UPPSALA
UNIVERSITET

IT 13 015
Examensarbete 15 hp
Mars 2013

Real-Time Resource Monitoring for Poly/ML Processes

Magnus Stenqvist

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Real-Time Resource Monitoring for Poly/ML Processes

Magnus Stenqvist

Poly/ML, a run time system of the Standard ML language, has with its latest version 5.5 included a statistics feature which provides information about the running system and ML programs running in that system. This feature provides information about the program threads running in the system, memory usages, garbage collection information and user defined data. The problem is that the statistics information is only accessible through the use of the language C. No way exist today that reads and displays this data automatically. It would be very convenient for developers, to have an easy access to this statistic data, to better evaluate their ML programs running in the Poly/ML system. This report presents a Java application that collects the statistics data of Poly/ML and displays it in a graphical environment easy to observe by the user. This application will help developers develop programs with Poly/ML, by letting them observe how their programs affect the Poly/ML run time system.

Handledare: Tjark Weber
Ämnesgranskare: Justin Pearson
Examinator: Olle Gällmo
IT 13 015
Tryckt av: Reprocentralen ITC

Contents

Chapter 1

Introduction

1.1 Background

A monitoring program can be an invaluable tool for every developer to monitor the behavior of programs and their effect on the system. Many of the program languages today have support for multi threading and a lot of programs uses this feature. It would help tremendously to have a monitor program to observe those threads together with other statistic data.

The idea for this project is to develop a monitoring program that displays information of Poly/ML [1] specific processes to aid the developer to write better code and what follows is more robust software. Poly/ML is a compiler and run-time system for the functional programming language Standard ML [2]. The data that Poly/ML provides is: number of running threads, size of the local heap, number of garbage collections etc.

1.2 Monitor programs

The task of a monitor program is to retrieve data from any system and display it on a computer screen, mostly data that is received over a period of time and where the data is constantly shifting. Monitor programs provides a historical presentation of the data and helps the users to get a clearer picture of how the data is changing over time, often by using some graphical representation such as graphs to plot the data. For example in an industrial environment there is CitectSCADA [7] which can be used to monitor industrial applications. There are all kinds of monitor programs depending on

the context but the common theme is to help the user interpret important information and potentially detect any anomalies in the system or improve the system. Monitor programs can also issue events in the form of warnings or notifications either locally or remotely, so that the user does not have to be present at all times. This project focuses on programs that monitors data that is generated from an application or operating system.

1.2.1 System monitor programs

From a computer scientific view a system monitor program monitors the resources of an operating system [8] whether it be, the states of the processes currently in execution, the memory consumption, the network traffic history or the workload of the CPU. A system monitor program displays, to name a few things, the state of all processes currently being alive together with their names and process ids, how much virtual and regular memory they currently are using and their priorities. The state of a process is always pending between running or waiting. Mostly a system monitor program is used to verify the state of a process. In case of any problem e.g. a process that has occupied to much of the computers memory the process can be killed through the system monitor program.

The idea with a system monitor program is for the user to get a clearer view of the operating system, and to some extent manipulating its resources, for example starting and stopping processes. The user can with the help of the system monitor program more easily analyze the efficiency of the system and potentially improve it or/and detect strange behaviors. System monitor programs can range from text based ones like htop, see fig. 1.1 for the Linux system, to more graphical ones like Windows Task Manager, see fig. 1.2.

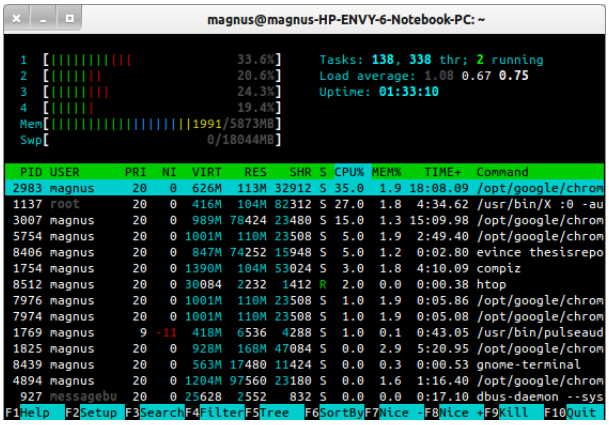

Figure 1.1: The system monitor htop for the Linux system

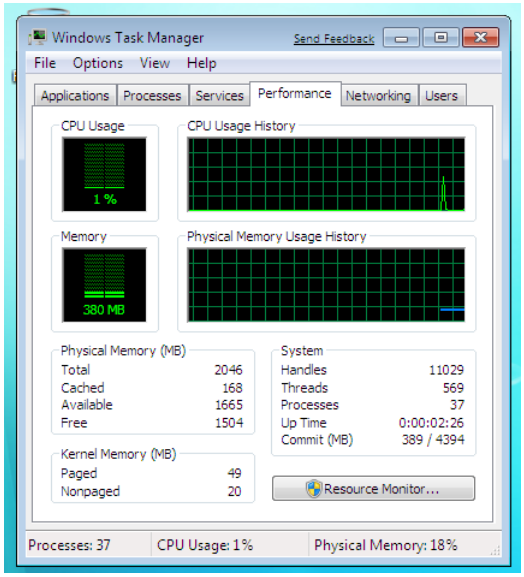

Figure 1.2: The task manager in Windows 7

1.2.2 Monitoring other applications

As previously stated a monitoring program can monitor just about everything. This section will describe monitor programs that monitor other applications. Basically every application that provides some sort of statistic

data can be monitored. It really depends of what type of data there is if it is going to be useful to the user or not.

An example is JConsole [15] (see fig. 1.3) a fairly simple monitor program for Java programs that monitors the Java Virtual Machine. It displays various information such as, number of threads currently running, number of classes currently loaded, how much of the heap size available and how much of it that is currently being used. This information can be used to observe running Java programs, to detect memory leaks, which classes are used for an application etc.

A slightly more advanced monitor program also for the JVM is VisualVM [20] (see fig. 1.4) that in addition to the data mentioned previously has features such as: how much time has been used for each method in the program, which class type occupies most memory, and the option to perform a heap dump. From the heap dump the user can see how many instances of every class type that exists currently and the user can even examine the classes instances to observe its data members. With the help of these kind of monitor programs the programmer can write more efficient Java programs which uses the possibilities of a multi threaded environment. A monitor program can be used as an additional tool to the debugger. The monitor program JConsole uses Java Management Extensions (JMX) which is an API that provides information about the Java Virtual Machine. JMX is written in Java.

