Petscope: A Standardized System for Ballistic Missile Guidance Data Analysis

by

Austin Casey McNurlen

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Masters of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2004 [June 2004]

Author .....

Department of Electrical Engineering and Computer Science

May 20, 2004

Certified by .....

Ray Magee

Charles Stark Draper Laboratory

Thesis Supervisor

Certified by .....

Barbara Liskov

Professor of Electrical Engineering and Computer Science

Thesis Supervisor

Accepted by .....

Arthur C. Smith

Chairman, Department Committee on Graduate Students

Petscope: A Standardized System for Ballistic Missile Guidance Data Analysis

by

Austin Casey McNurlen

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2004, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Computer Science and Engineering

Abstract

This thesis had the goals of standardizing and automating guidance data analysis at The Charles Stark Draper Laboratory, Inc. (Draper). This was accomplished via the first ever machine understandable knowledge of Portable Engineering Testing Stations (PETS) variables located in a comprehensive MySQL database. Automation was achieved via a MATLAB program designed to be flexible for a wide variety of inputs and outputs. The standardization required time-consuming interface analysis and a tedious programming effort, but has proved to be successful and is already streamlining the data collection and analysis process for another program at Draper.

Thesis Supervisor: Ray Magee
Title: Charles Stark Draper Laboratory

Thesis Supervisor: Barbara Liskov
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

This thesis was prepared at The Charles Stark Draper Laboratory, Inc., under Contract 00030-04-C-0010, sponsored by the U.S. Navy S.S.P.

Publication of this thesis does not constitute approval by Draper or the sponsoring agency of the findings or conclusions contained herein. It is published for the exchange and stimulation of ideas.

Acknowledgments

I would also like to acknowledge a few of the people who have significantly helped me through MIT.

My parents were the supporting benefactors of three very expensive undergraduate years at MIT.

My two academic advisers have over seen my progress. Prof. Saman Amarasinghe was my adviser for my first three years, before going on sabbatical. I was then fortunate enough to find Prof. Barbara Liskov; who in addition was nice enough to chair my thesis.

Dr. Bakhtiar Mikhak provided support and funding through NSF grants for my research in my first three years at MIT in the Media Lab. Andy Begel has been an influential aide in his vast technical knowledge of computer science.

This thesis would not have been possible without the help of the following individuals at Draper. Ray Magee and Dave Liguori have over seen the project and provided direction. Thomas Cheng has contributed experienced knowledge of PETS data, and has been essential to the standardization of it. Christ Shakhmalian has been heavily involved with the coordination between Petscope and evolving PETS database logging procedures at Draper.

ASSIGNMENT

Draper Laboratory Report Number T-1493.

In consideration for the research opportunity and permission to prepare my thesis by and at The Charles Stark Draper Laboratory, Inc., I hear by assign my copyright of the thesis to The Charles Stark Draper Laboratory, Inc., Cambridge, Massachusetts.

Chapter 1

Introduction

The Charles Stark Draper Laboratory, Inc. (Draper) has been a pioneer in guidance, navigation, and control for nearly fifty years. This expertise has been productized into systems operating on the manned Apollo space flights and nearly all of the nation's operational strategic missile systems. Recently, Draper was awarded a contract for a life extension through the year 2042 (known as MK6LE) of the Trident Mark 6 Guidance Missile Program [2]. The Trident program has been a successful program of Draper's for many years now. The main goals of the life extension are to upgrade to maintainable technology and to reduce the costs of the missile systems operation.

One important area of the project is the testing and quality assurance of the guidance system. As one can imagine testing the performance of a ballistic missile is not easy because it can't just be recompiled like a piece of software or re-fabricated like an electronic circuit. The guidance system however can be tested individually from the casing and rocket systems of the missile. To do this, Draper engineers have a variety of tests at their disposal.

One such test is called a centrifuge test. In this test the guidance system is placed on a 32 foot centrifuge that Draper owns in Bedford, Massachusetts. This centrifuge spins the guidance system around providing it with angular acceleration that it can process and react on appropriately. Another test commonly used is to shake the guidance system vigorously to simulate disturbances it encounters during flight. A more realistic test is to put the guidance system in a pod under a F-15 aircraft, and

let it experience flight similar to what it would experience in a missile launch. All of these tests are adequate and necessary, however repeating test failures and setting up the hardware for the tests can be difficult, time-consuming, and costly. So the new ideology at Draper has been to simulate flight via software and measure what the guidance system is doing.

The challenge I will be dealing with in my thesis will be to develop an efficient standardized testing system for the lab and involved third parties to use. Every time a test is run on a guidance system, a long binary stream is produced. Previously, analyzing this stream in the lab could take up to two hours. Cost reduction, and the foreseen increase of testing due to software based tests demonstrate a need for a more elaborate testing system at Draper. I have developed a system called Petscope which will analyze the data out of the Portable Engineering Testing Station (PETS) that Draper uses to perform tests on the guidance system. The main goals of the system are standardization of tests and improved automated testing processes.

