COMPONENT WORKLOAD EMULATOR UTILIZATION TEST SUITE

User and Developer's Guide

Version 1.0

Contents

Preface

Enterprise distributed systems (such as air traffic management systems, cloud computing centers, and shipboard computing environments) are steadily increasing in size (e.g., lines of source code and number of hosts in the target environment) and complexity (e.g., application scenarios). To address challenges associated with developing next-generation enterprise distributed systems, the level of abstraction for software development steadily increases as well. Distributed system developers therefore focus more on the system's "business-logic" instead of wrestling with low-level implementation details, such as development and configuration, resource management, fault tolerance. Moreover, increase the level of abstraction for software development helps promotes reuse of the system's "business-logic" across different application domains, which inherently reduces (re)invention of core intellectual property.

Although increasing the level of abstaction for software development is improving the software lifecycle of next-generation enterprise distributed systems, system quality-of-service (QoS) properties (*e.g.*, latency, throughput, and scalability) are not evaluated until late in the software lifecycle, *i.e.*, during system integration time. This is due in part to the *serialized-phasing developmen problem* where the infrastructure- and application-level system entities, are developed during different phases of the software lifecycle. Consequently, distributed system developers do not realize the system under development does not meet its QoS requirements until its too late, *i.e.*, during complete system integration time.

System execution modeling (SEM) tools is a model-driven engineering technique that helps overcome the effects of serialized-phasing development. SEM tools provide distributed system developers with the necessary artifacts and tools for modeling system behavior and workload, such as computational attributes, resource requirements, and network communication. The constructed models are then used to evaluate QoS properties of the system under development during early phases of the software lifecycle. This enables distributed system developers to pinpoint potential performance bottlenecks before they become to costly to locate and rectify later in the software lifecycle.

The Component Workload Emulator (CoWorkEr) Utilization Test Suite (CUTS) is a SEM tool designed for next-generation enterprise distributed systems. Distributed system developers and testers use CUTS to model the expected behavior and workload of the system components under development using high-level domain-specific modeling languages. Model interpreters then transform the constructed behavior and workload models into source code for their target architecture, *i.e.*, the components same interfaces and attributes as their real counterpart. Finally, system developers and testers emulate the auto-generated components in their target (or representative) environment and collect QoS metrics. This enables distributed system developers and testers to conduct system integration test at early stages of software lifecycle, instead of waiting

until complete system integration time to perform such testing.

This book therefore is the end-user's guide to CUTS. It contains information on building and installing CUTS, using its domain-specific modeling languages to model behavior and workload, creating system experiments, and details on CUTS many tools that help simpify evaluting QoS of next-generation distributed systems continuously throughout the software lifecycle.

Part I

Getting Started

Chapter 1

Building and Installing CUTS

CUTS has many small projects that comprise the entire system execution modeling (SEM) tool. Its many projects, however, can be divided into three main categories, or project groups:

- Runtime architecture
- Modeling tools
- Analysis tools

This chapter discusses how to build and install each of the aforementioned project groups for CUTS. Depending on your usage of CUTS, you may not need to build and install all projects in a category on the same machine. For example, you may install the runtime architecture of CUTS in your testing environment, and the modeling and analysis tools outside of your testing environment to minimize interference with the testing process when viewing collected performance metrics in real-time. We are aware of these needs, and have setup the build process to decouple projects within a group from projects external to its corresponding category. You can, therefore, refer to each of the following sections on building and installation in isolation since project groups do not explicitly depend on each other.

1.1 CUTS Runtime Architecture

The runtime architecture for CUTS allows developers and testers to emulate system experiments on their target architecture. CUTS also provides the mechanisms to monitor and collect performance metrics for the executing system. In order to build the CUTS runtime architecture, you will need the following technologies installed on the target machine(s):

- **DOC Group Middleware** - This set of middleware is used to abstract away complexities associated with implementing applications that operate on many different operating systems (ACE) and perform distributed communication (TAO). You can download DOC Group Middleware from the following location: `http://www.dre.vanderbilt.edu`.