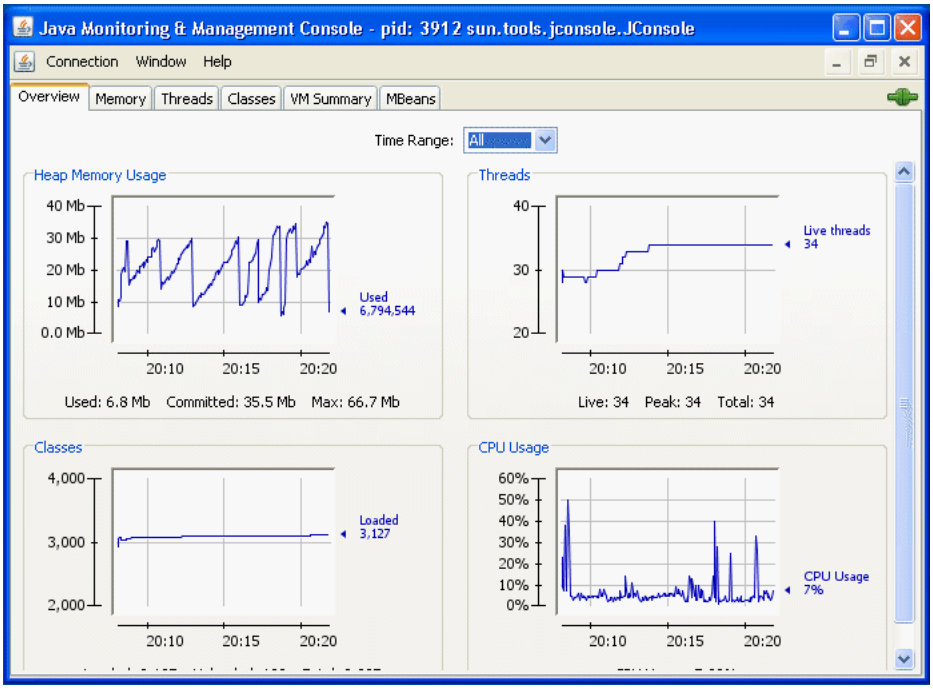

Figure 1.3: JConsole

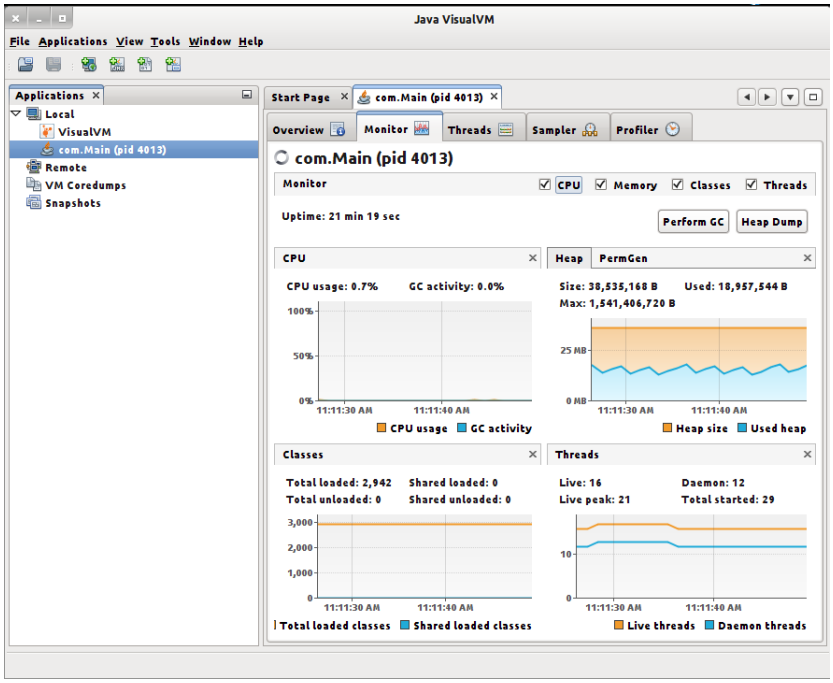

Figure 1.4: Visual VM

1.3 Poly/ML and functional programming

Poly/ML [1] is a compiler and run time system of the Standard ML language. The user can either type in ML code for evaluation directly in Poly/ML or load existing ML programs from files. Standard ML [2] is a modern implementation of the ML programming language, which is a functional programming language popular among compiler writers and in the development of theorem provers. The compiler is written in Standard ML while the runtime system is written in C++. Poly/ML supports the 1997 Definition of Standard ML (ML97) but also supports additional features, for example a windows programming interface, a symbolic debugger, a C language interface, and for this project: statistics information of running Poly/ML applications.

Functional programming [3] is a programming paradigm that leans heavily towards functions, where the functions is more similar to that of mathematical notations of functions compared to functions in imperative languages. In a functional programming language such as ML, functions returns the out-

put solely based on the inputs, avoiding state changes from a global scope. Functions should avoid side effects that is, only the parameters that is passed to the function should determine the outcome of the function, not any external data which can be the case in an imperative programming language like C++. Eliminating side effects can make it easier to predict the behavior of a program and having immutable data types will make the programs more thread safe, since an object once created cannot be altered later. However Standard ML is not purely functional, it allows for side effects but it is common for programmers to avoid them. Some features of a functional programming language includes pattern matching, type inference, immutable data types and in some cases shorter code compared to imperative programming languages.

1.4 Problem description

In version 5.5 of Poly/ML a new statistic feature has been introduced. During run time the Poly/ML run time system offers statistic information that can be retrieved by an outside source. This statistic data includes number of running threads, size of the local heap, number of garbage collections etc. This statistic data will help the programmer evaluate their programs and make them better. For example if there is a high number of threads in a program, that is waiting for a mutex, this will indicate that code sections hold by the mutexes is too long and therefore should be reprogrammed.

The problem with this statistic data is that, in its current state, the data is stored in memory as structures of the language C without any additional descriptive data. Without inspection of the source code, this data is not understandable. It is wanted to retrieve and show this data in a more clearer way and using information about the statistics as it is provided by Poly/MLs source code comments and documentation.

As a solution to this problem, this report presents a graphical monitoring program that retrieves the statistical data from all the running Poly/ML processes and displays it in a comprehensible and descriptive way, similar to the JConsole and JVisualVM programs. This will give the users all the benefits of having a monitor program when programming Poly/ML applications. The application is written in Java and C++ for Linux and Windows 7 64 bit and it is available for free as a repository on Bitbucket [6]. Mercurial [5] or a similar tool can be used to retrieve the sources.

