5-1-2014

Introducing Modularity into Complex Software Test Suite Frameworks

Lucille Frey

Lucille H Frey
Candidate

Computer Science
Department

This thesis is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Dorian Arnold , Chairperson

Patrick Bridges

Jeff Squyres

Introducing Modularity into Complex Software Test Suite Frameworks

by

Lucille Helen Frey

B.A., Astronomy, Case Western Reserve University, 2008

THESIS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Computer Science

The University of New Mexico

Albuquerque, New Mexico

May, 2014

Acknowledgments

I would like to thank my advisors Nathan Hjelm, at Los Alamos National Lab (LANL)[1], and Dorian Arnold, at the University of New Mexico (UNM), for working with me on the content of this thesis and helping me revise my document. I would also like to thank Dorian for serving as my committee chair, and my committee members, Patrick Bridges from UNM and Jeff Squyres from Cisco Systems, Inc. I thank Lynne Jacobsen, Lourdes McKenna, Cindy Leyba and all the people involved with the UNM Interactive TV program for helping me navigate the complexities of completing a degree from Los Alamos.

[1]This work was supported by LANL under the auspices of the National Nuclear Security Administration of the U.S. Department of Energy under contract No. DEAC52-06NA25396.

Introducing Modularity into Complex Software Test Suite Frameworks

by

Lucille Helen Frey

B.A., Astronomy, Case Western Reserve University, 2008

M.S., Computer Science, University of New Mexico, 2014

Abstract

Large, complex programming projects are often constructed by developers at multiple institutions working with a variety of computer architectures and environments. For such projects a useful testing scheme must test all functionality in the software, be able to run on all relevant architectures, and provide an efficient way for all developers to view and interpret the test results. As a code project evolves, the test suite framework must be able to display new result parameters and be easily extended with additional tests including new types of tests. The complexity of current software projects has led to the development of many individualized test suite frameworks, each specifically tailored to the tests and results which currently exist for that project.

In this work, we show that a more generic, modular test suite framework simplifies the addition of new phases of testing, types of test results, and new ways to display results. Such a test suite framework also can be applied to multiple programming projects with minimal modifications. Our case study was based on the MPI Testing Tool (MTT), which was created to provide a fully automated infrastructure

to run regression and performance tests on the Open MPI code base. We have reconstructed MTT using the the principle of separation of concerns, reorganizing and rewriting the code base to allow changes to be more easily made and simplifying the connections between each part of the code. This restructuring was based around the database schema as the model, so that changes to this schema are automatically propagated through to both the test results submission process and the user interface which displays test results. We show that the modular structure of the reconstructed framework reduces the amount of code changes that must be completed to modify the test suite or display while also allowing the framework to be applied readily to new projects. We tested this version of MTT to ensure that it continues to provide the same functionality of previous versions and that new features can be added efficiently. We conducted a series of timing tests on our new code and to compare it to the previous version and confirmed that our modifications did not affect performance negatively. We have also applied the user interface code to a sample test suite to demonstrate its applicability to new projects.

Contents

List of Figures

List of Figures

List of Tables

Chapter 1

Introduction

Programming projects involving large, complex software are often collaborative projects between many developers and institutions, involving multiple architectures and software environments. As software grows in size and complexity, testing new changes for proper functionality and performance becomes more crucial. To identify bugs in a timely fashion, comprehensive regression and performance tests must be constructed and implemented in a way that tests all supported platforms and provides the test results to all the developers. This suite of tests must be large enough to capture all current functionality and be easily modifiable as the software project evolves. The test suite also must be able to be run automatically on all the relevant architectures without requiring code revisions. The test results must then be available to all the developers in an easy to access format so that bugs can be quickly identified and studied in detail.

Such large software projects present many challenges to the development of an efficient, functional test suite framework. These include:

1. **Portability:** The software may need to be run and be tested in many different hardware and software environments.

2. **Comprehensiveness:** Many tests may be required to test the software completely.

3. **Complexity:** Complex test results may be generated that require user interpretation to be useable.

4. **Time Intensive:** Running and analyzing a comprehensive test suite comprise long turn-around times.

5. **Problem Diagnosis:** Identifying and tracking regressions and performance changes require the ability to compare to past test results.

6. **Extensibility:** New tests and ways of displaying new test results are required as the software develops.

7. **Simultaneous, Real-time Access:** Users and developers are at multiple physical locations, but all want to submit and view test results in real time.

8. **Versatility**: Test suite frameworks are tailored to one software project and the particular set of tests and results desired at one point in time, and so cannot be easily applied to new projects.

Current test suite frameworks exist that address one or more of these challenges, but do not solve all of them comprehensively. For many code projects, test suites exist which are intended to be run by individual developers before new or revised code is submitted. These are not automated and frequently must be modified to run on any individual architecture. The test results are then used by that individual developer to identify and fix bugs before they are added to the trunk, but are not available to other developers unless shared individually. Other projects have a centralized testing

system where tests are run regularly on a single test machine, sometimes with these results publicly available.

None of these current approaches addresses all of the issues that arise with a large multi-developer code base. Many of the test suites cannot be run without modifications on multiple platforms. Others are not run consistently in time or as a complete comprehensive suite. The test results are not always available to all the developers, or if they are the results are not well sorted and displayed so that bugs can be easily identified. The code for the test suite framework is also not modular and easy to add to or modify and is not freely available, requiring developers for a new programming project to start from scratch.

Software test suite frameworks are developed to fit a specific purpose and to supply the results needed for developers to perform their jobs. As software projects grow the associated test suites must grow as well, but it is generally easier to add to or modify whatever patchwork of tests exists rather than treat the test suite framework as its own programming project. Even where a test suite framework is incorporated into a project from the start, it is tailored to the programming project in question rather than treated as a project in its own right. The huge variety of languages and packages which exist help prevent test suite frameworks from being applied to new projects, as developers prefer to choose the languages, database or data storage format, and user interface that they know best.

Our goal in this thesis is to construct a test suite framework that addresses all of the challenges listed here: one that can run automated regression and performance tests on multiple architectures, report the results to a central database, display the results in a usable form in real-time, be easily modifiable as the project evolves and be applicable to new projects. Our test suite framework will be constructed in a modular way using pre-existing packages where possible, so that future changes can be made quickly and efficiently. All the code for the test suite, database, and online

user interface will be available online.

Our case study is based on the MPI Testing Tool (MTT)[1], created to provide a fully automated infrastructure to run regression and performance tests on Open MPI (Message Passing Interface)[2] implementations. The pre-existing version, MTT 3.0, tests whether MPI can be successfully installed, whether test programs can be compiled and linked, whether test programs run successfully and generate valid results, and then compiles the results of those tests into a central online database. This ability to run many automated tests and sort results solves the problems of comprehensiveness, complexity and time intensiveness listed above, and its online-accessible database of results addresses problem diagnosis and partially addresses the problem of simultaneous, real-time access. The user interface is not reactive, and so the most recent test results can be missed if the user does not manually refresh the page. MTT 3.0 can also be run on a wide variety of architectures without requiring any code modifications, addressing the problem of portability. New tests can be easily added, solving the first part of the problem of extensibility, but storing and displaying new test results requires extensive code modifications and additions. MTT 3.0 is well-tailored to the tests and test results currently run and generated by the Open MPI community and could not be easily applied to a different programming project with a different suite of tests and results.

Despite all of its functionality, MTT 3.0 does not address the challenge of versatility and only partially addresses extensibility and real-time access. In this thesis we restructure and modify MTT to maintain all of the preexisting functionality and add solutions to address these problems. We use the Model-View-Controller architecture to structure MTT 4.0, restructuring the test result submission process and the UI (both views in this architecture) to receive information from the database (the model) through separate controllers. This allows model-specific code to be replaced with calls to the database, simplifying the code base. The restructured code

allows new tests and parameters to be easily added and displayed in the UI, as changes in the database are automatically propagated through to the rest of the code. This dependence on the database schema allows the code to be easily applied to new projects. The user interface has been replaced with one based on the Meteor JavaScript framework, providing a reactive interface which displays the latest results without manual page refreshes.

To demonstrate the success and impact of this project, we have performed a variety of experiments that evaluate the functionality and performance of our newly extended framework. Specifically, we ran a series of timing tests on both MTT 3.0 and MTT 4.0 to ensure that the modifications will not increase the time required to run an automated series of tests. We also compared the functionality of the user interfaces of MTT 3.0 and MTT 4.0 to test that a bug in Open MPI can be identified. We added a new test, a new type of test results and a way to display these results in the UI to MTT 4.0 and compared the amount of code and the number of code sections which needed to be edited or added to MTT 3.0. To test the ability of MTT 4.0 to be applied to a new project, we created a sample test suite which generated results data and used MTT 4.0 to store this data in a newly constructed database and display the desired results in the UI.

The results from these tests, presented in Chapter 5, demonstrate that our approach to test suite framework modularity solves all of the challenges enumerated above. This greatly simplifies the addition of new tests, phases of testing and ways of displaying data for the Open MPI project. It also provides a code base which can be applied to new programming projects in the future.

The rest of this document is organized as follows: In Chapter 2 we provide further background information for the current project and related work. In Chapter 3 we describe Open MPI and the previous version (MTT 3.0) of the test suite framework used as a case study. In Chapter 4 we describe the work performed to modify MTT

for this thesis. We evaluate the success of the project in Chapter 5, demonstrating how it addresses all the challenges described in the introduction, and describe its impact. We provide a summary and a discussion of the limitations of the current project and an outline of future work in Chapter 6.

Chapter 2

Background

2.1 Domain

Large software projects are frequently developed by many programmers at multiple locations. Complex codes with large code bases require similarly complex test suites, to identify bugs and performance issues. Useful test suites must be runnable on all the relevant architectures without requiring modifications which might invalidate comparisons. Developers who run an automated test suite can quickly identify regressions and performance issues resulting from any of their modifications, and if the results can be shared can also see the effects of code written by other developers. A database of past results is also required so that developers can identify when any regression or performance cut was introduced.

Comprehensive test suites for large software projects produce large amounts of output, making it difficult for developers to sort through for the specific results they require. Well constructed user interfaces can filter and sort the results so that only the most frequently used are presented at first, and allows users to drill down to more detailed data when desired.

As the software evolves developers must add new tests to the suite, sometimes producing different types of results: performance data versus pass/fail results, for instance. A good test suite framework will streamline the process of adding new tests so that minimal changes are required to add a test and add the relevant results to the database and user interface. Choosing the right software model for the test suite framework can simplify the addition of tests and encourage the addition of new useful tests.

2.1.1 Software Models

A test suite framework which includes all of the functionality described in the previous section will be a complex system, including the source code to be tested as well as code for the tests, to run the tests, to copy test results to a central database, store these results, and display them to the user. Projects of this complexity are regularly encountered in software development, where the programmer must connect a user interface with some sort of database or data processing capability. If each part of a project is written independently, then adding or removing a feature from the UI will also require changing the code for the data storing and processing capabilities.

There are many different software models which have been developed to solve this type of problem, providing different ways of abstracting data, analysis and display. One of these, Model-View-Controller (MVC)[3], was developed in order to match the user's mental model of a set of data and allow them to easily edit and view this data. It was originally developed with the Smalltalk language [3][4], but has since been implemented with many different languages[5]. It has been used to simplify description and coding of many different problems, especially web applications [6][7][8][9][10]. MVC has also been applied to many applications including computer telephony applications [11], automatic generation of visualizations and GUIs [12] and adaptable

simulation software [13].

For test suite frameworks, the Model-View-Controller architecture is a good match to the flow of data. Tests are run and may generate large amounts of output, not all of which may be interesting to the user. The data of interest must somehow be specified and then stored in a database so that results can be viewed over time and regressions identified. Similarly, the exact data which a user wishes to view through the UI must be specified so that data can be read from the database and displayed. With the Model-View-Controller architecture, a single model (the database schema) can be linked to multiple views (the test submission code and the UI) by individual controllers which communicate data about the model to the views.

