The following paper was originally published in the

Proceedings of the 3rd USENIX Windows NT Symposium

Seattle, Washington, USA, July 12–13, 1999

MTEX—A BRIDGE FOR MIGRATING CAD DESIGN ENVIRONMENT FROM UNIX TO NT

Ty Tang, Vipul Lal, and Shesha Krishnapura

MTEX - A Bridge For Migrating CAD Design Environment From UNIX To NT

Ty Tang, Vipul Lal, Shesha Krishnapura
ty.tang@intel.com, vipul.lal@intel.com, shesha.krishnapura@intel.com
Design Technology, Intel Corporation

Abstract

This paper shares an innovative technology that we developed while migrating Intel CAD design environment from UNIX to Microsoft Windows NT operating system. The UNIX to Windows NT migration is a complex and challenging task because the chip design environment involves CAD applications, CAD infrastructure scripts, design flows made up of UNIX-centric scripts and design data.

Due to the wide differences between UNIX and NT scripting architectures, straight porting to native NT scripting environment is not feasible without a complete redesign and redevelopment of the CAD infrastructure and design flow scripts. Our approach to this problem was to develop a transition production-capable mixed NT-UNIX CAD environment technology, MTEX, with the eventual goal of complete migration to a Windows NT CAD environment. This technology solves the script migration problem and supports a seamless mix of UNIX and NT centric CAD tools.

In this paper we will present MTEX functionality, its internal design and architecture.

1. Introduction

Intel's current CAD environment is based on high-end UNIX based RISC workstations. To harness the power of the cheaper Intel Architecture (referred to as IA from here on) platform and to have a single workstation that supports both engineering and office tools, Microsoft Windows NT (referred to as NT from here on) was chosen as the operating system of choice. The UNIX to NT migration tasks were broken down into the following three activities:

- Porting of CAD tools
- Porting of the associated runtime environment
- Porting of the test environment

Most of the CAD tools are written in high-level language such as C or C++, whereas the runtime and the test environment were mostly UNIX centric C-shell and Perl scripts.

Initially we estimated, rather incorrectly, that most of our effort would be used in porting the design tools to native Win32 APIs. As we progressed further, it became evident that the porting of C and C++ tools to Win32 APIs does not constitute a big task.

Porting of runtime and test environment to Windows NT became a bigger challenge due to the incompatibility of the scripting environment between NT and UNIX. The current commercially and publicly available UNIX utilities on NT were not mature enough to support a production-worthy UNIX-like scripting engine in multi-user mode that provides the identical UNIX functionality. Other possibilities were evaluated that included going in for one of the native Win32 scripting environment, DOS batch files, which was very limited in functionality or Visual Basic which will require complete re-architect of Intel CAD runtime and test environment.

We finally realized that complete migration in one-step to IA-NT as a CAD platform was not possible and a creative but simple technique was needed to meet our aggressive migration goal. This paper describes the problems in detail and our innovative technology to over come to the migration problems.

2. The Problem

The main problems that prevent Intel CAD design environment from moving directly to a pure IA-NT compute environment:

- UNIX-centric runtime and test environment scripts can not be reliably ported to IA-NT
- Not having the complete set of design CAD tools on IA-NT (compile time and run time dependent tools)

Intel CAD runtime and test environment is made up of millions of lines of UNIX-centric Perl and shell scripts. The runtime environment drives the microprocessor design flow process. These infrastructure scripts act as gluing utilities that integrates the various CAD applications into a functional design environment. Some of the tasks include data format translation, data extraction, simulation, data analysis, waveform analysis, performance analysis, design validation, etc. The CAD runtime environment needs to be both accurate and reliable to support design processes that might run for days.

The test system for CAD tool suite consists of UNIX scripts which provide a dynamic and open environment in the sense that these scripts can take any number and type of test cases and analyzers as arguments. The test system scripts rely heavily on UNIX-centric process and user environment. Some tasks that appear straightforward on UNIX might turn out to be non-trivial on Windows NT. The following example illustrates the point.

Script 1: *foo*

```
#!/usr/local/bin/perl
system("foo1 'grep -i -v fail' datafile");
```

Script 2: *foo1*

```
#!/usr/local/bin/tcsh -f
cat $argv[2] | $argv[1]
```

ASCII file: *datafile*

```
test data - success
test data - fail
```

Example 1

On UNIX, executing *foo* will output:

test data - success

On Windows NT, however, executing the same script will fail due to couple of errors (even under tcsh with proper TCSHSUBSTHB environment setup).