To develop a well designed and structured application Iterative and Incremental development (IID) [9] has been used for the software development process. IID divides a project in a sequence of iterations where each iteration consist of the steps: requirement analysis, design, programming and test. The goal for an iteration is to release a stable and tested partially complete program. As this project consisted of only one developer the IID proved to be a good choice because it allowed to start creating a prototype which matched only a subset of the initial requirements and the prototype was constantly being refined with the steps described above. The requirement specification was created together with the student and the supervisor.

The final product is to be verified by the supervisor and the reviewer. One major application that could benefit through the use of this monitor program is Isabelle, a generic proof assistant program, written in Poly/ML and that uses multi threading for efficiency.

1.5 The statistic data available in Poly/ML v.5.5

- Multi threading information
  - Total number of threads
    - The total number of threads that are currently running inside of a Poly/ML process. Threads can be created and interrupted by the Poly/ML process.
  - Threads running ML code
  - Threads waiting for IO
    - Threads that are in a waiting state, waiting for IO (Input/Output)
  - Threads waiting for mutex
    - A mutex lets only one thread at a time execute a piece of code. If the number of threads that are waiting for mutexes is high it is an indication that the critical section, the code between locking and unlocking the mutex is too long and should be changed.
  - Threads waiting for a condition var

    * Condition variables are used in conjunction with mutexes to implement long-term waits for some condition to become true.
  - Special case - signal handling thread
    * A special purpose thread that waits for Unix-style signals and calls a signal handler registered using the Signal.signal function. There will almost always be one thread in this state except when it is actually in the signal handler in which case it will be "running". It is really included in the statistics only to make the numbers add up.
  - Number of full garbage collections
    * Garbage collection is an automatic memory management implementation. Whenever objects in the heap no longer are used by any process running, the garbage collectors purpose is to find and delete those objects to free up space in the heap.
  - Number of partial garbage collections
    * Partial (sometimes called "minor") GCs are fast collections which sweep up all the currently live cells created since the last partial collection and add the cells into a reserved area of the heap. Each partial collection adds more data to this area so periodically it is necessary to run a full ("major") garbage collection. This is more expensive than a partial collection but removes any cells that are now no longer accessible.

- Memory statistics
  - Total size of the local heap
    * The heap is a place in memory where all the variables created or initialized at run time are stored. The heap is local to a Poly/ML process.
  - Space free after last Garbage collection
  - Space free after the last full Garbage collection
    * Indicates the effectiveness of the garbage collection. If the free space after a full GC is too small this indicates that the overall heap may need to be increased. Normally this happens automatically but the user may limit the maximum heap size by command-line arguments to Poly/ML.

  - Size of allocation space
  - Space available in allocation area
    * The allocation area is the part of the heap where cells are allocated initially. As an ML program runs the space available will decrease until a GC is triggered. The partial GC empties this area by copying live cells into the longer-term area of the heap.

- Timer data
  - System and user times for the garbage collector as well as non garbage collections.

- User counters
  - User counters are there to allow for an Poly/ML application to record information along with the real time statistics data. The programmer could record the number of some objects that has been created to be able to observe it in the statistic data.

Chapter 2

Design

2.1 The software process

2.1.1 Iterative and incremental software development

Iterative and incremental software development [9] (see fig.2.1) is a foundation where several software development methods are built upon, including Scrum, Extreme Programming, Unified Process and Evo to name a few. IID and its extensions are called agile methods, where agile means for the developers to be adaptive to incoming changes in the project during the complete lifetime of the project.

The common theme of the IID methods is that a projects lifespan is divided into several iterations in sequence. Each sequence is in itself a small contained mini project, in which there are several steps similarly to the steps in the waterfall [9] model. The goal for an iteration is to release a stable and tested partially complete program which will then be evaluated by the client together with the developer. This allows for creating a prototype that in the beginning matches only a subset of the initial requirements and where the prototype later is constantly refined and extended for every iteration.

The benefit with IID is that errors can be found early in the project due to the prototype being a full workable part of the finished application and can therefore be thoroughly tested before continuing with the project. This technique is called Tracer Bullets [10] that is; get a fully working program from back end to front end without bothering to specify the complete system. Get it to work as a small program and work from there. With IID the cost of errors is at minimal even if these are found late in the project, because the

errors are much more likely to fixed in later iterations, as opposed to a waterfall type of project where the implementation part already is done. Because of that the IID methods are agile processes and uses adaptive (evolutionary) planning. This means that the planning of the project can adapt when more information about the project accumulates in contrast with predictive planning (in waterfall models) where all the planning is in the beginning and cannot change. Further drawbacks of the waterfall model is that often when the clients sees the product develop they change their minds and/or they have difficulty in the beginning of the project, stating what they want and stating all they want. Another drawback is the risk profile of the waterfall model. The longer the project goes on the higher the risk is to successfully tackle any problems should they arise. In the light of these drawbacks of the waterfall model, the agile methods seems all the more appealing.

2.1.2 The methods of IID

Scrum is the most popular method today, with daily team meetings called sprint meetings and self organizing teams. Scrum promotes autonomy during iterations, which means that no stake holders (clients) from the outside can change in the requirements during an iteration. Extreme programming focus on unit tests, pair programming, code revisions and oral communication. Unified process focuses on the riskiest requirements and try to get them done first, so called risk driven development. The thing that these methods has in common is that they all promote adaptive development, that is, under the projects lifetime, either adapt to the clients wishes when they arise and/or adapt to changes from an external source, for example new technologies available on the market.

This project has used a software development process that closely resembles that of the IID methods. The steps used in an iteration are requirements, design, programming, testing and evaluation and each iteration is approximately 1 week. Many ideas from the agile methods have been used including risk driven development, adaptive development, oral communication, autonomy during iterations etc. The benefit of an iterative process for this project has been weekly evaluation meetings with the supervisor, where a prototype has been evaluated and tested, with new requirements added or changed.