Chapter 2

Standardization

2.1 Overview

The first goal of the Petscope project is the standardization of PETS data analysis. Standardization is a well known problem in both historical and contemporary engineering. The primary idea is that when multiple parties are involved on the same project or team, everybody has to be speaking the same language.

The standardization problem with PETS data analysis is that even though the format and structure of PETS data is standardized, what is stored in the PETS data is variable. PETS data can store many different types of variables in it, in addition to different methods of representing and scaling the data. Problems can occur as data formats were not standardized during the design of the interface. Each engineer created his own formatting scheme, resulting in many different data formats.

I will define a PETS dictionary as a definition of the contents in a given set of PETS files. In general, most PETS dictionaries have been pretty similar throughout the 30 year history of the use of PETS. However, some changes are normally made from dictionary to dictionary, and in the future it is intended for some dictionaries to be drastically different than current dictionaries. Most engineers at Draper and third-party companies heavily involved with research at Draper rely on definitions of these dictionaries in officially produced Weapons Specification (WS)documents. The problem with relying on WS documents for analysis of PETS data is that every

engineer must constantly refer to these documents. As one could imagine problems arise when the engineer applies incorrect formatting when interpreting data. The other main problem with WS documents is that they can not be utilized by automated computer scripts and programs. These documents are highly complex, obscurely formatted, and thus can not easily be read and understood by a machine.

Petscope seeks to alleviate this problem by creating a central MySQL database in Draper that will contain a database structure representing what is in various different dictionaries of PETS files. It is intended that any piece of software that uses PETS data will utilize this database. This means that the effort exerted to standardize PETS data analysis by Petscope can be reused by many other research projects in the laboratory. The benefits of this will be reduced development time of future analysis programs (such as real-time data displays) and higher integrity of results from future PETS data analysis.

2.2 Database Overview

The main goal of any database engineer is to create a database structure that efficiently represents the data being stored and allows programs that operate on this data to be able to rapidly retrieve and process it. While the Petscope database must adhere to these design requirements, it also has two very distinct requirements that separate it from most other databases.

The first of these additional requirements is that the dictionaries that must be stored are designed to interpret data that comes off of electrical interfaces with the guidance system. This has huge influence over the database design because the database must be engineered so that its relation to the interfaces it analyzes data off of is clear. Petscope is designed to analyze data off of the four main interfaces that guidance data comes off of: Guidance to Fire Control (G2FC) Channel 1, Guidance to Fire Control (G2FC) Channel 3, Telemetry 1, and Telemetry 2. For the remainder of this thesis these four interfaces will be referred to by the following names and abbreviations: Channel 1 (ch1), Channel 3 (ch3), Telemetry 1 (tlm1), Telemetry 2(tlm2).

Each of these four interfaces retrieves a different kind of data in a different format. Because of this, it is difficult to develop one database structure that encompasses the subtleties and anomalies of each interface. I thus chose to develop a structure that has a few tables specifically made for each interface, and then any code that utilizes the database will have retrieval functionality designed for each of the interfaces. I felt this was the ideal approach because each of the four interfaces is not going to be changed or re-engineered anytime soon. Also, engineers at Draper and its affiliates are normally very involved with and knowledgeable about each particular interface so any feature that is considered specific to an interface will not be foreign to them. Based on this idea, it was easier and more logical to just develop a database structure that met the historical requirements of the various interfaces.

The other unique challenge that Petscope faces is that it will be a pioneer database in Draper. Draper is a very high tech research facility, but still must operate on legacy systems that must be maintained for many years. Because of Draper's requirements to still use existing technologies found in submarines, missiles, and many other government projects, Draper has never really utilized a database system in any guidance data analysis situation. Because of this, the database design must be implemented so that it will interface well with Draper's existing testing system.

In summary, the main two goals of the Petscope database structure are to create a structure that relates well to the historical requirements of the four data interfaces and interfaces well with Draper's existing testing system.

2.3 Physical Setup

The main considerations of where to host the database are that it must be on a server that is convenient to access and properly maintained. Like the general database structure, some special considerations must be made because of the context which Petscope will be used.

The Petscope database is hosted on a Sun Blade 1000 server known by its domain name, Sylvester. Sylvester is located in a locked testing lab at Draper. Access to this

room is permitted only with appropriate identification and knowledge of a keypad combination on the door. This ensures that Sylvester is kept in a facility that is physically safe enough to store the kind of data that the Petscope database will contain. Sylvester is kept running 24 hours a day and serves several purposes for the testing groups at Draper. The Petscope database will not require a demanding amount of processing or disk requirements, so the server should be adequate for several years. The server is also properly backed up by the IT department at the system level, which means that the database will automatically be backed up by nightly processes that operate at Draper.

