LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
Department of Information Technology

MASTER'S THESIS

PROTECTION OF PUBLICLY DISTRIBUTED SOFTWARE COMPONENTS

The Master thesis has been submitted for approval for the degree of Master of Science in Technology by the department council of the Department of Information Technology on 11 April 2007.

The supervisor from Motion Avenue company is Jarno Väkeväinen
The examiner is Professor, FT Kari Smolander
The second examiner is D.Sc. Uolevi Nikula

Lappeenranta 20.08.2007

Pavel Stepanov
Maauuninkuja 4d, 30
01450 VANTAA
050-3610338

ABSTRACT

Lappeenranta University of Technology
Department of Information Technology

Stepanov Pavel

Protection of publicly distributed software components.

Master's Thesis
2007
62 pages, 27 figures and 4 tables.

Supervisor: Jarno Väkeväinen
Examiners: Professor, FT Kari Smolander and D.Sc. Uolevi Nikula.

Keywords: Client-side architecture, software development, reverse-engineering, protection, object-oriented programming, methods of protection, obfuscation.

The nature of client-server architecture implies that some modules are delivered to customers. These publicly distributed commercial software components are under risk, because users (and simultaneously potential malefactors) have physical access to some components of the distributed system. The problem becomes even worse if interpreted programming languages are used for creation of client side modules. The language Java, which was designed to be compiled into platform independent byte-code is not an exception and runs the additional risk. Along with advantages like verifying the code before execution (to ensure that program does not produce some illegal operations) Java has some disadvantages. On a stage of byte-code a java program still contains comments, line numbers and some other instructions, which can be used for reverse-engineering.

This Master's thesis focuses on protection of Java code based client-server applications. I present a mixture of methods to protect software from tortious acts. Then I shall realize all the theoretical assumptions in a practice and examine their efficiency in examples of Java code. One of the criteria's to evaluate the system is that my product is used for specialized area of interactive television.

1. FOREWORD

This Master's thesis has been done for MotionAvenue International company. It is a research of protection for interactive television specialized software, based on Java code. It is shown that mixture of different protecting techniques (which for some reason has not been used before) can guard efficiently the whole programming complex.

I thank my supervisor professor Kari Smolander for support and valuable guidance. I would also like to thank my parents Vera and Arkadiy, who have helped and supported me throughout the years. I thank my wife Katja for her big love.

Lappeenranta 20.08.2006

Pavel Stepanov

Table of Contents

Terminology, symbols and abbreviations

TERMINOLOGY

Compiler – A compiler is a computer program (or set of programs) that translates text written in a computer language (the source language) into another computer language (the target language). The original sequence is usually called the source code and the output called object code.[1]

Compilation – Translation of source code into object code by a compiler.[1]

Cost – The additional run-time resources required to execute a program after a transformation has been applied to the program. [2]

De-compilation – Is a program transformation by which a high-level source code for an executable program is discovered. Decompilation is the inverse of program compilation.[1]

De-obfuscator – A tool designed to reverse obfuscating transformations that are applied to a program. [3]

Disassembly – Is the process of translating an executable program into its equivalent assembly representation. The greatest problem with disassembling is determining what is code (instructions) and what is data, as both are represented in the same way in current machines.[4]

Obfuscation – The process by which a program is made more difficult to understand, so that it is more resistant to reverse-engineering.[3]

Obfuscator – A tool designed to apply obfuscating transformations to a program. [3]

Potency – The degree to which an obfuscating transformation confuses a human who is trying to understand an obfuscated program. [4]

Reverse engineering – The analysis of an existing system regardless of ownership. It takes a finished product and recreates the knowledge needed to reproduce it. [2]

Resilience – The effort required to undo the effects of a transformation. [4]

Stealth – The degree to which the code added by an obfuscating transformation differs from the code in the original program. [4]

ABBREVIATIONS

Ear – Enterprise Archive (A file that contains an entire J2EE application including its components and deployment descriptors). [6]

GUI – Graphical User Interface.

IDE – Integrated Development Environment

IPG – Interactive program guides

ITV – Interactive Television

IVR – Interactive voice response

Jar – Java Archive (A file format that contains multiple files and is used to distribute a complete Java application). [5]

JDK – Java Development Kit.

JIT – Just-In-Time compiler [5]

JNI – Java Native Interface

JRE – Java Runtime Environment [7]

JSP – Java Server Pages

J2EE – Java 2 Platform, Enterprise Edition [6]

J2ME – Java 2 Platform, Micro Edition [8]

J2SE – Java 2 Platform, Standard Edition [5]

MIDP – Mobile information device profile

MMS – Multimedia Messaging Service (is a standard for telephony messaging systems that allows sending messages that include multimedia objects)

NVOD – Near video on demand

OpenGL – Open Graphics Library

SMS – Short Message Service (often called text messaging)

UML – Unified Modeling Language

VOD – Video on demand

War – Web Archive (A file that makes up a Java-based Web application including servlets, JSPs and other resources). [6]

1. Introduction

The reverse-engineering, which is defined as an analysis of an existing system is not illegal by itself. Moreover it is supposed that reverse-engineering helps to improve one's own software products. It becomes illegal in case it is used regardless of ownership. Reverse-engineering is based on reconstructing enough knowledge about the program to be able to modify it or use parts of it.

De-compilation is the process of turning object code backward into source code and object code is code which can be executed directly by real or virtual machine. For compiled languages like C or C++ the object code is executed on a hardware CPU whereas Java code runs on a virtual machine.

Almost all programs can be decompiled. Those programs written with interpreted languages (Java belongs to this group) can be decompiled easily. Java byte-code still contains some useful information: package names, names of all the classes, names of variables, parameters names and line numbers.

Due it intellectual property right and economical reasons, efficient protection of Java-based code is an important research area. This research deals with several techniques and approaches.

Previous economical researches [9], [10] show that billions of Euros are lost every year all over the world because of software de-compilation and computer piracy.

	Piracy Rates				Losses ($M)			
	2006	2005	2004	2003	2006	2005	2004	2003
WESTERN EUROPE								
Austria	26%	26%	25%	27%	$147	$131	$128	$109
Belgium	27%	28%	29%	29%	$222	$257	$309	$240
Bulgaria	69%	71%	71%	71%	$50	$41	$33	$26
Cyprus	52%	52%	53%	55%	$12	$13	$9	$8
Czech Republic	39%	40%	41%	40%	$147	$121	$132	$106
Denmark	25%	27%	27%	26%	$183	$199	$226	$165
Estonia	52%	54%	55%	54%	$16	$18	$17	$14
Finland	27%	26%	29%	31%	$149	$156	$177	$148
France	45%	47%	45%	45%	$2,676	$3,191	$2,928	$2,311
Germany	28%	27%	29%	30%	$1,642	$1,920	$2,286	$1,899
TOTAL EU	**36%**	**36%**	**35%**	**37%**	**$11,003**	**$12,048**	**$12,151**	**$9,786**

Figure 1. Economical losses, caused by reverse engineering. Source: [9], [10], [11]

Available protection approaches try to apply the same protection techniques even though each field of application has its own distinctive features.

Moreover, there is no any analysis on how the efficient protection of commercial software components can be done for different working areas. What are the pluses and minuses of each protection techniques for every separate working field?

This Master's thesis will try to:

- Classify the most common existing protection approaches.
- Emphasize on the appropriate protection techniques, which are up to date and have not become antiquated.
- Develop new protection technique, which can bring some newness and represents a mixture of different techniques.
- Develop the new software based on the theoretical proposals.
- Evaluate the efficiency of newly developed protection software.

All the theoretical assumptions will be implemented in a practice and as a test/implementation area I have chosen software for interactive television. The reasons of this choice are quite simple:

- Present-day television uses a lot of digital services, which are basically commercially distributed software components with on-screen visualization.
- These software components need protection from illegal customer's actions.
- Nowadays I work as software develop and create this kind of modules on my current workplace. I strictly understand the structure of system, all features and further improvements.

Organization of this Thesis

The remaining subsections of the thesis are arranged as follows:

Chapter 2 describes the particular features of a system to be protected. The main questions which should be answered in this chapter are following:

"What is the area of interactive television?"

"Why it is so important to develop a specialized protection system that takes into account all the distinctive features?"

Chapter 3 examines the different protection approaches and techniques.

The main question of this chapter is: “Whether the existing protection approaches have any meaningful disadvantages or why they are unsuitable in our particular case?”

Based on such evaluations new technique or mixture of existing techniques will be proposed.

Chapter 4 focuses on the practical implementation of the suggested protection approach. The chapter includes selection of programming tools, realization (UML diagrams, architecture diagrams and code).

Chapter 5 chooses the parameters to evaluate the system.

Some criteria should be chosen to evaluate the system. One of those criteria is that my product can be used for specialized area of software for interactive television.

Chapter 6 describes future work and possible improvements, which can make the developed product more effective and comfortable in use.

2. Application area for protection system.

Interactive Television is a common term for the mixture of digital technologies and usual television. Conjointly with traditional television environment ITV uses:

- Mobile devices
- Video recorders
- Computers
- Game consoles

The inclusion of mentioned devices into communication process between end-users and television makes a television increasingly interactive.

Simultaneously with using mentioned devices, interactive television involves a number of technologies, all of which have gripping affect on how the end product looks and works:

- The production tools and software
- The television set
- The transmission network or platform
- The set-top box

Mark Gawlinskii in his book *Interactive Television Production* [12] defines these terms as follows:

The production tools means programs to be broadcasted on ITV.

The television set means a device to display television programs. It can be a traditional television set or some personal computer with tv-tuner device.

The set-top box is a technical device receives and decodes digital television broadcasts into analogous form. Also set-top box can be used as a storage device for previously recorded TV-programs.

The transmission network means the network, delivering video signal in any existing format from television studio to end-users.

2.1 The specialized area of interactive television.

How to define precisely the area of interactive television?

- Television programming that allows viewers to participate in some way. This may involve voting for elimination of participants from a contest, picking the next action on a program, or choosing from a menu of content options. The return signal from homes may be via a touchtone telephone, the Web or directly over a two-way cable system. [12], [13]

Furthermore, most of interactive television is still under development. Interactive technologies in television will not replace traditional television technologies, but rather exist in parallel. Figure 2 presents the difference between traditional broadcast model and interactive model.

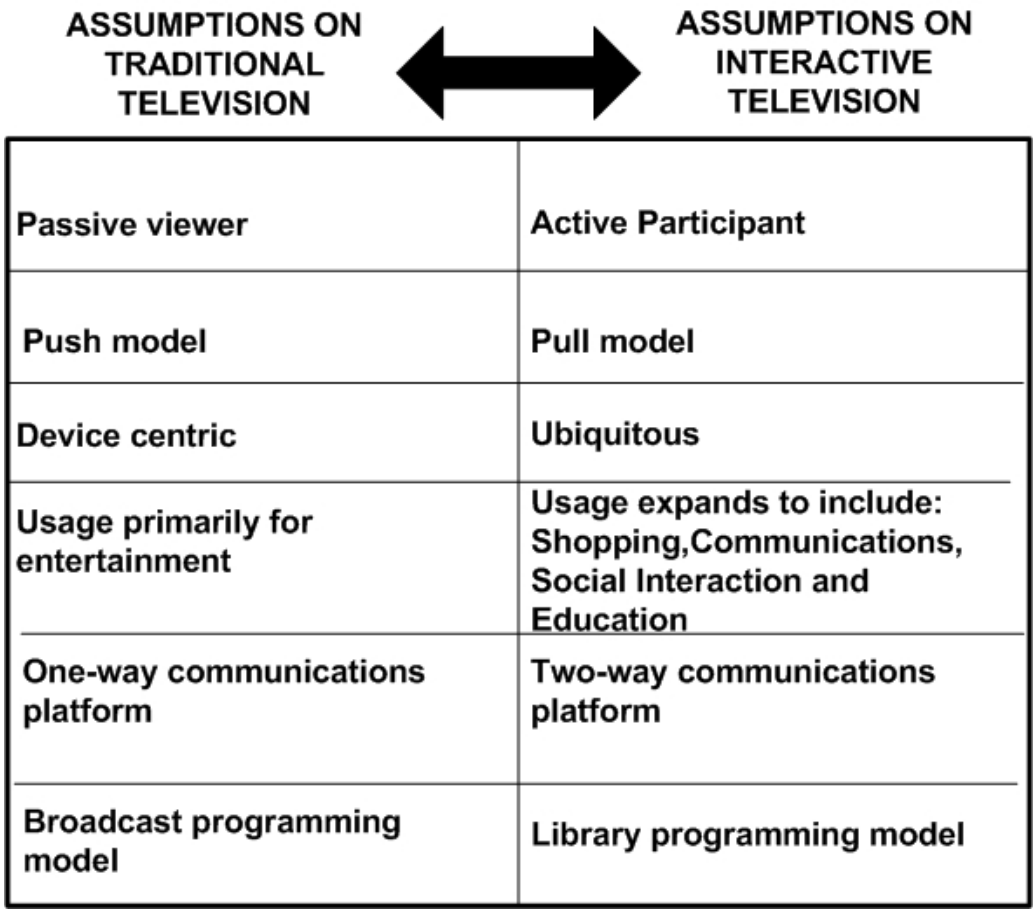


Figure 2. The differences between traditional and interactive television.

Source: [13]

The most important difference presented on the figure above relates to the broadcast programming model. Since ITV provides the possibility for end-user to

choose the concrete television program, it is called library programming model. Interactive television represents video on demand (also so-called VOD) behaviour [12], where viewers are allowed to download movies or any previously recorded TV-programs. This approach gives free rein to customers especially concerning time-shifting capabilities:

- Jump to specific video-frame whenever the end-user wants to do that
- Pause the program
- Fast-forward
- Rewind

Customers of interactive television services can watch some selected TV-program as many times as they want within a defined period of time (duration of that period depends from company providing interactive services). Mostly because of this fact the end-user can be called the active participant.