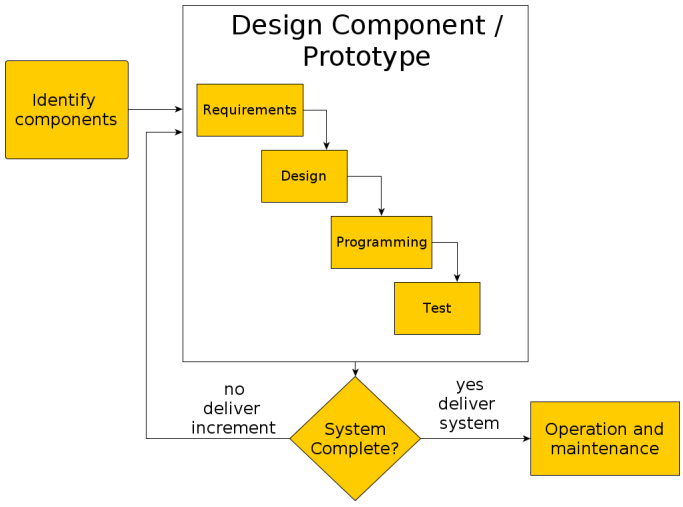

Figure 2.1: Iterative and incremental development

2.2 Interaction design

2.2.1 Who are the users?

The users of this application will be people with a lot of computer experience, most likely software developers, since they will know how to program in the ML language and use Poly/ML. This application will be a tool for the developer to use in his work. The interface will consist of a window environment, mostly used today for programs on the computer, with commonly used features such as buttons, toolbars, menus and drawn graphics. As the user is experienced in computers this will prove to be reasonable. The appropriate appearance of the interface is also based on what statistics data there is.

It is assumed that the user knows about the Poly/ML language, its multi threading features and a little more advanced “under the hood“ features of a programming language for example garbage collection. The user will be very experienced with the computer for interaction with the software, with the mouse and keyboard working as input and the screen as output. As this is a monitor program the major part lies on the output, but there will be input where the user will store and save log files and manipulate what data to be displayed.

2.3 Requirements

2.3.1 Validation and verification

Validation [11] means checking if the program does the right thing according to the clients wishes while verification [11] often is used internally by the programmer to verify that the program works without unexpected behavior. Validation of the prototype in this project has been checked by the supervisor after every iteration while the verification part has been performed by the developer. During the project the design on a high level must hold both for the customer or end user and also that the design of the program is internally consistent.

The requirements has grown during this project with new requirements being added at every iterations evaluation step. There are two types of requirements, functional requirements and non functional requirements [12]. The functional requirements describes what the application should do and the non functional describes what constraints there are on application as well as the software development process.

2.3.2 Non functional requirements

The application must run on the Linux operating system. Porting it to other operating systems, could be future work. The application must be written in Java, due to the fact that Java has better and faster tools for creating GUI graphics. Poly/ML stores the statistic data as compiled C structs to memory. Because of this the memory layout of the statistic data will be dependent on the C compiler of Poly/ML, since different compilers handles C structs in different ways. Since the Poly/ML statistic layout cannot be altered without modifying its source code the requirement is for Java to read the C structs in some way.

2.3.3 Functional requirements

- Continually poll data
  - The application must constantly read (poll) the available statistic data for every Poly/ML process that is running. There should be a list of currently running processes, displayed with their process

ids, where the user can click to select which process data to show. The statistic data should be displayed in graphs, one graph for each value in the statistic data. With graphs the user can see how an input behaves during the complete lifetime of a Poly/ML process.

- Logging data
  - When a process is finished it will disappear from the process list, so a logging facility must be present for the user to store the current data. This also requires an option to load previously saved log files.
- Display the most recent data
  - The graphs must not only display the statistics data arbitrarily but also display the most recent data first. The system (current) time on one of the graphs axis will let the user now when Poly/ML generated this information. This will make it convenient for the user to know at what time events happened during program execution.
- Adequate update frequency
  - The statistics data must, as mentioned above, be read and displayed in real time during a regular time interval. An interval of one second is feasible. If the interval is smaller there would be an unnecessary amount of data time stamps to store for each statistic value and the log files would be huge and therefore time consuming to load.
- Flexibility of the sizes of the views components
  - The components of the interface must be re sizable when the main window size changes. It is a common future among desktop applications and it is widely supported by Java swing which uses layout managers to do this. This will allow the user to enlarge the window and the graphs when needed.
- Scrollable graphs

  - As there could be thousands of values for each graph collected during a run, they could not all possibly fit in each graph. For example, if the application has been running for 12 hours and it retrieves the data every second, that will mean 43200 time stamps for each graph to display. As the sizes of the graphs are technically limited to the width of the computer screens some option to scroll back and forth through the graph data is needed. The monitoring could go on for a long time, and old data values should not be deleted just because they cannot fit in the graph.

- Selectable time ranges
  - A requirement is to let the user choose different time ranges depending on how much data the application has collected. The user can zoom in or out on the data when needed.

- Display customization
  - To prevent that the screen feels too cluttered with all the graphs, the user must have an option to customize what data to show. Some data could be of less importance for the time being, so the user must have the option to hide that data.

- Keyboard short cuts
  - To minimize the number of mouse clicks keyboard short cuts is a requirement.

The projects requirements should follow principle 11 of the Agile Principles which states that simplicity is essential, that is, do the simplest thing that could possibly work [9]. All requirements do not have to be known at the start of the project, but the earlier they are known the better.

2.3.4 Tool kits used

The Java library Swing [13] will be used for the interface. The Java Swing library is a platform independent class library to create graphical user interfaces (GUI) for a Java program that is recognizable to a large audience. It supports for example buttons, menus, text fields in a window environment and the components can easily be extended for customization. The free Java

library JFreeChart [18] is a class library that will be used to create and plot time series graphs. These graph classes comes bundled with features such as zooming, save graph as image and colors customization. This classes can easily be extended and customized according to the needs of the application.

2.3.5 A sketch of the design

An important aspect for a monitor program is that it should show as much information as possible to the user without being too cluttered. As few clicks as possible is desirable and keyboard short cuts are a must. A well supported feature in Java Swing is that the layout can be kept consistent even when the window resizes. See fig. 2.3 for a sketch of the design.