The essential idea of MVC is to separate the problem into three completely separate components, the model, the view and the controller, that only interact in specified ways, usually implementing each as an object. The model is an object or a structure made up of objects that represents the information stored by the system. This model should match the user's perception of the information in a logical way. The view represents the model, usually visually as a GUI or other UI. It receives information from the model and filters it to highlight the desired data. The view can send messages to the model to update it. The controller links the user to the system, receiving user input entered through the view and translating this into the appropriate commands to change the model. The controller also converts any changes in the model into commands to update the view [14]. Complex programs can be constructed out of multiple MVC components, with each controller communicating with its own model and view as well as the controllers associated with other model-view pairs[15]. Many other similar architecture models exist which divide responsibilities between components in slightly different ways. For example, the Presentation-Abstraction-Control model is very similar to hierarchical MVC, but the presentation component (similar to view) does not perform any communication

and can only receive and display data [14][16].

As an architecture for a complex test suite framework, the MVC model can provide an easily modifiable structure. With an MVC architecture, a developer will not need to edit the view to display different kinds of data, since any changes to the model are automatically propagated through to the view by the controller. The MVC architecture allows a localization of control that, when used in the context of a test suite framework, simplifies the necessary work to add new functionality.

2.2 Related Work

Recall the challenges from the introduction, namely: portability, comprehensiveness, complexity, time intensity, problem diagnosis, extensibility, simultaneous real-time access and versatility. Current test suite frameworks exist that address one or more of these challenges, but do not comprehensively solve all of them. For many code projects, test suites exist that are intended to be run by individual developers before new or revised code is submitted. Generally, these are not automated and frequently must be modified to run on any individual architecture. The test results are then used by that individual developer to identify and fix bugs before they are added to trunk, but are not available to other developers unless shared individually. Other projects have a centralized testing system where tests are run regularly on a single test machine, sometimes with these results publicly available. To evaluate the completeness of currently available codes, from our list of challenges we have created a taxonomy of important test suite framework features (see Table 2.2). These include whether the code is modular, whether each part of the source code is publicly available, and how the test results are stored and shared. Through literature and public web searches, we compiled a list of all the code testing projects which meet all of the following criteria:

- code development and use is performed at multiple locations
- test suite results are used by code developers
- the test suite can be automated
- some part of the test suite framework code or a description of it is available

Table 2.1: A summary of current code testing architectures, including the test case described in this thesis (MTT). We evaluate whether the code is modular and whether the source code for the testing, result submission, database of results or user interface is publicly available. Note that the number of developers is for the programming project being tested (i.e. Open MPI), not for the testing suite itself (i.e. MTT).

	Name	Firebug	CP2K	Lustre	PVFS	Mozilla	SDCC	MTT 3.0	MTT 4.0
Project details	Developers	25	∼12	∼100	∼10	>3000	40	>20	>20
	Modular	✗	✗	✗	✗	✗	✗	✗	✓
	Extensible	✗	✗	✗	✗	✗	✗	✗	✓
Code available	Testing	✓	✓	✓	✓	✓	✓	✓	✓
	Submit	✗	✗	✗	✗	✗	✗	✓	✓
	DB	✗	✗	✗	✗	✓	✗	✓	✓
	UI	✗	✗	✗	✗	✗	✗	✓	✓
Results storage and access	DB	✓	✗	✗	✗	✓	✓	✓	✓
	Online	✓	✓	✗	✓	✓	✓	✓	✓
	All results	✓	✓	✗	✗	✗	✓	✓	✓
	Organized	✗	✗	✗	✗	✓	✗	✓	✓
	Sortable	✗	✗	✗	✗	✗	✗	✓	✓
	Simultaneous	✓	✓	✗	✓	✓	✓	✓	✓
	Real-time	✗	✗	✗	✗	✗	✗	✗	✓

Many large software projects have distributed development, requiring some sort of centralized testing structure to allow for effective regression and performance testing. For many projects (for example, Mozilla, Lustre, Firebug, CP2K), the responsibility for testing rests on each developer. They are expected to run regression and sometimes performance tests before submitting any changes to the central repository. Even if each developer carefully runs up-to-date regression and performance tests before submitting any change, they have no way of knowing whether their revised code will behave identically on all the other systems. After a change is submitted, it could be discovered that what runs perfectly on system A shows a bug on system B. In addition to this developer-based testing system, some projects have a centralized automated testing system which posts the results online for developers to view (Firebug, CP2K, PVFS), but these are only run on a small set of test machines and the results are generally long and difficult to sift through.

Firebug is a debugging and editing tool for websites, supporting languages such as CSS, HTML and JavaScript [17]. The current development team consists of 25 people, although most of the development is performed by a small subset of these. There is a test suite which can be run by any developer, and the results are submitted to a database. The online UI for this database simply lists the raw results of each test in text form along with the number of failures which occurred. The test results are not indexed or sortable in any way.

CP2K is a freely-available program that performs atomistic and molecular simulations of solid state, liquid, molecular and biological systems, run in parallel [18]. It currently has about a dozen developers from multiple institutions. It has an automated tester which builds and tests CP2K after each new commit, running correctness tests on a single machine with one compiler. The raw data from the last run of these tests is available online in text format.

PVFS (Parallel Virtual File System) provides high performance I/O for parallel systems, supporting large datasets and high access rates [19]. The development team contains members from universities and industry groups. Nightly tests are run on the current code version on dedicated test machines. The tests have been tailored to run on the test machines, and it is the responsibility of the developers to run tests on their own machines, modifying the tests as necessary, before submitting changes. A brief summary of the latest test results is available online, consisting of pass/fail information for the build and test of 8 packages.

Lustre is another parallel, distributed file system developed for Linux-based cluster computing [20]. In addition to a core team of developers outside programmers can contribute patches. While these patches are reviewed before being added to the code, developers are responsible for performing their own tests before submission. There is an automated test suite which runs when new changes are checked in, but the test results are not freely available.

Mozilla has developed a python performance testing framework, Talos[21], which can be used on Windows, Mac and Linux operating systems. Test results can be pushed to a Graph Server, where they are accessible online, and regressions are found by comparing each push to both the previous 12 and subsequent 12 pushes.

SDCC (Small Device C Compiler) is a retargettable table, optimizing ANSI-C compiler suite targeting specific Intel microprocessors[22]. It has been developed by an international team of 40 programmers. Regression tests are automatically performed daily on each version currently in development with the raw results of the latest run available in text format online.

In Table 2.2, we compare the functionality currently available in each of these codes. We provide information about the number of developers involved in the software project being tested, and whether each test suite framework is modular or

extensible to new projects. We checked whether source code for each of four stages in the test suite framework (running tests, submitting test results to a database, storing results in a database and the user interface) is publicly available. We also studied how the test results are made available to developers, including whether they are stored in a database (allowing checks for regressions and performance issues), available online and if this online access is available simultaneously to all developers and in real-time. For test suite frameworks with results available online, we also checked whether all test results are available or if just the latest results or a small subset is presented, and whether the results are presented in some organized manner and if they can be further sorted and filtered by the user.

These codes were chosen because all of them have distributed development involving many individual programmers, and have the ability to conduct automated testing and have some part of their code publicly available. As described above, however, not all of these codes run automated tests on all the systems used for development, or run tests at regular time intervals. The only code constructed in a modular fashion to ease further development is the MTT code written for the current project. Despite some of these projects providing useful functionality, since most of the source code is not available it cannot be applied to other projects. All but one of the projects, Lustre, provide some sort of online UI with test results. These test results are presented as a brief summary of pass/fail results (PVFS, Firebug, Mozilla) and/or the raw text dump containing all the test output (Firebug, CP2K, Mozilla, SDCC). Two of these projects, Firebug and Mozilla, store past test results in a database but only Mozilla uses this data to identify regressions.

Chapter 3

Test Case: MPI Testing Tool

Our case study is the MPI Testing Tool (MTT)[1], developed to test the Open MPI code. Open MPI was initially created by the merger of MPI implementations from the University of Tennessee, Los Alamos National Laboratory (LANL), Indiana University and the University of Stuttgart. At LANL, Open MPI is preferred to vendor-supported codes because of the ability to have developers identify bugs, create a solution and immediately add that fix into the code base. The size, complexity and distributed development of the Open MPI project provides a good demonstration of the difficulties involved in implementing and using the results of test suites. Here we describe the Open MPI code and the pre-existing version of MTT, 3.0. In Chapter 4 we detail the new version we developed, MTT 4.0.

3.1 Open MPI

Open MPI[2] is an open source implementation of the MPI (Message Passing Interface) standards, compliant up to MPI-2.2 as of Open MPI version 1.7.3. It is a message-passing system designed to be used on many different parallel architectures.

The Open MPI project has over 30 organizations as members, contributors and partners, including universities, government labs and software and computing companies. Open MPI is constantly being developed and expanded, requiring a comprehensive testing tool which can be used by all of the partners involved. Stable major releases are produced approximately every two years along with new feature series as well as about five minor releases each year. Nightly snapshots of the development series are also produced and used for testing.

3.2 MPI Testing Tool (MTT)

The Open MPI project requires testing and reporting which have not previously been implemented by other code projects. Developers use multiple compilers, architectures, schedulers and environments and need to know whether the changes they make will fail when run on other systems. This requires that the test suite be highly adaptable and that the test results are immediately and freely available in an easy to interpret format. To be efficient and easy to use, one run of the test suite must be able to perform multiple compilations with varying compilers and options and then run a specified set of tests, again with varying options.

The MPI Testing Tool (MTT) provides a fully automated infrastructure to run regression and performance tests on MPI implementations (see Figure 3.1 for a flow chart). It can be run on many different machines with different environments and combine the results into a central database. MTT tests whether MPI can be successfully installed, whether test programs can be compiled and linked, and whether test programs run successfully and generate valid results. It is designed for use with Open MPI, but will work with any MPI implementation. It was specifically designed to incorporate previously existing tests, so that adding additional tests is simple. The separation between running MTT and submitting results allows it to be run behind

firewalls. MTT can automatically test a given MPI version with a wide variety of different parameters, such as compilers, compiler options, mpirun or mpiexec options or environment variables.

The modular format of MTT input files allows many options to be easily specified. A single run of MTT, run on one machine, takes a list of MPI versions (typically tarballs containing the latest nightly snapshots from the development heads), a list of compilers and environment variables, and a list of tests. It compiles each MPI version using each compiler, and runs each of the tests. The results from all of these steps are then submitted to an online database that is connected to a UI from which results from all institutions running MTT are displayed (http://mtt.open-mpi.org). This UI can be used by developers to identify under what conditions (if any) their additions or modifications have broken the code, and in which part of the code the errors originated (see Figures 3.2, 3.3, 3.4 and 3.5 for screencaps of this UI).

MTT 3.0 is specifically tailored to the project and data involved. The code for submitting test results to the database and displaying database results in the user interface is hard-coded for the specific tests and results desired. Displaying additional results, such as performance data, or adding a new type of test requires modifying many parts of the code: the code to run tests, submit results to the database, store results in the database and display data in the user interface (see the bold-outlined steps in Figure 3.1 for a graphical representation).

Referring back to the challenges listed in the introduction, MTT 3.0 does address some of them, specifically portability, comprehensiveness, complexity, time intensity and problem diagnosis and partly addressing simultaneous real-time access. Test results are accessible in the database as soon as they are submitted from any location, but the UI is only updated when the user manually refreshes the page. As described above, displaying new types of data or adding new types of tests requires extensive modifications to the code, discouraging useful additions. The single-purpose nature

of this code, closely linked to the Open MPI project, also would make it very difficult to apply any part of the MTT 3.0 code to a different software project despite it being publicly available.