The first error happens when the script *foo* calls *foo1*. Since *foo1* is a tcsh script and Windows NT does not support execute permission on scripts, "tcsh -f" needs to be added to system() to invoke *foo1* as follows:

system("tcsh -f foo1 'grep -i -v fail' datafile");

The second error is due to the fact that Win32 Perl does not escape the single quote (') character. As a result, the second argument to *foo1* on NT will be:

'grep -i -v fail'

instead of:

grep -i -v fail

as in UNIX. Of course, changing the single quote to double quote (") will get the script to work under NT.

In brief, we want to bring up a point that even though the commercial and public UNIX-like utilities are available on Windows NT, the native NT scripting environment does not remotely resemble UNIX scripting environment. During the porting of our CAD runtime and test system, we realized that the currently available UNIX utilities for NT can not support and maintain a production-level quality UNIX-like infrastructure environment for Intel CAD tool development and design activities on Windows NT.

Besides the many concerns we have with the UNIX-like tools' reliability and functional incompatibility on Windows NT, the main discouragement comes from the high failure rate of UNIX-centric scripts performing in the integrated UNIX-like scripting environment under Windows NT. We have pointed out some of the problems in example one above. The many "little" differences between UNIX and Windows NT such as escape characters, executable loading mechanism, search-path support for scripts, limited DOS shell, etc. make the runtime environment highly unstable, especially in cases where executable flows are moving back and forth between Perl and shell scripts. These scripts rely heavily on idiosyncratic UNIX environment to behave properly.

Example: *system($argv[1]),* where *$argv[1]* can be an executable, a Perl script, a shell script, or a command series such as *'awk ... | grep ... | head ...'.*

We came to the conclusion that we might be able to put a patch here and there to fix the immediate problems as they occur in our runtime and test environment. However, with millions of lines of scripts, it is very unlikely that we can achieve the stability level we need for our CAD development and design activities. We realized that we must not pretend that we are still using UNIX on NT and began to investigate alternatives to our problems.

3. The Analysis

Our project goal is clear and simple that is to move Intel CAD design environment to IA-NT. Two primary boundary conditions while meeting this goal are:

1. The skill set of Intel CAD tool developers and chip design engineers is UNIX-centric. Any chosen solution to migrate Intel CAD environment from UNIX to Windows NT needs to be least disruptive to these developers and design engineers.
2. Intel CAD environment consist of both internally developed and external CAD applications and libraries. The chosen solution must be in accordance with the computing standard of the EDA industry to assure the co-existence of internal tools with external tool libraries.

The right solution is to re-architect Intel CAD tools, runtime and validation environment to take advantage of native Windows NT features (Windows NT Logo standards, ActiveX controls, DCOM, etc.) coupled with Windows NT centric extension language interface to achieve the complete benefits of native Windows NT environment. However, this is undoubtedly a long-term project and we are working towards that direction.

We realized that the practical short-term solution is to use Windows NT CAD applications with pre-existing UNIX centric Intel CAD runtime and test environment scripts to enable Intel developers and design engineers a gradual exposure and a smooth migration to NT.

Our investigation of using exiting Win32 emulated UNIX scripting environment can not meet our need as we have illustrated in the earlier problem section. The second approach in trying to use the POSIX subsystem (both native Windows NT version and third-party version) also did not solve our problem because Intel CAD tools need to be native to Win32 subsystem due to external vendor libraries dependency. The POSIX subsystem does not work well in the areas of multi-level inter-process invocation between WIN32 applications and POSIX utilities, symbolic links for project data sharing, automatic environment variables conversion, and NT-UNIX shared file-systems support.

After several attempts to get a complicated UNIX scripting environment to work flawlessly and reliably on Windows NT, we re-examined our approach from a different angle. Instead of moving the UNIX centric CAD design environment to Windows NT, why not integrate native Windows NT applications into our pre-existing UNIX CAD environment?

With this refreshing idea, we set out to develop an in-house product, the MTEX technology (Multi-platform Tool Execution eXtensions). MTEX allows CAD tools to be configured to run on either UNIX or Windows NT workstations while the bulk of CAD runtime and test environment scripts remain on UNIX. This transition technology serves as a bridge to allow the gradual migration of Intel CAD tools and environment and the gradual update of Intel engineers' mind-set and skill-set from UNIX to Windows NT.