“Near video on demand” (also so-called NVOD) [13] systems stream the video in a way where customers are queued for the next start time which happens with some intervals. Video on demand systems are based on a pull model, while near video on demand systems is more like push model (traditional television) with some additional services provided by the television networks.

True pull modelled systems allow the end-users to make complicated enough actions:

- Listen and watch some educational courses and even generate some feedback
- Shopping
- Communication with similar end-users (chats and even conferences)

Even fully remote educational process can be organized with the help of interactive television, since it provides two-way communications mechanism.

To answer completely the preliminary formulated question: “Why it is so important to develop a specialized protection system that takes into account all the distinctive features” – let see the commercial size and commercial importance of interactive television industry which always need some kind of protection. The table below presents an amount of subscribers for digital interactive television only on United States market.

The most popular providers of digital interactive television are presented in table 1 (only for United States of America market).

Platform name	**Satellite or cable**	**Number of Subscribers. Approximate values**
Comcast Cable Communications	Cable	22 millions
AT & T broadband	Cable	14 millions
America-On-Line Time Warner	Cable	11 millions
DirecTV	Satellite	10 millions
DISH (echostar)	Satellite	9 millions
Charter Communications	Cable	6 millions
Cox Communications	Cable	6 millions
Cablevision (iO)	Cable	3 millions
Adelphia Communications	Cable	2 millions
Mediacom LLC	Cable	1 million
Insight Communications	Cable	1 million
Total		85 millions

Table 1. Amount of subscribers for interactive television industry on a market. Source: [14]

Table 1 shows that market of interactive television is huge. On example of United States of America it covers a quarter of population. This fact can be the obvious evidence of commercial industry importance.

2.2 Detailed structure of the system to be protected.

Below I will present the structure of real system for interactive television. The project is tested and realized as a real application at "MotionAvenue International" company (www.motionavenue.com) which produces software for interactive television.

The main idea is that end-users can manipulate the interactive content shown on TV by sending SMS messages to the gateway.

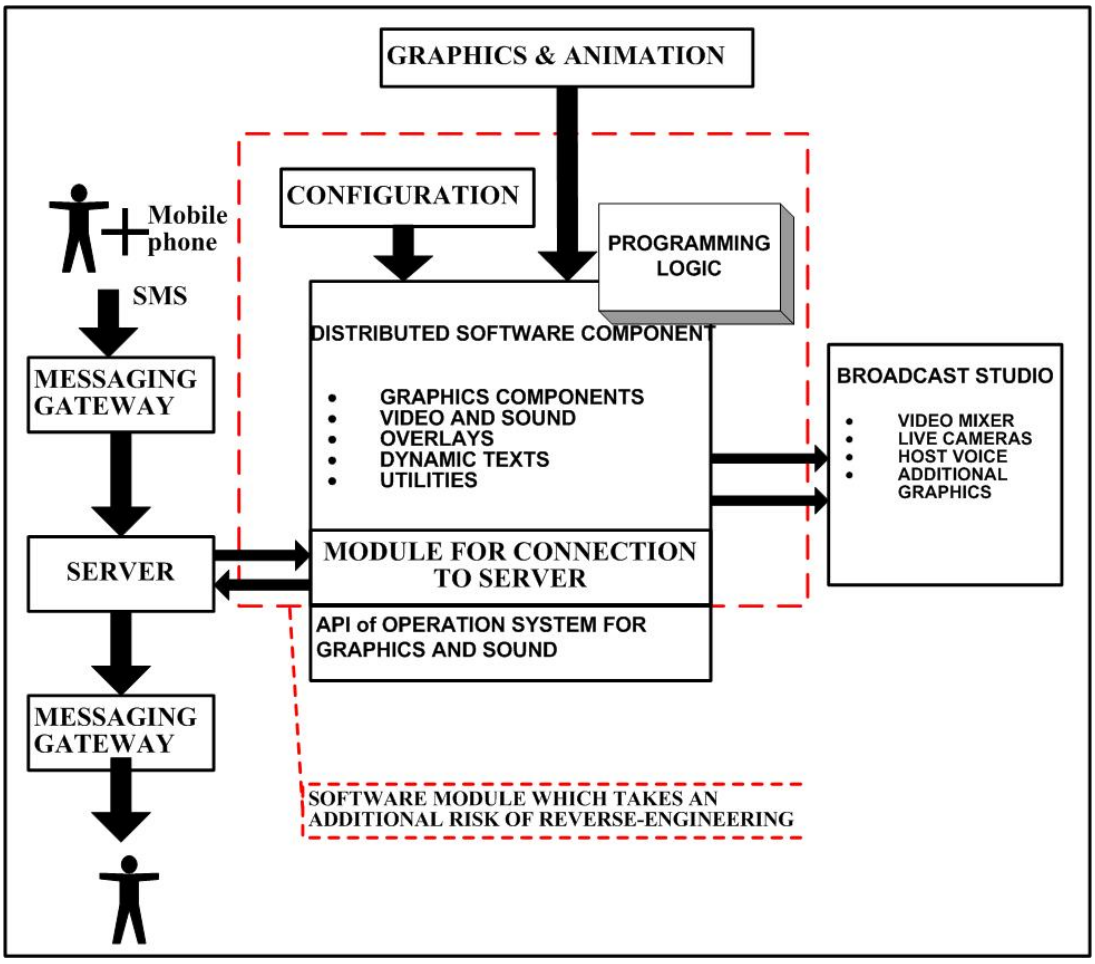


Figure 3. System to be protected

The graphic above illustrates the technical architecture of the system, which contains the following key components:

- Messaging gateways: Transmit end user messages (e.g. SMS and MMS) to and from server side using standardized protocols.

- Back End server: The server application that handles all messages and sends them to the client application. Server receives each message and commits its work to the data central. In addition to message processing, server provides services for user profiling, message tracking, high score management, billing and scheduling, among other things.
- Distributed client software component: Includes on-screen applications that represent user requests and responses in graphical form. Also contains all administrative tools. Client-based software communicates with server, retrieving messages, handling scheduling and reporting possible error situations, for example.
- Broadcast studio: The studio systems that take the video input from the applications and broadcast them to the end user.

The part outlined in red on the figure takes an additional risk of reverse-engineering, because physically it is delivered to television station and can be decompiled by potential malefactor.

2.3 Software Package to be broadcasted.

To be more concrete about which software package to be broadcasted in case of interactive television, I present an example of interactive application on Figure 4.

The idea of this exact interactive application is that every registered user is visualized as 3D model on the TV screen. End-user can manipulate by his/her interactive avatar and also participate in some quiz by sending SMS.

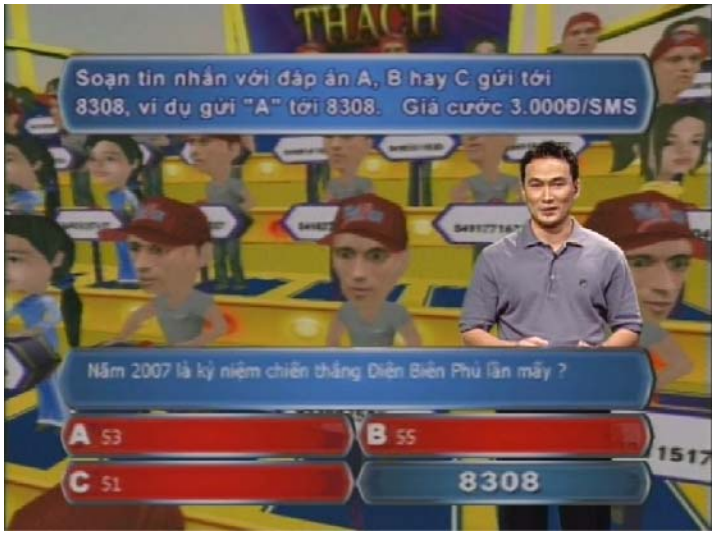


Figure 4. Interactive content and real video stream from TV-studio. Source: [15]

Presented live broadcast is a combination of video signal from studio camera and Java-based application. Next figures illustrate the separation of final video-stream into several parts. Figure 5 presents java application which emulates three-dimensional virtual world. Any user can register and control his/her own avatar. Figure 6 shows television studio and presenter.

THE FINAL VIDEO STREAM

Figure 5. Java based application for interactive TV. Source: [15]

Figure 6. TV-Studio. Source: [15]

At the same time Java based application (from Figure 5) consists of several modules. Some of them are shown on the Figure 7 and Figure 8.

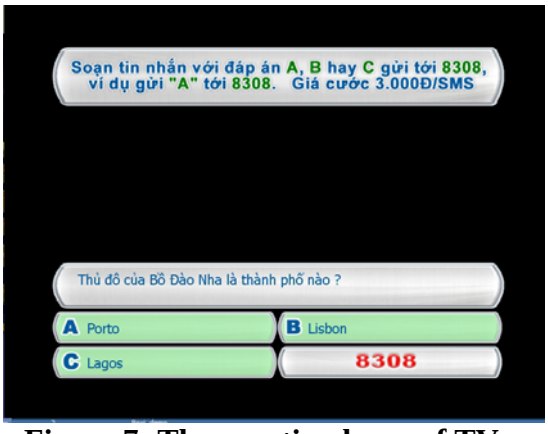


Figure 7: The question layer of TV-programme Source: [15]

Figure 8. 3D layer of TV-programme. Source [15]

The referred TV-programme was produced by MotionAvenue International and was broadcasted in Vietnam during spring 2007. [15]

2.4 Reverse-Engineering software, which can be used to decompile ITV software distributed components

A situation concerning an illegal de-compilation of software products is complicated enough, because a lot of tools and programs for this purpose (transformation by which a high-level source code for an executable program is discovered) are accessible freely from Internet. There is a short list of commercial java de-compilers:

- **Jdec** [http://jdec.sourceforge.net/]
- **Jode** [http://jode.sourceforge.net/]
- **Jad** [http://kpdus.tripod.com/jad.html]
- **Dava** [http://www.program-transformation.org/Transform]
- **Mocha** [http://www.brouhaha.com/~eric/computers/mocha.html]
- **SourceTec Java Decompiler** [http://www.srctec.com/decompiler.htm]
- **JreversePro** [http://www.program-transformation.org/Transform]
- **SourceAgain** [http://www.program-transformation.org/Transform]
- **ClassCracker 3** [http://mayon.actewagl.net.au/index.html]
- **Decaf [**http://ourworld.compuserve.com/homepages/teleobjet/decaf.htm]

All mentioned de-compilers can be used to reverse-engineer any part of publicly distributed software components.

2.5 Potential risks and protection of software components

The whole system is a set of programming modules, which logically can be separated to server side and client side modules. Client side modules should be sent to different television stations. Companies which produce client side applications (also called as content for interactive television) require technical protection of their software

components from illegal attempts to change produced software. One of the important requirements is that the protection should not affect the connection speed between server and client.

The field has some principal constraints that make tasks sophisticated enough. The constraints are:

a) Bandwidth of connection between server and client is limited.
b) The end-users (people on TV station) have the physical access to the client-side software.
c) All the calculations on the client end are resource-intensive.

Since this thesis considers the protection applied to commercial software, the main goal can be formulated as following: protect means increase the time and effort required for reverse-engineering, so that it is economically infeasible for a client-company to reverse engineer an application.

We faced with a problem of reverse-engineering (or decompiling) and we should change the program so that it behaves identically as before, but it should not be possible for humans (customers) to understand the principals of its functionality, which makes a very interesting protection task for interactive television.

3. ANALYSIS OF EXISTING PROTECTION APPOACHES and TOOLS

The main idea behind any software protection tool is to make reverse-engineering much more hard, economically unprofitable or even technically impossible.

We discuss the technical protection approaches based on obfuscation, client code encryption, using compiled programming languages and execution on the server side. The mentioned types of protection are most suitable for Java-based software.

3.1 Obfuscators

Overview:

Obfuscation means changing and cutting the names of packages, classes, methods, parameters, variables and attributes of particular project from something significant to totally non-significant values. This approach can be satisfactorily applied to large projects which are absolutely incomprehensible without clear code comments and documentation. Figure 9 presents Obfuscation mapping.

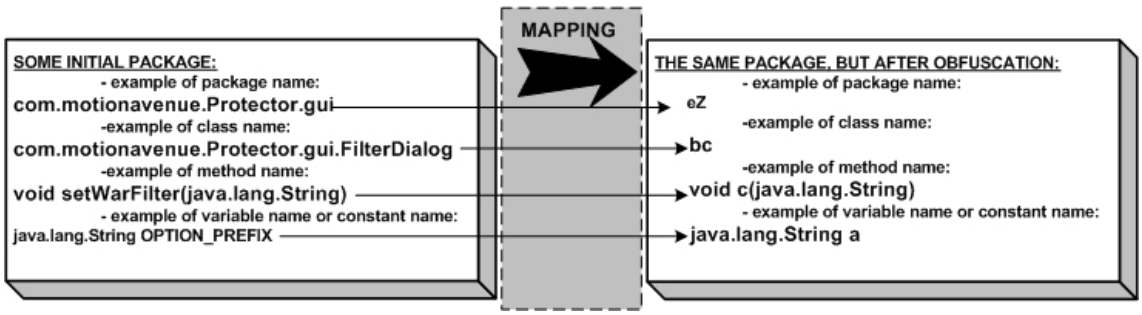


Figure 9. Obfuscation process. Source: [16]

Advantages:

After these operations most probably the code will be as following:

```
public abstract class ay implements cU
{
public int a;
public int b;
public int c;
public cL[] a;
public String a(cp var_cp){
 cE var_cE= (cE)a(var_cp, "Code");
 if (var_cE==null)
            return null;
 O o = (O) var_cE/a(var_cp, "LineNumberTable");
 if (o==null)
            return null;
 return""+o.a(0)+":"+o.a(2147483647);
}
...
}
```

This small example shows that this kind of protection still allows user to make some reverse engineering, but if the project structure is complicated enough, it will take a lot of resources to understand the structure and to make even small changes in the project.