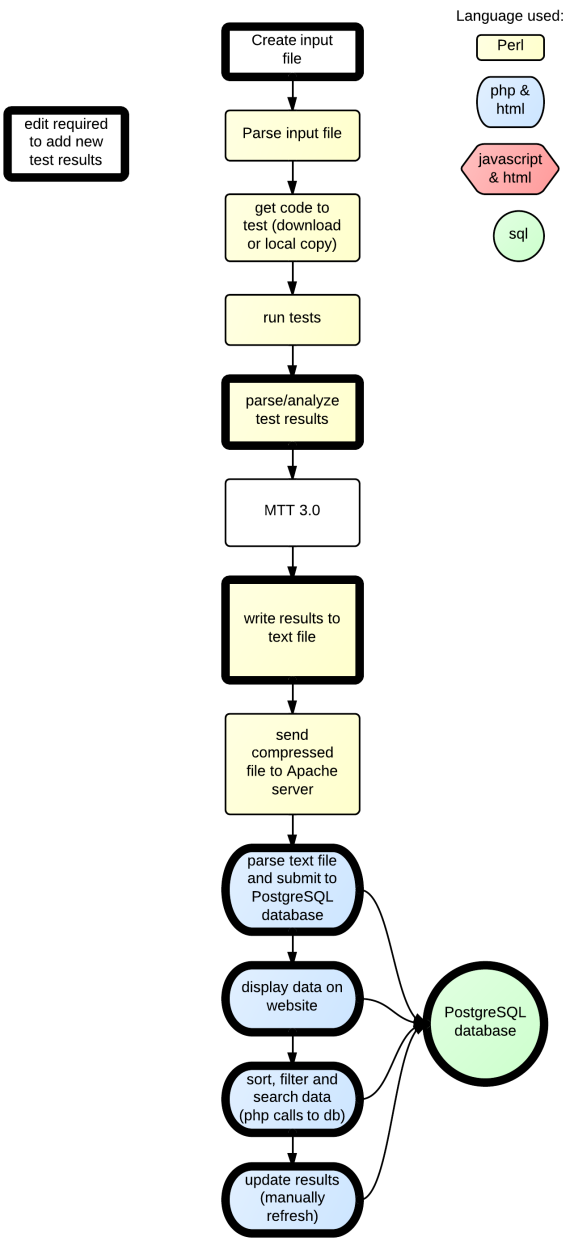
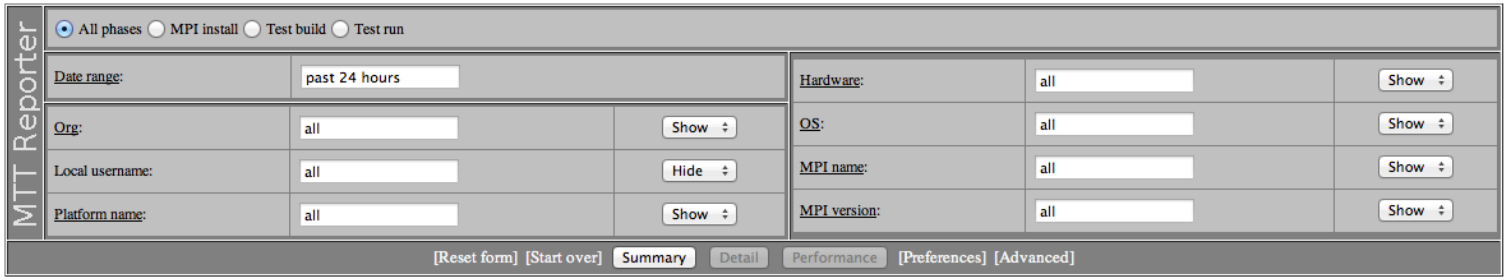


Figure 3.1: Flow chart for MTT 3.0



Current time (GMT): 2014-02-07 17:47:18
Date range (GMT): 2014-02-06 17:47:18 - 2014-02-07 17:47:18
Phase(s): MPI install, Test build, and Test run *(Via Summary)*
Number of rows: 13

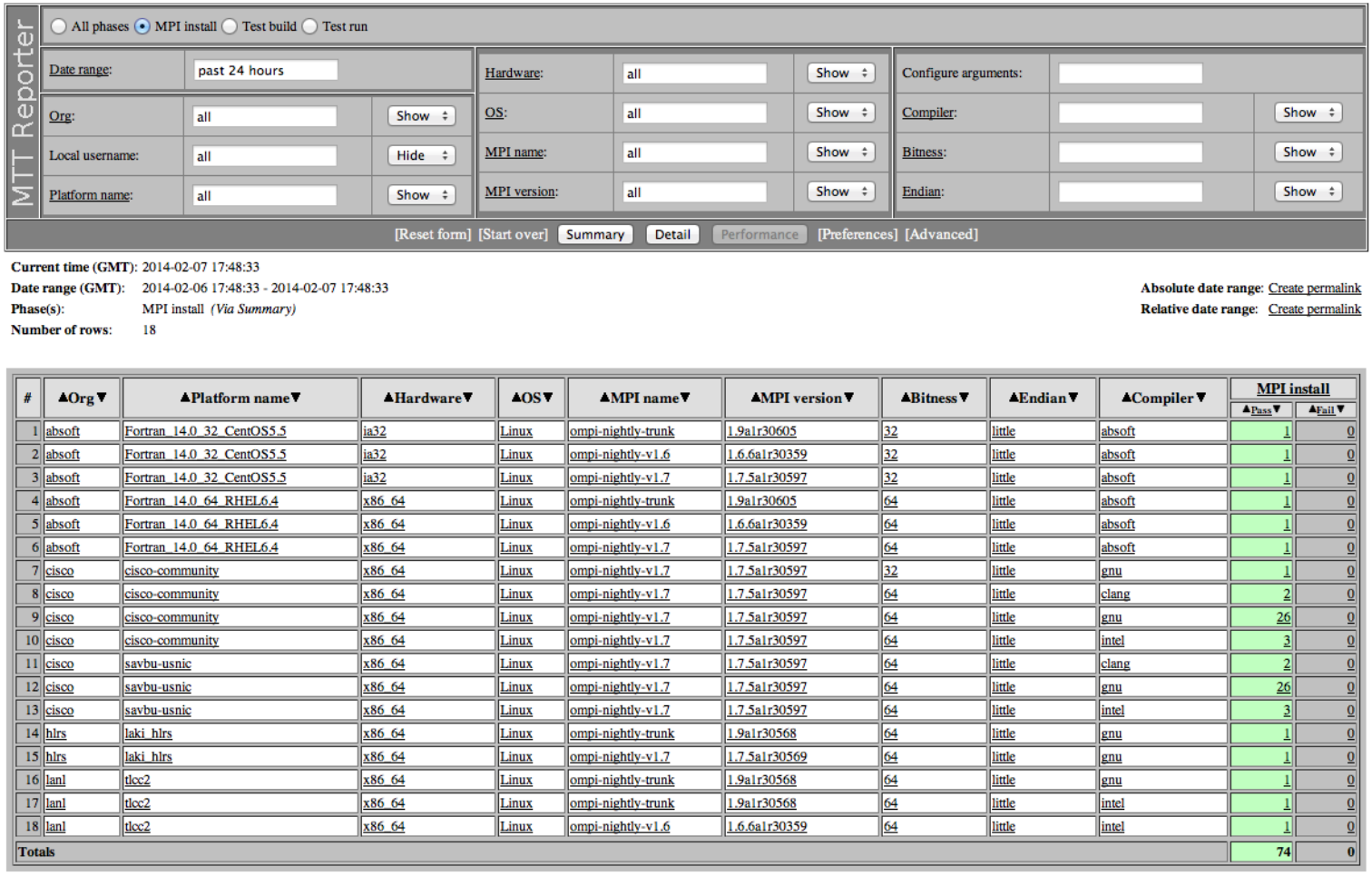
Absolute date range: Create permalink
Relative date range: Create permalink

#	▲Org▼	▲Platform name▼	▲Hardware▼	▲OS▼	▲MPI name▼	▲MPI version▼	MPI install		Test build		Test run				
							▲Pass▼	▲Fail▼	▲Pass▼	▲Fail▼	▲Pass▼	▲Fail▼	▲Skip▼	▲Timed▼	▲Perf▼
1	absoft	Fortran_14.0_32_CentOS5.5	ia32	Linux	ompi-nightly-trunk	1.9a1r30605	1	0	1	0	0	24	0	0	0
2	absoft	Fortran_14.0_32_CentOS5.5	ia32	Linux	ompi-nightly-v1.6	1.6.6a1r30359	1	0	1	0	24	0	0	0	0
3	absoft	Fortran_14.0_32_CentOS5.5	ia32	Linux	ompi-nightly-v1.7	1.7.5a1r30597	1	0	1	0	30	0	0	0	0
4	absoft	Fortran_14.0_64_RHEL6.4	x86_64	Linux	ompi-nightly-trunk	1.9a1r30605	1	0	1	0	24	0	0	0	0
5	absoft	Fortran_14.0_64_RHEL6.4	x86_64	Linux	ompi-nightly-v1.6	1.6.6a1r30359	1	0	1	0	24	0	0	0	0
6	absoft	Fortran_14.0_64_RHEL6.4	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	1	0	1	0	30	0	0	0	0
7	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30569	0	0	0	0	1521	14	30	6	0
8	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	32	0	287	0	79474	233	2040	248	160
9	cisco	savbu-usnic	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	31	0	278	0	30344	239	774	85	62
10	hlrs	laki_hlrs	x86_64	Linux	ompi-nightly-trunk	1.9a1r30568	1	0	1	0	0	1	0	0	0
11	hlrs	laki_hlrs	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30569	1	0	1	0	1	0	0	0	0
12	lanl	tlcc2	x86_64	Linux	ompi-nightly-trunk	1.9a1r30568	2	0	9	1	566	2	0	2	24
13	lanl	tlcc2	x86_64	Linux	ompi-nightly-v1.6	1.6.6a1r30359	1	0	9	1	0	574	0	0	0
Totals							**74**	**0**	**591**	**2**	**112038**	**1087**	**2844**	**341**	**246**

Time: 0.312 sec. (PHP: 0.283 / SQL: 0.029)

Overall MTT contribution graph (updated nightly): All Time or 1 Year Window

Figure 3.2: MTT 3.0 UI with results for all phases



#	▲Org▼	▲Platform name▼	▲Hardware▼	▲OS▼	▲MPI name▼	▲MPI version▼	▲Bitness▼	▲Endian▼	▲Compiler▼	MPI install	
										▲Pass▼	▲Fail▼
1	absoft	Fortran_14.0_32_CentOS5.5	ia32	Linux	ompi-nightly-trunk	1.9a1r30605	32	little	absoft	1	0
2	absoft	Fortran_14.0_32_CentOS5.5	ia32	Linux	ompi-nightly-v1.6	1.6.6a1r30359	32	little	absoft	1	0
3	absoft	Fortran_14.0_32_CentOS5.5	ia32	Linux	ompi-nightly-v1.7	1.7.5a1r30597	32	little	absoft	1	0
4	absoft	Fortran_14.0_64_RHEL6.4	x86_64	Linux	ompi-nightly-trunk	1.9a1r30605	64	little	absoft	1	0
5	absoft	Fortran_14.0_64_RHEL6.4	x86_64	Linux	ompi-nightly-v1.6	1.6.6a1r30359	64	little	absoft	1	0
6	absoft	Fortran_14.0_64_RHEL6.4	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	64	little	absoft	1	0
7	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	32	little	gnu	1	0
8	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	64	little	clang	2	0
9	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	64	little	gnu	26	0
10	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	64	little	intel	3	0
11	cisco	savbu-usnic	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	64	little	clang	2	0
12	cisco	savbu-usnic	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	64	little	gnu	26	0
13	cisco	savbu-usnic	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	64	little	intel	3	0
14	hlrs	laki_hlrs	x86_64	Linux	ompi-nightly-trunk	1.9a1r30568	64	little	gnu	1	0
15	hlrs	laki_hlrs	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30569	64	little	gnu	1	0
16	lanl	tlcc2	x86_64	Linux	ompi-nightly-trunk	1.9a1r30568	64	little	gnu	1	0
17	lanl	tlcc2	x86_64	Linux	ompi-nightly-trunk	1.9a1r30568	64	little	intel	1	0
18	lanl	tlcc2	x86_64	Linux	ompi-nightly-v1.6	1.6.6a1r30359	64	little	intel	1	0
Totals										**74**	**0**

Figure 3.3: MTT 3.0 UI with results for MPI Install

MTT Reporter

All phases | MPI install | Test build | Test run

Date range:	past 24 hours		Hardware:	all	Show	Suite:	all	Show
Org:	all	Show	OS:	all	Show	Compiler:		Show
Local username:	all	Hide	MPI name:	all	Show	Compiler version:	all	Show
Platform name:	all	Show	MPI version:	all	Show	Bitness:		Show