4. Design Goal

As a UNIX to Windows NT transition technology, the MTEX has a distinguished design goal:

Support a mixed NT-UNIX CAD design environment which enable gradual migration of Intel CAD design tools and design environment from UNIX to Windows NT. This requirement has two advantages:

a. Allows to selectively migrate the high compute usage CAD design tools from UNIX to native Windows NT to start taking advantage of the cheaper IA-NT computing cycles early.
b. Allows CAD developers to use the existing UNIX-centric test systems and test vectors to validate their design tools on Windows NT. This eliminates the uncertainty with the correctness of the newly developed test cases on Windows NT.

With time Intel's CAD design environment will be moving from UNIX majority to Windows NT majority design tools and scripts.

5. MTEX Overview

MTEX is a transition technology to enable the integration of a mixed NT-UNIX design environment. The idea behind the MTEX capability is to extend the remote procedure call concept to a remote execution environment. The result is a tool environment that allows transparent execution of CAD tools in mixed NT-UNIX platforms. In this cross-platform environment, the user will be working on an NT desktop and executing design commands from a remote UNIX xterm window using the same familiar design flow written in scripts. The user view is illustrated in Figure 1.

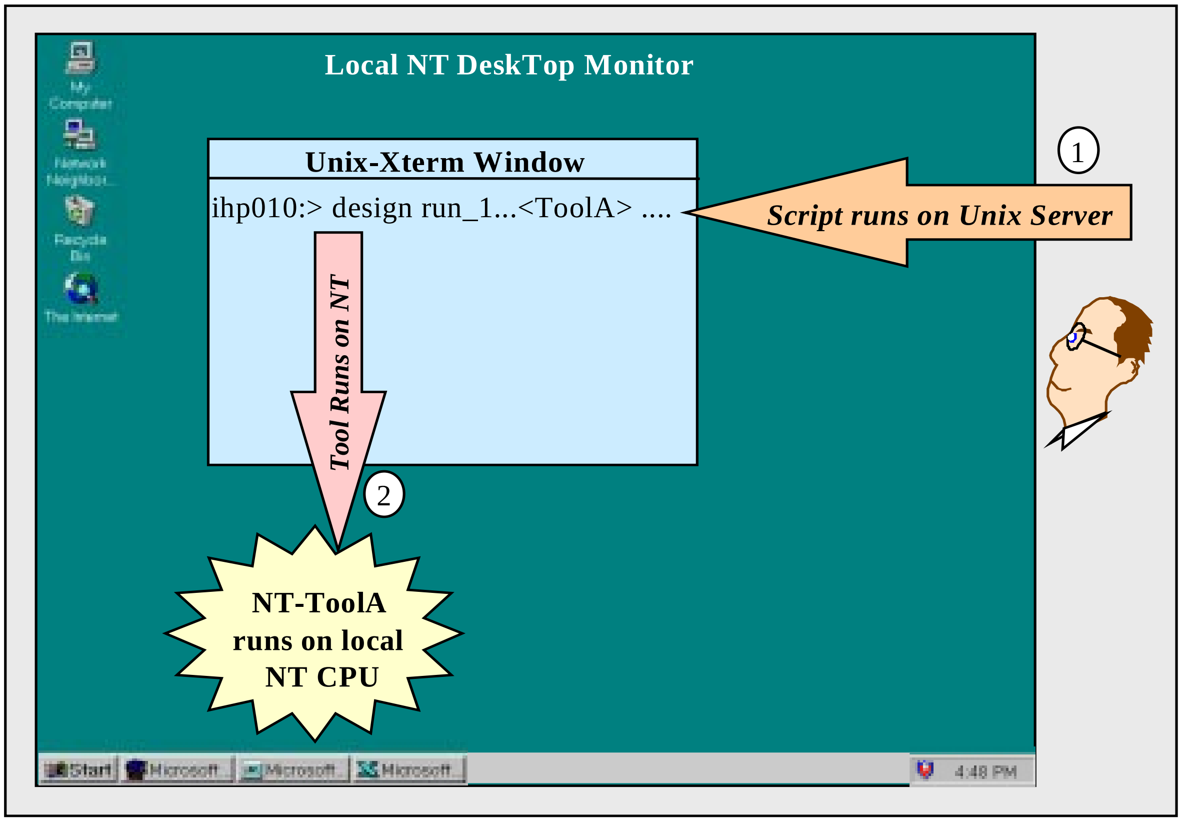

Figure 1: User View of MTEX Environment

6. MTEX Architecture

The following block diagram provides a high level architecture of the MTEX application.