Disadvantages:

Since obfuscation is based on mapping of all the names to some non-significant values, it also means that with appropriate resources anyone can create the correct mapping in backward direction step-by-step.

By this scenario, people who are doing reverse engineering can create simple UML diagrams of obfuscated packages and based on some assumptions give significant names to all the packages, classes and elements. Figure 10 illustrates the possible way of reverse-engineering.

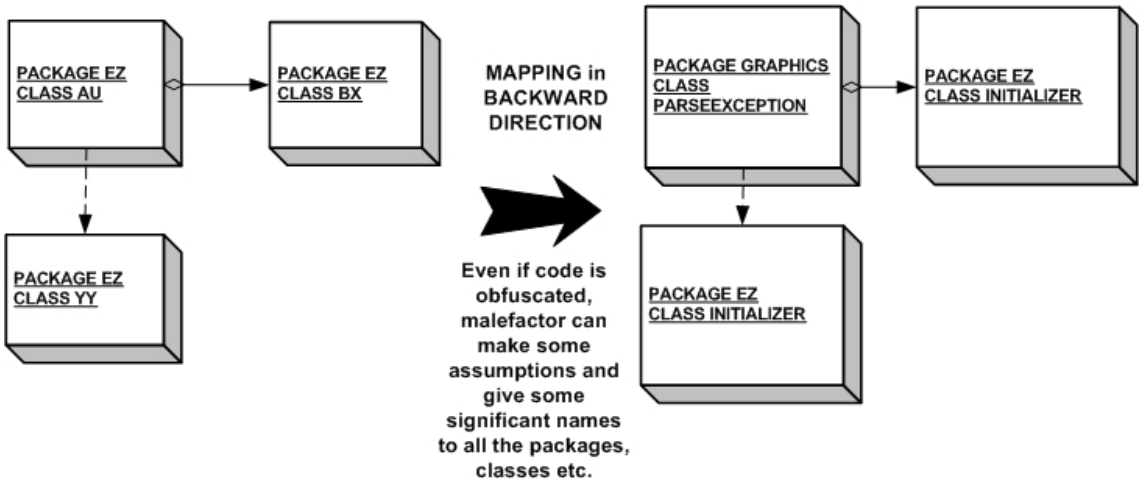


Figure 10. Backward mapping for reverse engineering.

Conclusion: Obfuscation is a very good protection method with some disadvantages. In real usage, obfuscation can not protect the code completely, but it can make reverse engineering of existing code much harder and resource-intensive.

Obfuscation should be supplied with more secured approaches simultaneously.

3.2 Encryption of code

Overview:

Encryption is another possible way to protect the code from reverse engineering. Encrypting data at rest can be accomplished by software, but have substantive disadvantages in performance and ease of use.

Advantages:

Software encryption is available for free or for little cost on many operating systems.

Disadvantages:

The problem with deploying this is performance. Software encryption is extremely CPU intensive [16]. Decrypting data is just as CPU-intensive as encrypting it. Hardware encryption is more preferable in comparing with software case. But since this more reliable encryption process takes place on a hardware level – it is possible that user intercepts and decrypts the code [17]. The next disadvantage in case of hardware encryption – is limitation for program portability. The device that executes the encryption/decryption processes should be supported by the appropriate operation system (which is not always possible). The traditional scheme of encryption protection is drawn on the figure below.

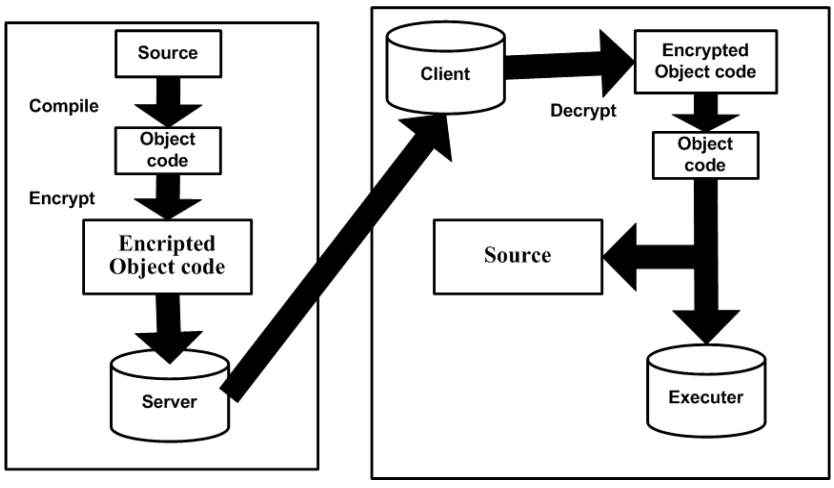


Figure 11. Protection based on encryption. Source: [17]

Conclusion:

The fact that software encryption is accessible for free on many operation systems makes it very attractive. However the described imperfections like resource-

intensity and limitation of program portability drops any further attempt to be applied on protection system.

3.3 Using compiled languages (Signed Native Code)

Overview:

Java code can be executed only by Java Virtual machine **Error! Reference source not found.**, which is also called Java Runtime. Means that before instructions can be sent to processor they should go through an interpreter. Thus execution of Java code is slower than traditional compiled programs, created with languages like C++ or C. To increase the performance of java based programs, special Just-In-Time compilers were developed, which translate java code to C++ code. Figure 12 presents the described behavior.

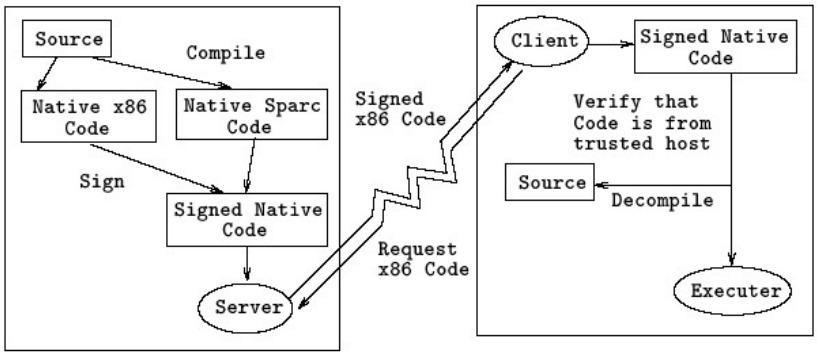


Figure 12. Protection based on the use of native code

Advantages:

Due to the functionality of compiled languages - Programs, created with languages like C++ do not contain additional information that can be used for decompiling. [18]

Disadvantages:

Translation of Java code to C++ code does not guarantee if the code is absolutely safe. Additionally a JIT compiler is required to translate the Java byte-codes into a native code of each architecture while Java-based programs should just have an appropriate interpreter and byte-code can be the same for every architecture.

Conclusion:

Using compiled programming languages is an efficient way to protect the inner structure of developed software. Simultaneously with other protecting paradigms the described approach can be very useful.

3.4 Execution on the server side

Overview:

The first obvious solution to prevent reverse engineering is to restrict the physical access to the executed code. Figure 13 illustrates this technical approach.

Advantages:

In some cases, part of the code can be moved to the server side of the distributed system. Client side can just receive the results and visualize them.

Disadvantages:

There are two major imperfections of this approach:

a) It requires permanent connection between server and client. Limited bandwidth can affect the performance of the whole application
b) If connection breaks, the end users are not able to use the client side of the program.

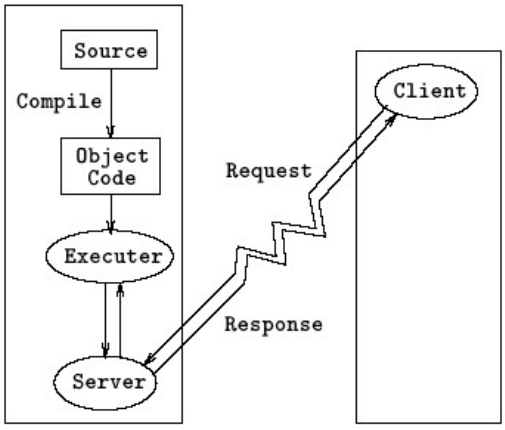


Figure 13. Functionality has been moved to the server side

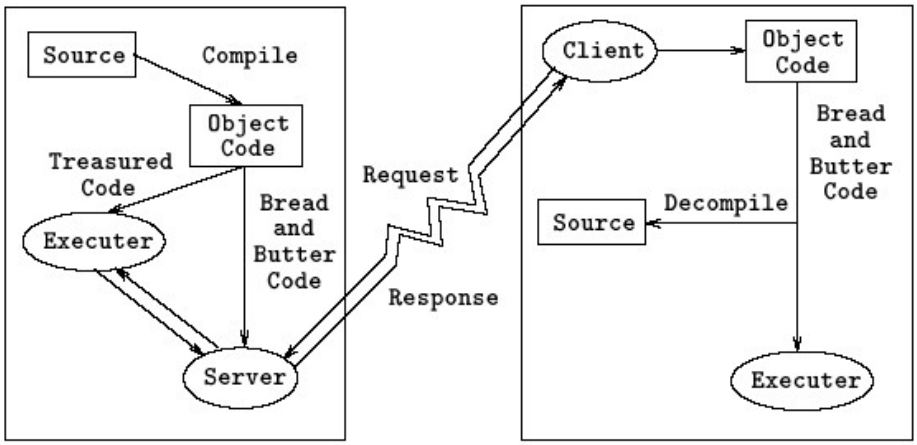


Figure 14. Partial functionality has been moved to the server side

Conclusion:

Connection bandwidth of connection between server and client can be limited. Thus the protection approach with execution on the server side can be chosen only if information traffic between client and server is not very intensive. The information traffic also depends on the nature of application and the efficiency of practical implementation.

3.5 Summary of the section:

As it was described before, all the considered approaches have some disadvantages, which in our case do not allow to rely on only one single solution.

Server side execution depends on a network bandwidth, which in our case can be very critical. Encryption can not be effective without special hardware, but for programming languages like Java, it means that encryption device should be supported for a variety of hardware platforms. Partial translation of Java-code to native code is possible and makes the task of reverse-engineering very hard. It also restricts portability, because of the need to provide Java byte-codes to native code translators for multiple architectures.

Finally, it is clear that using only one approach is not enough to protect the code. For this reason I would like to use the following ways simultaneously:

- Use the changed byte code (Obfuscation)

- write some parts of the code with C++, compile it into libraries and after that connect those libraries with Java modules using JNI.

The mixture of different approaches will be the basic idea of practical realization. The sequence of actions is shown in the figure below:

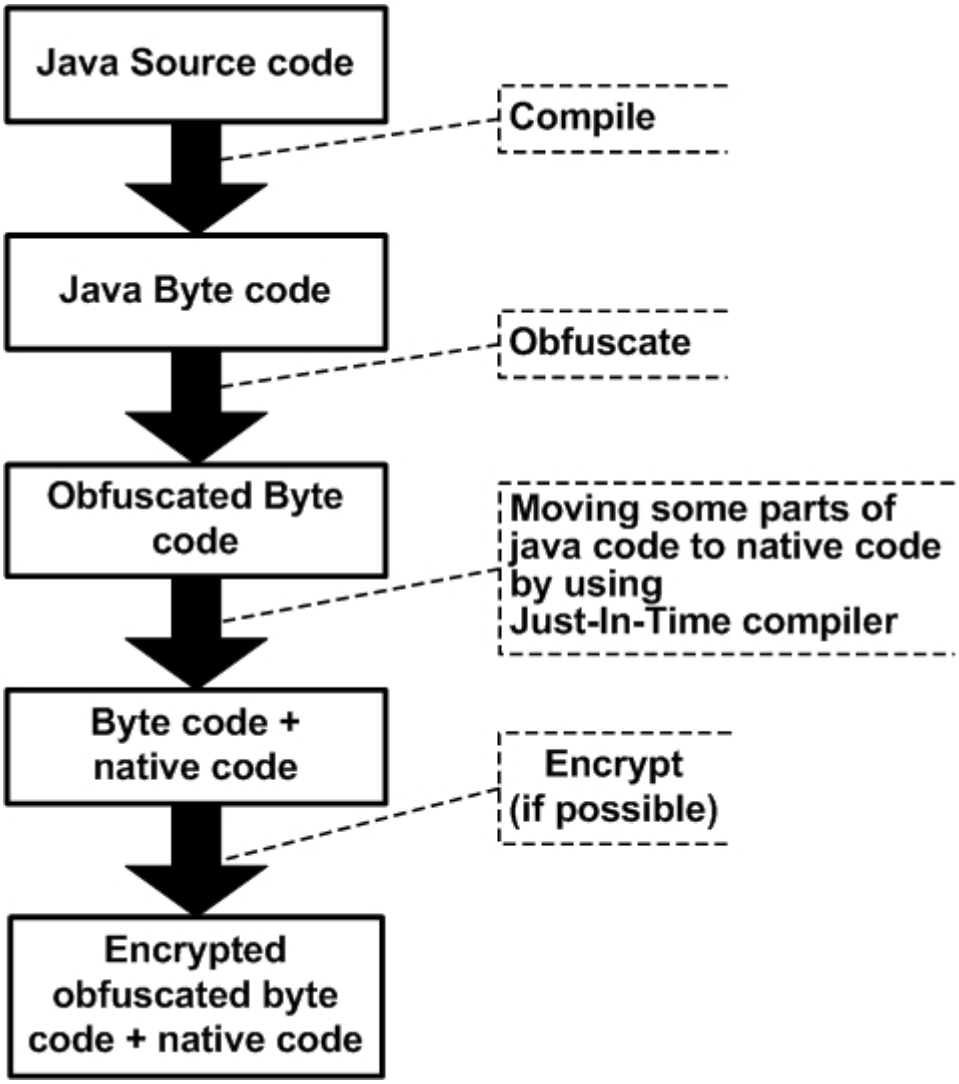


Figure 15. Final developed protection behavior to be realized on a practice

4. SELECTION AND REALIZATION

From the selected project environment, realization will depend on few points mentioned below:

- Speed of programming development
- Convenience of user interface
- Possibility to change/improve the code easily
- Performance of software
- Possibility to create useful documentation automatically

Consequently, Selection of programming tools is one of the most serious steps in development cycle.