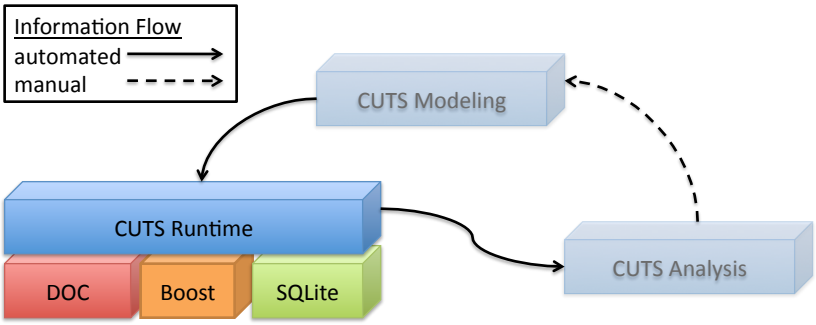

Figure 1.1. Building blocks for the CUTS runtime architecture

- **Boost** - This set of libraries is used for implementing the parsers (Boost Spirit) used within different artifacts of CUTS. You can download Boost from the following location: `http://www.boost.org`.
- **SQLite** - This set of libraries is used to support flat file archives for test results. You can download SQLite at the following location: `http://www.sqlite.org`.
- **PCRE (not pictured)** - This set of libraries is used to enable PERL regular expression support in CUTS. You can download PCRE at the following location: `http://www.pcre.org`.
- **XSC (not pictured)** - This set of libraries and applications is used to convert XML documents to/from objects. You can download XSC from the following location: `svn://svn.dre.vanderbilt.edu/XSC/trunk`.

1.1.1 Obtaining Source Files

You can obtain the latest snapshot of the CUTS runtime architecture from the DOC Group Subversion repository at the following location:

```
svn://svn.dre.vanderbilt.edu/DOC/CUTS/trunk/CUTS
```

If you need to download a stable version of the source code, then you can access at one of the subdirectories in the repository at the following location:

```
svn://svn.dre.vanderbilt.edu/DOC/CUTS/tags
```

1.1.2 System Configuration

Before you can build CUTS, you must for configure you environment. Please set the following environment variables:

```
%> export CUTS_ROOT=location of CUTS
%> export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CUTS_ROOT/lib
%> export PATH=$PATH:$CUTS_ROOT/bin
```

Please see Appendix A for instructions for configuring, building, and installing third-party libraries needed by CUTS, such as DOC Group Middleware, Boost, and SQLite.

1.1.3 Installation

We use Makefile, Workspace, Project Creator (MPC) [1] to assist in building CUTS on different operating systems, and with different compilers. Before you can build the runtime architecture, you must first generate the target workspace. Use the following command to generate the workspace:

```
%> $ACE_ROOT/bin/mwc.pl -type TYPE [-features FEATURES] CUTS.mwc
```

where TYPE is your compiler type, and FEATURES is a comma-separated list of enabled/disabled features for your build of the CUTS runtime architecture.[2] The complete set of features for CUTS is located in the following file:

```
%> $CUTS_ROOT/default.features.tmpl
```

It is recommended that you copy the file above to the following location:

```
%> $CUTS_ROOT/default.features
```

and enable/disable the appropriate features for your workspace by modifying `default.features`. This way you do not have to use the `-features` command-line option unless you want to override your default feature selection. Once you have generated the workspace, you can build the solution using your specified compiler.

1.2 CUTS Modeling Tools (Windows-only)

CUTS modeling tools provide system developers and testers with an enviroment for rapidly constructing experiments for distributed component-based systems, and generating testing system for their target architecture. The modeling tools are built on the following technologies:

- **GME** - The Generic Modeling Environment (GME) is a graphical modeling tool for creating domain-specific modeling languages (DSMLs). You can download GME from the following location: http://www.isis.vanderbilt.edu/projects/GME.
- **CoSMIC** - The Component Synthesis via Model Integrated Computing (CoSMIC) is a tool suite designed to address the complexities of building large-scale component-based systesm, such as sytem assembly, deployment, packaging, and planning. You can download CoSMIC from the following location: http://www.dre.vanderbilt.edu/cosmic.

[1] For more information on MPC, please see the following location: http://www.ociweb.com/products/mpc
[2] You can type `$ACE_ROOT/bin/mwc.pl --help` to view the command-line options for MPC.

