation.

MATT attacks the challenging problems involved in testing real-time systems. Testing real-time systems involves hardware and software and takes risk of damage to hardware. In aeronautic and aerospace applications, the result of this damage is costly and can be dangerous. In order to overcome these difficulties, large amount of effective testing needs to be completed in advance to the real-time application of software to hardware.

Engineers often employ simulation model prior to, or in conjunction with, hardware-software integration. However, at the same time, other problems arise when simulation is utilized, such as generation and use of potentially huge data sets, time-consuming and
error-prone output analysis, and testing time constraints. For example, a real-time system samples 100 input values at 100 millisecond update times, then a simulation must supply 1,000 input values per second. A modest one-hour simulation would require 3,600,000 values. If these values are floating point numbers, a 14.4-megabyte file is required for this simulation. In addition, if the system produces 100 output values every 100 milliseconds; the output file would contain another 14.4 megabytes of data. Creating the input file is overwhelming to a project team without the support of automated tools.

Further, many real-time systems include the requirement to execute for months or years without interruption. Testing of this length of time is not feasible within most projects.

Without question, testing real-time systems presents a number of complex and time-consuming problems. Automated testing tools need to be developed to improve software quality, increase testing productivity, and enhance management insight into process and product risk. MATT is this kind of tool.

-Current status of MATT

Under the instruction of Dr. Henry, a team in East Tennessee State University developed MATT 1.0 for MatrixX employed on both Windows and Sun Solaris platforms in 2000. The product is very reliable and validated on NASA wind tunnel systems.

According to the new requirement of NASA, the cooperate software of MATT will be changed from MatrixX to MATLAB. A team in the University of Montana is working on
this assignment. Both Beta Version for Windows and Sun Solaris Platforms of MATT 2.0 for MATLAB will be released.

Goals

- Port MATT from MatrixX to MATLAB in Windows operating system

MATT 2.0 will run in the Windows operating system. The cooperate software will be changed from MatrixX to MATLAB. And MATT 2.0 will keep main functions as same as MATT 1.0.

- Port MATT for MATLAB to Sun Solaris operating system

In academic and scientific domain, lots of scientists or engineers are working with Unix operating system. It is important to port MATT to Sun Solaris. Moreover, MATLAB has both Windows and Unix version, too.

- Same functionality on both Windows and Sun Solaris

The functionality for both versions should be the same. Both of them can create test cases in different test type, specify test number and test step, run simulation, export or import testing data and have help function.

Approach

A software engineering approach would be used in the development. It includes requirement analysis, design, implementation, and testing.
1. Requirement

The requirement for MATT 2.0 is almost the same as MATT 1.0. It will be changed according to the feedback from NASA engineers after the demonstration of Pre-Beta version.

2. Design

The design for MATT 2.0 is almost as same as MATT 1.0. It will be revised after changing requirement.

3. Implementation

MATT 2.0 for Windows version will be developed with Microsoft Visual C++. MATT 2.0 for Sun Solaris version will be developed with text editor and Makefile.

The order is to develop Windows version first, and then Sun Solaris version.

4. Testing

The testing plan for MATT 2.0 would follow the plan for MATT 1.0. Unit testing, integration testing, and system testing will be performed for MATT 2.0.

Challenges and Solutions

1. The connection between MATT and MATLAB
MATT will send data and commands to MATLAB. Some commands, such as reading model’s information, importing data, and running simulation in MATLAB, should be figured out. The method to get information includes reading MATLAB User Manual, reading MathWorks on-line help files, contacting help engineers.

2. The compatibility of Windows and Sun Solaris

Windows version is developed by IDE – Microsoft Visual C++, while Unix version is developed by text editor and compiled by g++ compiler. Some code, especially for pointer and array, can be compiled successfully in Visual C++ but probably can cause an error by g++ compiler because the systems have different standard. So during the development, we should use standard C++ language approved by the American National Standards Institute (ANSI) for the common files that will be used for both Windows and Sun Solaris operating system. This method will reduce the code incompatibility on two different platforms.