A more important feature of Sylvester is that it sits on the unclassified network within Draper. This means that any computer either physically on Draper's network or connected to Draper through a Virtual Private Network (VPN) will be able to access the Petscope database with the proper authentication procedures. The down-side to this however is that the two telemetry channels will not be able to contain all the information necessary to analyze telemetry data. See section 2.4 for a complete discussion on how this challenge was handled.

In addition to the actual MySQL server, it is useful to have a web application known as phpMyAdmin installed. This is a popular open source web application that is operated through any normal web browser. Unfortunately, it requires that the server running it have Apache and PHP installed. I briefly attempted to do this on Sylvester, but abandoned this idea because PHP does not ship compiled binaries for the Solaris operating system on Sylvester. Instead, phpMyAdmin is operated on several Microsoft Windows machines. Since phpMyAdmin operates a socket connection to the database server just as any other program utilizing it would, the application does not have to reside on the same machine that the database server does. This seemed like a better solution than custom compiling PHP for Solaris.

2.4 Classified Data

Because of treaties with foreign nations, the data obtained from telemetry 1 and 2 is classified. This is because this data is released over radio waves during missile flights, and the U.S. must release this data to other nations after every flight. Because I did not have security clearance, I was never able to see any semantic meaning of what is in either telemetry channel. What this essentially means is that all the mnemonics are given generic names such as "var123", all descriptions are removed, all scalers are set to "1", and all units are set to "BITS".

While this may seem to make the process of standardizing the telemetry interfaces in a database useless, I will instead argue that it is irrelevant. The reason for this is because of the classified nature of the interfaces. Regardless of whether or not I had clearance, classified information can only be stored on classified computers that are not connected to the Internet or any other type of external network. Since the Petscope database must sit on a server that all Draper employees and affiliates must be able to access, this means that the semantic meaning of the variables would have to be encrypted to be in the database anyways.

Not having security clearance did cause difficulties in building a database for the telemetry interfaces. I was assisted by a Draper engineer to build the hierarchies for these two interfaces, as it was impossible to group the variables without knowing what they represent. Also, the database structure for the two telemetry interfaces is not as elegant and relational because performing these kinds of optimizations on the variables is not possible without knowing the meaning of the variables. Because of this, the same semantic variable may appear multiple times in the database due to a variety of reasons such as multiple units for the same variable or different transmission rates for the same variable.

Any engineer who must analyze telemetry data will know how to access text files that were created to allow personnel to get the actual mnemonic, description, scaler, and unit information from the generic variable name. With this information, the engineer can then perform the necessary transformations on the raw bit values to get

the actual information out of the telemetry interfaces.

2.5 Naming Issues

Problems arose in the standardization process due to naming conflicts with MySQL. The main problem occurred because one of the lists in Channel 1 is called "UPDATE". This provides a conflict with MySQL because "UPDATE" is a traditional reserved word in SQL. MySQL [1] claims that "you're allowed to do this". Reese [3] describes a solution to this problem using escape characters. However, after initial testing, I found no escape characters or other solution to solve this problem. Because of this, I had to break convention with the historical name of the Channel 1 list, and rename it to "UPDT".

In channel 3, mnemonics have historically been assigned by word, rather than by variable. This means that two mnemonics that describe the high and low words of a variable might be "VARH" and "VARL". Instead of choosing to name the variable one or the other words, in these select few cases, I just chose to drop the word identifier suffix (in this case, the variable name would be "VAR").

2.6 Channel 1 Setup

Channel 1 is an interface that provides data across multiple lists and segments. Naturally, these segments and lists made for wise choices for the hierarchical menus to select channel 1 variables. The data that comes across channel 1 is not stored in standard form however. The interface uses various bit stuffing patterns to store extra things like sign bits. The variables in channel 1 can be either simple or complex matrices.

ch1variables Table

Field	Type	Attributes	Null	Default	Extra	Action
list	varchar(10)		No			Change Drop Primary Index Unique Fulltext
segment	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
mnemonic	varchar(10)		No			Change Drop Primary Index Unique Fulltext
dimension	varchar(10)		No	0		Change Drop Primary Index Unique Fulltext
type	varchar(7)		No			Change Drop Primary Index Unique Fulltext
scaler	double		No	0		Change Drop Primary Index Unique Fulltext
unit	varchar(10)		Yes	*NULL*		Change Drop Primary Index Unique Fulltext
description	varchar(50)		No			Change Drop Primary Index Unique Fulltext
location	varchar(10)		No			Change Drop Primary Index Unique Fulltext

- **list** - the list that the variable belongs to
- **segment** - the segment that the variable belongs to
- **mnemonic** - a unique mnemonic to channel 1 for the variable
- **dimension** - describes the dimension of the variable:

  x - the variable is a x-by-1 matrix

  x,y - the variable is a x-by-y matrix
- **type** - a string that describes the bit-stuffing of the variable
- **scaler** - a floating point value to scale all the values by
- **unit** - the engineering unit of the variable
- **description** - an English description of the variable
- **location** - where the variable is located in the guidance system's memory, not used for analysis, but may be used for future applications