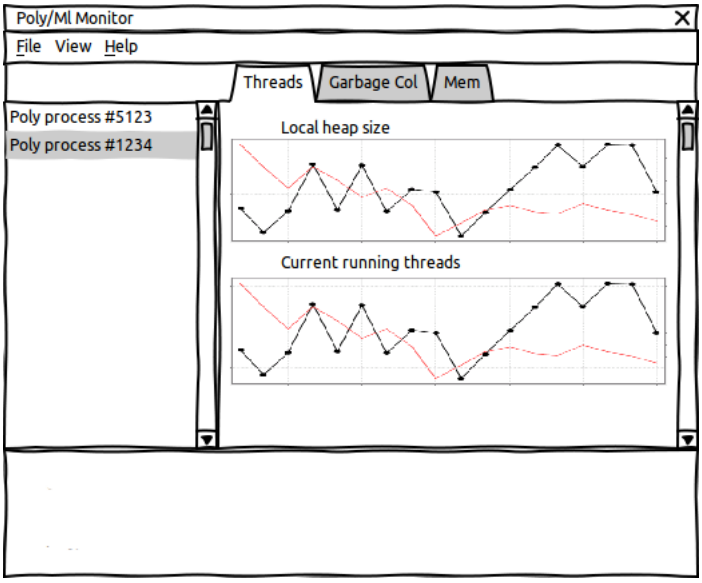

Figure 2.2: A sketch of the design for the monitor application

2.3.6 Architectural design and Model View Control

The architectural design [11] is a follow up to the requirement specification. While the requirements states *what* the program should to do, the architectural design describes *how* it should do it.

A model-view-control (MVC)[14] based design will be used for the internal design of this program. The MVC is used to separate the data (model) from the user interface (view) with the controller handling the input. The model consists of all the application data which then can be requested by the view for representation. For examples this makes it easier to develop different views for a model, or to change the model without affecting the view. It is good to handle all the input from the user at one place and that is the controllers job. The controller receives user input via the view and then makes appropriate actions based on these inputs, often to update the models data.

The reason for using MVC is to be able to work on the interface without bothering with how the data gets there or working on the model knowing that the interface wont be affected. More about the actual implementation of the architectural design available in the implementation section.

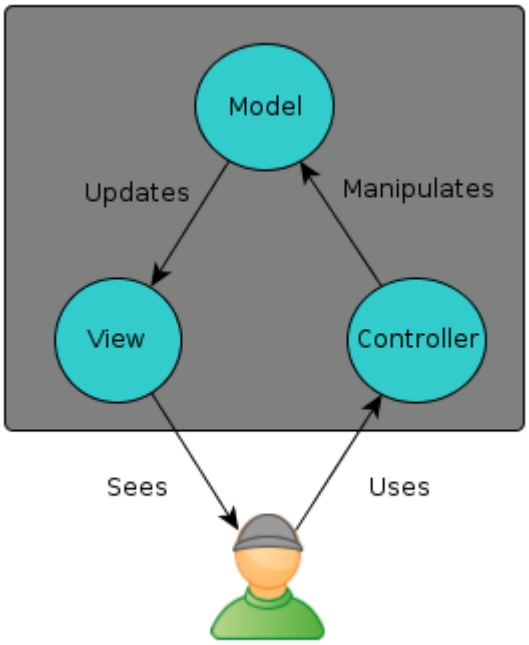

Figure 2.3: Model View Control

Chapter 3

Implementation

3.1 How to get the data to monitor

3.1.1 How Poly/ML provides statistic data

Poly/ML is written in C/C++ and uses a C struct called *polystatistics* (defined in *polystatistics.h*) as a container for the statistics data. The struct has several arrays with integer values, where each element corresponds to a statistics value of some kind. Each running Poly/ML process has a corresponding struct filled with that process statistic data.

```
//The memory layout of the statistics data

typedef struct polystatistics {
    unsigned long psCounters[N_PS_COUNTERS];
    ...
    ...
    int psUser[N_PS_USER];
} polystatistics;
```

This structure is saved either in shared memory or in memory mapped file in the file system depending on whether the operating system is Linux or Windows.

3.1.2 How the data is provided in Linux

mmap [19] is a POSIX-compliant Unix system function that allocates a part of the memory to be used by processes to read from and write to. *mmap* is similar to *malloc* except that while *malloc* allocates a part of the memory at a random address, *mmap* uses a file in the file system as an identifier for the allocated part in the memory. If a process wants to read or write to this part in memory it can use *mmap* with the path to the file as one of the arguments. Other arguments needed with *mmap* is the size to allocate and some read and write rules. *mmap* returns the starting address of the memory and the process can start read or write to this memory.

Each *polystatistics* struct is memory mapped, using *mmap*, to one file per running Poly/ML process, stored in the Linux $HOME/.polyml directory. For example a running Poly/ML process with PID 23452 is saved as a file with the name *poly-stats-23452*. To read these files the linux functions *readdir* and *opendir* functions can be used to get the names of all files in the directory. These names are then used with *mmap* to retrieve the start address of each files allocated memory. The returned start address from *mmap* is casted to struct *polystatistics* and can then be used by the process by just reading the structs member values to obtain the statistics data.

3.1.3 How the data is provided in Windows

The *windows.h* functions *OpenFileMapping* and *MapViewOfFile* are used to read from and write to a part in the page memory of the operating system. Each Poly/ML process uses these functions to store its corresponding *polystatistics* struct to memory, using its process id as an identifier for the space in memory. To retrieve the statistic data for each Poly/ML process the windows function *CreateToolhelp32Snapshot* is used to get all running processes on the operating system. Then the processes are traversed, all the Poly/ML processes are identified and their process ids are stored. With the process ids the *OpenFileMapping* and *MapViewOfFile* can be used to read the memory spaces allocated for each Poly/ML process.

3.2 Java and JNI

Although it is easy to retrieve the data with C code, how to make the Java code read that memory to display it in the GUI, when it is stored as a C

struct? One unreliable option is to try and read the C struct from Java using the class *DataInputStream* or similar. This technique reads from the memory directly cell by cell. Unfortunately, because of the C standard which does not enforce padding or aligning members of a struct each struct may end up different in memory between different compilers. The C compiler may rearrange the members of the struct and when Java reads the cell in memory expecting a certain statistic value it may end up getting a totally different one. Different compilers will also perform padding differently. Padding is adding extra space in memory between struct members. Even if the memory can be read satisfactory with Java on one machine, it is not portable. If the members of the C struct were to be saved to the memory in some order one at a time without using a struct, and then read with Java this would then be portable. Another option that let Java read C structs from memory is for Java to execute native code (in our case C++) to successfully handle the data structs and then transform the data for Java to read.

Java Native interface [16] is a method that lets the user integrate native languages from within Java code. This can be useful if you want to run for example a C module for performance reasons and then get some data from the C module to your Java program, or maybe there is a huge library of C files that there is no time to rewrite into Java which which then can be used by native code instead. In this project C++ code will retrieve the statistics data that has been stored in a C struct in the memory file, and send the data back to the Java code.