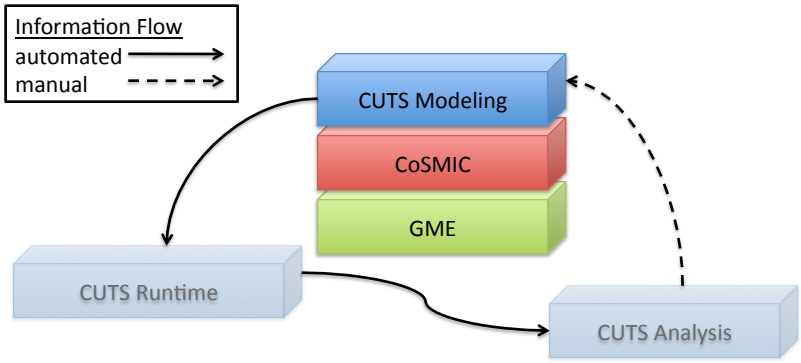

Figure 1.2. Building blocks for the CUTS modeling tools

1.2.1 Installing Prebuilt Version

The easiest and quickest way to install the CUTS modeling tools is to install them using the .msi installer. You can download the latest version of the CUTS modeling tools from the following location: http://www.dre.vanderbilt.edu/CUTS/downloads. Before you can install the CUTS modeling tools, please make sure you have installed GME and CoSMIC. Once both GME and CoSMIC are installed (in that order), then you can install the CUTS modeling tools.

1.2.2 Building from Sources (advanced)

If you choose, you can build the modeling tools from source. This is given you have downloaded all the source from the CUTS source code repository (see Section 1.1.1). To build the CUTS modeling tools, first install the latest version of GME. Then, you **MUST** build CoSMIC from source as well. This is required because the CUTS modeling tools are built on top of CoSMIC, and its therefore dependent on CoSMIC. You can find instructions for building CoSMIC from source at the following location:

http://www.dre.vanderbilt.edu/cosmic/downloads

After you have built and installed CoSMIC from sources, you are ready to build the CUTS modeling tools from source. Please use the following commands to build the modeling tools using your flavor of Visual C++:

```
%> cd %CUTS_ROOT%
%> mwc.pl -type [vc type] -features modeling=1,cosmic=1 CUTS_CoSMIC.mwc
%> open solution and build
```

The build process will ensure that all the interpreters are installed and configured for your environment.

1.3 CUTS Analysis Tools (Windows-only)

The CUTS analysis tools allow developers to view and analyze system performance metrics. The analysis tools are developed using the following key technologies:

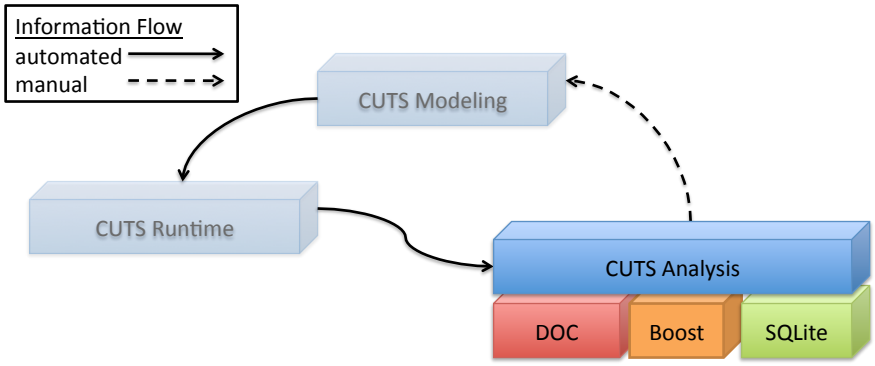

Figure 1.3. Building blocks for the CUTS analysis tools

As shown in Figure 1.3, the same technologies used to develop the runtime toolset (see Figure 1.1) are used to develop the analysis toolset. Because the same technologies are used in the analysis toolset, the core projects in analysis toolset is also built while building the runtime toolset.

Part II

CUTS Runtime Infrastructure

Chapter 2

CUTS Logging Facilities

This chapter discusses the CUTS logging facilities. These facilities are designed to simplify collection of system execution traces in a distributed environment. Moreover, the facilities simplify partitioning system execution traces with different tests. This therefore makes it possible to execute multiple tests in the same environment while preserving data integrity. Finally, the logging facilities discussed in this chatper are extensible via intercepters, and are highly configurable so they adapt to different application domains.

2.1 Overview

Distributed systems consist of many software components executing on many hosts that are connected via a network. When validating distributed system function and quality-of-service (QoS) properties, *e.g.*, end-to-end response time, scalability, and throughput, it is necessary to collect data about the systems behavior in the target environment. Such behaviors could be the events executed of the execution lifetime of the system, the state of the system at different points in time, or data points needed to calculate the end-to-end response of an event.

