Boss-Spy Design Document

Brendan Budine    Paul J. Melici    John Novatnack    Abbas Omar
{bdb23, pmelici, jmn27, aao23}@drexel.edu

Advisers: Drs. Ko Nishino and Ali Shokoufandeh (Department of Computer Science)
{kon, ashokouf}@cs.drexel.edu

March 15, 2006

Contents

1 Introduction ......................................................... 2
   1.1 Design Goals ................................................... 2

2 General Architecture ............................................ 3
   2.1 Overview ....................................................... 3

3 Structural Model ................................................ 4
   3.1 Graphical User Interface ...................................... 4
       3.1.1 The Logon screen .......................................... 4
       3.1.2 The Sign up screen ......................................... 4
       3.1.3 The Home Tab ............................................... 4
       3.1.4 The Monitor Tab ............................................. 4
       3.1.5 The Faces Tab .............................................. 7
       3.1.6 The Console Tab ............................................ 7
       3.1.7 The Preferences Tab ....................................... 7
       3.1.8 The About Tab ............................................. 7
   3.2 Java Core Classes ............................................. 8
       3.2.1 BossSpy .................................................... 8
       3.2.2 saveProps() ................................................ 8
       3.2.3 loadProps() ............................................... 8
       3.2.4 BackendDriver ............................................ 13
       3.2.5 BSUser ...................................................... 13
       3.2.6 BSFace ...................................................... 13
       3.2.7 BSAction ................................................... 13
       3.2.8 Properties file ............................................ 14
       3.2.9 Backend interface ......................................... 14
   3.3 Computation Unit ............................................. 15

4 Behavioral Model ................................................ 16
   4.1 General Behavioral Model .................................... 16
   4.2 Computation Unit ............................................. 16

5 Future Directions .............................................. 18
List of Figures

1. Overall Architecture .................................................. 3
2. The logon screen. Figure (a) shows the GUI logon screen. Figure (b) shows the UML diagram for the classes implementing the logon screen. ........................................... 5
3. The signup screen. Figure (a) is a screenshot of the GUI signup screen. Figure (b) illustrates the UML diagram of the associated classes in the design. ......................... 5
4. The home GUI screen. Figure (a) is a screenshot of the home screen of the GUI. Figure (b) illustrates the UML diagram of the associated classes in the design. ......................... 6
5. The monitor tab. Figure (a) is a screenshot of the GUI. Figure (b) illustrates the UML diagram of the associated classes in the design. ......................................................... 6
6. Graphical User Interface: The Faces Tab ........................................ 6
7. UML: GUI - Faces Tab .................................................. 7
8. UML: GUI - Associated Actions Pane (Faces Tab) ......................... 8
9. The console tab. Figure (a) is a screenshot of the GUI. Figure (b) illustrates the UML diagram of the associated classes in the design. ......................... 9
10. The preferences tab. Figure (a) is a screenshot of the GUI. Figure (b) illustrates the UML diagram of the associated classes in the design. ......................... 9
11. The about tab. Figure (a) is a screenshot of the GUI. Figure (b) illustrates the UML diagram of the associated classes in the design. ......................... 10
12. UML: Java Core - BossSpy ........................................... 11
13. Figure (a) UML: Java Core - BackendDriver. Figure (b) UML: Java Core - BSUser. ........................................... 12
14. Figure (a) UML: Java Core - BSFace. Figure (b) UML: Java Core - BSAction. ........................................... 14
15. Computation unit .................................................... 16
16. UML Sequence diagram illustrating a typical user scenario .................. 17
17. UML state diagram illustrating the computation unit behavior .................. 17

List of Tables

1. Introduction

For the modern day corporate employee, the work place is a suspicious and intimidating environment. With bosses of varying importance constantly "checking in", an employee must be in a constant state of alertness. Boss-Spy is a product designed to relax the work environment of employees. By making use of recent advances in facial detection and recognition, Boss-Spy provides early warning of a bosses' presence by delivering software warnings.

In this document we describe the design of the Boss-Spy system. The system is split into two components, (1) the graphical user interface (GUI) and (2) the computation unit. The design is such that these two components may be developed independently of each other, and then composed. The aggregate system will be one that is both visually appealing to the user, and also computationally efficient.

1.1 Design Goals

The Boss-Spy system was designed with two independent components, the graphical user interface and the computation unit. The GUI is designed to be implemented in Java using Swing. For the sake of efficiency, the computation unit is designed for C++.