3.2.1 The usage of JNI in this application

1. In the class *ReadStat*, the method *writedatatofile* is declared with the keyword native and this method is the link between Java and the C code. Once this method is called the native code will be executed. The *System.loadLibrary* statement loads a shared library that contains the compiled native code.

```
//Readstat.java

class ReadStat {

    ....
```

```
    public native void writedatatofile();

    try {
System.loadLibrary("ReadStat");
    } catch (UnsatisfiedLinkError e) {
System.err.println("Native code library failed to load.\n" + e);
System.exit(1);
    }

    ....
}
```

2. The command *javah ReadStat* is used to create a header file, that will be referenced to in the native code. This header file will automatically include JNI libraries and the declaration of the *writedatatofile* method.

```
//com_control_ReadStat.h
#include <jni.h>

 ....
//Declaration of the native method
JNIEXPORT void JNICALL Java_com_control_ReadStat_writedatatofile
  (JNIEnv *, jobject);

 ....
```

3. Then the native method can be defined in the native code as

```
//writedatatofile.cpp

#include <stdio.h>
#include "com_control_ReadStat.h"

JNIEXPORT void JNICALL Java_com_control_ReadStat_writedatatofile
    (JNIEnv *env, jobject obj)
{
    // Native C code ready to be executed!
}
```

3.2.2 Alternatives to JNI

JNA [17] is a class library that lets Java programs access native code without the requirement to generate native headers with the javah command. The JNA class library uses native functions allowing code to load a library by name and retrieve a pointer to a function within that library. The developer uses a Java interface to describe functions and structures in the target native library. However with JNA for this project, it is still required to compile shared libraries from the native code to be used by Java. Because of the little amount of native code required for this project (only one function call) it makes little difference whether to go for JNI or JNA.

3.2.3 Portability issues

Java code is compiled into byte code which are then interpreted by the Java Virtual Machine. The benefit with this is that a Java program can run on any system that supports Java as opposed to C++ where the compiler needs to compile versions for every system. A benefit with native code is that it often runs faster. Because the Poly/ML source code is unmodifiable for this project the requirement of JNI is forced even though this will loose the platform in dependency.

3.3 The architectural design

3.3.1 The JNI and Java communication

The method *writedatatofile* returns nothing, instead it retrieves Poly/MLs statistic data, using the methods described in above sections 3.1.1 and outputs it to a plain text file. Once that is done, the Java code can read this file using common file processing tools. It is the controllers job to execute the writedatatofile method at regular time intervals, to obtain a new version of the text file, read the text file with the statistics data and send the data to the model. The changes in the model is immediately reflected in the view.

In fig. 4.1 the layout of the programs architectural design is shown where native and Java code is clearly separated as well as model, view and control. Here the compiled shared library is communicating with the Java source through the use of JNI. The reason for using a text file is that it is simple, and there is no need to create JNI specific Java objects in the C++ code to

return to the Java side. Instead we rely on purely C++ code on the native side.

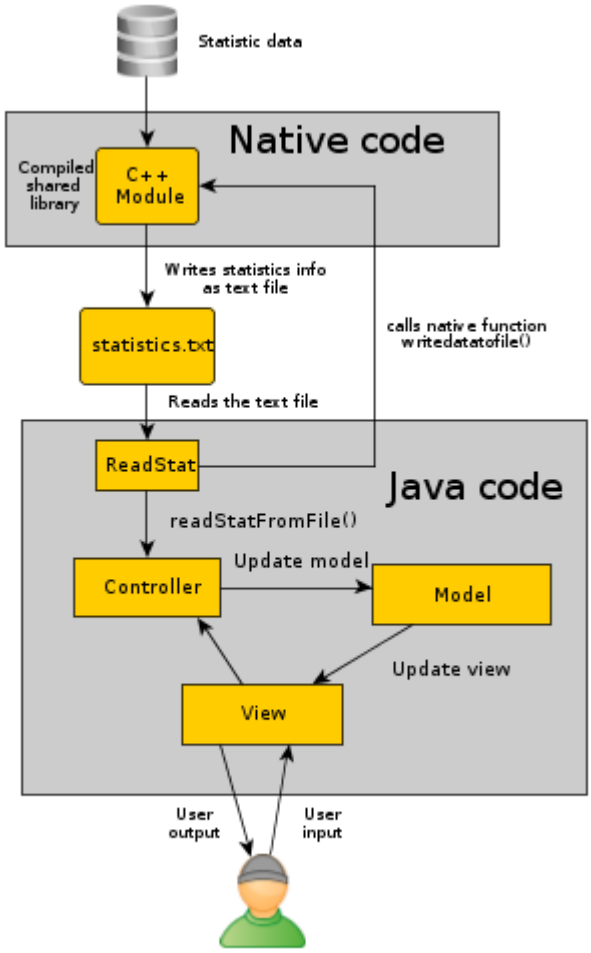

Figure 3.1: Layout of the native and JAVA communication

3.3.2 The class layout

The classes are divided into three groups, controller, view and model according to the MVC design paradigm. The *MainControl* class receives the user actions and depending on them, updates the view and/or manipulates the

model classes. It is also polling statistics data from any running Poly/ML processes during regular intervals.

The model classes are basically containers for settings of the application that needs to be stored during run time. The *ProcessListModel* class stores which Poly/ML processes that are running and the *ProcessList* class in the view shows the processes in the list. The *ProcessStat* class contains a time stamp of the statistics data for all Poly/ML processes and is used by the controller to update the graphs. *ProcessDisplay* stores which graphs to be displayed on the screen and *ProcessStatLogger* logs data to temporary log files during run time. When user wants to save a log file, the data in the temporary log files are used.

In the view group the *MainWindow* class is the window frame, with the menu bar and title bar. The *MainPanel* is basically a canvas where the other two *ProcessList* and *ProcessGraphs* classes are drawn on. *ProcessGraphs* is a tabbed panel displaying all the graphs.