When dealing with data collection in a distributed environment, such as those that host distributed systems, data is collected and analyzed either *offline* or *online*. In offline collection and analysis, data from each host is written to local persistent storage, such as a file, while the system is executing in its target environment. After the system is shutdown, the collected data in local storage on each host is combined and analyzed. The advantage of this approach is network traffic is kept to a minimum since collected data is not transmitted over the network until after the system is shutdown. The disadvantage of this approach is collected data is not processed until the system is shutdown. This can pose a problem for systems with a long execution lifetime, *e.g.*, ultra-large-scale systems that must run 24x7, or when trying to monitor/analyze a system in real-time using collected data.

In online collection and analysis, data is collected and transmitted via network to a host outside the execution environment. The advantage of this approach is that it allows analysis of metrics data an environment that does not use the systems resources. This also helps not to skew the results while the system is running. The disadvantage of online collection and analysis is the difficulty of devising a strategy for efficiently collecting data in a distributed environment and submitting it to a central location without negatively impacting

the excuting systems QoS—especially if the system generates heavy network traffic.

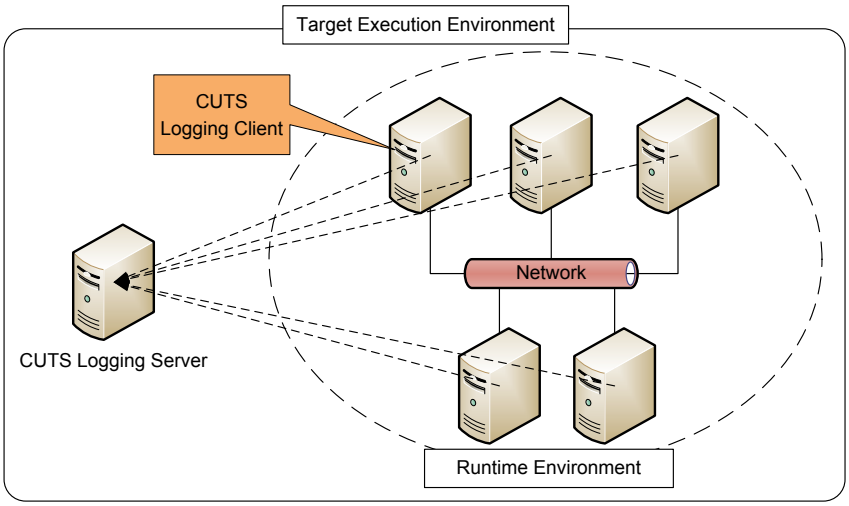

Figure 2.1. Overview of the CUTS logging facilities.

Because online collection and analysis is a more promising approach to data collection and analysis, CUTS logging facilities implement online data collection. As illustrated in Figure 2.1, there is a target execution environment and a runtime environment. The runtime environment is where the system is executing and generating data for collection. Each host in the runtime environment has a logging client (see Section 2.3). The logging client is responsible for collecting data from individual software/hardware components on its host. After collecting data on the host, the logging client periodically submits collected data to a logging server (see Section 2.2), which is executing outside of the runtime environment.

The remainder of the chapter therefore presents the basics for using the CUTS logging facilities in a production-like environment. In particular, the following information is covered in this chapter:

- **CUTS Logging Server** - The CUTS Logging Server is responsible for collecting data from individual CUTS Logging Clients executing in the runtime environment. Details of the CUTS Logging Server are presented in Section 2.2.

- **CUTS Logging Client** - The CUTS Logging Client is responsible for collecting data from individual client loggers and submitting it to the CUTS Logging Server. Details of the CUTS Logging Client are provided in Section 2.3.

- **Client Logger** - The client logger is responsible for collecting actual data from the runtime environment and submitting it to the CUTS Logging Client executing on its host. Details of the client logger are provided in Section 2.4.

2.2 The Logging Server

The logging server is responsible for collecting data from individual logging clients in the runtime environment (see Section 2.3). As shown in Figure 2.1, there is typically one instance of a logging server running in the target execution environment. There can also be cases when more than one instance of a logging server is desired, such as using multiple logging servers to balance the workload due to the amount of data collected by individual logging clients in the runtime environment. This user's manual, however, assumes that only a single instance of a logging server is used in the runtime environment.