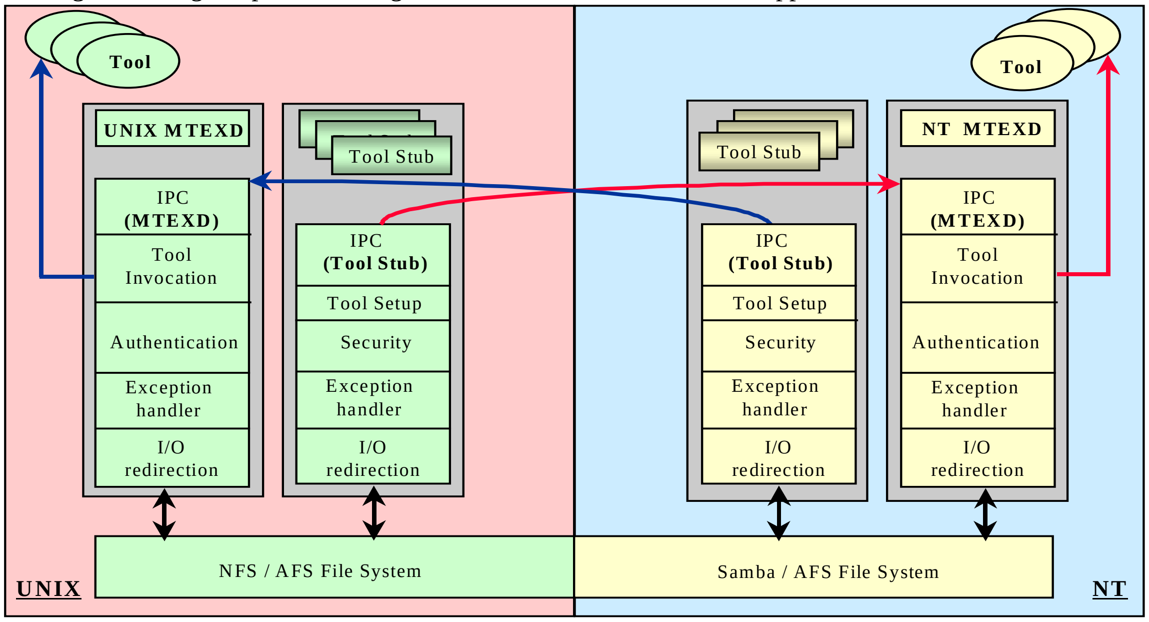

Figure 2: MTEX Architecture

6.1. MTEX Components

MTEX is implemented with three components: a UNIX server daemon, an NT server service, and CAD tools and scripts that are set up to run under MTEX.

Every tool executable in this environment has two complementary parts, namely the tool executable itself and the tool stub, each of which is running on different platforms. The tool stub is a wrapper running on local system that launches the tool executable on the remote system.

When a MTEX tool stub is executed on the local system, it takes a snapshot of the local run-time environment and sends this information to the server daemon (or service). The MTEX server on the remote system impersonates the local user, duplicates the local run-time environment with necessary platform specific adjustments, and invokes the actual tool executable. The MTEX server also handles the routing of STDIO streams to the tool stub and passes the exit code of the tool executable to the tool stub. The tool stub exits with the same exit code. An overview of how MTEX components work together is shown in Figure 3

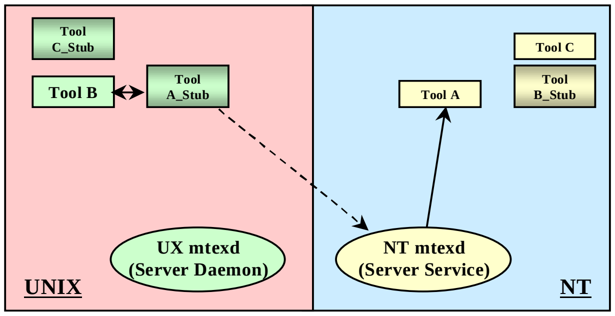

Figure 3: Overview of MTEX Components

6.2. MTEX Features

MTEX provides the following features to enable transparent remote tools execution:

1. Conversion and customization of the run-time tool environment. The run-time environment on the local system is duplicated on the remote system for tool execution.
2. NT and UNIX shared file-system support so that it can handle the tool stubs, executables, data, and output files residing on shared file systems.
3. Handling the redirection of standard input, standard output, and standard error streams between the tool stub on the local system and the tool executable on the remote system.
4. Providing the capability to execute binary files on the remote system (UNIX or NT).
5. Providing the capability to launch the scripts on the remote system (UNIX or NT).
6. Duplication of the shared file system credentials for transparent user access from local to remote system.
7. Impersonation of local system user on the remote system and execution of the tool with the same user credential.