2.7 Channel 3 Setup

Channel 3 is the most popular interface used for analysis testing. Because of this, it also has the largest number of different data formats and making a standardized

dictionary for it was the most challenging. The data in channel 3 is all simple one-dimensional variables. Some of the variables are 32-bit double words. Through the course of this thesis, with the help of Draper's engineers, I developed a list, segment structure similar to channel 1 for channel 3.

ch3variables Table

mnemonic	varchar(10)		No			Change	Drop	Primary	Index	Unique	Fulltext
description	varchar(50)		No			Change	Drop	Primary	Index	Unique	Fulltext
ch3high	varchar(10)		Yes	*NULL*		Change	Drop	Primary	Index	Unique	Fulltext
ch3low	varchar(10)		Yes	*NULL*		Change	Drop	Primary	Index	Unique	Fulltext
list	varchar(30)		Yes	*NULL*		Change	Drop	Primary	Index	Unique	Fulltext
segment	varchar(30)		Yes	*NULL*		Change	Drop	Primary	Index	Unique	Fulltext

- **mnemonic** - a unique mnemonic to channel 3 for the variable

- **description** - an English description of the variable

- **ch3high** - the ch3locations mnemonic of the high word, NULL if variable is only a 16-bit word

- **ch3low** - the ch3locations mnemonic of the low word

- **list** - the list that the variable belongs to

- **segment** - the segment that the variable belongs to

ch3locations Table

Field	Type	Attributes	Null	Default	Extra	Action
mnemonic	varchar(10)		No			Change Drop Primary Index Unique Fulltext
address	varchar(4)		No			Change Drop Primary Index Unique Fulltext
rate	tinyint(4)	UNSIGNED	No	0		Change Drop Primary Index Unique Fulltext
wordf1	tinyint(3)	UNSIGNED	Yes	*NULL*		Change Drop Primary Index Unique Fulltext
wordf2	tinyint(3)	UNSIGNED	Yes	*NULL*		Change Drop Primary Index Unique Fulltext
wordf3	tinyint(3)	UNSIGNED	Yes	*NULL*		Change Drop Primary Index Unique Fulltext
wordf4	tinyint(3)	UNSIGNED	Yes	*NULL*		Change Drop Primary Index Unique Fulltext
wordf5	tinyint(3)	UNSIGNED	Yes	*NULL*		Change Drop Primary Index Unique Fulltext
wordf6	tinyint(3)	UNSIGNED	Yes	*NULL*		Change Drop Primary Index Unique Fulltext
wordf7	tinyint(3)	UNSIGNED	Yes	*NULL*		Change Drop Primary Index Unique Fulltext
wordf8	tinyint(3)	UNSIGNED	Yes	*NULL*		Change Drop Primary Index Unique Fulltext
wordf9	tinyint(3)	UNSIGNED	Yes	*NULL*		Change Drop Primary Index Unique Fulltext
wordf10	tinyint(3)	UNSIGNED	Yes	*NULL*		Change Drop Primary Index Unique Fulltext

- **mnemonic** - a unique mnemonic for every 16-bit word in channel 3, referenced to by either ch3high or ch3low from ch3variables, this mnemonic is not for the variable

- **address** - where the variable is located in the guidance system's memory, not used for analysis, but may be used for future applications

- **rate** - the rate that the variable is transmitted at

- **wordf1 ... wordf10** - the word number to find the data at in the frame, NULL if the word does not appear in the frame

ch3units Table

mnemonic	varchar(10)		No			Change Drop Primary Index Unique Fulltext
unit	varchar(10)		No			Change Drop Primary Index Unique Fulltext
scaler	double		Yes	*NULL*		Change Drop Primary Index Unique Fulltext

- **mnemonic** - the mnemonic of the variable from ch3variables

- **unit** - the engineering unit of the variable

- **scaler** - a floating point value to scale all the values by

2.8 Telemetry 1 Setup

Telemetry 1 is the most complex of all the interfaces. Because of classification issues discussed in Section 2.4, all variables in telemetry 1 have descriptions of "CLASSIFIED", units of "BITS", and scalers of "1". As transmitted, the telemetry 1 interface has no hierarchical arrangement to the variables within it. Through the course of this thesis, with the help of Draper's engineers, I developed a list, segment structure similar to channel 1. Telemetry 1 variables can either be simple one-dimensional variables, array variables, matrix variables, low throughput variables, or high throughput variables.

A low throughput variable is a variable that is only transmitted at 2 Hz. This is accomplished by the mod50 field in telemetry 1 data. The variable is only present in its frame locations if the 50th word in the frame has the variable's corresponding mod50 value. This essentially lets some physical word locations in telemetry 1 store 50 times as many variables as normal. Telemetry 1 is by far the largest data set because of this feature.