2.2.1 Running the Logging Server

Assuming the CUTS runtime architecture has been built and installed correctly, the CUTS Logging Server is installed at the following location:

```
%> $CUTS_ROOT/bin/cuts-logging-server
```

To see a complete list of command-line options, use the following command:

```
%> $CUTS_ROOT/bin/cuts-logging-server --help
```

2.2.2 Configuring the Logging Server

There are only a handful of command-line options for configuring the logging server. The most important of the available command-line options is `--register-with-iortable=NAME`, where `NAME` is an user-defined name, such as `LoggingServer`. When this option is combined with the `-ORBEndpoint ENDPOINT` CORBA command-line option, an external reference to the logging server is exposed. This is what allows the CUTS Logging Client (see Section 2.3) to connect to the logging server.

The following is an example of exposing the logging server named `LoggingServer` on port 20000 for logging clients to connect:

```
%> $CUTS_ROOT/bin/cuts-logging-server -ORBEndpoint \
   iiop://`hostname`:20000 --register-with-iortable=LoggingServer
```

As shown in the example above, the logging server is listening using port 20000 on its host [1], and has the name `LoggingServer`. The logging client can therefore connect to the logging server using the following CORBA reference:

```
corbaloc:iiop:`hostname`:20000/LoggingServer
```

The next section therefore discusses how to configure the logging client to connect to the correct logging server in the runtime environment.

[1] 'hostname' is used because it is assumed that Unix command-line substitution is available on the host in the example. If such a feature is not available, it is possible to hardcode the hostname in place of 'hostname', *e.g.*, host.foo.bar.com

2.3 The Logging Client

The logging client is responsible for collecting data from individual client loggers in the runtime environment (see Section 2.4). As shown in Figure 2.1, each host in the runtime environment typically runs one instance of a logging client. Similar to the logging server, there can also be cases when more than one instance of a logging client is desired, such as using multiple logging clients to balance the workload due to the amount of data collected by individual client loggers in the runtime environment. This user's manual, however, assumes that only a single instance of a logging client running on each host in the runtime environment.

2.3.1 Running the Logging Client

Similar to the CUTS Logging Server, the CUTS Logging Client is installed at the following location:

```
%> $CUTS_ROOT/bin/cuts-logging-client
```

To see a complete list of command-line options, use the following command:

```
%> $CUTS_ROOT/bin/cuts-logging-client --help
```

2.3.2 Configuring the Logging Client

The CUTS Logging Client is configured similar to how the CUTS Logging Server is configured (see Section 2.2.2). For example, the `-ORBEndpoint ENDPOINT` defines the CORBA endpoint for the logging client. The logging client's endpoint is used by the client loggers (see Section 2.4) that need to connect to the CUTS Logging Client executing on its local host.

The addition to configuration parameters similar to the CUTS Logging Server, to the logging client's configuration requires the location of the CUTS Logging Server. In Section 2.2.2, the logging server was given an endpoint that is used by the logging client when connecting to it. To specify the location of the CUTS logging server, use the `-ORBInitRef LoggingServer=LOCATION` CORBA command-line option, where `LOCATION` is the location of the logging server [2].

The following is an example of configuring the logging client to connect to the logging server discussed in Section 2.2.2 and listens to data from client loggers on port 20000:

```
%> $CUTS_ROOT/bin/cuts-logging-client -ORBEndpoint \
   iiop://`hostname`:20000 --register-with-iortable=LoggingClient \
   -ORBInitRef LoggingServer=corbaloc:iiop:server.foo.bar.com:20000/LoggingServer
```

The next section discusses how to use client loggers to submit data for collection and analysis to the logging clients executing on each host in the runtime environment.

[2] The location of the `LoggingServer` can be a `corbaloc`, `corbaname`, or `IOR`.

2.4 The Client Loggers

The client loggers are responsible for collecting log messages from an application and submitting them to the logging client on its host machine. Since client loggers interact directly with the application, they are language dependent. Currently, CUTS supports client loggers for C++ and Java applications. In addition, CUTS support the following logging frameworks: ACE Logging Facilities and log4j. The remainder of the section dicusses how directly integrate client loggers into an application and indirectly intergrate them using an pre-existing logging framework.

2.4.1 Direct Integration

Direct integration is the simplest method for integrating CUTS logging facilities into an existing application. It is also the most instrusive approach for integrating the logging facilities into an existing application. This is because you have to augment existing code to include the client logger(s). Only then can the application use to client logger to collect messages and submit them to the logging client executing on the localhost.