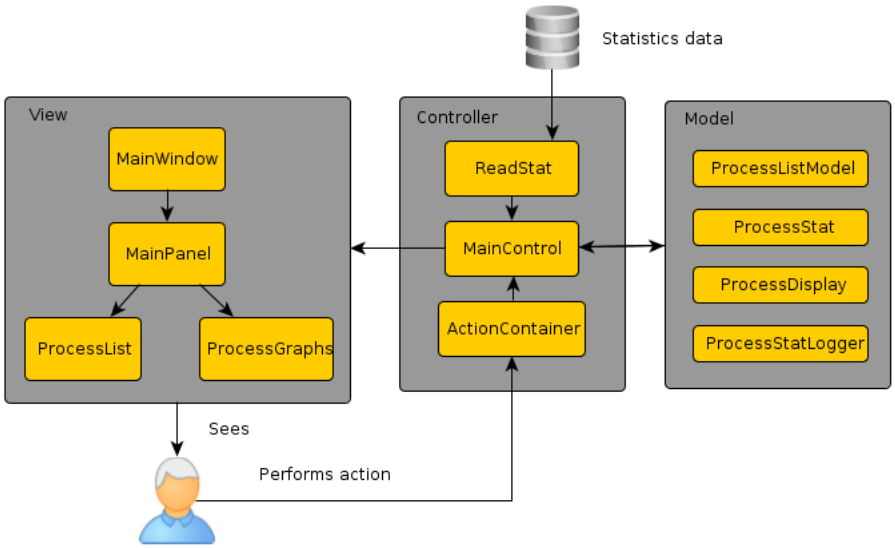

Figure 3.2: Layout of the most important Java classes

Classes	Lines of code
ActionContainer	446
ProcessStatLogger	320
Graph	283
ProcessGraphs	256
MainControl	241
MainWindow	144
TabbedGraphPanel	168
ProcessViewFrame	156
MainPanel	142
ProcessDisplay	135
ProcessListModel	134
ReadStat	126
Statistics	105
CustomButton	103
ProcessList	78
GraphPanelGroup	63
ProcessStat	57
Main	23
Total	2980

3.3.3 Testing

During the implementation part testing has usually been performed by using debuggers and extensive user tests performed by the developer. Due to this project's small size, special unit tests where not considered necessary, although the Extreme Programming method would recommend it. In the Java code, whenever bugs and errors occurred, the Java debugger jdb was used to find the cause of the problem. The application had to run for a long time monitoring sample ML programs and was checked for memory leaks and potentially faults reading the statistic data. The application was evaluated when monitoring several running Poly/ML programs that executed threads, to generate some dummy statistics data to read. The monitor program JVisualVM was used to look for memory leaks and to observe that the right amount of instances where running.

The native C code had to be tested in isolation due to the lack of debugging facilities for JNI. When native code is executed in Java code and an error happens, there is no information from the JVM where the error is,

just that the error has happened somewhere inside the native code. This is a serious drawback. Luckily the only communication between Java and the native code is only one function call so the native code can be tested in isolation by a C debugger before connecting it with Java. The native code also writes the statistic data to a text file, which can easily be examined for correctness. After each iteration user tests have been performed by the supervisor to evaluate the requirements of the program.

Chapter 4

Result

4.1 Conclusion

All of the requirements set up in the requirement specification were met due to the use of IID. In the evaluation part of every iteration more requirements were added as old ones were met. This is a part of the adaptive planning in the agile strategy.

One requirement was that the program should always be reading and displaying statistic data. The application continually polls any information available of running Poly/ML processes until the user terminates the program. If a Poly/ML process starts it is immediately shown in the applications process list available to click on. If there are only one process running that process statistic data is shown, otherwise user can choose to click on any currently running process in the list to show the most recent statistic data in the graphs which was a requirement.

The update frequency is every second which was a requirement, however no option exists to allow the user to edit it. The application has a logging facility due to the requirement, which lets the user save the data, that has been observed during the Poly/ML monitors up time, to a log file. The main view consists of a list of all the current Poly/ML processes that are running on the system and a tabbed graph pane of the currently selected process's. The tabbed pane is divided into four main categories.

An option to set the time range is provided so that more or less data points can be shown in the graphs. If the user wants to see previously recorded data that does not fit in the graph there is the option to scroll through the time

range of the graph, for example to watch data values from a previous time. These two options fulfill the time range and scrollable graphs requirements.

In the menu bar there are the menu items File, View and Help. The File menu provides options to save the currently collected data of a process to a file which can be opened for later inspection. The View menu lets the user set various view settings i.e. what data to show, which was a requirement called display customization. There is support for keyboard short cuts for easier access to the menu options. The window is resizable to let the user see more graphs.

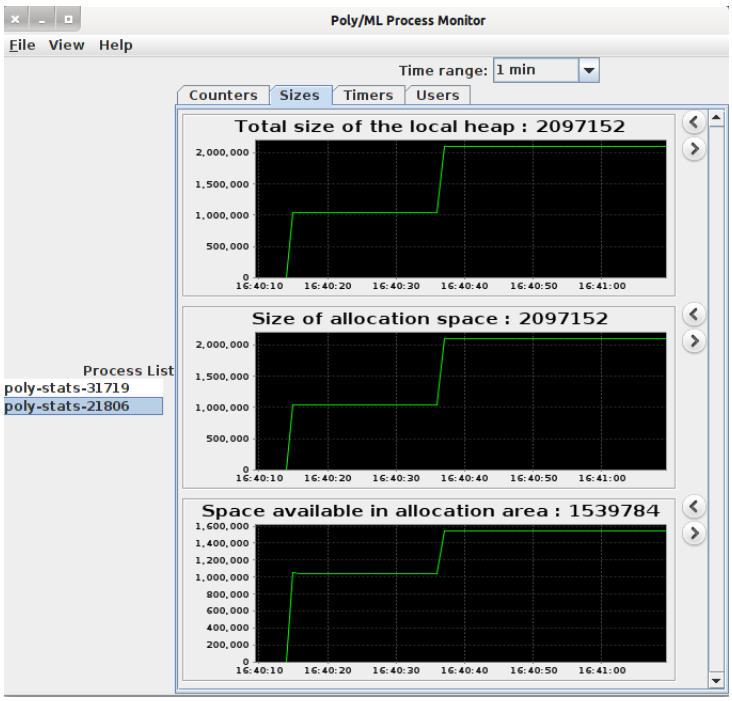

Figure 4.1: The final application: a Poly/ML monitor application

4.2 Discussion

The purpose of this project was to find a way to retrieve and display the statistics data available for the Poly/ML run time system. In its previous state of Poly/ML no such way existed because the format of the statistic data was stored in memory as structures of the language C. This project has

aimed to solve this problem and the solution is a tool, a monitor program that grabs this previously unreadable statistics data and transforms it into graphs with descriptive headers clearly stating the meaning of each data value. The application tries to monitor the statistic features of the Poly/ML environment in an easy to use program that resembles many of the monitor programs that exist today and where some of them are described in section 1.2.