4.1 Selection of programming tools and programming patterns to realize the project

First of all, the concrete programming language should be chosen for realization of described protection system.

Programming language for core development:

Based on the advantages listed below, Java programming language has been chosen for the practical implementation:

- **Portability:** Java runs on most of the hardware and software. Theoretically, by using this language, we make protection system compatible with these platforms. The major problem with C or C++ code porting is related with the precision of calculation (the basic data types in C and C++ depend on actual platform). The perfect thing about Java is that it defines the size of basic types, for all implementations equally. [19]

- **Java is a very simple programming language:** Many software developers consider this language simpler than programming language C++.

- **Reliability:** Java runtime realizes multiple checks of byte-code to avoid any inconsistency and to verify correctness of code. As compared to C and C++,

some features were deleted from Java language (like pointers and automatic type conversion). They could cause a lot of problems with wrong applications (for example, overwriting memory and corrupting data.)

- **Support of multithreading:** This feature can be defined as “the ability of a program or an operating system process to manage its use by more than one user at a time. It even manages multiple requests by the same user without the need to have multiple copies of a program running in a computer[20]. In our case multithreading can be used as an advantage for GUI creation.
- **Robustness:** It includes early checking of possible errors. Java compiler is able to detect many problems. Java realizes runtime exception handling feature, which means that it can catch and respond to an exceptional situation so that the program can continue its normal execution and terminate gracefully when a runtime error occurs [21].

- **Java is an interpreted language:** It means that a virtual machine is running in between the Java program and the machine. This property of described programming language is considered as an advantage or disadvantage depending on a context.

Finally, Java has a lot of advantages and one main disadvantage – speed of execution. This is a retribution for producing reliable, architecturally neutral and portable code.

By technical estimation made before the programming realization, advantages which Java provides to developer are more ponderable then it’s single imperfection.

The following environment version has been chosen:

Java version "1.6.0"
Java(TM) SE Runtime Environment (build 1.6.0-b105)
Java HotSpot(TM) Client VM (build 1.6.0-b105, mixed mode, sharing)

Integrated development environment:

Four integrated development environments for Java programming considerably exceed all others. These four are:

- **Netbeans:**
  It is a cross-platform open source IDE for Java that comes with a syntax highlighting code editor that supports code completion, annotations, macros, auto-indentation, etc. It includes visual design tools (wizards) for code generation. It integrates with numerous compilers, debuggers, Java Virtual Machines and other tools. [22]

- **Eclipse IDE:**
  It is an open source extensible IDE. At present, it works well as a Java IDE, and includes Java development tools. It requires that you have the Sun Java runtime environment (JRE) installed. The IDE supports Windows XP, Windows 2000, Windows 98, Windows ME, Linux, Solaris, QNX, AIX, HP-UX, Mac OS X, and possibly other systems as well. [23]

- **Oracle JDeveloper:**
  It supports every step of a development life cycle, including modeling, coding, debugging, testing, profiling, tuning and deploying of applications. All these tasks are done from a single IDE using a set of integrated features. [24]

- **IntelliJ IDEA:**
  It is a code-centric editor characterized by following features:

  - This IDE based code assistance works much faster than other packages e. g. the proposal window for importing classes.
  - Unlike Eclipse, IDEA is not a platform based on plugins, but it’s much easier to find and install different software plugins for it.

It is also very important to take account of developer’s predilection. Since I am not familiar with Oracle JDeveloper, thus I will consider and compare three residuary development environments.

Before the final acceptance of chosen IDE, small research and comparison was done on their possibilities/properties.
Since Eclipse IDE concentrates more on functional capabilities than NetBeans and IntelliJ IDEA, I will choose it for the development process. At the same time I decided to use NetBeans as the main tool for XML diagrams creation.

Integrated Development Environment:

Eclipse SDK Version: 3.1.2

For development of Visual part of the system:

Visual Editor SDK plugin 1.1.0.1

which requires installing some additional plugins:

- Graphical Editing Framework 3.1.1
- EMF SDK 2.1.2

For automatic compiling:

Apache Ant version 1.6.5 compiled on June 2 2005

For Semi-automatic Documentation:

Javadoc module from JDK

Programming pattern:

Visitor pattern is used under one of the packages. The main idea of using this pattern is to somehow divide object structures and logical algorithm [25].

For UML Modeling:

UML Modeling plug-in for NetBeans IDE.

4.2 Short description of developed packages

Limited amount of packages should realize the following functionality: reading information from files with *.class extensions; writing the information to the same format; obfuscation and moving defined parts to C++ modules.

To name all the basic elements of the project - I will follow the rules of java naming conventions [26].

System for protection of publicly distributed Software components should contain some compulsory parts:

- Package **com.motionavenue.Protector** – main package.
- Package **com.motionavenue.Protector.gui** – all the graphical user interfaces.

- Package **com.motionavenue.Protector.obfuscate** – one of the most important system parts, intended for code obfuscation.
- Package **com.motionavenue.Protector.classfile –** to represent different elements of classes
- Package **com.motionavenue.Protector.classfile.visitor –** to separate an algorithm from an object structure

Classes contain lists of elements of different types - attributes, entries etc. Most of these types will not be changed in further Java releases. But we can wish to expand and change the operations on these class elements. The basic idea of using Visitor pattern [27] is to separate objects and operations. In other words, using of visitor pattern is a separation of an algorithm from an object structure.

- Package **com.motionavenue.Protector.cppjni**

For additional protection - some parts of the code should be rewritten with C++, compiled to libraries and after that they should be connected with the rest of logical structure with Java native Interface (JNI). That’s what this package does.

4.3 UML diagrams

Before any programming realization – all the technical requirements for compulsory packages were written in a form of UML diagrams. I skipped creation of Use-Case diagrams because the functionality and goals of all the packages were described textually. The main idea was to create the class diagrams – the core of object-oriented design. It describes the types of all the objects in the system and different relationship between them.

UML class diagrams which are presented in this section:

- package com.motionavenue.Protector.io is described on figures 16 – 18.
- package com.motionavenue.obfuscate is overviewed on figures 19 -23.

These packages have been chosen to be overviewed in details, because they realize the basic operations of developed software program.

The package com.motionavenue.Protector.io is the abbreviation of **I**nput**O**utput. It contains classes to read and write files from and to different java archives with *.jar, *.war and *.ear extensions.

Any package from developed protection software contains a number of interfaces, because “Implementing an interface allows a class to become more formal about the behavior it promises to provide. Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler.”[21]. Figure 16 provides the part of **I**nput**O**utput package.

package com.motionavenue.Protector.IO

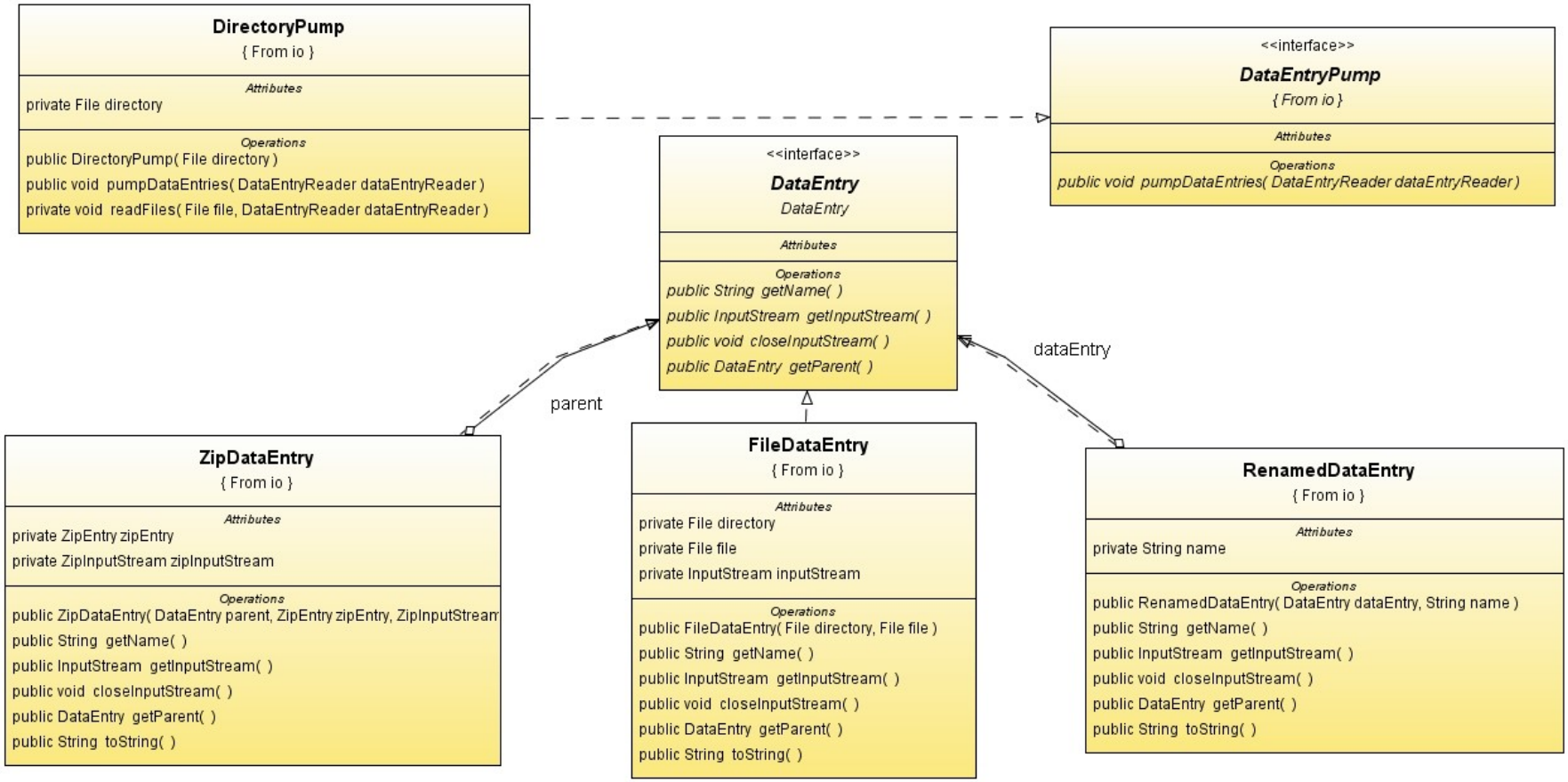


Figure 16. Package com.motionavenue.Protector.io (part 1)

The package com.motionavenue.Protector.io presented on figures 16, 17 and 18 contains the following five interfaces:

- com.motionavenue.Protector.io.DataEntry

  This interface describes the high-level objects which can be written or read by developed program.

- com.motionavenue.Protector.io.DataEntryFilter

  This interface provides a filter which should check the correctness of input DataEntry object.

- com.motionavenue.Protector.io. DataEntryPump

  This interface provides the methods to evacuate the data from DataEntry object

- com.motionavenue.Protector.io.DataEntryWriter

  This interface provides methods to write the information from any DataEntry object.

- com.motionavenue.Protector.io.DataEntryReader

  This interface provides methods to read the information from any DataEntry object.

- com.motionavenue.Protector.io.Finisher

  This interfaces defines the methods which should be called before the reading or writing stream is closed.

Figure 17 presents the next logical part of **I**nput**O**utput package. It contains several interfaces (com.motionavenue.Protector.io.DataEntryReader and com.motionavenue.Protector.io.DataEntryWriter) and almost all the classes, which implements the mentioned interfaces.