The following code illustrates integrating the client logger using the C++ version:

```
1  #include "cuts/utils/logging/client/logger/Client_Logger.h"
2
3  int main (int argc, char * argv []) {
4    try {
5      // connect client logger to logging client.
6      CUTS_Client_Logger logger;
7      if (-1 == logger.connect (argv[1]))
8        return 1;
9
10     // Send a single, yet simple message.
11     logger.log (LM_DEBUG, "this is a simple message");
12
13     // Disconnect from the logging client.
14     logger.disconnect ();
15     return 0;
16   }
17   catch (const ::CORBA::Exception & ex) {  }
18   catch (...) {  }
19
20   return 1;
21 }
```

Listing 2.1. Example illustrating integration of the C++ client logger.

Likewise, this code illustrates the same concept using the Java version of the client logger:

```
1  import CUTS.client.logging.Logger;
2
```

```
3  public class TestApplication {
4    public static void main (String [] args) {
5      String loggingClientLoc = /* location of logging client */
6      Logger clientLogger = new Logger ();
7
8      // connect to the logging client
9      clientLogger.connect (loggingClientLoc);
10
11     // send a log message
12     clientLogger.log ("this is a simple message");
13
14     // disconnect from the logging client
15     clientLogger.disconnect ();
16   }
17 }
```

Listing 2.2. Example illustrating integration of the Java client logger.

In both the C++ and Java example, the `connect()` method takes as its input parameter the location of the logging client. This location can be a CORBA IOR, corbaname, or corbaloc (*i.e.*, anything that is a valid parameter for the `string_to_object()` method implemented on the ORB). After a connection is made to the logging client, the client logger submits messages to the logging client using the `log()` method. When the client logger is done (or the application is ready to exit), it disconnects from the logging client using the `disconnect()` method. This is a critical step because it ensures the logging client does not get into an inconsistent state when trying to keep track of the number of active client loggers.

2.4.2 Indirect Integration

Indirect integration is a non-intrusive method for integrating the CUTS logging facilities with an existing application. This approach only work if the application is already using a logging framework, such as ACE Logging Facilities or log4j. Indirect integration works by intercepting log messages sent to the original logging framework and submitting them to the CUTS logging facilities (*i.e.*, the logging client).

Unlike direct integration, system developers do not need to update their existing code base to leverage indirect integration. Instead, they must update the existing logging frameworks configuration files to "install" the necessary inteceptors. The remainder of this section therefore presents details on using indirect integration with supported logging frameworks in CUTS.

log4j

Language. Java
How to integrate. Update `log4j.properties` (or similar file)
Illustrative example.

```
# define the loggers
log4j.rootCategory=ALL, Console, LoggingClient
```

```
# console appender
log4j.appender.Console=org.apache.log4j.ConsoleAppender
log4j.appender.Console.layout=org.apache.log4j.PatternLayout
log4j.appender.Console.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n

# CUTS appender
log4j.appender.LoggingClient=cuts.log4j.LoggingClientAppender
log4j.appender.LoggingClient.LoggerClient=corbaloc:iiop:localhost:20000/LoggingClient
```

ACE Logging Facilities

Language. C++
How to integrate. Update `svc.conf` (or similar file)
Illustrative example.

```
dynamic CUTS_ACE_Log_Interceptor Service_Object * \
 CUTS_ACE_Log_Interceptor:_make_CUTS_ACE_Log_Interceptor() active \
 ''--client=corbaloc:iiop:localhost:20000/LoggingClient''
```

Part III

Appendix

Appendix A

Building and Installing Third-Party Libraries

The following appendix contains best practices for building and installing third-party libraries used by CUTS projects. It is not a requirement to install third-party libraries per the instructions in this appendix. If you are experiencing compilation problems with CUTS due to third-party libraries, however, the instructions in this appendix may be of assistance. To make help explain the installation process of the thirdparty libraries, it will be assumed that all third-party libraries will be installed at the following location: `/opt/thirdparty`.[1]

A.1 Boost

The Boost libraries are required when building and installing CUTS. All artifacts for Boost, such as prebuilt versions and source files, can be downloaded from the following location: `http://www.boost.org`.

A.1.1 Installation

Non-Windows

If you are installing Boost on a non-Windows system, *e.g.*, Linux, Soloris, or MacOS X, then you can download the source code directly from its main website (`http://www.boost.org`), build it, and install it using standard procedures.

