A PROFILER FOR TESTING JAVA PROGRAMMES
(Socket Server and the Analyser)

A PROJECT REPORT PRESENTED BY

NISHANTHA SRIPALI WEERAKOON

to the Board of Study in Statistics and Computer Science of the

POSTGRADUATE INSTITUTE OF SCIENCE

in partial fulfilment of the requirement

for the award of the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

of the

UNIVERSITY OF PERADENIYA

SRI LANKA

2003

ABSTRACT

Java programming language was first designed to be used as a controlling language for consumer electronic devices but later this was redesigned to be used as a platform independent programming language, which uses an interpreter that can be embedded in other applications. With this redesign, Java became very popular among, especially, web based application programmers. Java programmes began to grow larger and larger. In the meantime, most of larger Java programmes were not satisfactory in performance and tools were required to analyse and measure the performance of larger Java programmes.

Many tools were built for testing Java programmes and most of these tools used profiling as their approach. Almost all of the early Java profilers used an instrumented Java VM to obtain the necessary information. There were many drawbacks of those profiling tools so that the Sun Microsystems Inc. introduced a general-purpose profiling instrumentation, JVMPI, with their standard Java release. At the moment this is used as a standard for the development of profiling tools. The most interested areas of profiling are the memory usage, CPU usage and monitor contention. In this project a general-purpose memory profiler, which is useful in analysing memory leakage problems in a single Java VM is designed.

In this profiling process, JVMPI is programmed to provide information on class loading, object allocation, object release and object movement events. These data are then sent to the profiler front-end and stored there. Since memory problems can be identified by visualising the obtained information graphically, all information regarding the classes and objects are presented to the user through a graphical user interface. The system uses advanced Java features such as JVMPI to obtain the information from the Java VM, JNI to incorporate native codes written to receive information from the profiler agent and Swing to build a decent graphical user interface.

A makefile is provided to automate the compilation and installation of all the source files using Unix make utility.

TABLE OF CONTENTS