Results

- Release Beta Windows Version of MATT 2.0
- Release Beta Sun Solaris Version of MATT 2.0
- Documentation
  1. Design diagram of the product
  2. Testing plan for the product
  3. User guild for system configuration and installation
4. Readme file
5. Revised help file
6. Personal effort and time data

[1] Reference:


# Appendix B

**MATT System Requirements**

<table>
<thead>
<tr>
<th>MATT Windows Release</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processor</strong></td>
<td>Pentium 90 MHZ - Minimum</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>32 MB - Minimum</td>
</tr>
<tr>
<td><strong>Disk Space</strong></td>
<td>10 MB</td>
</tr>
<tr>
<td><strong>Display</strong></td>
<td>Any display capable of 1024 X 768 resolution</td>
</tr>
<tr>
<td><strong>Browser</strong></td>
<td>Netscape Version 2.0 or Internet Explorer 3.2 - Minimum</td>
</tr>
<tr>
<td><strong>Operating Systems</strong></td>
<td>Windows 95, Windows 98, or Windows/NT</td>
</tr>
<tr>
<td><strong>Matrix-X</strong></td>
<td>6.0.3 or higher</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MATT Sun Solaris Release</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Disk Space</strong></td>
<td>250MB</td>
</tr>
<tr>
<td><strong>Display</strong></td>
<td>Any display capable of 1024 X 768 resolution</td>
</tr>
<tr>
<td><strong>Browser</strong></td>
<td>Netscape Version 2.x - Minimum</td>
</tr>
<tr>
<td><strong>Operating Systems</strong></td>
<td>Any version currently supported by Matrix-X</td>
</tr>
<tr>
<td><strong>Additional Libraries</strong></td>
<td>Qt Version 2.x</td>
</tr>
<tr>
<td><strong>Matrix-X</strong></td>
<td>6.0.3 or higher</td>
</tr>
</tbody>
</table>

56
Appendix C

How to Run MATT 2.0

1) Launch MATLAB 6.1

2) Set the MATT path (if not already done):
   A) Click 'File'
   B) Click 'Set Path...'
   C) Click 'Add Folder...'

   D) - Locate and select the MATT installation directory (Windows version)
      - Type the MATT installation directory (Sun Solaris version)

   E) Click 'OK'

   F) Click 'Save'

   G) Click 'Close'

3) Launch MATT by typing the following at the MATLAB command window:
   A) matt('<Simulink_Model_Name>');

   B) Press 'Enter'
Note:

1) The model's path should already be put into MATLAB Search Path. Otherwise, MATT cannot find the model.

2) A Simulink model must contain at least one Input and one Output variable to be properly loaded into MATT.
Appendix D

INSTALL file for MATLAB Automated Testing Tool (MATT) Beta Release V 2.0
for Sun Solaris
December 20, 2001
Copyright (C) 2000 The University of Montana

PREREQUISITES:

1. Qt must have already been built on the target machine. Qt is available from http://www.trolltech.com.

2. The QTDIR environment variable (and possibly others) must be set as described in the Qt INSTALL file.

TO BUILD AND INSTALL MATT:

1. Get Source File:

   Put source file of MATT 2.0 named "MATT2.tar.gz" into matt_install_directory
2. Uncompress Source File:

   gunzip MATT2.tar.gz

   tar -xvf MATT2.tar

3. Set Environment:

   I. Set Environment for Matt's installation directory:

      Bourne/Korn shell:

         MATTDIR=matt_install_directory

         export MATTDIR

      C shell:

         setenv MATTDIR matt_install_directory

   II. Set Environment for Compiler:

      Bourne/Korn shell:

         CC=c-compiler-executable-name

         export CC
CXX=c++-compiler-executable-name

export CXX

For Sun Solaris:

CC=CC
export CC
CXX=CC
export CXX

C shell:

setenv CC c-compiler-executable-name
setenv CXX c++-compiler-executable-name

For Sun Solaris:

setenv CC CC
setenv CXX CC

III. Set Environment for MATLAB
Boume/Korn Shell:

```
ISIHOME=matlab_install_directory
export ISIHOME
MATLABARCH=your_system_architecture
export MATLABARCH
```