Windows

Installing Boost from sources on Windows is not a trivial process. Moreover, the prebuilt version of Boost available for download from the main website does not contain the correct configuration or directory lay-out. It is therefore recommended that you download and use the prebuilt version of Boost available for

[1] This location is representative of a common location for installing third-party libraries, and may be different on your system.

download from the CUTS website at the following location: http://www.dre.vanderbilt.edu/CUTS/downloads/thirdparty/boost. Please download the correct version of Boost that matches your version of Microsoft Visual C++ and extract it to a common location on disk.

A.1.2 System Configuration

To build CUTS correctly, the Boost environment variables must be defined. Please define the following required environment variables:

```
%> export BOOST_ROOT= location of Boost
%> export BOOST_VERSION= name of subdirectery under $BOOST_ROOT/include, e.g., boost-1_34_1
%> export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_ROOT/lib[2]
%> export PATH=$PATH:$BOOST_ROOT/lib
```

In some cases, Boost libraries will have an optional configuration appended to their filename, such as `-mt-d`. To use a specific configuration of Boost, the `BOOST_CFG` environment variable must be defined. The following is an example of defining the Boost configuration environment variable:

```
%> export BOOST_CFG=-mt-d
```

The configuration of Boost is now complete. Once all the other third-party libraries are installed you will be able to build CUTS (see Chapter 1).

A.2 Xerces-C

Xerces-C 3.x is required when building and installing CUTS and other third-party libraries. All artifacts for Xerces-C, such as prebuilt versions and source files, can be downloaded from the following location: http://xerces.apache.org/xerces-c.

A.2.1 Installation

Non-Windows

If you are installing Xerces-C on a non-Windows system, *e.g.*, Linux, Soloris, or MacOS X, then you can download the source code directly from its main website (http://xerces.apache.org/xerces-c), build it, and install it using standard procedures.

Windows

Installing Xerces-C from sources on Windows is not a trivial process. Moreover, the prebuilt version of Xerces-C available for download from the main website does not contain the correct configuration or directory layout. It is therefore recommended that you download and use the prebuilt version of Xerces-C available for download from the CUTS website at the following location: http://www.dre.vanderbilt.

[2] OnWindows-based systems, please add library paths to `PATH`. On Mac OS-basedsystems, please add library paths to `DYLD_LIBRARY_PATH`.

edu/CUTS/downloads/thirdparty/xerces-c. Please download the correct version of Xerces-C that matches your version of Microsoft Visual C++ and extract it to a common location on disk.

A.2.2 System Configuration

To build CUTS correctly, the Xerces-C environment variables must be defined. Please define the following required environment variables:

```
%> export XERCESCROOT= location of Xerces-C
%> export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$XERCESCROOT/lib
```

The configuration of Xerces-C is now complete. Once all the other third-party libraries are installed you will be able to build CUTS (see Chapter 1).

A.3 DOC Group Middleware

DOC Group Middleware is required when building and installing CUTS. All artifacts the DOC Group Middleware can be downloaded from the following location: http://www.dre.vanderbilt.edu/CIAO. The following section therefore covers building and installing ACE, TAO, CIAO, and DAnCE from the DOC Group Middleware.

A.3.1 System Configuration

Unlike Boost, you must configure you system installing ACE and TAO. Once you have downloaded the ACE and TAO tarball from the website above [3], please define the following environment variables:

```
%> export ACE_ROOT= location of ACE
%> export TAO_ROOT= location of TAO
%> export CIAO_ROOT= location of CIAO
%> export DANCE_ROOT=$CIAO_ROOT/DAnCE

%> export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ACE_ROOT/lib
%> export PATH=$PATH:$ACE_ROOT/bin:$CIAO_ROOT/bin:$DANCE_ROOT/bin
```

After defining the ACE, TAO, and CIAO environment variables, the next step is configuring the middleware for the target platform. This is accomplished by defining the `$ACE_ROOT/ace/config.h` header file, which will include the correct configuration for your platform. The included configuration file has the form `config-*`, where `*` is the platform of choice. The following is an example `config.h` file for building ACE, TAO, and CIAO on Linux platforms:

```
// -*- C++ -*-

#ifndef _ACE_CONFIG_H_
```

[3] It is recommended you download either the latest snapshot or beta version of ACE and TAO tarball. This will ensure you have the must up-to-date version of ACE and TAO that is compatible with CUTS.

```
#define _ACE_CONFIG_H_

#include ``config-linux.h''

#endif // !defined _ACE_CONFIG_H_
```

Non-Windows

Configuring ACE, TAO, and CIAO on non-Windows systems requires one more step. The configuration for the build engine also needs to be defined. Please use the following step to define the build engine's configuration:

```
%> cd $ACE_ROOT/include/makeinclude
%> ln -s platform_*.GNU platform_macros.GNU
```

where * is the architecture for your platform of choice.