Our goal is separate the responsibilities of these two components such that they may be independently developed, and then composed using messages passed through standard input and output. The design will be considered a success if this document leads to an implementation that reaches this goal.
2 General Architecture

2.1 Overview

Figure 1 presents a visual guide to Boss-Spy at a high level. First, we shall briefly discuss how the Boss-Spy backend operates (this appears on the right hand side of Figure 1):

- The Face Detection algorithm receives an image from the camera.
- The algorithm decides whether or not this particular image contains a face. If it does, the Face Recognition component is notified. If not, no further action is taken and the Face Detection component waits for the next image from the camera.
- The Face Recognition algorithm processes the image. It decides which face, if any, is present in the image. If a known face is observed then it sends a message to the GUI via standard output. Otherwise, the Face Recognition component waits for the next image to be received from the Face Detection component.

Next, let us examine how the GUI operates at a very high level:

- The Logon screen is displayed.
- The user either logs in (by supplying a valid username and password) or the user clicks sign up.

  If the user clicks sign up then the Sign Up Screen is presented at which point the user enters information such as name, username and password in order to create a new Boss-Spy account.

  If the user enters a valid username/password and clicks Log on the main screen is presented and the Home Tab is displayed. (see Figure 1)
• See the Behavioral Model section for further details.

3 Structural Model

3.1 Graphical User Interface

Each of the following subsections correspond to a particular screen or tab within the BOSS-SPY GUI. Before discussing each screen and tab, however, let us first discuss some common elements and design patterns. Private and protected data members which correspond to GUI elements use the following abbreviations:

• Btn: JButton
• Fld: JTextField or JPasswordField
• Lbl: JLabel
• List: JList
• CB: JComboBox
• TP: JTextPane
• Spn: JSpinner
• Sld: JSlider

Another heavily used design pattern involves the way buttons work. JButton s are linked to an implementation of the AbstractAction class. This has proven effective in increasing modularity and testability in other projects. The AbstractAction implementation class is typically an inner class of whatever Tab or Screen contains the JButton. For example, ALogOn is linked the the Log on button of the Log on screen (see Figures 2).

Like many Object Oriented programs, the getter and setter design pattern is used. Where appropriate each Screen and Tab class has getters and setters for fields. The following list describes the structural design of the components in the graphical user interface.

3.1.1 The Logon screen

See Figure 2(a) for the screen shot and Figure 2(b) for the UML diagram. The logon screen is fairly straightforward; the user types his user name and password in the appropriate fields, then clicks Log on. If the user is new to BOSS-SPY he may click Sign Up to create a new account.

3.1.2 The Sign up screen

See Figure 3(a) for the screen shot and Figure 3(b) for the UML diagram. This is where the user may create a new BOSS-SPY account. The SignUpPane class is very simple, consisting only of getters/setters and one derivative of AbstractAction.

3.1.3 The Home Tab

See Figure 4(a) for the screen shot and Figure 4(b) for the UML diagram. This class is also very straightforward. The UML diagram is self-explanatory.

3.1.4 The Monitor Tab

See Figure 5(a) for the screen shot and Figure 5(b) for the UML diagram. This class is very straight-forward and consists only of getters/setters.
Figure 2: The logon screen. Figure (a) shows the GUI logon screen. Figure (b) shows the UML diagram for the classes implementing the logon screen.

Figure 3: The signup screen. Figure (a) is a screenshot of the GUI signup screen. Figure (b) illustrates the UML diagram of the associated classes in the design.
Figure 4: The home GUI screen. Figure (a) is a screenshot of the home screen of the GUI. Figure (b) illustrates the UML diagram of the associated classes in the design.

Figure 5: The monitor tab. Figure (a) is a screenshot of the GUI. Figure (b) illustrates the UML diagram of the associated classes in the design.
3.1.5  The Faces Tab

See Figure 6 for the screen shot and Figures 7 and 8 for the UML diagrams. While by far the most complex class in the Java GUI the `FacesTab` class and the `AssocActionsPane` class remain fundamentally simple, consisting of getters/setters.

Information is stored in `BSUser`, `BSFace`, and `BSAction` objects local to the `FacesTab` instance. Thus, changes are not propagated back to the `BossSpy.user BSUser` instance until the user clicks **Apply/Save Changes**.