C shell:

```
setenv ISIHOME matlab_install_directory
setenv MATLABARCH your_system_architecture
```

IV. Add MATLAB Library to Environment for LD_LIBRARY

Bourne/Korn Shell:

```
 LD_LIBRARY_PATH=$ISIHOME/extern/lib/{$MATLABARCH}:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

C shell:

```
setenv LD_LIBRARY_PATH

$ISIHOME/extern/lib/{$MATLABARCH}:$LD_LIBRARY_PATH
```

4. Configure MATT:
./build_matt.sh -configonly -installdir $MATTDIR

5. Build MATT:

   make

6. Install MATT:

   make install
Appendix E

A Real File of .cshrc

# @(#)cshrc 1.11 89/11/29 SMI

umask 077

#set path=(/bin /usr/local/bin /usr/ucb /etc )

#set environment variable for QT installation

setenv QTDIR /work/matt/qt

set path=( /usr/bin /usr/dt/bin /usr/local/bin /usr/local/ssh/bin /usr/ucb /usr/ccs/bin /etc $QTDIR/bin )

setenv LD_LIBRARY_PATH /usr/lib:/usr/local/lib:/usr/openwin/lib:/usr/dt/lib

setenv LD_LIBRARY_PATH_64 /usr/lib/64:/usr/openwin/lib/64:/usr/dt/lib/64

setenv MANPATH


#set environment variable for MATT installation

setenv MATTDIR /work/matt

64
#setenv ISIHOME /usr/local/matrixx

setenv QTDIR /work/matt/qt

setenv MATTDIR /work/matt

#setenv ISIHOME /usr/local/matlab

#set environment variable for MATLAB installation

setenv ISIHOME /opt/sfw/matlab

#set environment variable for compiler

setenv CC /usr/local/bin/gcc

setenv CXX /usr/local/bin/g++

#set environment variable for MATLAB architecture

setenv MATLABARCH sol2

if ( ! $?QTDIR ) then

    setenv QTDIR /work/matt/qt

endif

#if ( $?PATH ) then

    # setenv PATH $PATH:$QTDIR/bin

#else

    # setenv PATH $QTDIR/bin

65
if ( $?MANPATH ) then
    setenv MANPATH $QTDIR/man:$MANPATH
else
    setenv MANPATH $QTDIR/man
endif

if ( $?LD_LIBRARY_PATH ) then
    setenv LD_LIBRARY_PATH $QTDIR/lib:$LD_LIBRARY_PATH
else
    setenv LD_LIBRARY_PATH $QTDIR/lib
endif

#set MATLAB lib to LD_LIBRARY_PATH
setenv LD_LIBRARY_PATH

#if ( $?LD_LIBRARY_PATH ) then
    #setenv LD_LIBRARY_PATH
endif

$ISIHOME/extern/lib/{SMATLABARCH}:$LD_LIBRARY_PATH

66
#else

#setenv LD_LIBRARY_PATH
$QTDIR/lib:$ISIHOME/extern/lib/{$MATLABARCH}
#endif

if ( ! $?LIBRARY_PATH ) then
    setenv LIBRARY_PATH $LD_LIBRARY_PATH
endif

if ( $?CPLUS_INCLUDE_PATH ) then
    setenv CPLUS_INCLUDE_PATH $QTDIR/include:$CPLUS_INCLUDE_PATH
else
    setenv CPLUS_INCLUDE_PATH $QTDIR/include
endif

alias setprompt 'set prompt = "$cwd % "'
setprompt

alias cd 'cd \!*; setprompt'
alias pushd 'pushd !*; setprompt'

alias popd 'popd !*; setprompt'

r your machine. Note that
Appendix F

Reference

[10] Code Guild
[16] InstallShield for Microsoft Visual C++ 6 Help Documentation
[18] Real-Time workshop documentation