[Reset form] [Start over] Summary Detail Performance [Preferences] [Advanced]

Current time (GMT): 2014-02-07 17:48:44
Date range (GMT): 2014-02-06 17:48:44 - 2014-02-07 17:48:44
Phase(s): Test build *(Via Summary)*
Number of rows: 163

Absolute date range: Create permalink
Relative date range: Create permalink

1 2 ▶

#	▲Org▼	▲Platform name▼	▲Hardware▼	▲OS▼	▲MPI name▼	▲MPI version▼	▲Bitness▼	▲Compiler▼	▲Compiler version▼	▲Suite▼	Test build ▲Pass▼	Test build ▲Fail▼
1	absoft	Fortran_14.0_32_CentOS5.5	ia32	Linux	ompi-nightly-trunk	1.9a1r30605	32	absoft	GCC: 4.1.2, Absoft: Absoft 32-bit Pro Fortran 14.0.2	trivial	1	0
2	absoft	Fortran_14.0_32_CentOS5.5	ia32	Linux	ompi-nightly-v1.6	1.6.6a1r30359	32	absoft	GCC: 4.1.2, Absoft: Absoft 32-bit Pro Fortran 14.0.2	trivial	1	0
3	absoft	Fortran_14.0_32_CentOS5.5	ia32	Linux	ompi-nightly-v1.7	1.7.5a1r30597	32	absoft	GCC: 4.1.2, Absoft: Absoft 32-bit Pro Fortran 14.0.2	trivial	1	0
4	absoft	Fortran_14.0_64_RHEL6.4	x86_64	Linux	ompi-nightly-trunk	1.9a1r30605	64	absoft	GCC: 4.4.7, Absoft: Absoft 64-bit Pro Fortran 14.0.2	trivial	1	0
5	absoft	Fortran_14.0_64_RHEL6.4	x86_64	Linux	ompi-nightly-v1.6	1.6.6a1r30359	64	absoft	GCC: 4.4.7, Absoft: Absoft 64-bit Pro Fortran 14.0.2	trivial	1	0
6	absoft	Fortran_14.0_64_RHEL6.4	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	64	absoft	GCC: 4.4.7, Absoft: Absoft 64-bit Pro Fortran 14.0.2	trivial	1	0
7	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	32	gnu	4.8.2	ibm	1	0
8	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	32	gnu	4.8.2	imb-check	1	0
9	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	32	gnu	4.8.2	imb-general	1	0
10	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	32	gnu	4.8.2	intel	1	0
11	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	32	gnu	4.8.2	mpicxx	1	0
12	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	32	gnu	4.8.2	mpi-test-suite	1	0
13	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	32	gnu	4.8.2	netpipe	1	0
14	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	32	gnu	4.8.2	onesided	1	0
15	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	32	gnu	4.8.2	trivial	1	0
16	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	64	clang	3.1	ibm	1	0
17	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	64	clang	3.1	imb-check	1	0
18	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	64	clang	3.1	imb-general	1	0

Figure 3.4: MTT 3.0 UI with results for test builds

MTT Reporter

All phases | MPI install | Test build | Test run

Date range:	past 24 hours		Hardware:	all	Show	Suite:	all	Show
Org:	all	Show	OS:	all	Show	Test:	all	Hide
Local username:	all	Hide	MPI name:	all	Show	np:	all	Show
Platform name:	all	Show	MPI version:	all	Show	Command:		Show

[Reset form] [Start over] Summary Detail Performance [Preferences] [Advanced]

Current time (GMT): 2014-02-07 17:49:15
Date range (GMT): 2014-02-06 17:49:15 - 2014-02-07 17:49:15
Phase(s): Test run *(Via Summary)*
Number of rows: 69

Absolute date range: Create permalink
Relative date range: Create permalink

#	▲Org▼	▲Platform name▼	▲Hardware▼	▲OS▼	▲MPI name▼	▲MPI version▼	▲Suite▼	▲np▼	Test run				
									▲Pass▼	▲Fail▼	▲Skip▼	▲Timed▼	▲Perf▼
1	absoft	Fortran_14.0_32_CentOS5.5	ia32	Linux	ompi-nightly-trunk	1.9a1r30605	trivial	4	0	24	0	0	0
2	absoft	Fortran_14.0_32_CentOS5.5	ia32	Linux	ompi-nightly-v1.6	1.6.6a1r30359	trivial	4	24	0	0	0	0
3	absoft	Fortran_14.0_32_CentOS5.5	ia32	Linux	ompi-nightly-v1.7	1.7.5a1r30597	trivial	4	30	0	0	0	0
4	absoft	Fortran_14.0_64_RHEL6.4	x86_64	Linux	ompi-nightly-trunk	1.9a1r30605	trivial	4	24	0	0	0	0
5	absoft	Fortran_14.0_64_RHEL6.4	x86_64	Linux	ompi-nightly-v1.6	1.6.6a1r30359	trivial	4	24	0	0	0	0
6	absoft	Fortran_14.0_64_RHEL6.4	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	trivial	4	30	0	0	0	0
7	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30569	ibm	1	15	0	0	0	0
8	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30569	ibm	3	30	0	0	0	0
9	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30569	ibm	8	1476	14	30	6	0
10	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	ibm	1	145	0	0	160	0
11	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	ibm	3	320	0	0	0	0
12	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	ibm	8	49819	233	815	88	0
13	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	intel	8	22425	0	1220	0	0
14	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	mpicxx	8	310	0	0	0	0
15	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	mpi-test-suite	8	160	0	0	0	0
16	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	netpipe	2	160	0	0	0	160
17	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	onesided	8	4895	0	5	0	0
18	cisco	cisco-community	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	trivial	8	1240	0	0	0	0
19	cisco	savbu-usnic	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	ibm	1	56	31	0	31	0
20	cisco	savbu-usnic	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	ibm	3	62	62	0	0	0
21	cisco	savbu-usnic	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	ibm	32	19254	116	316	54	0
22	cisco	savbu-usnic	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	intel	32	8382	0	456	0	0
23	cisco	savbu-usnic	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	mpicxx	32	90	30	0	0	0
24	cisco	savbu-usnic	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	mpi-test-suite	32	62	0	0	0	0
25	cisco	savbu-usnic	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	netpipe	2	62	0	0	0	62
26	cisco	savbu-usnic	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	onesided	32	1896	0	2	0	0
27	cisco	savbu-usnic	x86_64	Linux	ompi-nightly-v1.7	1.7.5a1r30597	trivial	32	480	0	0	0	0

Figure 3.5: MTT 3.0 UI with results for test runs

Chapter 4

Reconstructing MTT

For this thesis, we produced a new version of MTT to address the testing challenges described in the previous section. We used the Model-View-Controller architecture model to structure the code, allowing changes to the database schema to be automatically propagated through to both the user interface and test results submission code. This simplifies the process of adding additional tests or testing phases to the code, as well as allowing large parts of MTT to be applied to different programming projects.

4.1 Concepts

The primary goal of this project is to simplify the structure of the test suite, results database and user interface so that changes and additions can be easily made. To aid this goal, we have used previously existing packages where possible, to make understanding the code easier and to take advantage of future external developments. We have also restructured the code, including the results submission process, the database and the user interface, to follow the modular approach.

Many different parts of computer science have embraced the idea of modularity, or separation of concerns. This design principle advocates splitting a program into separate parts, each of which solves a distinct subproblem. This can be applied on small scales, with object orientation and classes, or on larger scales with architectural patterns or separating software from hardware. Modularity requires that some time be spent on planning the structure of a software package before it is implemented, so that the code can be well structured. Separating out various functionalities allows small changes to be made without requiring large portions of the code, and simplifies testing procedures for newly added code. Having many well-defined parts makes it easier to divide tasks between developers so that multiple people can work on the same software at once. If each individual part of a problem is solved separately, then depending on the project some of these parts can be reused in the future for another project.

Applying a structured, modular architecture to a software project requires that the specifications for a project are defined before the code is written. In the case of software projects which start out small and focused and gradually evolve to be much larger and more comprehensive, this is generally not the case. With test suites, what starts out as a few simple tests written by one developer to catch bugs in a small chunk of code can develop into a large comprehensive test suite framework, without any official design process being conducted. This results in code which is functional but not easily comprehensible or changeable.

4.2 Implementation

The code in MTT 3.0 that governs the reporting of test results to the database and the sorting and displaying of these results by the user interface is not easily separable and contains many hard-coded references to the current set of test results

and database schema. Changes to the schema for the database require changes to both the reporter code and the UI code as well. In our revised MTT 4.0 the database schema is used as the central model, with both the results submission process and the UI referencing the database.

We have reconstructed the MTT code base using the MVC model (see Figure 4.1 for a flow chart). We simplified the process by submitting test results to the database directly from the code used to run tests (written in Perl), replacing the previous process of using Perl to write test results to a text file and then using php to parse that text file and submit results to the database. In the revised version, both the UI and the test code receive information from the schema so that changes to the schema are automatically propagated through. This new modular design makes the process of adding new test suites more efficient. The modular design will also allow this code to be applied to other testing projects.

4.2.1 Reconstructing the results submission process

MTT 3.0 has a well designed and comprehensive schema for its database which includes all the data produced by the test runs that developers would then want to view on the website (see Figure A.1). This schema was not fully implemented in the code, and some of the variables were either never set or never viewable by the user. Also, since these parameters are hard-coded in the code to submit test results, any changes or additions require changes to both the database schema and to the results submission code. In MTT 4.0, we simplified this by directly submitting test results from the test running code, written in Perl. Directly accessing the database at this point allows the results submission code to query the database to determine which data to submit. There are several requirements imposed on the database schema so that this process can work. There must be a `phase` table containing

a row of data for each phase of the testing process. Each row matches the name of the phase in the MTT input file to the name of the table in the database which should receive results from that phase. A table must also exist, then, for each of these phases with column names matching the parameters generated by MTT. When these requirements are met, as MTT completes each phase it first queries the `phase` table to find the corresponding table for results, and then queries that results table to determine the correct parameters to submit to the database. Any additional phases or parameters added to the database will then be automatically incorporated into the results submission phase without requiring any code changes.

This method of directly submitting to the database, without an intermediate text file step, was used as a proof of concept and as a way to explore how to make the process more efficient. For the production system, with the database server accessed remotely by individual developers possibly running behind firewalls, this process will need to be further studied and modified. We maintained an option of writing a text file, either for debugging purposes or for use in a modified submission scheme in the future (see section 6.2).

4.2.2 Reconstructing the database

To enable the use of Meteor, a preexisting javascript package that connects the database and the UI (described in section 4.2.4), we migrated the database from PostgreSQL to MySQL (see Figure A.2 for the revised schema). The type of data stored in the database was not changed, instead being restructured. The PostgreSQL database used in MTT 3.0 makes use of the `INHERITS` functionality to group common variables together and share them between tables. It is possible to replicate this in MySQL with foreign keys and triggers, but we found that instead storing all data for a phase in a single table speeds up and simplifies data calls. As described in section

6.2, the structure of this database will be further evaluated and changed before it is moved onto production systems. The database for MTT 3.0 was optimized for searching using the concept of database normalization, since sorting and filtering of data in the UI was performed with direct calls to the database. Meteor provides a different way of accessing data and filters and sorts with javascript rather than PHP/SQL calls, so the performance of the UI depends on the database structure in different ways and will require further testing.

In addition to this reorganization, three new tables were created, two to allow the results submission code and the UI to receive information about the database and a third to store standard values for tests. The modular structure of MTT 4.0 requires the existence of these first two tables, shown with bold outlines in Figure A.2. The `phase` table described in section 4.2 is accessed by the results submission code, finding the MySQL table which corresponds to each phase and the types of data (column names) which should be submitted to that table. The UI accesses the `phase` table as well, to determine which phases to display as tables by accessing the `display` boolean. If display is true for a table, then that table will be displayed in the UI. For each of these displayed tables, the columns to be displayed in summary or detail mode are determined by entries in the `display_data` table. Views are created (see Figure A.3) which filter the columns of each table listed in `phase` with the entries in `display_data`, and so contain only the data which should be displayed in a single table. With the current version of Meteor (described in the next section) views cannot be accessed, and so instead new tables were created with the desired filtering and are updated with triggers every time data is added to the database. When the user interacts with the UI to sort and filter the displayed data, this is done entirely on the client side with javascript and html. These actions do not change the values stored in the database defining default display options. As described in section 6.2, the ability for users to define their own default options can be implemented in the future.