A high throughput variable is a variable that is transmitted at a faster rate than the interface. For instance, if a variable was a 200 Hz variable, then the variable would be transmitted at times 1, 1.5, and 2 1/100ths of a second. Because frames in all interfaces are only transmitted at 100 Hz, the variable's intermittent 1.5 value will be placed at a second word location on every frame. Any analysis program must then fetch this extra location and interpolate it into the correct intermittent time value.

tlm1variables Table

Field	Type	Attributes	Null	Default	Extra	Action
mnemonic	varchar(10)		No			Change Drop Primary Index Unique Fulltext
list	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
segment	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
longword	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
dimension	varchar(10)		No			Change Drop Primary Index Unique Fulltext
description	varchar(40)		No			Change Drop Primary Index Unique Fulltext
rate	int(4)		No	0		Change Drop Primary Index Unique Fulltext
unit	varchar(15)		No			Change Drop Primary Index Unique Fulltext
scaler	double		No	0		Change Drop Primary Index Unique Fulltext
bits	varchar(10)		No			Change Drop Primary Index Unique Fulltext
flagmode	varchar(10)		No			Change Drop Primary Index Unique Fulltext
mod50	varchar(4)		No	0		Change Drop Primary Index Unique Fulltext

- **mnemonic** - a unique mnemonic to telemetry 1 for the variable

- **list** - the list that the variable belongs to

- **segment** - the segment that the variable belongs to

- **longword** - describes the size of the variable:

  0 - the variable is a single 16-bit word

  1 - the variable is two 16-bit words, words in tlm1locations are mnemonic(pos)_H and mnemonic(pos)_L

- **dimension** - describes the dimension of the variable:

  x - the variable is a x-by-1 matrix, tlm1locations mnemonics are mnemonic(1) ... mnemonic(x), if dimension is 1 then the tlm1locations mnemonic has no position suffix

  x,y - the variable is a x-by-y matrix, tlm1locations mnemonics are mnemonic(1,1) ... mnemonic(x,y)

***x** - the variable is a high throughput variable with multiple locations in each frame, tlm1locations mnemonics are mnemonic(*1) ... mnemonic(*x)

- **description** - an English description of the variable
- **rate** - the rate that the variable is transmitted at
- **unit** - the engineering unit of the variable
- **scaler** - a floating point value to scale all the values by
- **bits** - a string describing the bits in each word that the data is encoded in
- **flagmode** - a string describing what flag modes of the guidance system in which the variable can appear, not used for analysis, but may be used for a future application
- **mod50** - used for 2 HZ variables, the variable only appears in frames where the 50th word is equal to this value

tlm1locations Table

Field	Type	Attributes	Null	Default	Extra	Action
mnemonic	varchar(15)		No			Change Drop Primary Index Unique Fulltext
wordf1	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
wordf2	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
wordf3	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
wordf4	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
wordf5	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
wordf6	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
wordf7	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
wordf8	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
wordf9	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
wordf10	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext

- **mnemonic** - the mnemonic derived from the mnemonic, dimension, and longword fields of tlm1variables to describe the particular word

- **wordf1 ... wordf10** - the word number to find the data at in the frame, 0 if the word does not appear in the frame

2.9 Telemetry 2 Setup

Telemetry 2 is a similar interface to telemetry 1. Because of classification issues discussed in Section 2.4 all variables in telemetry 2 have descriptions of "CLASSIFIED", units of "BITS", and scalers of "1". As transmitted, telemetry 2 interface has no of hierarchical arrangement to the variables within it. Through the course of this thesis, with the help of Draper's engineers, I developed a list, segment, section structure similar to channel 1. The section is an extra level of hierarchy in telemetry 2 beneath a segment. Telemetry 2 variables can be either simple one-dimensional variables, array variables, matrix variables, or high throughput variables. See Section 2.8 for a discussion of high throughput variables.

tlm2variables Table

Field	Type	Attributes	Null	Default	Extra	Action
mnemonic	varchar(10)		No			Change Drop Primary Index Unique Fulltext
list	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
segment	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
section	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
dimension	varchar(10)		No			Change Drop Primary Index Unique Fulltext
description	varchar(40)		No			Change Drop Primary Index Unique Fulltext
rate	int(4)		No	0		Change Drop Primary Index Unique Fulltext
unit	varchar(15)		No			Change Drop Primary Index Unique Fulltext
scaler	double		No	0		Change Drop Primary Index Unique Fulltext
bits	varchar(10)		No			Change Drop Primary Index Unique Fulltext
flagmode	varchar(10)		No			Change Drop Primary Index Unique Fulltext

- **mnemonic** - a unique mnemonic to telemetry 2 for the variable

- **list** - the list that the variable belongs to

- **segment** - the segment that the variable belongs to

- **section** - the section that the variable belongs to

- **dimension** - describes the dimension of the variable:

  x - the variable is a x-by-1 matrix, tlm1locations mnemonics are mnemonic(1) ... mnemonic(x), if dimension is 1 then the tlm1locations mnemonic has no position suffix

  x,y - the variable is a x-by-y matrix, tlm1locations mnemonics are mnemonic(1,1) ... mnemonic(x,y)