In Figure 4 we illustrate how MTEX can be integrated into pre-existing UNIX CAD design environment. In the figure, a shared complementary directory structure is set up between UNIX and NT. The execution flow begins when the script *driver_script* invokes the tool named *testtool* on UNIX. Testtool is a MTEX client, which invokes the real testtool.exe on Windows NT. After *testtool.exe* has finished execution, the execution control returns to testtool stub on UNIX-end which then exits with testtool.exe's exit status. At this point, the *driver_script* continues with the execution flow to invoke the *analysis_script*, on UNIX.

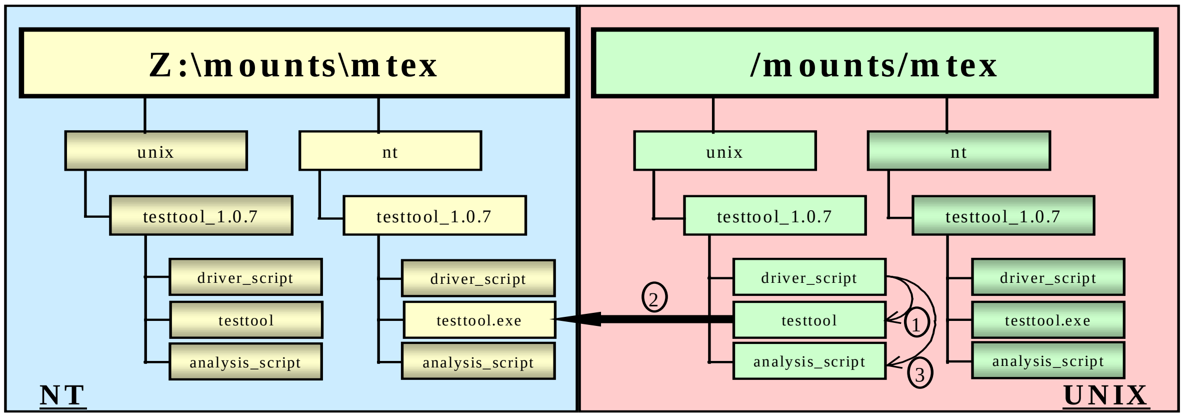

Figure 4: Complementary Directory Structure and Execution Flow under MTEX

6.3. MTEX Internals

MTEX features are implemented across the MTEX server and client. This section shows how the features work and how they are structured in MTEX server and MTEX client to provide a platform independent execution environment.

6.3.1. MTEX Server Internal Modules

As shown in Figure 5, the MTEX server has four major modules. They are IPC, Authentication, Tool Invocation, and I/O Redirection. For clarity purpose, we will omit the error and exception handling routines in this section and in the diagram.

MTEXD, the MTEX server, is to be started by a superuser on UNIX and a user with administrative privileges on NT. It can also be setup to be invoked as part of the boot-time initialization. The MTEX server listens for connection requests by MTEX clients. On detection of a connection request, the server starts a new thread of execution (or a new server process) to handle this request. The original thread returns to listen for client requests.

The IPC module receives the message packets from MTEX client (stub). It decodes the message and to retrieve the following important information:

- User's login information
- AFS token information
- Current working directory
- Customized runtime environment
- Application to be invoked and its arguments.

The Authentication module takes user login and AFS token information and authenticates with local operating system and the AFS server. If the authentication fails, appropriate error handling action is taken; otherwise, the execution proceeds with the Tool Invocation module.

The Tool Invocation module sets the current working directory and the tool runtime environment customized for the local platform. It then spawns a new process to invoke the tool with the correct command line arguments.

The I/O redirection module redirects the STDIO streams for the newly invoked tool back to the MTEX stub's STDOUT/STDERR sockets as well as redirects the stub's input data to the tool's STDIN stream. When the tool exits, the MTEX server gets the exit code of the tool and sends it back to the stub.

The execution flow of the above four MTEX server modules is illustrated in Figure 5