The `standards` table was introduced to store standard values for performance tests where a test result must be compared against the standard to decide whether the test passed or failed. The test result parameter it will be compared against, the architecture, compiler, Open MPI version, and any other relevant parameters can be specified for each standard value. When a new performance test result is added to the database, a trigger checks whether any standard values are associated with that test and if so performs a check against that value. The result of this check is used to set the pass or fail values for that test. The structure of the `standards` table and details of the checks performed will be extended as additional performance tests are added to the framework.

4.2.3 Meteor.js

Meteor is a node.js-based open source JavaScript framework designed to support reactive web applications using a single language [23]. It contains a library of packages which provide basic functionality such as HTTP connections and HTML templates, as well as many other official or user-provided packages providing functionality such as email, passwords and accounts, or MySQL support. Meteor also contains a command-line build tool which combines all the JavaScript, HTML and CSS for one project with the required packages and produces a single standalone application bundle. For this thesis, we use meteor to provide the connection between the database and user interface and automatically update the UI in real-time when changes are made to the database.

The current release of Meteor natively only supports MongoDB as a data storage format. We have used a package to enable MySQL support which was written for a previous release and does not provide full functionality. The planned Meteor 1.0 release will provide MySQL support and enable views in the database to be accessed

by Meteor in the same way as tables. When this functionality is available, views constructed in the database which filter each data table using the parameters in the `display_data` table will be accessed directly. The current version of MTT 4.0 contains a work-around. Instead of views, new tables were constructed for each desired display with columns selected based on the values in `display_data`. We then added triggers to the database so that every time data is inserted into one of the tables in `phase` the relevant columns of data are also added to these new display tables.

4.2.4 Reconstructing the UI

Test results in the MTT database must be accessible in some manipulable format to developers in real-time so that they can check the effects of any new code development on test results and performance. In MTT 3.0 the code for this UI was closely tailored to the currently existing set of test results, so that any additions or changes require edits to all parts of the code. Results are received and displayed in the UI by php-mediated calls to the database which are performed every time the user refreshes the page or selects a different option. The most recent results can be missed if the user does not refresh the page frequently. We have reconstructed this part of MTT 4.0 using the Meteor package to provide a reactive interface which interacts with the database to determine which results to display rather than requiring hard-coded tables. This functionality allows us to remove schema-specific data from the UI code, instead querying the database to determine which tables and columns of data to display.

The data displayed in the UI is formatted using the DataTables JavaScript package[24] which provides sorting and filtering functionality as well as the ability to switch between tables for multiple testing phases and display more detailed data all

in one web page. Using this packages rearranges where filtering and sorting is accomplished. Previously all filtering and sorting of data was done with php/sql queries to the database. In our revised code, the UI receives all the data to be displayed at once (and not any other data) and performs any further sorting and filtering with JavaScript commands. Meteor automatically updates the data displayed by the UI whenever the database is changed, so no additional calls are necessary. With these modifications adding an additional testing phase (such as performance testing) or parameter only requires changing the related data table and `display_data` entries in the database, not any of the UI code. Figure 4.2, containing flow diagrams of MTT 3.0 and MTT 4.0, illustrates how we have reduced the number of steps which must be edited to add a new test by substituting hard-coded values for dependence on the database.

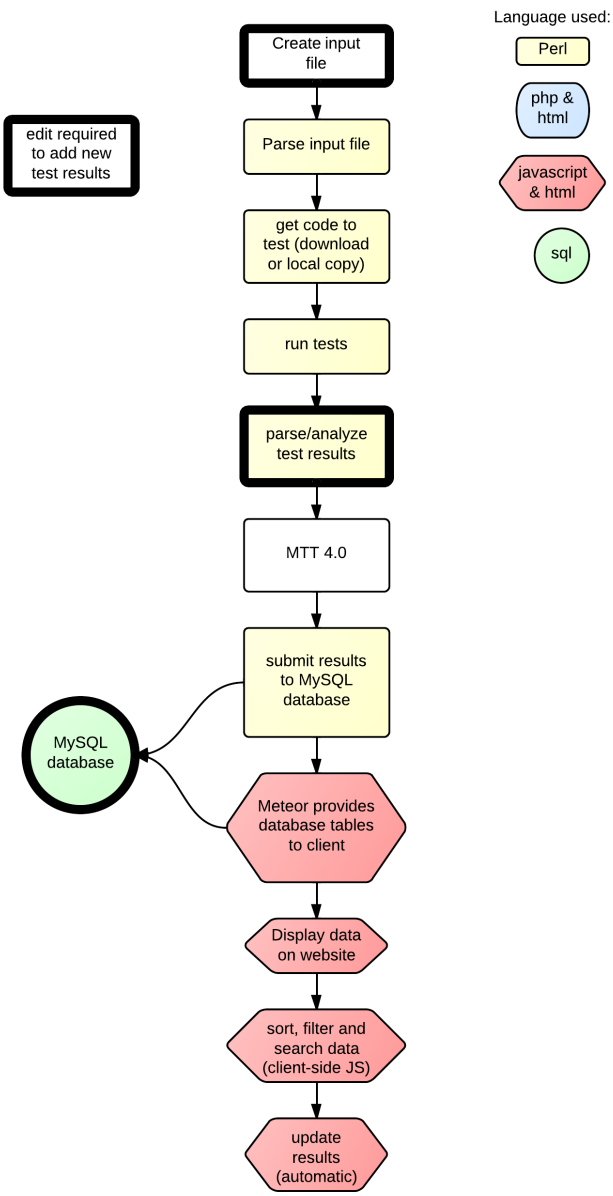


Figure 4.1: Flow chart for MTT 4.0

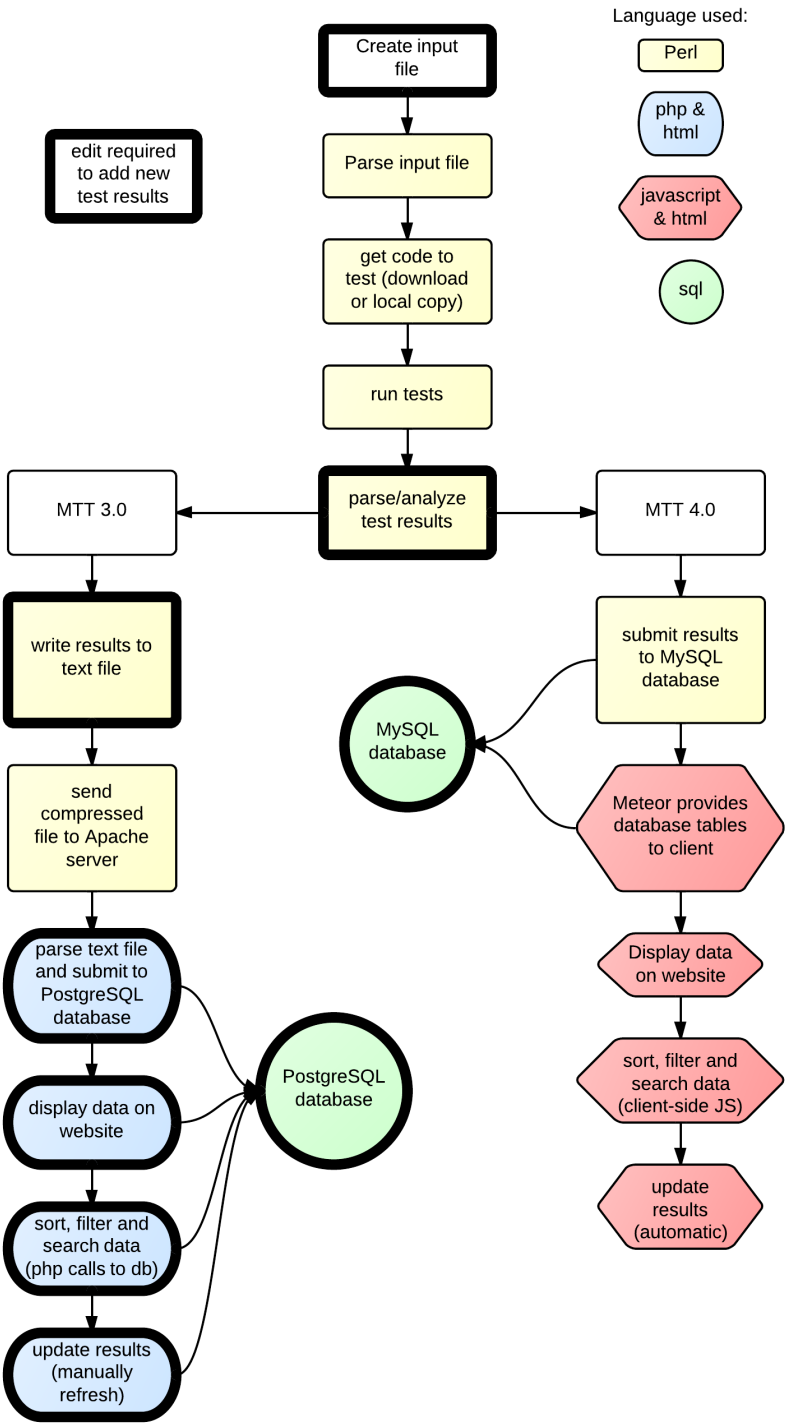


Figure 4.2: Flow chart comparing MTT 3.0 and MTT 4.0

Chapter 5

Evaluation and Assessment

The goal of this thesis was to produce a software test suite framework that can address all eight of the challenges enumerated in the introduction. We conducted tests to ensure that the functionality present in MTT 3.0 was successfully maintained in MTT 4.0 as well as tests to demonstrate the added functionality. Specifically, our experiments were designed to answer the following questions:

- Can MTT 4.0 be used to identify a bug in Open MPI in the same way as MTT 3.0?
- Does MTT 4.0 add additional overhead?
- Can the following be easily and efficiently generated and displayed in the UI?
  - an already-generated test result
  - a new test
  - a new type of test
- Can MTT 4.0 be easily applied to a new project?

5.1 Methodology and Results

All of the tests described in this section were run locally on a MacBook Pro running OSX 10.8.5, with a 4-core 2.2 GHz Intel Core i7 processor. All tests were conducted while the local database servers for both versions of MTT, the Apache server for MTT 3.0 and the Meteor server for MTT 4.0 were all running locally. This was to ensure that any timing or performance differences were due to code we introduced rather than processes running in the background. As described in the next chapter, further testing will be required before this code is moved to a production system. For the purposes of this thesis, the ease of access and control over the whole system afforded by using a local system was preferred.

5.1.1 Maintaining Functionality

We first tested that the new code can match the performance and functionality of the previous version of MTT. Figures 5.1 - 5.4 show the user interface for MTT 4.0 displaying results, which can be compared to the MTT 3.0 UI in Figures 3.2 - 3.5. We compared test results from MTT 3.0 and MTT 4.0 run on three different Open MPI tarballs (versions 1.6a1r30359, 1.7.4rc2r30528, and 1.9a1r30527) and confirmed that the results were identical. For version MTT 4.0, we also compared test results directly output by the tests to the results submitted to the MySQL database and then displayed in the UI and confirmed that these match. We also ran both MTT 3.0 and MTT 4.0 on a version of Open MPI with a purposely introduced bug, and were able to successfully identify the bug with both versions. These tests confirmed that the new version of MTT still addresses the challenges of comprehensiveness, complexity and problem diagnosis. We viewed the UI and command line output simultaneously while MTT was running, and confirmed that new results are automatically added to the UI in real time without any action from the user. This confirms that we address

the challenge of real-time, simultaneous access.