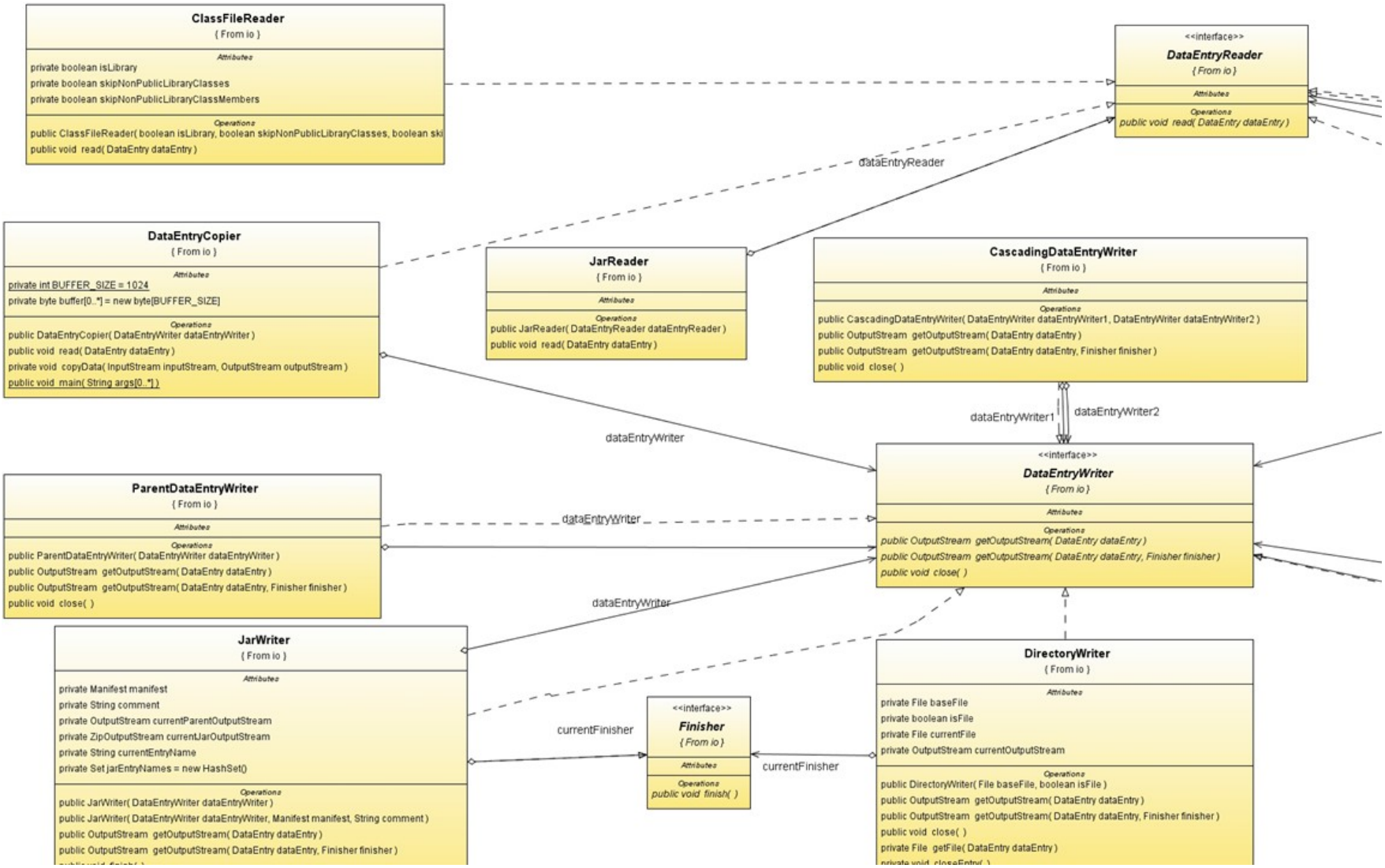


Figure 17. Package com.motionavenue.Protector.io (part 2)

Figure 17 shows that interface com.motionavenue.Protector.io.DataEntryReader is implemented by the following classes:

- com.motionavenue.Protector.io.ClassFileFilter

  The main purpose of this class is to read the given data entry.

- com.motionavenue.Protector.io.ClassFileReader

  This implementation of com.motionavenue.Protector.io.DataEntryReader applies visitor software pattern [27] to the classes.

- com.motionavenue.Protector.io.JarReader

  As it was described before – DataEntry object can represent different java archives with *.jar, *.war and *.ear extensions. This particular class reads all the information from files with *.jar extension

- com.motionavenue.Protector.io.ClassFileRewriter
- com.motionavenue.Protector.io.DataEntryCopier
- com.motionavenue.Protector.io.FilteredDataEntryReader,

Figure 18 is the last part of package com.motionavenue.Protector.io

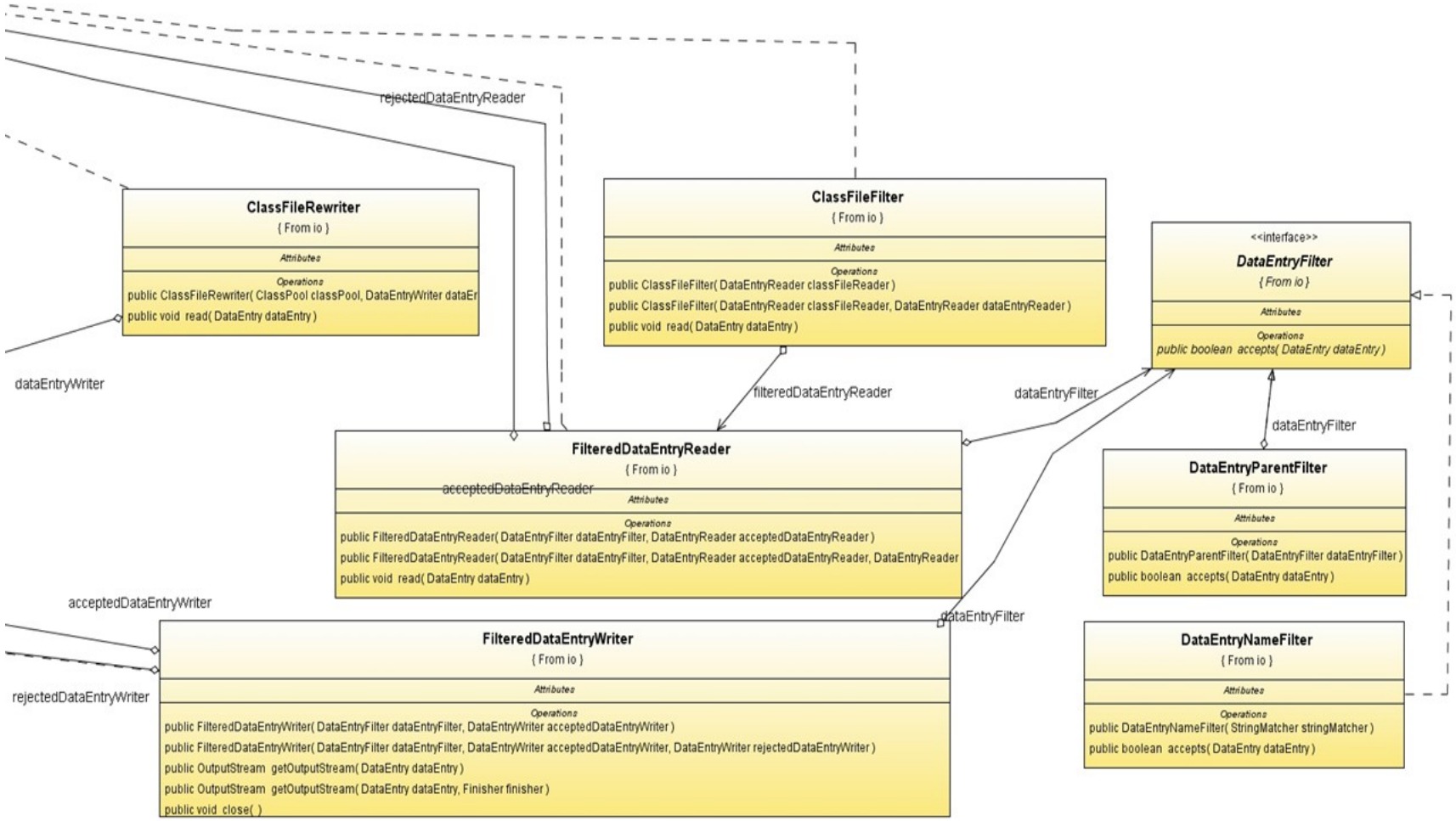


Figure 18. Package com.motionavenue.Protector.io (part 3)

The package **com.motionavenue.Protector.Obfuscate** equally with the package **com.motionavenue.Protector.cppjni** is the central core of developed protection system and realizes on a practice some theoretical assumptions stated in section three of this master's thesis. The package consists of 26 classes and 2 interfaces. Its basic functionality can be described as follows:

- Provide unique generated meaningless names for all the project elements: packages, classes, methods, attributes
- Keep the initial hierarchy and logical connections between newly named elements identically to situation before obfuscation
- Solve naming conflicts between different elements
- Use mixed-case characters
- Appends some meaningless suffixes to newly generated class names

Figure 19 illustrates some part of functionality described above. The rest of UML diagrams are defined in Appendix 1

.

package com.motionavenue.Protector.Obfuscate

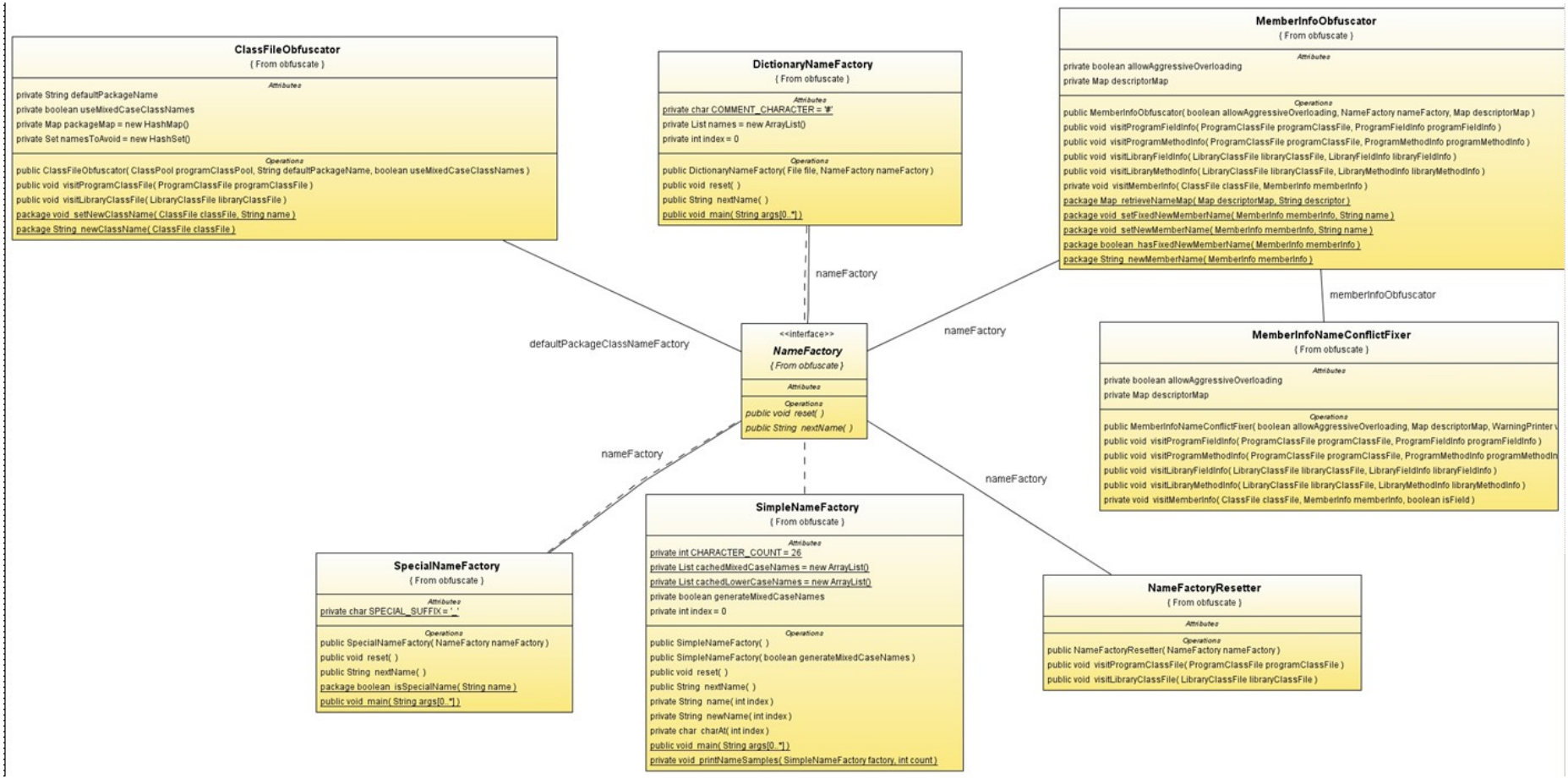


Figure 19. Package com.motionavenue.Protector.obfuscate (part 1)

4.4 Graphical user Interface

Developed software contains graphical user interface module, which allows end-users to define all the necessary parameters and go step-by-step through all the procedures of protection.

Considered graphical software package contains the following functionality:

a. Choosing java based archives to be protected (it can be jar/war/ear/zip files)
b. Choosing the name of an archive – to be outputted after protection. It means that it should contain the same functionality, but in some protected form.
   Also some additional features have been implemented:
   - Changing the order of packages to be processed.
   - Complete removing of a package from the list of packages to be protected.
   - Moving the package to library list (Libraries are not protected. They are just included into final java archive without any obfuscation or any other type of protection)
c. Choosing all the libraries which should be included into the final protected java archive. (For example, most probably JRE should be included for any project to be protected)
d. Defining some package filters for classes that are not to be included into the final java archive.
e. Defining the obfuscation (which can be chosen or skipped) and its options:
   - Print mapping
   - Obfuscation dictionary
   - Default package
   - Overload aggressively
   - Use mixed-case class names
   - Keep attributes

   Etc.
f. Checking the consistency of output package by manipulating the following options:
   - Warn about missing libraries
   - Ignore warnings about missing libraries
   - Skip non-public library classes.

g. Defining how to import part of the source code to C++ language and connecting this module by using Java Native Interface.