  ***x** - the variable is a high throughput variable with multiple locations in each frame, tlm1locations mnemonics are mnemonic(*1) ... mnemonic(*x)

- **description** - an English description of the variable

- **rate** - the rate that the variable is transmitted at

- **unit** - the engineering unit of the variable

- **scaler** - a floating point value to scale all the values by

- **bits** - a string describing the bits in each word that the data is encoded in

- **flagmode** - a string describing what flag modes of the guidance system in which the variable can appear, not used for analysis, but may be used for a future application

tlm2locations Table

Field	Type	Attributes	Null	Default	Extra	Action
mnemonic	varchar(15)		No			Change Drop Primary Index Unique Fulltext
wordf1	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
wordf2	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
wordf3	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
wordf4	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
wordf5	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
wordf6	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
wordf7	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
wordf8	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
wordf9	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext
wordf10	tinyint(4)		No	0		Change Drop Primary Index Unique Fulltext

- **mnemonic** - the mnemonic derived from the mnemonic and dimension fields of tlm2variables to describe the particular word

- **wordf1 ... wordf10** - the word number to find the data at in the frame, 0 if the word does not appear in the frame

Chapter 3

Processing

3.1 Introduction

The second goal of the Petscope project is a processing environment for PETS data. Previously at Draper, many engineers relied on testing data delivered in PETS format; however, stripping and processing a PETS file requires the attention of a very experienced data analysis engineer to manually ensure correct alignment of merged data files.

The problem is further complicated because processing PETS data previously took about two hours. This is because the process is a series of scripts and program written by many different programmers over the past 20 or so years. These scripts and programs must often be recompiled to be readjusted for different PETS dictionaries or to produce plots of different variables. Some of the legacy programs can not be recompiled on modern personal computers which makes the task even more daunting. The process produces many intermittent processing files, is not formally described in any kind of internal document, and requires inside knowledge so that only an experienced engineer can perform the task.

The idea is that with Petscope any engineer would be able to process his PETS files in a matter of minutes rather than sending it to a highly skilled test engineer and waiting a couple of hours for the data to come back. One can quickly see how big of a time sink this is because if a certain switch was forgotten to be turned on for

a test, it could take a couple of hours for the engineer to realize the mistake.

The Petscope processing is implemented in Matlab because of its compatibility with a wide variety of data processing algorithms and visualization used throughout the lab. The Petscope process can be performed in two different modes. The first mode is a Matlab GUI where the engineer will be able to open a PETS data file, connect to the MySQL database that stores the PETS dictionary for processing it, and then via a hierarchical menu be able to select what variable they are interested in. The resulting variables will be processed with the results being displayed in a Figure window or output to a standard Matlab MAT file. Most engineers however are usually only interested in looking at a set group of variables over a large number of tests. The second mode of Petscope is designed for this purpose. Once the engineer selects the group of variables he or she is interested in from the GUI mode, he or she will be able to save this grouping to the database. Under the second Petscope mode he or she will be able to invoke this predetermined processing via Matlab scripting rather than having to go through the GUI over and over again. It gives the new ability to easily plot data from multiple runs on the same graph.

3.2 Modularity Requirements

One of the main requirements of Petscope will be its ability to adapt to different testing environments at Draper. Draper's testing procedures are currently being upgraded, and so the interaction that Petscope will have with other programs, scripts, and procedures is not fixed.

Essentially, Petscope is a piece of software that knows how to fetch meaningful variables out of binary PETS data. What will change, however, is how Petscope receives this binary data, and what it does with the processed results. Petscope must thus be engineered in a way that it can utilize multiple input engines and multiple output engines. More importantly, in the future when other engineers have to develop different engines, they should be able to just create a few simple functions that input or output data to their needs, without having to know or understand most of the

code involved in Petscope.

The field of computer science normally meets software requirements like this with a concept known as object oriented programming (OOP). This approach has been largely popularized over the past years with the languages C++ and Java. Petscope unfortunately though is not written in C++ or Java, and instead Matlab. While the choice to use Matlab is a wise one because it is much better suited for data processing and visualization than C++ or Java are, unfortunately Matlab is a scripting language and not a programming language. Essentially Petscope needs to utilize classes for input engines and output engines, and then every time when a new input or output must be developed, the software engineer can just extend the abstract inputEngine and outputEngine class and overload the functions Petscope calls from the class. Since Matlab does not have the concept of a class or member functions in it, I thus needed to develop a way to best simulate this idea in Matlab.