A.3.2 Installation

After you have configured your system for ACE, TAO, CIAO, and DAnCE, you are ready to build and install ACE and TAO. Please use the following steps to build and install ACE, TAO, CIAO, and DAnCE:

```
%> cd $CIAO_ROOT
%> $ACE_ROOT/bin/mwc.pl -type [build tool] CIAO_TAO_DAnCE.mwc
```

This will generate the workspace for building and installing ACE, TAO, CIAO, and DAnCE. The name of the workspace depends on the build tool specified when generating the workspace. Finally, using your build tool of choice, you can build and install ACE, TAO, CIAO, and DAnCE.

A.4 XML Schema Compiler (XSC)

XSC is required when building and installing CUTS. All artifacts for XSC can be downloaded from the following location: svn://svn.dre.vanderbilt.edu/XSC/trunk. The remainder of this section discusses how to build and install XSC.

A.4.1 System Configuration

You must configure you system installing XSC. Once you have downloaded XSC from the website above please define the following environment variables:

```
%> svn co svn://svn.dre.vanderbilt.edu/XSC/trunk XSC
%> export XSC_ROOT= location of XSC
%> export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$XSC_ROOT/lib
%> export PATH=$PATH:$XSC_ROOT/bin
```

A.4.2 Installation

After you have configured your system for XSC, you are ready to build and install it. Please use the following steps to build and install XSC:

```
%> cd $XSC_ROOT
%> $ACE_ROOT/bin/mwc.pl -type [build tool] -features xsc=1,xerces3=1 XSC.mwc
```

This will generate the workspace for building and installing XSC. The name of the workspace depends on the build tool specified when generating the workspace. Finally, using your build tool of choice, you can build and install XSC.

A.5 Perl-Compatible Regular Expressions (PCRE)

PCRE is required when building and installing several projects in CUTS. You can download the source code for PCRE from the following location: http://www.pcre.org

A.5.1 Installation

Non-Windows

If you are installing PCRE on a non-Windows system, *e.g.*, Linux, Soloris, or MacOS X, then you can download the source code directly from its main website (http://www.pcre.org), build it, and install it using standard procedures.

Windows

Installing PCRE from sources on Windows is not a trivial process. It is therefore recommended that you download and use the prebuilt version of PCRE available for download from the CUTS website at the following location: http://www.dre.vanderbilt.edu/CUTS/downloads/thirdparty/pcre. Please download the correct version of PCRE that matches your version of Microsoft Visual C++ and extract it to a common location on disk.

A.5.2 System Configuration

To build CUTS correctly, the PCRE environment variables must be defined. Please define the following required environment variables:

```
%> export PCRE_ROOT= location of PCRE
%> export PCRE_VERSION= version of PCRE, blank if not applicable
%> export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PCRE_ROOT/lib
```

The configuration of PCRE is now complete. Once all the other third-party libraries are installed you will be able to build CUTS (see Chapter 1).

A.6 SQLite

SQLite is required when building and installing several projects in CUTS. You can download the source code for SQLite from the following location: http://www.sqlite.org

A.6.1 Installation

Non-Windows

If you are installing SQLite on a non-Windows system, *e.g.*, Linux, Soloris, or MacOS X, then you can download the source code directly from its main website (http://www.sqlite.org), build it, and install it using standard procedures.

Windows

Installing SQLite from sources on Windows is not a trivial process. It is therefore recommended that you download and use the prebuilt version of SQLite available for download from the CUTS website at the following location: http://www.dre.vanderbilt.edu/CUTS/downloads/thirdparty/sqlite. Please download the correct version of SQLite that matches your version of Microsoft Visual C++ and extract it to a common location on disk.

A.6.2 System Configuration

To build CUTS correctly, the SQLite environment variables must be defined. Please define the following required environment variables:

```
%> export SQLITE_ROOT= location of SQLite
%> export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$SQLITE_ROOT/lib
%> export PATH=$PATH:$SQLITE_ROOT/bin
```

The configuration of SQLite is now complete. Once all the other third-party libraries are installed you will be able to build CUTS (see Chapter 1).