All these options and variations are visualized on appropriate graphical panels. Also, I should mention that some of the described options can be defined by using graphical user interface or by editing text configuration file. Let's consider all the graphical implementation of Protection system step-by-step:

On the picture below, you can see the title page of considered project

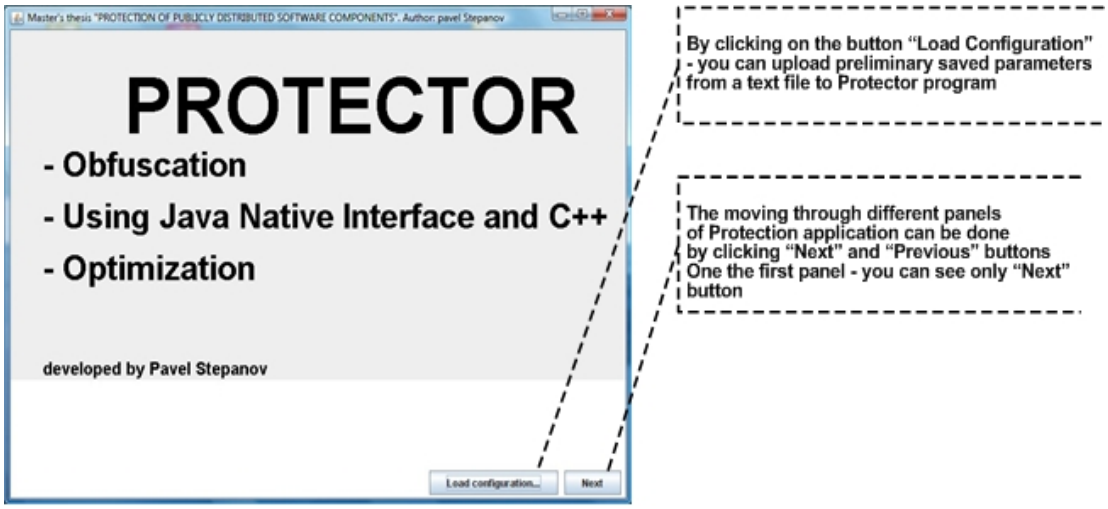


Figure 20. Title page of Protector package

By clicking on "Load configuration" panel you can import a preliminarily prepared configuration file. The file by itself should present some text instructions saved in UTF-8 format. The example of such kind of file can be the following:

```
-injars ./test/example.jar(!protector/ant/**,!protector/wtk/**)
-outjars ./test/example_out_fromProtector.jar
-libraryjars <java.home>/lib/rt.jar
-printmapping ./protector_mapping.txt
-overloadaggressively
-defaultpackage ''
```

The end-users can use specially designed File-Open dialog (which shows the files with only *.pro extensions) to open configuration files,

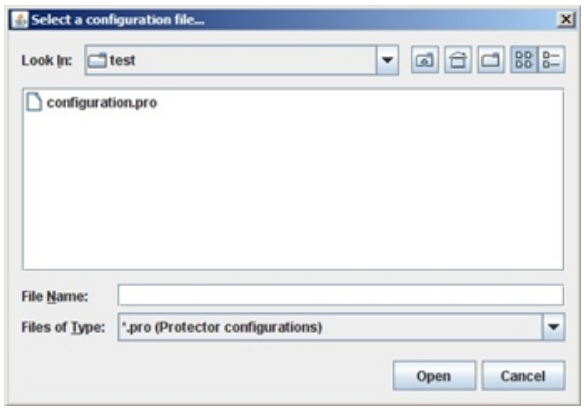


Figure 21. File-Open dialog for files with *.pro extension

The next Panel, which is called “Input/Output” provides the possibility to define all the input/output options for java archives, libraries etc.

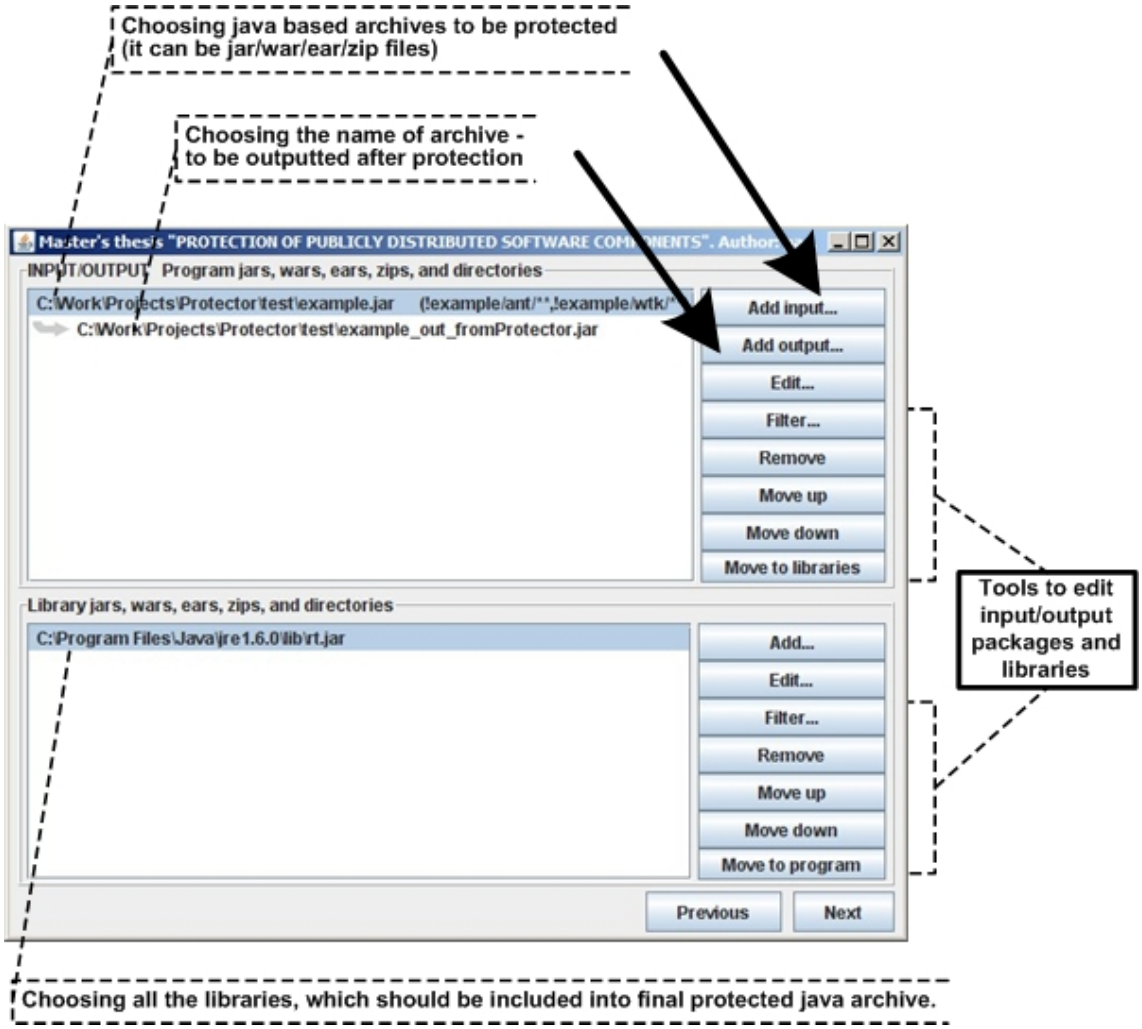


Figure 22. Graphic panel to define input/output options

Implemented software can protect not only single package in a time, but the sequence of packages. For that you should choose them one-by-one and put into mentioned list of java archives to be protected.

All other graphical interfaces are presented on the figures below.

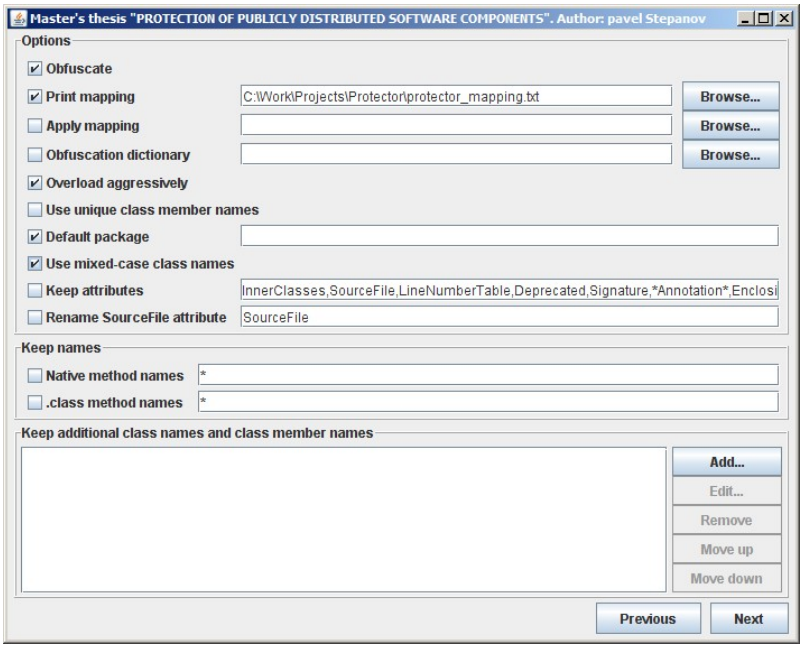


Figure 23. Graphic panel to define Obfuscation options.

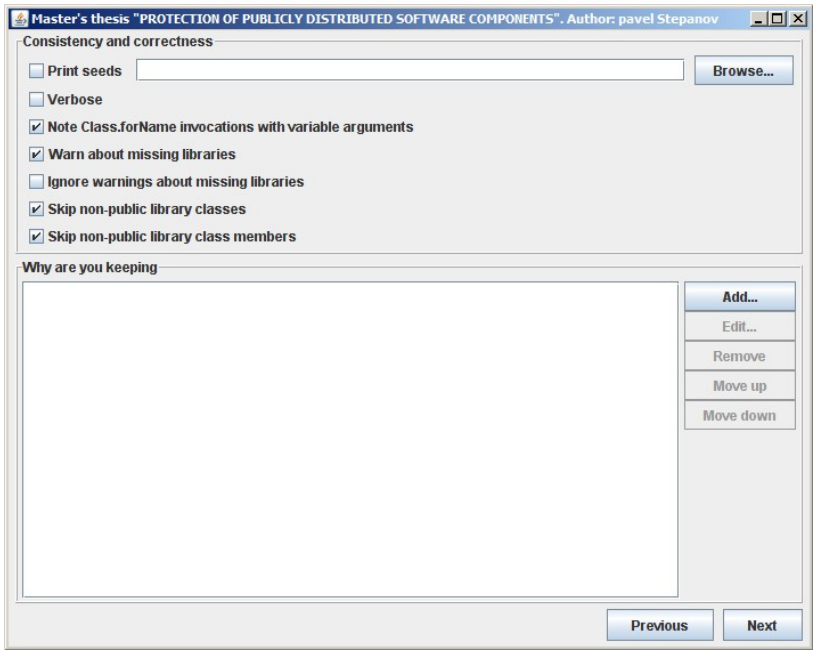


Figure 24. Graphic panel to define Consistency options.

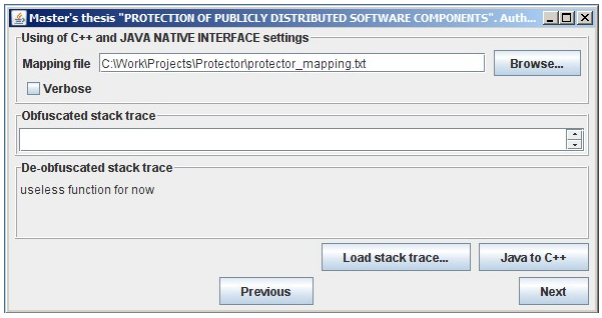


Figure 25. Graphic panel to define C++ and Java Native Interface options.

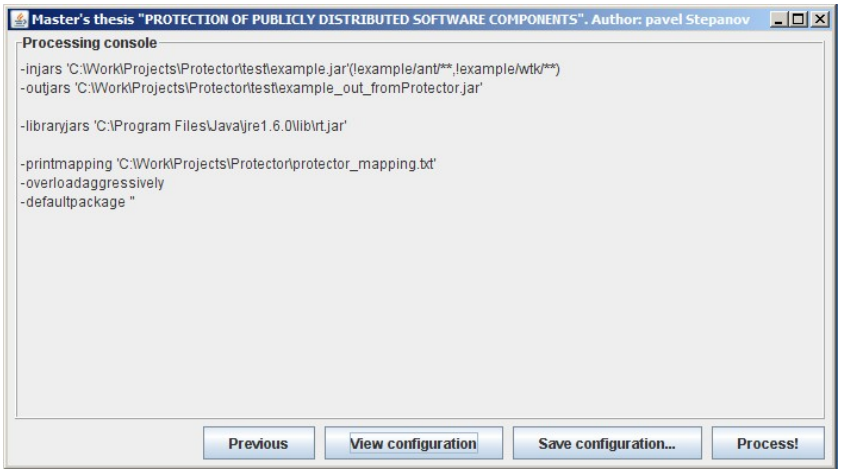


Figure 26. Processing console with information about current configuration.

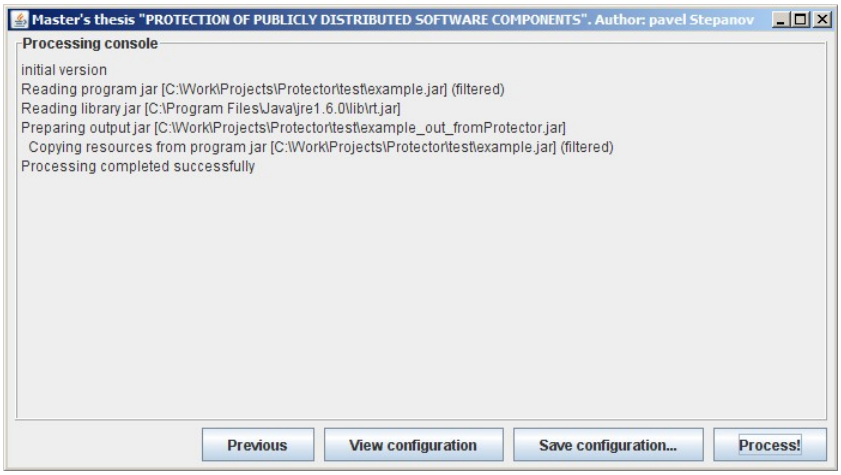


Figure 27. processing console with information about protection process and results.