The Petscope code will be distributed within Draper as source code, and not as a binary. Matlab gives you the ability to compile the code and distribute it as a binary, but this is unnecessary for Petscope. When a new user wants to use Petscope, they will be given a directory with all the Petscope code in it. Then they must add this directory's path to the Matlab PATH variable. This variable tells Matlab where to search for the scripts when you type them into the console or call them from another script. So the way Petscope achieves this concept of a class is to make the abstract class directories inside the Petscope directory. At runtime, Petscope will look at each of these abstract class directories, and get a listing of each of the directories within it. These directories represent the extended classes that overload all of the functions in the abstract class, and within them are the scripts which represent the overloaded functions of the class. So this means when a new input or output engine has to be developed for a new way of processing PETS data within Draper all that must happen is the software engineer must clone a directory of a preexisting engine, and then customize all the scripts within the directory to work with the new form of input or output. Petscope chooses which "extended class" to use for the functions by adding and removing that particular class' path to the Matlab PATH variable.

This thus enables Petscope to not know anything about how a particular input or output engine works, or even how many such engines exist. All of this is figured out at runtime.

The negatives of this approach are worth mentioning. While this approach can allow some modularity, it is far from a compiler checking to make sure that each class has the proper functions overloaded. Essentially, if Petscope calls a function in a class that does not exist or has syntax errors in it, Petscope will crash just like any other Matlab script would at runtime. While this is not ideal, this would presumably only happen during engine development, so the engineer would immediately know that their engine code has problems in it. When Matlab crashes, it prints its stack trace so the developer will be able to quickly check the broken function. Also, during development the engineer can just call his or her own functions from within Matlab in the same manner that Petscope will during runtime. This will enable engineers to quickly debug their engine without having to boot up Petscope every time.

3.3 Input Engines

Petscope has currently been developed with two input engines. Both engines have been developed within the requirements described by Section 3.2. Any input engine must overload the following functions:

- **openEngine** - Called before the engine is ever used. Normally this function is used to open up some kind of data connection and store any type of intermediate variables needed for data fetching. Engines should store variables in global variables with arcane names.

- **getCh1Word** - This function takes a list, segment, and array or word numbers to retrieve from the frames in the list and segment. The function is expected to return an array of the word values in chronological order, as well as a time vector associated with the word values. In the case when multiple words from the same time frame are requested the words should

be returned in increasingly numerical order by word number.

- **getCh3Word** - This function takes a list of frames and a list of word numbers to retrieve from all the frames in the frame list. The function is expected to return an array of the word values in chronological order, as well as a time vector associated with the word values. In the case when multiple words from the same time frame are requested, the words should be returned in increasing numerical order by word number.

- **getTlm1Word** - This function takes a list of frames and a list of word numbers to retrieve from all the frames in the frame list. The function is expected to return an array of the word values in chronological order, as well as a time vector associated with the word values. In the case when multiple words from the same time frame are requested, the words should be returned in increasing numerical order by word number.

- **getTlm2Word** - This function takes a list of frames and a list of word numbers to retrieve from all the frames in the frame list. The function is expected to return an array of the word values in chronological order, as well as a time vector associated with the word values. In the case when multiple words from the same time frame are requested, the words should be returned in increasing numerical order by word number.

- **closeEngine** - This function is called after all the data has been fetched from the engine. This function will normally close data connections and deallocated intermediate variables.

3.3.1 Database

One of the major other projects in the testing group is upgrading how PETS data is recorded from the missile. Historically, PETS data was streamed out of the guidance system and placed onto magnetic tape. Getting the data off of this tape required physically walking the tape over to the confidential analysis computer. This was yet

another bottleneck in the analysis process. In addition, the data collection computer is out of date and ready for retirement. The solution being developed is to log the data into a MySQL database. This database is separate from the Petscope databases, however for convenience is served off of Sylvester also. Unfortunately, inserting into a database requires writing to the hard disk, which is extremely slow. Benchmark tests have shown MySQL can only handle about 6,000 INSERTs a minute, which is too slow for the 100 Hz data that comes out of the guidance system. So this database logging system will be taking a temporary binary file produced from a new C program in the lab which basically probes the guidance system for data during simulations. This process will be automated so that the database logging happens right after the simulation is over. The benefits of having the data in a database are significant, the most important one being that programs that use this data will not have to read in an entire binary file for processing. These files can be up to 100 MB, and Petscope's file input can take up to four minutes to input these files. The other benefit is that the data is stored in the database in records consisting of frame and word numbers. This means applications that utilize it can fetch only the word and frame combinations they need.

Currently this database logging system is only able to handle channel 3 data because the hardware and software mechanisms to handle the other interfaces have not been developed yet. The database input engine of Petscope can successfully retrieve channel 3 data from the database. On the other three interfaces, it simply throws an error saying that this interface is not available from the database engine. After other engineers make the logging system compatible with the other interfaces, the associated retrieval functions in this input engine can be correctly overloaded, and Petscope will work with them properly.