5.1.2 Maintaining Performance

To demonstrate that the new structure of MTT 4.0 did not increase the time necessary to run the test suite framework, so that it still addresses the challenge of time intensiveness, we ran a series of timing tests on MTT 3.0 and MTT 4.0. While adding new functionality to the code we did not add any new steps to the process, and in fact removed a step. MTT 3.0 wrote test results to a text file with Perl, compressed that file and sent it to an Apache server and then decompressed the text file, parsed it and submitted its contents to a database. In MTT 4.0 we instead directly connect to the database with Perl and remove the intermediate steps. To maintain the ability to save results and submit them at a later time, in case of firewall requirements or technical issues, we have maintained an option to write results to a text file and to parse and submit this file to the database.

We ran MTT 3.0 and MTT 4.0 with identical input files, with three versions of Open MPI and six groups of tests. This combination of code and tests was selected to match the current MTT runs on LANL production systems and to contain tests that will (purposely) fail to build, and tests that will pass, fail, be timed out and produce performance results. For each version of MTT, we ran five runs without performing any results submission and five runs that submitted test results to the database, timing each. The averaged results for each set of runs, shown in Table 5.1, show both that the revised structure of MTT 4.0 did not have an adverse affect on timing and that the database submission process in both cases does not take a significant percent of the total run time. This will be further tested on larger scales before MTT 4.0 is introduced on production systems.

All phases

Show 10 entries Filter time:

	Submit Time	Org	Platform name	Hardware	OS	MPI name	MPI version	MPI Install		Test Build		Test Run				
								Pass	Fail	Pass	Fail	Pass	Fail	Skip	Timed	Perf
	2014-02-05 21:44:33	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-nightly-trunk	1.9a1r30527	1	0	1	0	41	4	2	0	2
	2014-02-05 23:32:26	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-nightly-trunk	1.9a1r30527	1	0	1	0	41	4	2	0	2
	2014-02-05 23:50:58	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-nightly-trunk	1.9a1r30527	1	0	0	1	43	4	0	0	2
	2014-02-06 00:51:04	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-nightly-trunk	1.9a1r30527	1	0	1	0	43	4	0	0	2
	2014-02-06 01:23:16	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-nightly-trunk	1.9a1r30527	1	0	1	0	43	4	0	0	2
	2014-02-06 01:47:03	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.6	1.6.6a1r30359	1	0	1	0	43	4	0	0	2
	2014-02-06 01:52:51	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.7	1.7.4rc2r30528	1	0	1	0	43	4	0	0	2
	2014-02-06 02:00:16	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-nightly-trunk	1.9a1r30527	1	0	1	0	43	4	0	0	2
	2014-02-06 17:39:04	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.6	1.6.6a1r30359	1	0	0	1	43	4	0	0	2
	2014-02-06 17:45:08	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.7	1.7.4rc2r30528	1	0	1	0	1	1	1	1	1

Showing 1 to 10 of 23 entries Previous Next

1.9a1r30527
1.6.6a1r30359
1.7.4rc2r30528
openmpi-1.9a1r30656M-buggy

Figure 5.1: MTT 4.0 UI with results for all phases

MPI Install

Show 10 entries Filter time:

	Submit Time	Org	Platform name	Hardware	OS	MPI name	MPI version	Bitness	Endian	Compiler name	Mpi Install	
											Pass	Fail
⊖	2014-02-11 00:48:31	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.6	1.6.6a1r30359	64	little	gnu	6	0

Configure Arguments: CFLAGS=-pipe --enable-debug
Exit Value: 0
Exit Signal: -1
Duration: 00:05:35
Result Message: Success
Stdout/Stderr:
/usr/bin/ranlib: file: .libs/libasm.a(asm.o) has no symbols
ranlib: file: .libs/libasm.a(asm.o) has no symbols
/usr/bin/ranlib: file: .libs/libopalutil.a(qsort.o) has no symbols
ranlib: file: .libs/libopalutil.a(qsort.o) has no symbols
/usr/bin/ranlib: file: .libs/libmca_memchecker.a(memchecker_base_wrappers.o) has no symbols
ranlib: file: .libs/libmca_memchecker.a(memchecker_base_wrappers.o) has no symbols
/usr/bin/ranlib: file: .libs/libopen-pal-lt.a(tsd.o) has no symbols
/usr/bin/ranlib: file: .libs/libopen-pal-lt.a(asm.o) has no symbols
/usr/bin/ranlib: file: .libs/libopen-pal-lt.a(qsort.o) has no symbols
ranlib: file: .libs/libopen-pal-lt.a(tsd.o) has no symbols
ranlib: file: .libs/libopen-pal-lt.a(asm.o) has no symbols
ranlib: file: .libs/libopen-pal-lt.a(qsort.o) has no symbols
ranlib: file: .libs/libopen-pal-lt.a(memchecker_base_wrappers.o) has no symbols
/usr/bin/ranlib: file: .libs/libopen-rte-lt.a(asm.o) has no symbols
/usr/bin/ranlib: file: .libs/libopen-rte-lt.a(memchecker_base_wrappers.o) has no symbols
/usr/bin/ranlib: file: .libs/libopen-rte-lt.a(qsort.o) has no symbols

Figure 5.2: MTT 4.0 UI with results for MPI Install

Test Build

Show 25 entries Filter time: 2014-02-10

Submit Time	Duration	Org	Platform name	Hardware	OS	MPI name	MPI version	Bitness	Compiler	Compiler version	Suite	Test build	
												Pass	Fail
2014-02-10 04:31:36	00:00:01	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-nightly-trunk	1.9a1r30527	64	gnu	5.0	trivial	1	0
2014-02-10 04:31:37	00:00:03	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.7	1.7.4rc2r30528	64	gnu	5.0	trivial	1	0
2014-02-10 04:31:40	00:00:03	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.6	1.6.6a1r30359	64	gnu	5.0	trivial	1	0
2014-02-10 04:31:45	00:00:03	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-nightly-trunk	1.9a1r30527	64	gnu	5.0	intel	1	0
2014-02-10 04:31:49	00:00:03	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.7	1.7.4rc2r30528	64	gnu	5.0	intel	1	0
2014-02-10 04:31:53	00:00:02	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.6	1.6.6a1r30359	64	gnu	5.0	intel	1	0
2014-02-10 04:31:55	00:00:00	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-nightly-trunk	1.9a1r30527	64	gnu	5.0	imb	0	1
2014-02-10 04:31:55	00:00:01	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.7	1.7.4rc2r30528	64	gnu	5.0	imb	0	1
2014-02-10 04:31:56	00:00:00	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.6	1.6.6a1r30359	64	gnu	5.0	imb	0	1
2014-02-10 04:31:56	00:00:00	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-nightly-trunk	1.9a1r30527	64	gnu	5.0	netpipe	1	0
2014-02-10 04:31:56	00:00:03	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.7	1.7.4rc2r30528	64	gnu	5.0	netpipe	1	0
2014-02-10 04:32:02	00:00:01	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.6	1.6.6a1r30359	64	gnu	5.0	netpipe	1	0
2014-02-10 04:32:03	00:00:14	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-nightly-trunk	1.9a1r30527	64	gnu	5.0	onesided	1	0
2014-02-10 04:32:18	00:00:12	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.7	1.7.4rc2r30528	64	gnu	5.0	onesided	1	0
2014-02-10 04:32:31	00:00:10	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.6	1.6.6a1r30359	64	gnu	5.0	onesided	1	0
2014-02-10 04:32:41	00:00:18	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-nightly-trunk	1.9a1r30527	64	gnu	5.0	amg	1	0
2014-02-10 04:33:01	00:00:19	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.7	1.7.4rc2r30528	64	gnu	5.0	amg	1	0
				MacBook Pro 8.2 IntelCore i7 2.2									

Figure 5.3: MTT 4.0 UI with results for test builds

Test Run

Show 10 entries Filter time:

	Submit Time	Duration	Org	Platform name	Hardware	OS	MPI name	MPI version	Suite	np	Test Run				
											Pass	Fail	Skip	Timed	Perf
	2014-02-11 22:13:28	00:00:28	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.9-bug	openmpi-1.9a1r30656M-buggy	amg2006	32	0	1	0	0	1
	2014-02-11 22:02:37	00:00:00	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.9-bug	openmpi-1.9a1r30656M-buggy	onesided	2	10	4	0	18	0
	2014-02-11 22:01:50	00:00:46	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.9-bug	openmpi-1.9a1r30656M-buggy	netpipe	2	1	0	0	0	1
	2014-02-11 22:01:45	00:00:01	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.9-bug	openmpi-1.9a1r30656M-buggy	intel	2	0	8	0	0	0
	2014-02-11 22:01:44	00:00:01	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.9-bug	openmpi-1.9a1r30656M-buggy	trivial	2	2	0	0	0	0
	2014-02-11 21:21:34	00:00:05	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.6	1.6.6a1r30359	amg2006	32	1	0	0	0	1
	2014-02-11 21:21:01	00:00:00	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.6	1.6.6a1r30359	onesided	2	28	4	0	0	0
	2014-02-11 21:20:15	00:00:46	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.6	1.6.6a1r30359	netpipe	2	1	0	0	0	1
	2014-02-11 21:20:06	00:00:01	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.6	1.6.6a1r30359	intel	2	8	0	0	0	0
	2014-02-11 21:20:01	00:00:02	lanl	OS X	MacBook Pro 8.2 IntelCore i7 2.2 GHz	Darwin	ompi-v1.6	1.6.6a1r30359	trivial	2	4	0	0	0	0

Showing 1 to 10 of 373 entries Previous Next

Figure 5.4: MTT 4.0 UI with results for test runs

Table 5.1: Timing results for MTT 3.0 and MTT 4.0, run with identical input parameters, with and without database submission. The times presented here each the average of five identical test runs.

code version	run time		
	no results submission	results submitted to DB	time to submit
MTT 3.0	40m 6.348s	41m 34.688s	54.053s
MTT 4.0	40m 45.646s	41m 29.755s	43.236s

5.1.3 Adding Functionality

We also conducted a series of tests to demonstrate the new functionality which was added to MTT 4.0. To test whether MTT 4.0 completely solves the challenge of simultaneous, real-time access, (providing results in real time, without action by the user), we ran MTT with the UI displayed in a browser. We compared the MTT command-line output to the results in the UI and confirmed that new results are added without requiring a manual refresh of the UI.

To demonstrate that new tests and test results can be more easily added (solving the challenge of extensibility), we first added one new parameter to store and display, then a new test and finally a new phase of testing.

We first went through the steps necessary to display a new test result parameter (test duration) in the UI. In Figure 4.2, the thick outlines show which steps in the code needed to be modified for MTT 3.0 and MTT 4.0. Both versions require code to be written to calculate the new parameter or parse it from test output. For this test, we chose a parameter which was already generated by the testing code but not saved or displayed. In MTT 3.0 the parameters to be submitted to the database and then displayed are all hard-coded, so adding a new parameter required editing the code at each step after tests are run. MTT 4.0 simplifies this process by having the test result submission code and the UI code directly access the database for information about

what parameters to submit and display. To add the `test duration` parameter, we added a new column, `duration`, to the `Test Run` table in the database and added a row to the `display_data` table indicating that `duration` should be displayed in the `Test Run` table. When the test result submission code accesses the MySQL table for the phase `Test Run`, it now receives the new `duration` parameter along with the previously existing ones, and submits the relevant data. The UI code then receives all of this data along with the information to display the `duration` column, and does so automatically.