5. EVALUATION

In this section we will try to measure the effectiveness of the proposed theoretical approach and also measure the performance of different programs after new protection technique is applied on them.

5.1 Quality of transformation

We can measure the effectiveness of made protection/transformation by the following criteria:

-potency

-stealth

-cost

-resilience

For example, **resilience** can be defined as following: How well the code resists to decompiling attempts [4]. Let us say that τ can be a transformation after obfuscation and also after applying Just-in-Time compilers to byte-code.

At the same time: P - initial program to be protected

P' - final protected program

E(P) - complexity of the program

Cost is the additional run-time resources required to execute a program after a transformation has been applied to the program and **potency** is the degree to which a transformation confuses a human who is trying to understand a program .

Also we should define some complexity metrics [4]. Usually any complexity metric counts some property of source code. Complicated programs have greater amount of these properties than simpler programs. Marciniak [4] describes the situation as follows: “If program P and P' are identical, except that P' contains more of property q than P, then P' is more complex than P”. Thus proposed protection technique should make the program more complex. The most popular complexity metrics are presented in table 2.

Metric	Metric name	Description	Reference
$\mu 1$	Program length	Complexity E(P) increases with the number of operators	[29]
$\mu 2$	Object-Oriented metric	This metric has a complicated description, which will be described in Table 3.	[30]
$\mu 3$	Data structure metric	Complexity E(P) increases with the number and complexity of static data structures like arrays, records and variables	[31]
$\mu 4$	Nesting complexity metric	Complexity E(F) will be increased in case if software designer enlarge the nesting level in some function F.	[32]
$\mu 5$	Data flow complexity metric	Complexity increases with the number of inter basic block variable references in methods.	[33]
$\mu 6$	Cyclomatic complexity	Complexity increases with the number of predicates in class methods	[34]

Table 2. Different complexity metrics. Source: [29], [30], [31], [32], [33], [34]

The complexity of software component described as E(X). For example, E(P) means the complexity of a program, while E(F) means the complexity of some function (method). Object-Oriented complexity metric will be used to evaluate an extension of **potency** after transformation from byte-code to native code. Table 3 shows the factors which affect the value of this metric.

Factor	Description
μ_2^1	The number of methods in native code after transformation the code from byte code to C++ code.
μ_2^2	The depth distance from the root class to considered class in terms of object-oriented inheritance.
μ_2^3	The number of direct subclasses in native code.

μ_2^4	The number of other classes to which considered class is coupled.
μ_2^5	The number of methods that can be executed in response to a message sent to an object of considered class

Table 3. The description of Object-Oriented complexity metric. Source: [30]

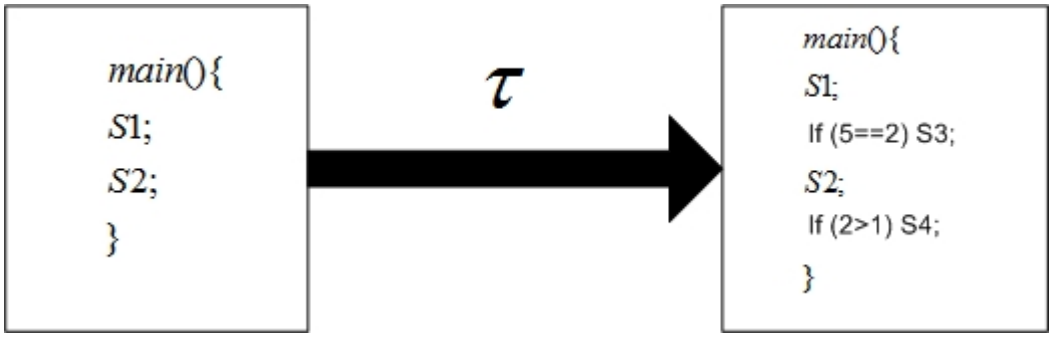
The complexity E(P) of developed protection mechanism defined by the following metrics:

- Program Length
- Data flow complexity
- Data structure complexity
- Cyclomatic complexity
- Object-Oriented complexity
    - The number of methods in C or in C++ language after using JIT.
    - Amount of subclasses from root to the last subclass in the tree of inheritance

Transformation potency can be calculated with the help of formula

$$\tau_{POT} = \frac{E(P')}{E(P)} - 1 \qquad (1)$$

According to description of cyclomatic metric the complexity can be increased by the simple programming transformation.



We can put these additional programming blocks, just to make reverse-engineering much more complicated.

In ideal case we want potency to be as high as possible, one-way resilience be free and stealthy

5.2 Speed of execution

Here we can evaluate the execution delays, which appear after protecting of input java-archives by developed software:

Input program	Original size	After Obfuscation	After using JIT compiler	Time (how much time does it take to transform the package to protected form)	Memory Usage	Performance losses after protection
OliveBeeDrone.jar	1.5 Mb.	1.2 Mb.	980 Kb.	31 s.	39 Mb.	19 %
Protector.jar itself	505 Kb.	363 Kb.	361Kb.	18 s.	13 Mb.	5%
ant.jar	2.3 Mb.	2.0 Mb	1.8 Mb	20 s.	49 Mb.	31%
Tomcat (the Apache servlet container)	1.2 Mb.	538 Kb.	530 Kb.	23 s.	39 Mb.	22%

Table 4. Comparison of packages before and after protection

The following software and hardware were used for these tests:

Processor type: Pentium 4

Memory: 512 MB

Java(TM) SE Development Kit 6 Update 2.

Operation system: Windows XP Professional.

Memory usage – is the amount of memory used by protector program during the obfuscation and transformation from byte-code to native code.

5.3 Summary of the section

This chapter has presented the classification of complexity metrics. Also it was shown which metrics were used, while trying to increase the complexity of developed protection software. Evaluation has shown that the performance losses after obfuscation, programming transformation and using Just-In-Time compiler are not critical for this type of software.

6. FUTURE WORK

This study by no means exhausts the subject of commercial software protection. The performance of software package after protection should be improved since some fields can have more strict technical requirements than area of interactive television. Reliability of protection also can be improved. We should solve many other technical issues. Some of further tasks are presented here.

6.1 Increase the complexity of mapping for obfuscation process.

One of the central tasks of considered protection system is the obfuscation of existing code. Obfuscation in this context means changing the names for packages, classes, variables and practically for everything else. For now, these mapping rules are quite simple. Below you can see the concrete example

```
protector.ConfigurationParser -> h:
protector.WordReader reader -> a
java.lang.String nextWord -> a
java.lang.String lastComments -> b
```

As one of possible improvements I propose to increase the complexity of this transformation.

6.2 Realize shrinking and optimization

Java programs are distributed as java archives, which contain Java byte code. The files with intermediate language are much more compact than source code files, but they still can contain a lot of unused code. The purpose of shrinking and optimization should be to analyze the structure of program and remove unused elements.

6.3 Improve the speed of execution for protected package. (Decrease performance losses after obfuscation)

Evaluation of protected packages showed that we have minor performance losses because of protection process. But even these minor losses can be diminished by optimizing the code.

6.4 Practical realization forJ2ME platform

Theoretically the developed package can be applied to programs written for mobile devices. Different kind of applications for mobile phones gets more and more popularity nowadays. It means that more and more software will be developed for mobile phones in the nearest future. The J2ME platform has a flexible structure. The profile for mobile phones is called the Mobile Information Device Profile (MIDP). Java applications created for this profile is called MIDlets [8]. To integrate developed protection system with MIDP, we should integrate protection system with J2ME Wireless Toolkit (produced by Sun Microsystems)

7. Conclusion

The thesis has made an overview of existing protection approaches for publicly distributed software components. This thesis mostly considered interpreted programming languages and protection concentrated especially on Java byte-code. The thesis proposed one new protection approach based on mixture of existing methods. Developed protection technique was realized as finished software product and evaluated in practice.

The simultaneous mixture of different protecting approaches (obfuscation, usage of native code and encryption) provides quite good result and does not allow any reverse-engineering. The practical implementation shows that protected code has enough resilience against decompiling attempts. These results are satisfactory for commercial software.

The following problem appeared during the practical realization*: cycle time of program execution has been increased because the program now is a mixture of byte code and native code.*

There is a trade-off between a highly protected program and increased run-time resource requirements. These factors should be taken into account while protecting the program or any java-based archive.

References

[1] Aho, A.V., Sethi, R., Ullman, J.D. 2006. *Compilers: Principles, Techniques, and Tools*. Baltimore: Addison Wesley.

[2] Muchnick, S. 1997. *Advanced Compiler Design and Implementation*. San Francisco:Morgan Kaufmann.

[3] Cifuentes, C. and Gough, K. J. 2005. De-compilation of binary programs. *Software - Practice and Experience*, **25**(7), pp. 811-829.

[4] Marciniak, J. J. (ed). 1994. *Encyclopedia of Software Engineering*. New Jersey:John Wiley & Sons, Inc.

[5] Sun Microsystems Java 2 Platform, Standard Edition Web site. http://java.sun.com/j2se

[6] Sun Microsystems Java 2 Platform, Enterprise Edition Web site. http://java.sun.com/j2ee

[7] Lindholm,T., Yellin F. 2006. *The Java™ Virtual Machine Specification. Second Edition*. [Accessed 31 May 2007]. Available from World Wide Web: <http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html>

[8] Sun Microsystems Java 2 Platform, Micro Edition Web site. http://java.sun.com/j2me

[9] The Business Software Alliance. 2005. *Software Industry Suffers From 5-year Cumulative Impact of Global Software Piracy, Estimated at $59.2 Billion*. Press release, issued 24 May 2005.

[10] Takeyama, L. N. 2004. The Welfare Implications of Unauthorized Reproduction of Intellectual Property In The Presence of Demand Network Externalities. *Journal of Industrial Economics*, **42**, pp.155-165.

[11] Business Software Alliance. *Global Software Piracy Study: European Union.* Available from World Wide Web: http://w3.bsa.org/globalstudy Press release, issued May 2007.

[12]Gawlinski, M. 2003. *Interactive Television Production*. Northwood:Focal Press.

[13]Srivastava, H.O. 2005. *Interactive TV Technology & Markets*. London:Artech House.

[14] Hawley, S. 2007. *Narrowing home network shortlist*. IPTV News Analyst [online]. **2** [Accessed 9th May 2007], pp.8-15. Available from World Wide Web: <http://www.broadbandbananas.com/images/stories/IPTV%20News%20Analyst%20January%202007.pdf>

[15] *Boxi/Nero Episode 1*. TV, HTV7 (National Vietnamese TV channel), 2007 March 19. 22.00 hrs.

[16] Boyd, C. 2003. Information Security. *In 6th International Conference, ISC, 1/3 October 2003. Bristol.* Conference paper. London: International Conference ISC, pp. 29-35.

[17] Galbreath, N. 2002. *Cryptography for Internet & Database Applications*. Chichester:John Wiley & Sons, Inc.

[18] Liang, S. 2007. *Java(TM) Native Interface: Programmer's Guide and Specification.* Baltimore:Addison Wesley.

[19] McLaughlin, B. 2007. *Head First Object-Oriented Analysis and Design.* Sebastopol:O'Reilly.

[20] Cohen, A. Woodring, M. 1997. *Win32 Multithreaded Programming*. Sebastopol:O'Reilly.

[21] Eckel, B. 2006. *Thinking in Java (4th Edition)* Stockholm**:**PED AB, Pearson Education.

[22] Myatt, A. 2007. *Pro NetBeans IDE 5.5 Enterprise Edition.* Berkeley:Apress.

[23] Burnette, E. 2005. *Eclipse IDE Pocket Guide.* Sebastopol:O'Reilly.

[24] Oracle Jdeveloper Overview 2007 [online]. [Accessed 9th May 2007]. Available from World Wide Web: <http://www.oracle.com/technology/products/jdev/collateral/papers/1013/jdev1013_overview.pdf>

[25] Gamma, E. 2004. *Design Patterns: Elements of Reusable Object-Oriented Software*. Baltimore: Addison Wesley

[26] Java Naming Conventions 1999 [online]. [Accessed 13th June 2007]. Available from World Wide Web: <http://java.sun.com/docs/codeconv/html/CodeConventions.doc8.html>

[27]WIKIPEDIA. 2007. *Visitor pattern* [online]. [Accessed 23rd January 2007]. Available from World WideWeb: <http://en.wikipedia.org/wiki/Visitor_pattern >

[28]Thomasson, M. (2003). *Winky dink, the history of interactive television, and You* [online]. [accessed 11th April 2007]. Available from World Wide Web: http://www.gooddealgames.com/articles/Winky_Dink.html

[29]Halstead, M.H. 1977. *Elements of software science (Operating and programming systems series).* Amsterdam:Elsevier North Holland Inc.

[30] Chidamber, R. 1994. A metrics suite for object oriented design, *IEEE Trans. Software Eng.*, vol. 20, pp. 476–493.

[31] Munson, J.C. 2003 *Software Engineering Measurements.* New-York:Auerbach-publications

[32] Harrison, W.A. 1981**.** *A complexity measure based on nesting level*. New Jersey:John Wiley & Sons, Inc.

[33] Oviedo, E.I. 1983 *Control flow, data flow and program complexity*. New-York: State University of New York at Buffalo, Department of Computer Science.

[34]McCabe, Thomas J. 1976. A Complexity Measure. *IEEE Transactions of Software Engineering*, **2**, pp. 25-44.