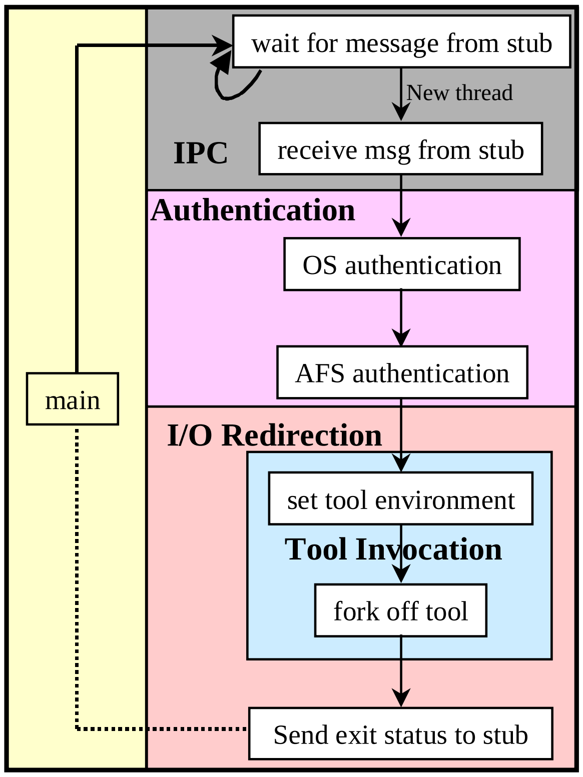

Figure 5: MTEX Server Execution Modules

6.3.2. MTEX Client Internal Modules

The MTEX client also has four major modules. They are Tool Setup, Security, I/O Redirection, and IPC.

The Tool Setup module collects user environment related data and converts the environment variables according to the conversion rules specified in the configuration file for the target platform. An example of this conversion may include adding default search paths.

The Security module collects the user login information as well as the AFS information to be processed by MTEX server's Authentication module.

The I/O redirection module redirects the local MTEX stub input data to the appropriate MTEX server socket and redirects the remote tool's output data to the matching local stub's STDIO streams. The I/O redirection is totally transparent to the user.

The IPC module encodes the data collected by the Tool Setup and Security modules into message packets and sends them to the MTEX server. It then goes into a send/receive state until the lifetime of the tool. When it receives the exit message from MTEX server, it puts the stub to exit with the same status as that of the tool on the remote system.

The execution flow of the above four MTEX client modules is illustrated in Figure 6

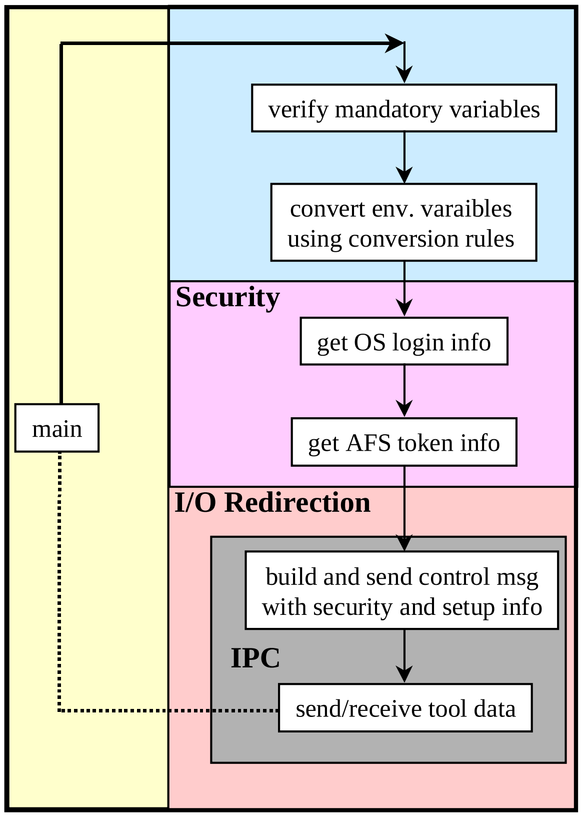
verify mandatory variables
convert env. varaibles using conversion rules
Security
get OS login info
main
get AFS token info
I/O Redirection
build and send control msg with security and setup info
IPC
send/receive tool data

Figure 6: MTEX Client Execution Modules

6.4. MTEX Implementation

MTEX is developed for HP-UX 10.01, HP-UX 10.20, AIX 4.1 and Windows NT 4.0. The MTEX server has been implemented as a daemon on UNIX and as a Win32 service on Windows NT.

The IPC communication between MTEX server and client is over TCP using BSD Sockets on UNIX and Winsock 2.0 on Windows NT. AFS library calls are used by the Security module to pass in user's AFS tokens to MTEX server and by the Authentication module to establish user's AFS credentials on the server end. User authentication on Windows NT is achieved by using the Win32 API CreateProcessAsUser() to impersonate the current logged-on user. This has the implication that the user has to be logged-on to a Windows NT desktop in order to use that system as the remote system under MTEX.

The Tool Setup module copies relevant current user runtime environment such as working directory, process environment block, etc. while the Tool Invocation module recreates the user runtime environment on MTEX server end. Appropriate adjustments are made to the environment to account for syntax differences, system variables and path differences, etc. between the local and remote system as shown in the following example.