We then added a new test (amg2006) to the process. For both versions this required editing the input file to include the path to the testing code and the correct build and run-time parameters. Both MTT versions also required a new Perl module to parse the test results into the desired parameters. Adding these parameters to the database and UI required the same steps as for a single parameter, described above. The AMG test, along with the previously-run NetPipe test, generates performance-related data. This needs to be processed and displayed in a different manner than simple pass/fail results and so requires new parameters in the database and a new way of displaying data in the UI. This data is generated in the same way as other test results, so no additional changes to the test running or submitting processes are required. In MTT 3.0, adding a new way of processing data requires additions to all the post-results-submission steps as with adding a single parameter. With MTT 4.0, the database must be edited to store the new data and code must be added to the display part of the UI to specify how this new type of results are to be displayed. Unlike in the previous version, however, no changes are required to submit the results to the database, or serve them to the UI. In the MySQL database, an additional row, `performance`, is added to the `phase` table along with a new column to indicate that this is performance data. Future work will include displaying this performance data in graphical format (triggered by this additional flag), but at present the data is displayed as text. Performance testing requires that

some result (bandwidth or latency for NetPipe or the figure of merit for AMG) be compared over time. This can easily be done with the already generated timestamps and the newly produced performance parameters. In the case of AMG results we also want to compare results to a standard value. To store this value, we added a `standards` table to the MySQL database which can store standard values for a given parameter per architecture. Values in this table can be added directly to the database or through the MTT input file. The method of input file submission to the Standards table might need to be changed before this system is moved to production to reduce the possibility of inadvertently replacing values. Using triggers in the MySQL database (to be replaced by the more efficient views when Meteor 1.0 is released), the performance data which should be displayed is selected, compared to standard data where required, and passed through Meteor to the UI to be displayed (see Figure 5.5). When it is possible to display charts through Meteor, this addition to the UI code will require only a single if/else statement referencing the `phase` table to determine if a set of data is performance data or not, and then the additional JavaScript/HTML to produce the desired chart format.

AMG 2006

Show 10 entries Filter time:

	Start Time	Duration	Org	Platform	MPI name	MPI version	Compiler	Compiler version	figure of merit	residual_norm
	2014-02-11 18:22:20	00:00:05	lanl	OS X	ompi-v1.6	1.6.6a1r30359	gnu	5.0	148432.5	0.000000197132
	2014-02-11 19:02:51	00:00:05	lanl	OS X	ompi-nightly-trunk	1.9a1r30527	gnu	5.0	139041.4	0.000000197132
	2014-02-11 19:02:56	00:00:04	lanl	OS X	ompi-v1.7	1.7.4rc2r30528	gnu	5.0	167902	0.000000197132
	2014-02-11 19:03:00	00:00:05	lanl	OS X	ompi-v1.6	1.6.6a1r30359	gnu	5.0	129936.2	0.000000197132
	2014-02-11 19:35:17	00:00:06	lanl	OS X	ompi-nightly-trunk	1.9a1r30527	gnu	5.0	144885.9	0.000000197132
	2014-02-11 19:35:23	00:00:05	lanl	OS X	ompi-v1.7	1.7.4rc2r30528	gnu	5.0	147440.2	0.000000197132
	2014-02-11 19:35:28	00:00:05	lanl	OS X	ompi-v1.6	1.6.6a1r30359	gnu	5.0	174358.4	0.000000197132
	2014-02-11 20:19:26	00:00:06	lanl	OS X	ompi-v1.6	1.6.6a1r30359	gnu	5.0	5	0.000000197132
	2014-02-11 20:39:50	00:00:05	lanl	OS X	ompi-v1.6	1.6.6a1r30359	gnu	5.0	157821.1	0.001
	2014-02-11 20:54:55	00:00:05	lanl	OS X	ompi-v1.6	1.6.6a1r30359	gnu	5.0	167817.1	0.000000197132

Showing 1 to 10 of 12 entries Previous Next

Figure 5.5: MTT 4.0 UI with performance results from the AMG2006 test. Green and red specify whether comparison to the standard value passed or failed for each test result parameter.

To address the challenge of versatility, we created a test database with dummy tables. We replaced the code build, test build and test run phases in MTT with Perl scripts which generated text to be stored in the database (simulating a new software project and tests). We populated the `phase` and `display_data` tables and ran MTT to populate our new MySQL database, constructed with the schema in Figure A.4. We performed manual checks of the results displayed in the UI and confirmed that the displayed data matched the data produced by the test code.

Chapter 6

Conclusions

Our goal in this thesis was to construct a test suite framework which can solve all of the challenges involved in testing large software projects which we identified: one that can run automated regression and performance tests on multiple architectures, report the results to a central database, display the results in a usable form in real-time, be easily modifiable as the project evolves and be applicable to new projects. As a test case, we created a new version of the MPI Testing Tool (MTT) code, maintaining its previous functionality while reconstructing it in a modular format using the Meteor JavaScript package to link the database and UI. This simplifies adding new parameters, tests and test phases to MTT and allows the code to be easily applied to different projects.

6.1 Contributions

In this thesis, we produced a code test suite framework (MTT 4.0) which provides all the functionality required for testing Open MPI and can also be applied to new programming projects. We applied the Model-View-Controller model to the pre-

existing MTT 3.0, allowing us to remove hard-coded values and functions from the results submission and UI code and substitute dependence on the MySQL database. This simplifies the process of adding new parameters, tests and testing phases which will allow Open MPI developers to more easily produce and display relevant data as the code evolves. We have demonstrated that MTT 4.0 can be applied to a completely different set of tests and data than those used by Open MPI. This functionality was introduced into MTT 4.0 without adding any additional overhead to the testing process. As with previous versions MTT 4.0 will be freely available online, so it can be used and developed by other programming projects.

6.2 Limitations and Future Work

Further testing will be required before MTT 4.0 can be implemented on production systems, including timing tests and testing the ability of the system to handle the large number of test results generated by multiple MTT users. Once MTT 4.0 is introduced onto production systems, then further tests and related test results can be added into the pipeline. When Meteor 1.0 is released, providing a more completely functioning interface between MySQL and Meteor, further tests will also be required to determine whether the current system of creating new tables of data-to-be-displayed with triggers in MySQL can be more efficiently replaced by views.

If Meteor 1.0 is capable of connecting to multiple types of SQL databases, then testing can also be performed to determine whether MySQL is the best type of database for this project. The database structure will also need to be optimized with respect to access from the UI. The database for MTT 3.0 was optimized for searching so that the PHP and SQL commands that sorted and filtered data for the UI could be performed quickly. With MTT 4.0, we instead use Meteor to access whole data tables and perform search and searching with javascript without further

queries to the database. Further testing must be performed to determine the most efficient way of performing this sorting and filtering of data when the full-scale set of test results is involved. An intermediate solution can also be explored, where some default amount of data (defined by a time interval) is initially passed through Meteor to the UI and a request by the user for older data would prompt Meteor to access and send through a larger set of data.

The exact method of submitting test results to the database will also need to be studied further before being used by the whole Open MPI community. The current method in MTT 4.0 of directly connecting to the database server will not be practical on this scale due to security concerns and memory limits. A possible workaround would be to apply a similar method as in MTT 3.0, but maintaining the modularity we have introduced. This could use the option to write results to a text file, and then move the results submission code from client-side MTT to the database server. Here, the code could directly query the database to link results in the text file to the correct tables in the database. It may also be possible to set up a second user interface with Meteor which can be used to submit test results.