Legal Notices

APPENDIX 1. UML diagram of package com.motionavenue.Protector.obfuscate (part 2)

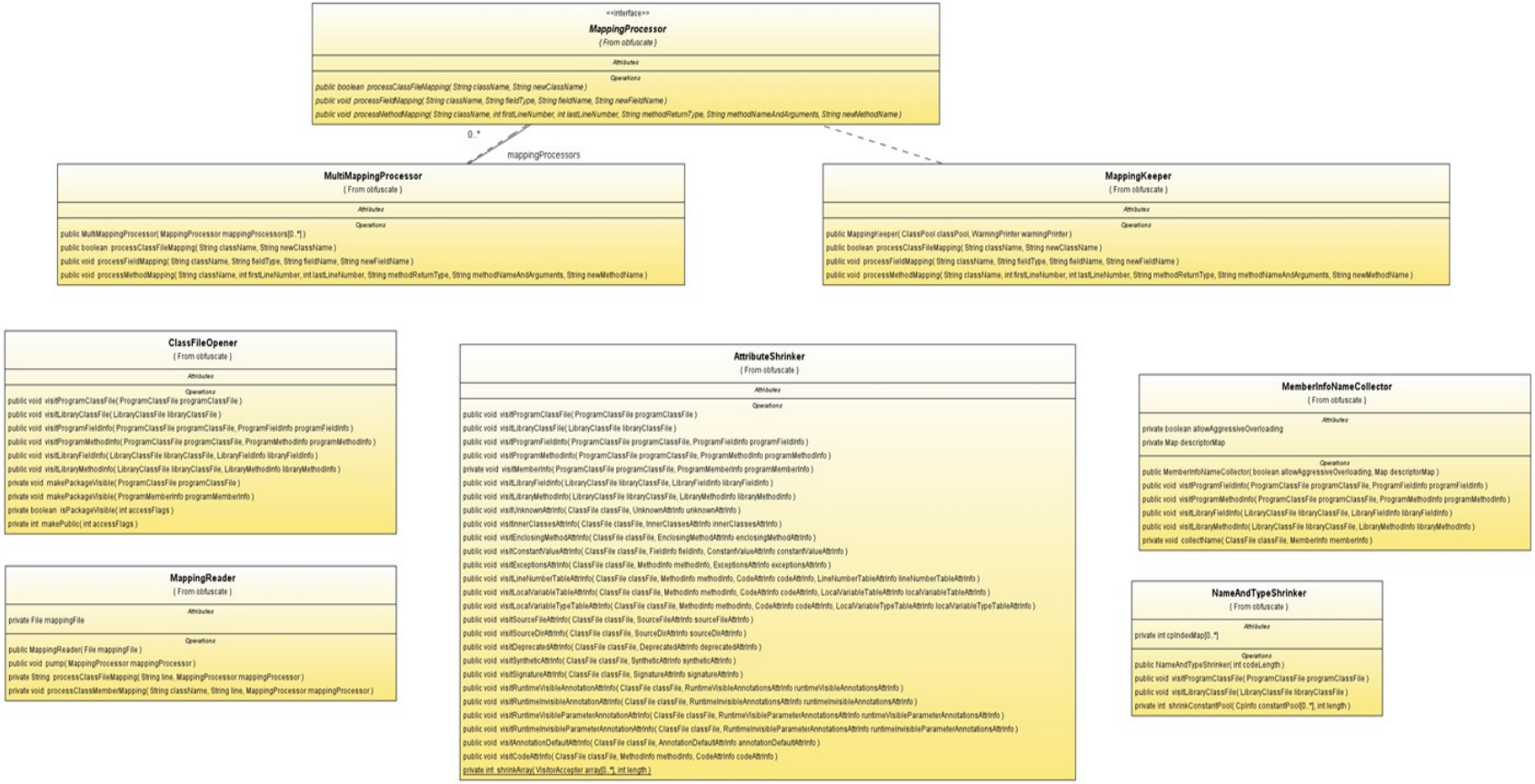


Figure 1.1 package com.motionavenue.Protector.obfuscate (part 2)

APPENDIX 1. UML diagram of package com.motionavenue.Protector.obfuscate (part 3)

Figure 1.2. package com.motionavenue.Protector.obfuscate (part 3)

APPENDIX 1. UML diagram of package com.motionavenue.Protector.obfuscate (part 4)

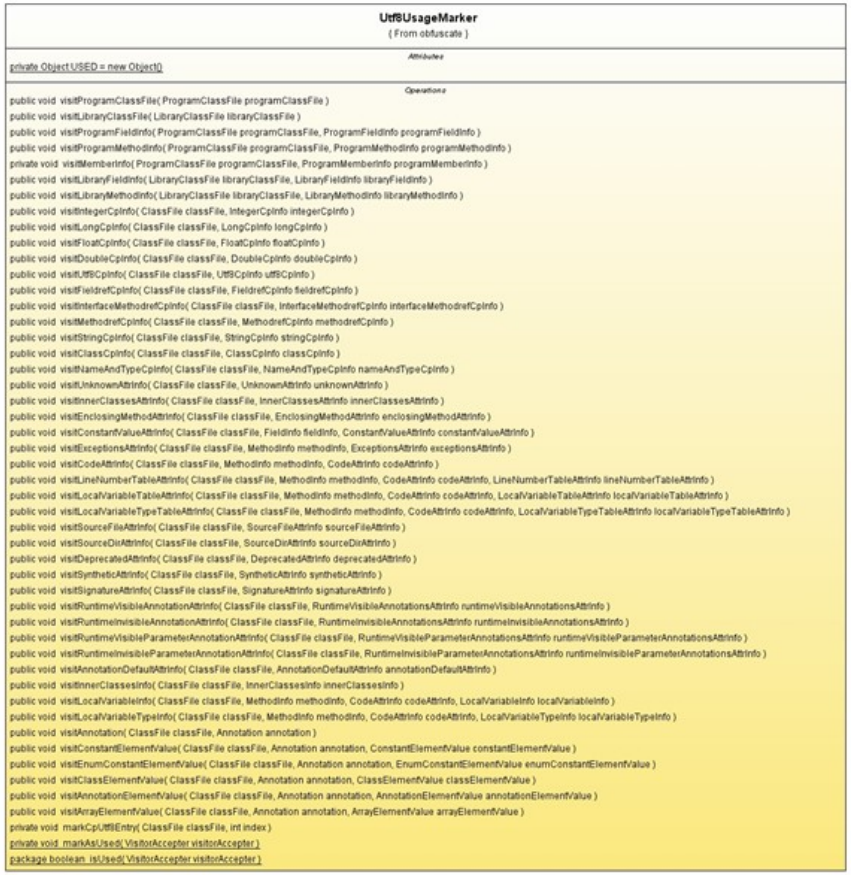


Figure 1.3. package com.motionavenue.Protector.obfuscate (part 4)

APPENDIX 1. UML diagram of package com.motionavenue.Protector.obfuscate (part 5)

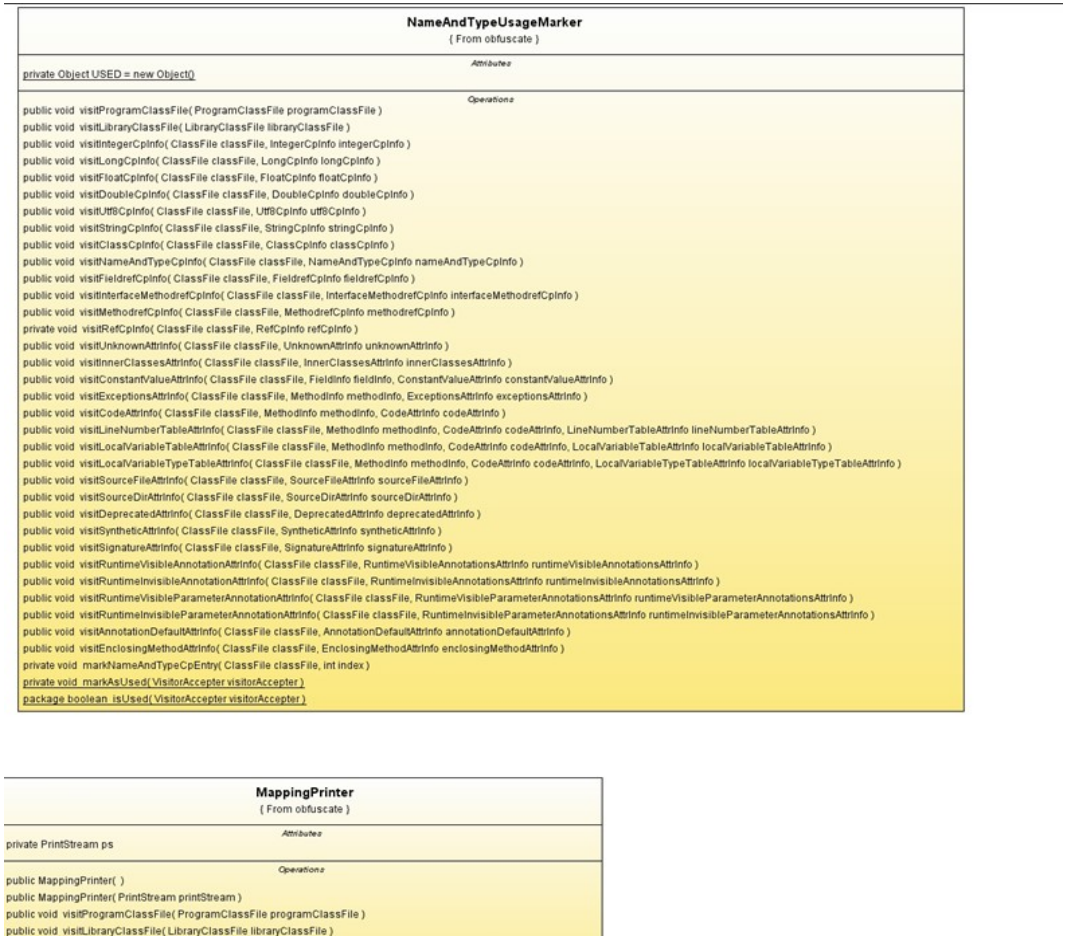


Figure 1.5 package com.motionavenue.Protector.obfuscate (part 5)

APPENDIX 2. Listing of build.xml file (part 1)

```
<project default="compile">
	<!-- == PROJECT NAME == -->
	<property name="projectname" value="protector"/>
	<!-- == DEBUG MODE ? == -->
	<property name="DEBUG" value="false" />
	<!-- == DIRECTORIES AND FILENAMES == -->
	<property name="src" value="src"/>
	<property name="dist" value="dist"/>
	<property name="javadoc.dir" value="javadoc"/>
	<!-- libraries needed for compiling the source -->
	<property name="buildlib" value="lib" />
	<!-- libraries needed in deployment -->
	<property name="deploylib" value="lib" />
	<!-- == TARGETS == -->
	<target name="init" description="Initialization">
		<tstamp />
		<mkdir dir="${dist}"/>
		<mkdir dir="${dist}/classes" />
		<mkdir dir="${dist}/jar" />
		<mkdir dir="${javadoc.dir}"/>
	</target>
	<!-- compile -->
	<target name="compile" depends="init" description="Compile project">
		<echo message="DEBUG? ${DEBUG}" />
		<javac source="1.5" target="1.5" encoding="UTF-8" srcdir="${src}"
destdir="${dist}/classes"
			debug="${DEBUG}" debuglevel="lines, vars, source">
			<compilerarg value="-Xlint:unchecked" />
			<classpath>
				<fileset dir="${buildlib}">
					<include name="**/*.jar"/>
					<include name="**/*.zip"/>
				</fileset>
			</classpath>
		</javac>
	</target>
	<!-- build JAR archive -->
	<target name="jar" depends="compile" description="Build JAR Class Archive">
		<jar destfile="${dist}/jar/${projectname}.jar" basedir="${dist}/classes">
			<manifest>
			<attribute name="Main-Class" value="com.motionavenue.Protector.Protector"/>
			</manifest>
		</jar>
	</target>
	<!-- clean up -->
	<target name="clean" description="Clean up distribution directories">
		<delete dir="${dist}"/>
```

APPENDIX 2. Listing of build.xml file (part 2)

```
        </target>
        <!-- == MISCELLANOUS == -->
        <!-- Javadoc generation -->
  <target name="javadoc" depends="init">
    <javadoc sourcepath="${src}" packagenames="com.motionavenue.*"
      destdir="${javadoc.dir}"
      author="true" version="true" use="true" package="true">
      <link
        href="http://java.sun.com/j2se/1.5.0/docs/api/" />
      <link
        href="http://www.junit.org/junit/javadoc/3.8.1/" />
                <classpath>
                        <fileset dir="${buildlib}">
                                <include name="**/*.jar"/>
                        </fileset>
                </classpath>
    </javadoc>
  </target>


        <!-- create ZIP file of .java and .jsp files (helps to keep everybody on track) -->
        <target name="srczip" depends="init" description="Create source package">
                <zip destfile="${projectname}-src-${DSTAMP}.zip">
                        <fileset dir="." includes="**/*.java,**/*.jsp,**/*.css,**/*.html,**/*.js" />
                </zip>
        </target>
        <!-- create a distribution directory and copy files. -->
        <!-- Remember to delete files that belong to clients that are not needed!!! -->
        <target name="dist" depends="jar" description="copies files for distribution">
                <delete dir="C:/Work/Projects/Protector"/>
                <copy todir="C:/Work/Projects/Protector/dist/jar">
                  <fileset dir="dist/jar"/>
                </copy>

                <copy todir="C:/ITVClient/lib">
                  <fileset dir="lib"/>
                </copy>
                <copy todir="C:/ITVClient/skins">
                  <fileset dir="skins"/>
                </copy>
                <copy todir="C:/Work/Projects/Protector">
                  <fileset dir=".">
                        <include name="*accelerated*.bat"/>
                        </fileset>
                </copy>
                <copy file="log4j.properties.dist" tofile="C:/Work/Projects/Protector/log4j.properties"/>
        </target>
</project>
```