Original list of variables

```
DISPLAY=sccia677.sc.intel.com:0.0
HOME=/home/vlal
HOSTTYPE=hp9000s700
MTEX=/mtex_cad/hp700_ux100/mtex/v1.1
PATH=/bin:/usr/bin:/usr/vue/bin:/usr
/bin/X11:/usr/ccs/bin:.
```

Modified list of variables

```
DISPLAY=sccia677.sc.intel.com:0.0
HOSTTYPE=nt_4.0
MTEX=/mtex_cad/nt_4.0/mtex/v1.1
PATH=C:/WINNT;C:/WINNT/system32;X:/c
ygnus-b18/h-i386-cygwin32/bin;
X:/Perl/5.004_04/bin;.
PERL5LIB=X:/Perl/5.004_04/lib
```

Example 2: Environment Variables Modification

The variables can be divided into the following 4 categories based on the operations performed on them.

- Variables that remain unchanged (e.g. DISPLAY)
- Variables that are modified for the target platform (e.g. HOSTTYPE, MTEX,PATH)
- New variables that are introduced for the target platform(e.g. PERL5LIB)
- Variables that are blocked from being sent over to the target platform (e.g. HOME)

Besides the major modules, MTEX routines also support error and exception handling and recovery, remote execution configuration and customization (e.g. MTEX_PREPATH=xxx will prepend xxx to the $PATH environment variable on remote machine), etc. To make the technology simple and easy to understand, we only cover the main features in this paper. All MTEX modules are implemented with documented library calls, system calls, and Win32 APIs.

```
#!/usr/intel/bin/perl

($scriptName = $0) =~ s/.*\///g;

# …< check for required environment variables > ...

exec("$mtexDir/mtexstub", <Remote system tool
path>, $scriptName, "@ARGV");

die "$scriptName -E- exec failed...";
```

Example 3: A simple UNIX front-end tool stub

A MTEX tool stub consists of two parts: the front-end stub and a back-end generic MTEX client. The front-end stub gets the name of its complementary tool on the remote system (e.g. the stub for foo.exe is foo).

On UNIX, the front-end stub is a Perl script that simply invokes the generic MTEX stub with all of its command line arguments. Example 3 presents a sample UNIX front-end tool stub.

The back-end generic MTEX client, known as mtexstub in the above example, is a thin client executable on UNIX. It implements the MTEX client modules described in section 6.3.2.

On Windows NT, the front-end stub is a simple 'C' application and the back-end MTEX client is an explicitly loaded DLL that contains the code for MTEX client modules. Example 4 presents a sample Windows NT front-end tool stub.

Special utilities are developed to facilitate the generation of tool stubs. These utilities can be integrated into a tool makefile to automate the entire process of build, test, release, and the creation of required MTEX stubs for the tool.

```
#include <windows.h>
#include <stdio.h>
int main(int argc, char **argv)
{
     int retCode = 1;
     HINSTANCE hMtexLib;
     Char buffer[260];
     FARPROC mtexEntry;

     Buffer[0] = '\\0';
       .
       .
   // Get the MTEX environment variable. to load the mtexstub DLL
   sprintf(buffer, "%s/mtexstub.dll", getenv("MTEX"));
   // Load the MTEX stub library
   //
     if ((hMtexLib = LoadLibrary(buffer)) == NULL) {
         printf("%s: -E- Failed to load the library %s - %d\\n", argv[0], buffer, GetLastError());
         exit(GetLastError());
     }
    // Get the address of startMTEX function
   //
     if ((mtexEntry = GetProcAddress(hMtexLib, "startMTEX")) == NULL) {
        printf("%s: -E- Failed to get the entry point of the stub startMTEX -  %d\\n", argv[0], GetLastError());
        FreeLibrary(hMtexLib);
        exit(GetLastError());
     }

     // Invoke the startMTEX function with the requisite arguments
     // Expand the toolPath is it starts with a '\$'- env. Variable
    //
     retCode = (\* mtexEntry)(argc, argv,<toolPath……>);

   // Free the MTEX library
   FreeLibrary(hMtexLib);

     // Exit with the exit code returned by startMTEX(...) function
    exit(retCode);
}
```

Example 4: A simple MTEX NT front-end tool stub

7. User Experience

Over the past year, the CAD organization at Intel has successfully used MTEX to validate more than 40 IA-NT ported CAD tools (3 million lines of Win32 ported code) using the same test system from UNIX by running more than 1000 test flows in a seamless mixed NT-UNIX platform. The test flows require execution of both the IA-NT ported CAD tools as well as the UNIX dependency CAD tools that are not yet available on Windows NT (MTEX enables runtime client-server modeling).