Appendices

Appendix A

Database Schemas

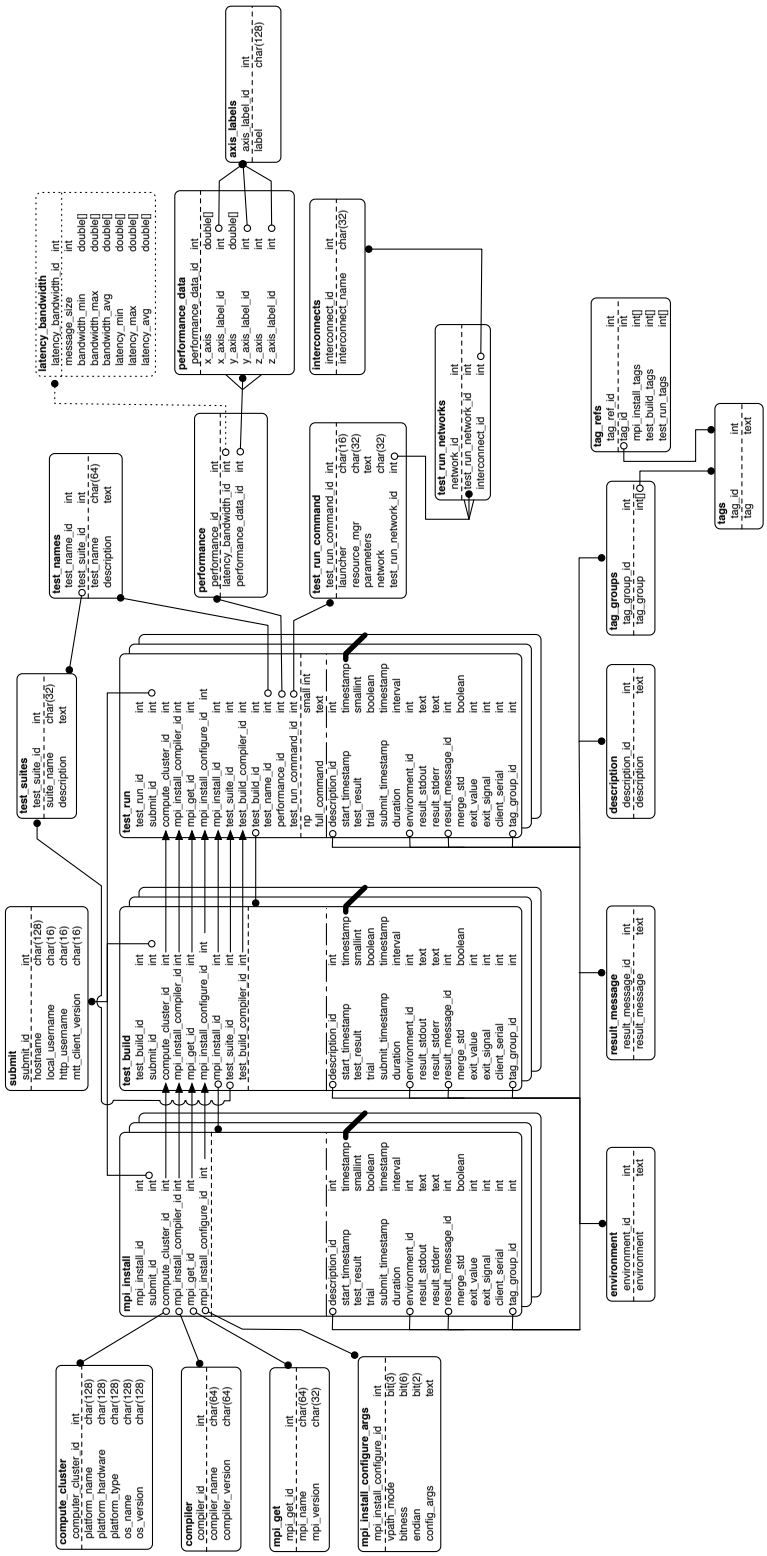


Figure A.1: Database Schema for MTT 3.0

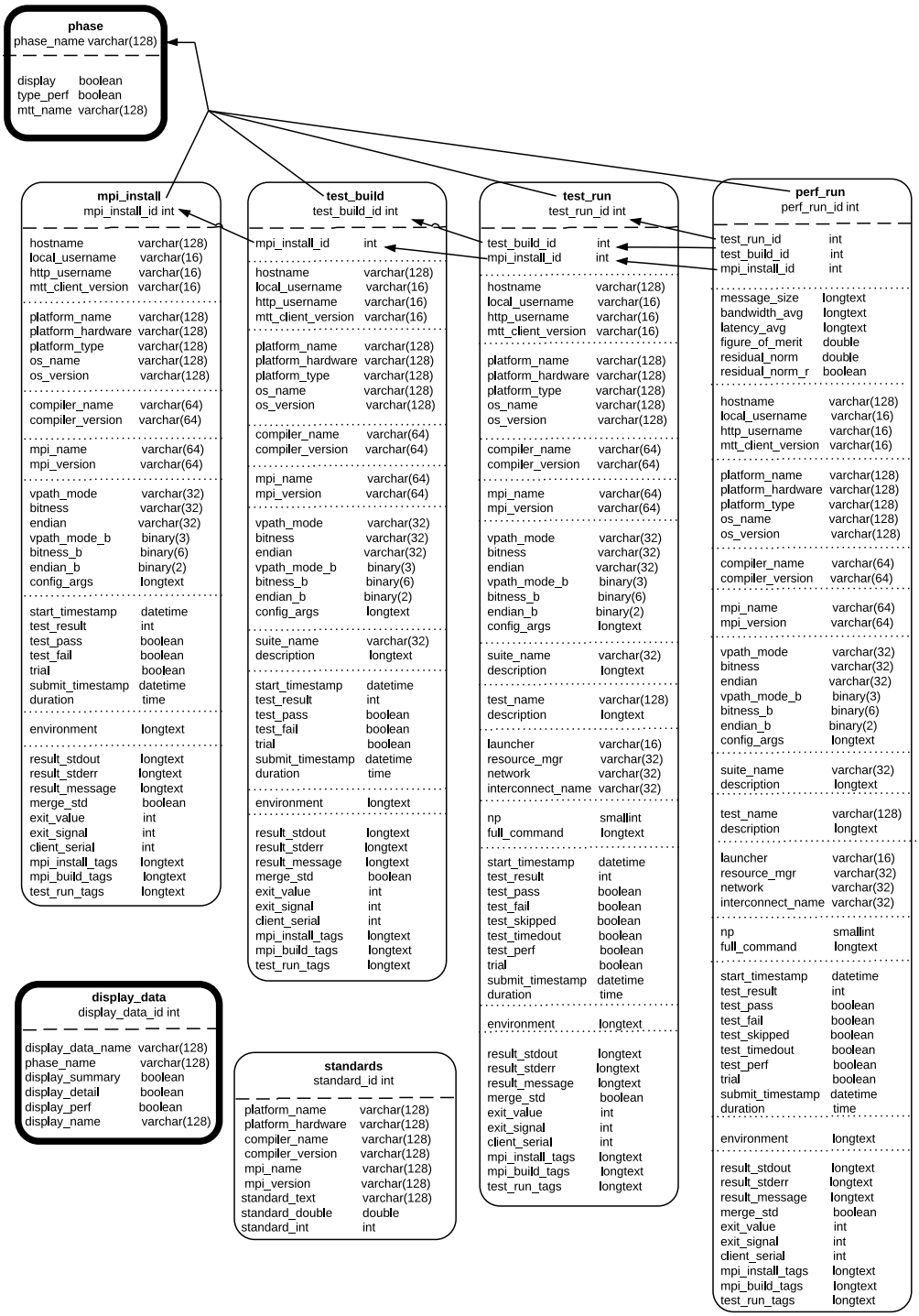


Figure A.2: Database Schema for MTT 4.0

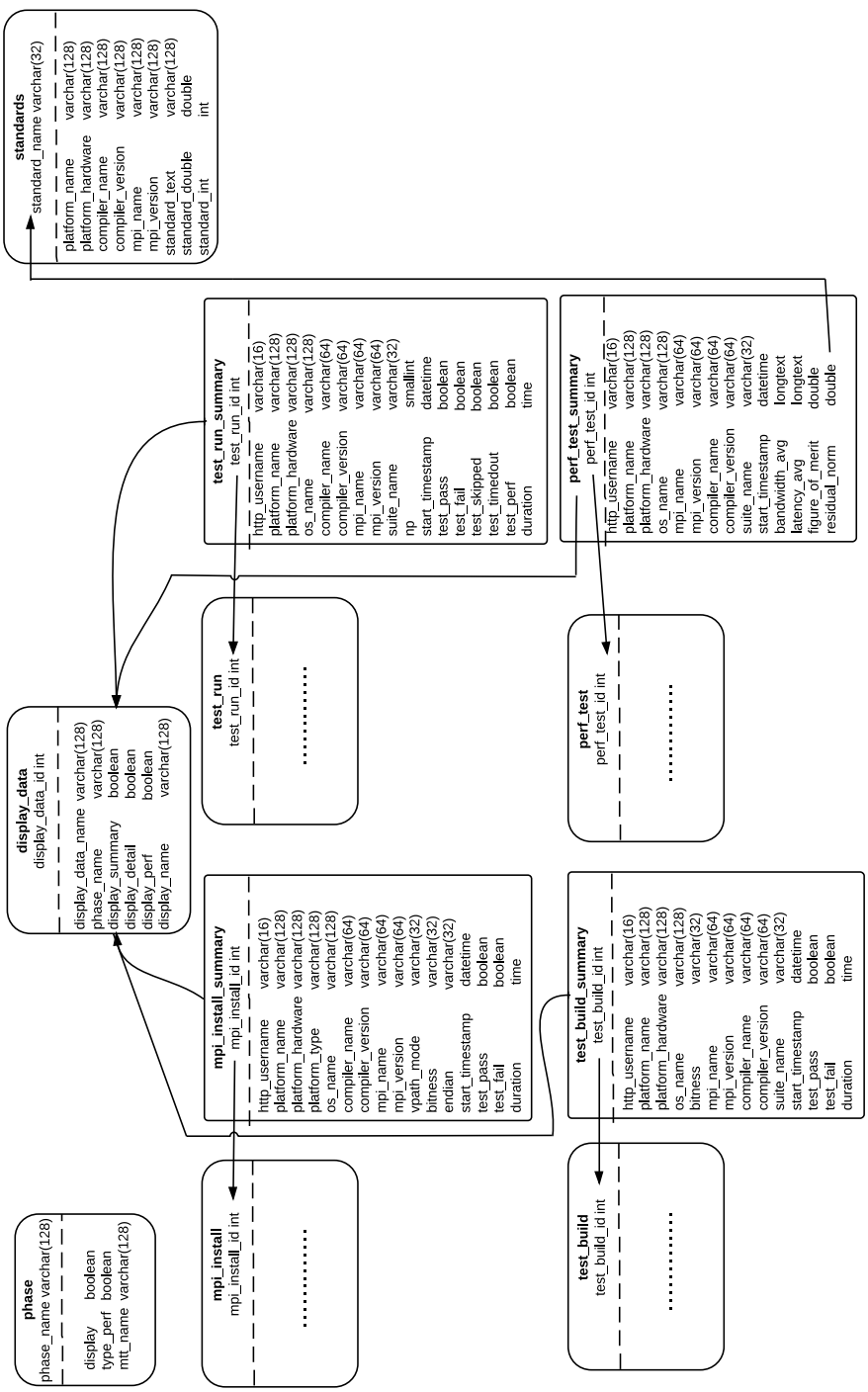


Figure A.3: Views in the database schema for MTT 4.0

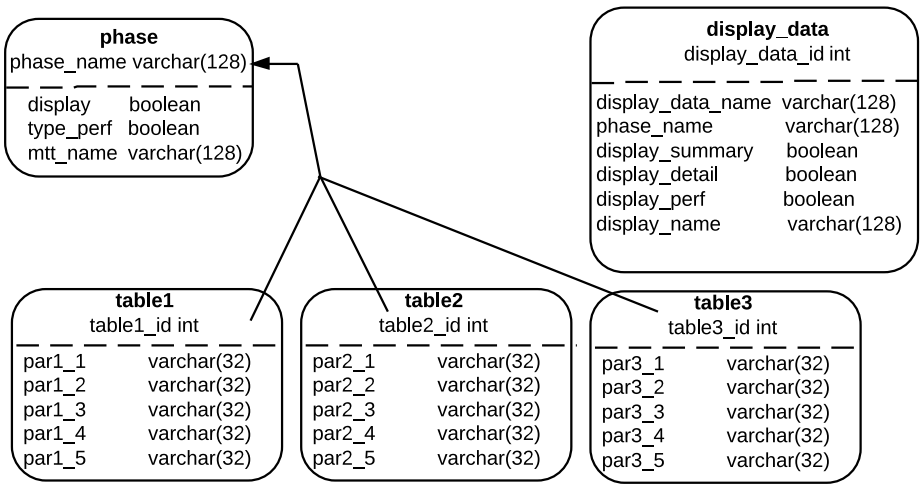


Figure A.4: Database Schema for test case, simulating results from a test suite

References

[1] J. Hursey, E. Mallove, J. M. Squyres, and A. Lumsdaine, *An Extensible Framework for Distributed Testing of MPI Implementations*, in Proceedings, Euro PVM/MPI, Paris, France, October 2007.

[2] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, *Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation*, in Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, September 2004, pp. 97–104.

[3] T. Reenskaug, *THING-MODEL-VIEW-EDITOR: an Example from a planning system*, http://heim.ifi.uio.no/~trygver/1979/mvc-1/1979-05-MVC.pdf.

[4] G. E. Krasner and S. T. Pope, *A Cookbook for Using the Model-View Controller User Interface Paradigm in Smalltalk-80*, J. Object Oriented Program. **1** (1988), 26–49.

[5] M. Veit and S. Herrmann, *Model-View-Controller and Object Teams: A Perfect Match of Paradigms*, in AOSD03 ACM Press, 2003, pp. 140–149.

[6] R. Sridaran, G. Padmavathi, K. Iyakutti, and M. Mani, *SPIM Architecture for MVC-based Web Applications*, Journal of Advanced Networking and Applications **1** (2009), 63–68.

[7] D. Walker and A. Orooji, *Metrics for Web Programming Frameworks*, in Proceedings of the International Conference on Semantic Web and Web Services, Las Vegas, NV, July 2011.

[8] A. Valikov, A. Akhounov, and A. Schmidt, *A Model-Transformers Architecture for Web Applications*, in Technologies for E-Services, Proceedings, Vol. 2444, 2002, pp. 29–37.

[9] Z. Sari, M. Sarosa, and H. Nurwasito, *Article: Concept of Designing an Optimized Pull Model View Controller Type Content Management Framework*, International Journal of Computer Applications **57** (2012), 9–13 (Published by Foundation of Computer Science, New York, USA).

[10] Y. Ping, K. Kontogiannis, and T. Lau, *Transforming Legacy Web Applications to the MVC Architecture*, in Software Technology and Engineering Practice, 2003. Eleventh Annual International Workshop on, 2003, pp. 133–142.

[11] O. Niese, T. Margaria, A. Hagerer, B. Steggen, G. Brune, W. Goerigk, and H. Ide, *Automated Regression Testing of CTI-Systems*, in Test Workshop, 2001. IEEE European, 2001, pp. 51–57.

[12] P. Hocov, T. Dymek, and K. Miroslav, *Adaptable Visualization Service: through Uniformity towards Sustainability*, in Proceedings of the First International Workshop on Model Driven Interoperability for Sustainable Information Systems (MDISIS'08) held in conjunction with the CAiSE'08 Conference, CEUR Workshop Proceedings, Montpellier, France, 2008, pp. 59–71.

[13] J. Nutaro and P. Hammonds, *Combining the Model/View/Control Design Pattern with the DEVS Formalism to Achieve Rigor and Reusability in Distributed Simulation*, Journal of Defense Modeling and Simulation: Applications, Methodology, Technology **1** (2004), 19–28.

[14] A. Karagkasidis, *Developing GUI Applications: Architectural Patterns Revisited*, in Proceedings, 13th Annual European Conference on Pattern Languages of Programming, 2008.

[15] J. Cai, R. Kapila, and G. Pal, *HMVC: The Layered Pattern for Developing Strong Client Tiers*, http://www.javaworld.com/jw-07-2000/jw-0721-hmvc.html.

[16] A. Hussey and D. Carrington, *Comparing Two User-Interface Architectures: MVC and PAC*, in MVC AND PAC. FAHCI'96 Springer Verlag, 1996, pp. 3–21.

[17] J. Hewitt, *Firebug: Web Development Evolved*, http://getfirebug.com.

[18] I. Bethune, A. Carter, K. Stratford, and P. Korosoglou, *CP2K: Scalable Atomistic Simulation for the PRACE Community*, in PRACE White Paper, 2012.

[19] P. H. Carns, W. B. Ligon, III, R. B. Ross, and R. Thakur, *PVFS: A Parallel File System for Linux Clusters*, in In Proceedings of the 4th Annual Linux Showcase and Conference MIT Press, 2000, pp. 391–430.

[20] R. Grigoryev, *Xyratex Testing Tool*, https://github.com/Xyratex/xperior.

[21] J. Maher, *Buildbot/Talos*, https://wiki.mozilla.org/Buildbot/Talos.

[22] S. Dutta, *SDCC - Small Device C Compiler*, http://sdcc.sourceforge.net/.

[23] M. D. Group, *Meteor*, http://www.meteor.com/.

[24] SpryMedia, *DataTables*, http://datatables.net/.