This application will help ML programmers using Poly/ML to evaluate their programs and write more efficient code. It will be an additional tool together with Poly/MLs debugger to detect faulty behaviors of programs especially memory and thread issues. Potentially memory leaks, an ineffective garbage collection or too many threads sharing some memory leading to an ineffective program could all be derived from the monitor program. The programmer must by himself understand and interpret the available statistic data to be able to make the correct actions. With this tool it will make the programmers job much easier. As an additional feature Poly/ML provides the option for the programmer to create his own statistic data which will add more usage flexibility. For example a programmer can store the count of some objects that he is created during run time and this will immediately be displayed in the monitor application.

This project used a software methodology that closely resembled the iterative and incremental development method. As IID is a broad field with various extension methods (Scrum, XP), this suited fine, since this project was only a 10 week project and using Scrum for instance would be overkill. The benefits of IID in general was the iterative process and prototyping. For each of the four weeks during implementation, the application went through design, implementation, test and evaluation. The idea to create a small working prototype in the beginning, that constantly was incremented as requirements where added is close to how the agile methods work. After each iteration (end of week) the application was evaluated with the supervisor to get the most important and riskiest requirements done first. An example of a risky requirement was getting the application to read and display one single statistic value for one Poly/ML process only. Once that was achieved more functionality was added, such as support for more statistics data and the handling of multiple running Poly/ML processes.

The technique of JNI was used, because the statistics data provided by Poly/ML was hard coded in C structures. This led to the drawback of not using only Java code, which would have been preferable because of Javas

write once, run everywhere motto. There are possibilities to read the members from a C struct from Java but it proved unreliable, since C compilers stores the structs differently on different compilers as well as on different architectures. If these problems are fixed this application would be an even nicer addition to the Poly/ML distribution.

Chapter 5

Future work

5.1 Further portability

Even though this project fulfilled the requirements in the requirements specification there is always ways to improve the application in the future. Currently the application has support for the platforms Linux (32 and 64 bit) and Windows 7 (64 bit) but should also have support for Mac OS X, Windows 8, Windows Vista and so forth. For the application to work with different architectures, the native code (C++ library) that Java uses, needs to be compiled for each of the architectures and shipped together with the Java classes. In Windows 7 64 bit, for example, a DLL-file need to be compiled for a 64 bit architecture together with the headers that the DLL-file requires. For the application to use native code, this is a drawback, as pointed out earlier in the discussion section, compared to using pure Java which can be run on any system as long as there is a Java Runtime Environment available.

5.2 Pure Java?

Because of the issues with native code in Java there would be a major improvement if the statistic feature was saved in some other way by Poly/ML other than hard coded C structs. A suggestion for the next version of Poly/ML could be saving the data to a simple text file. Now Poly/ML writes to the statistics memory whenever something happens, for example a new thread is created and this is of course more efficient than opening and writing to a text file whenever something happens. To be more efficient with

a text file there could be a separate process polling the statistic memory and outputting a simple text file every second or so. A text file would solve the portability issues regarding the java program. A text file is simple and almost every programming language has techniques for reading text files. Then this application could be pure Java without any native code.

5.3 Maintenance issues

As a part of the descriptive information of the Poly/ML statistics data is hard coded into Java, future changes of the Poly/ML statistics data require changes in the monitor application source as well. The reason for this is that the descriptive strings, that describe the data, for each statistics value is hard coded in Java. If these descriptions could be implemented in Poly/ML instead, retrieving and showing this data could be more automatic.

5.4 Improvements

The application could benefit from the following future improvements.

- Additional configurable display of process data.
  - Drag and drop to sort the graphs in a window.
  - Possibility to create new tabs in the tabbed panel and drag graphs there.
- Have an option to show multiple series in the same graph.
- The application could notify the user when some value goes under or above a certain threshold. This could occur either remotely through the means of for example mobile text messages or occur locally on the computer.
- Currently the statistic data is retrieved every second. An option could be to let the user choose the polling interval, by selection between a couple of options. This would also need the graphs time axis to be updated to a finer resolution than one second.

Bibliography

[1] Poly/ML [on-line] Available at `http://www.polyml.org/index.html` [Accessed April 25, 2013]

[2] Milner R., Harper R., MacQueen D., Tofte M. *The Definition of Standard ML*. The MIT Press, 1997.

[3] Bosworth R. 1995 *A practical course in functional programming using Standard ML*. McGraw-Hill Publishing Co, 1995.

[4] Isabelle [on-line] Available at `http://isabelle.in.tum.de/index.html` [Accessed April 25, 2013]

[5] Mercurial [on-line] Available at `http://mercurial.selenic.com/` [Accessed April 25, 2013]

[6] Bitbucket [on-line] Available at `https://bitbucket.org/` [Accessed April 25, 2013]

[7] CitectSCADA [on-line] Available at `www.schneider-electric.com` [Accessed April 25, 2013]

[8] Nutt G. *Operating Systems*. Addison-Wesley Professional, 3rd edition, 2003.

[9] Larman C. *Agile & Iterative Development*. Addison-Wesley Professional, 1st edition, 2004.

[10] Hunt A., Thomas D. *Agile & The Pragmatic Programmer - From journeyman to master*. Addison-Wesley Professional, 1st edition, 1999.

[11] Dix A., Finlay J., Abowd G. D., Beale R. *Human-Computer Interaction*. Prentice Hall, 3rd edition, 2004.

[12] Preece J. *Interaction Design - Beyond human-computer interaction*. Wiley, 1st edition, 2002.

[13] Zukowski J. *The Definitive Guide to Java Swing*. Apress, 3rd edition, 2005.

[14] Model View Controller [on-line] `http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller` [Accessed April 25, 2013]

[15] JConsole [on-line] `http://openjdk.java.net/tools/svc/jconsole/` [Accessed April 25, 2013]

[16] Java Native Interface [on-line] `http://docs.oracle.com/javase/6/docs/technotes/guides/jni/` [Accessed April 25, 2013]

[17] Java Native Access [on-line] `http://en.wikipedia.org/wiki/Java_Native_Access/` [Accessed April 25, 2013]

[18] JFreeChart [on-line] `http://www.jfree.org/jfreechart/` [Accessed April 25, 2013]

[19] mmap [on-line] `http://en.wikipedia.org/wiki/Mmap/` [Accessed April 25, 2013]

[20] Java Visual VM [on-line] `http://visualvm.java.net/` [Accessed April 25, 2013]