We have also received encouraging feedback from our UNIX CAD developers who participated in the early IA-NT productization and deployment effort. With MTEX, they are able to pick up the IA-NT ported codes for their tools from migration engineers and integrate back into their environment with little effort. They can use the pre-existing test suite and test system to validate the correctness of the ported codes. With no environment change, the deployment task to IA-NT has proven to have negligible impact to their daily work.

These CAD engineers appreciate the flexibility of the MTEX that enabled them to combine tools from both NT and UNIX environment to form a single seamless platform independent design flow execution environment. Having a familiar tool working under Windows NT, it would be a good starting ground for them to pick up Windows NT knowledge and to work on future improvements to their tool environment to take advantage of native Windows NT features.

8. Limitations & Future Work

The limitations of current MTEX version are:

- Scripts need to be modified to remove all references to local paths such as /tmp since the mixed UNIX and NT environment depends on the shared file system for their data exchange.
- If application processes communicate with each other through IPC, all of these processes need to be executed on the same platform.
- For MTEX to work correctly, the user has to be logged on to the NT system.

The next version of MTEX will offer the following features:

- Utilities to display MTEX invoked processes and to terminate MTEX invoked processes from any system where MTEX is installed.
- Removal of the limitation of the user to be logged on to the NT system for MTEX to work correctly.
- Conversion rules moves to the server end so that support for a new platform can be added on the fly.
- GUI interface for MTEX management.

9. Conclusion

The Multi-platform Tool Execution eXtensions (MTEX) technology is a simple but powerful technique to provide a transition path, a bridge to migrate a very complicated UNIX-centric environment to Windows NT.

MTEX is by no means THE solution to the UNIX scripting problems on Windows NT. However, we believe that MTEX is a very important transition technology to enable a successful UNIX to NT migration. By supporting a platform independent runtime environment, MTEX is implicitly enforcing the classic computer science "divide and conquer" strategy. What seems to be an impossible mission of moving all Intel CAD design tools, CAD runtime and test environment, user training, etc. from UNIX to Windows NT turns out to be a feasible project by using MTEX. The reader can refer to our paper in Intel Technology Journal (Q1/99 issue, details in reference section) for another innovative compile-time client-server technology for their mixed NT-UNIX applications.

Our motivation for writing this paper is to share our experience on an innovative transition technology we engineered to meet our CAD migration needs. MTEX is generic and can be easily proliferable to any other environments in any organization. However, we believe that the best migration strategy is the one that have been carefully studied and evaluated according to the specific requirement of the organization.

10. Acknowledgements

We thank Intel management for providing the opportunity to work on this exciting project. We are grateful to our department manager Tae Paik for his vision, inspiration, patience, and continuous encouragement during this project. We wish to acknowledge Tae Paik for the brainstorming session that helped us to come up with a run-time client-server model. Thanks to Athena-NT Technical Working Group members for ratifying the technical concepts and documents. Special thanks to Tzvi Melamed, Greg Hannon, Chenwei Chiu, and Ming Lin for their technical feedback and encouragement during the development work. We would like to acknowledge Rumi Zahir for motivating us to write this paper.

11. Trademarks

12. References

[1] Shesha Krishnapura et al. "CAD Design Flows Development in a Cross-Platform Computing Environment", Intel Technology Journal Q1 1999.

[2] Alexander Wolfe, "Intel taps Windows NT in design-software shift." EETIMES, issue 948, April 7, 1997, pages 1, 148.

[3] Richard Goering, "Can NT win in IC design?" EETIMES, issue 992, page 70.

[4] Jeffrey Richter, Advanced Windows (3rd Ed) Microsoft Press 1997.

[5] Marshall Brain, Win 32 System Services: The Heart of Windows 95 and Windows NT. Prentice Hall 1995.

[6] Andrew Lowe, Porting Unix Applications to Windows NT. Macmillan 1997

[7] David A. Solomon, Inside Windows NT (2nd Ed). Microsoft Press 1998.

[8] Jeffrey Richter, Win32 Q&A. Microsoft Systems Journal June 1998.

[9] Jeffrey Richter, Manipulate Windows NT Services by Writing a Service Control Program, Microsoft Systems Journal February 1998.

[10] Jeffrey Richter, Design a Windows NT Service to Exploit Special Operating System Facilities, Microsoft Systems Journal October 1997.