3.3.2 Binary Files

Original versions of Petscope were made to read binary files that came off the magnetic tapes of the original PETS system. While the database engine will most likely completely replace the magnetic tapes, an output engine to analyze binary files is still

important to have. Old files will still exist for a period of time, and remote testing locations may have these binary files for a while. Because of this, a binary files output engine has been developed. The engine can produce data from all four interfaces.

Unfortunately though because of its file based nature, the engine is very slow. It takes approximately four minutes to open a file. This is still many times faster than legacy analysis systems that Petscope will be replacing.

3.4 Output Engines

Output engines are how Petscope returns the processed data back to the user. Essentially these output engines take the processed variable data, and output it into a form convenient to the user. Petscope has currently been developed with two output engines. Both engines have been developed within the requirements described by Section 3.2. Any output engine must overload the following functions:

- **openEngine** This function is the first function called in an output engine. Normally it is for initializing some type of output data source.

- **outputVariable** This function is called once for each variable that is being output.

- **closeEngine** This function is called after Petscope is finished with the output engine. Normally it is for closing some type of data source.

3.4.1 M File

The most basic form of output is the M File engine, which essentially just takes all the variable structs (which consist of the variable data, a corresponding time axis, and a short string descriptor of the variable), and binds them into a MAT file. With this MAT file the user can then take the data and do whatever kind of manipulations and analysis they want on it.

3.4.2 Figures

The Petscope visualization mechanism is done in the Figures output engine. This output engine enables engineers to look at MATLAB figure windows with four variables in them each. With these figure windows, engineers can use MATLAB's built-in functionality for zooming, touching up, and saving figure windows.

The figures output engine only works with simple one-dimensional variables. This is because there is not a standardized way to look at complicated matrix variables. Most of these variables involve complex transformation functions to make sense of the data. Petscope is designed to be a basic analysis program, so adding all this customized functionality particular to specific variables would contradict the goals of Petscope. Instead, these variables should be output via the M File output engine, and then can be transformed and visualized in external MATLAB scripts.

Chapter 4

Conclusion

The true legacy of Petscope will be actually using it in some of the guidance system tests in development over the coming years at Draper. This is essential to Petscope for a variety of reasons, the obvious one being that it will demonstrate that Petscope works and raise confidence in it. The more important reason however is it will let various engineers in Draper see this new tool and become hooked on it so they will start using it in all of their tests. The integration of Petscope into lab testing procedures is the main goal of the Petscope project.

4.1 Future Directions

One of the main tests that I foresee Petscope getting involved in is a project called Hardware-In-The-Loop (HWIL). The Trident missiles are divided into a few major subsections. The main two sections Draper is involved with are the Inertial Measurement Unit (IMU) and the Electronics Assembly (EA). The IMU is essentially a collection of sensors that the EA reads in order to make computations to direct the missile's flight. The challenge in testing a missile is that a test launch is very time consuming, expensive, and is very hard to repeat. This then leads to other testing procedures such as testing the guidance system on a centrifuge or an F-14 aircraft. These procedures help, but still present challenges in pinpointing issues to the IMU or the EA. The initial step in HWIL is to simulate the IMU, and then pass this data to

the EA to see how it reacts. Petscope has the potential to be a tremendous aid here because it will enable the engineer to quickly see what the EA did after the simulated testing sequence. As Petscope becomes integrated into this testing procedure the benefits of having done the processing component in Matlab will become apparent because now the whole test can be a simple Matlab script: simulate an environment in Simulink and send it to the EA, process the data out of the EA with Petscope, do statistical and quantitative tests in Matlab on the data that Petscope produces.

4.2 Contributions

I feel that Petscope has made the following significant contributions to research at Draper.

- **Standardization** The Petscope database has the most comprehensive, standardized, and up-to-date knowledge of the variables stored in the Channel 1, Channel 3, Telemetry 1, and Telemetry 2 interfaces in Draper. More significantly, this knowledge is in the first ever machine-usable format. This will enable quicker and simpler updating to the data than would ever be conceivable without the help of Petscope.

- **Hierarchy** Petscope contributes the first ever hierarchy of Channel 3, Telemetry 1, and Telemetry 2 variables. These hierarchies enable researchers for the first time to quickly see all the variables related to a particular topic. In addition, these hierarchies provide a standardized way for all testing applications to display a user interface to select variables. This would be unwieldy without the hierarchies because the user would have a flat list of hundreds of variables to choose from. In addition, creating such a large, flat menu inhibits a personal computer's performance.

- **Automation** Petscope provides the first automated method for post-processing PETS data. In addition, Petscope is designed to be flexible

with a wide variety of inputs and outputs.

Bibliography

[1] *MySQL Reference Manual.* http://dev.mysql.com/doc/mysql/en/Reserved_words.html.

[2] Kathleen Granchelli et al., editors. *Explorations*, chapter Draper Leads Modernizing of the Navy's Trident MK6 Guidance System. Draper Laboratory's Office of Public and Employee Communications, 2004.

[3] George Reese et al. *Managing & Using MySQL.* O'Reilly and Associates, Inc., 2002.