Upon clicking **Apply/Save Changes** any local changes are propagated back to `BossSpy.user, boss_spy.properties` is written, and the backend is restarted via `BackendDriver.restart()`.

Clicking **Discard Changes** causes the fields to be re-populated with the data from the most recent `boss_spy.properties`.

3.1.6  The Console Tab

See Figure 9(a) for the screen shot and Figure 9(b) for the UML diagram. This class interfaces with the `BackendDriver` class to provide both observation and interaction with the Backend.

For example, clicking **Restart Backend** calls `BackendDriver.restart()`. Typing a command in the **Command:** field and clicking **Send** results in `BackendDriver.sendCommand()` being executed.

3.1.7  The Preferences Tab

See Figure 10(a) for the screen shot and Figure 10(b) for the UML diagram. This class works in a similar fashion to `FacesTab` in that it stores changes locally until the user clicks **Apply/Save Changes**.

Upon clicking **Apply/Save Changes** any local changes are propagated back to `BossSpy.user, boss_spy.properties` is written, and the backend is restarted via `BackendDriver.restart()`.

3.1.8  The About Tab

See Figure 11(a) for the screen shot and Figure 11(b) for the UML diagram. This is a very simple class to display the credits for **BOSS-Spy**.
3.2 Java Core Classes

3.2.1 BossSpy

See Figure 7 for a UML diagram of this class. This is the core class for the application. In order to increase modularity and avoid having one giant class many tasks have been delegated into other classes discussed later.

The BossSpy class is responsible for starting the application and initializing the other components. The MainFrame class represents the overall main screen of Boss-Spy. All of the GUI Tab classes are aggregated by this class.

3.2.2 saveProps()

Save the contents of the props data member to boss_spy.properties using Properties.store()

3.2.3 loadProps()

Load boss_spy.properties into private data member props using Properties.load()

See Figure 7 for a UML diagram of this class. The class BossSpy contains the following methods:

- saveProps()
  
  Save the contents of the props data member to boss_spy.properties using Properties.store()

- loadProps()
  
  Load boss_spy.properties into private data member props using Properties.load()
Figure 8: UML: GUI - Associated Actions Pane (Faces Tab)

Figure 9: The console tab. Figure (a) is a screenshot of the GUI. Figure (b) illustrates the UML diagram of the associated classes in the design.
Figure 10: The preferences tab. Figure (a) is a screenshot of the GUI. Figure (b) illustrates the UML diagram of the associated classes in the design.

Figure 11: The about tab. Figure (a) is a screenshot of the GUI. Figure (b) illustrates the UML diagram of the associated classes in the design.
Figure 12: UML: Java Core - BossSpy
Figure 13: Figure (a) UML: Java Core - BackendDriver. Figure (b) UML: Java Core - BSUser.
3.2.4 BackendDriver

See Figure 13(a) for a UML diagram of this class. This class is responsible for interfacing with the backend (which does the actual face detection and face recognition). See section 3.2.9 for the specification of this interface.

3.2.5 BSUser

See Figure 13(b) for a UML diagram of this class. This class can be thought of as a wrapper class for the Java Properties class. To understand how this class works, one must understand the getProp() and setProp() methods. Here is the psuedo-code for each:

```java
void setProp(String key, String value) {
    props.setProperty( "user." + userId + "." + key, value);
}
String getProp(String key) {
    return props.getProperty( "user." + userId + "." + key);
}
```

Thus, the implementation for the getUserName() method might look as follows:

```java
String getUserName() {
    return getProp("user_name");
}
```

The other methods operate in the same manner, with a few exceptions. These more complicated methods are discussed in the following list.

- **getFaceById(face_id:int): BSFace**
  
  This method returns a BSFace object with the specified face_id. Most of the work is done in the BSFace.loadFrom() method.

- **getFaces(): List**
  
  Returns a List of all BSFaces present. This is achieved by iteratively calling the getFaceById() method and storing the results in a List.

- **isValidPassword(passwd:String): boolean**
  
  Computes the MD5 sum for the passwd argument and compares it to the value stored in the .properties file. Returns true if the two match, false otherwise. The password is stored and authenticated this way to avoid storing the user’s password in plain text.

3.2.6 BSFace

See Figure 14(a) for a UML diagram of this class. The class serves as a wrapper for a "face object" stored in the boss_spy.properties file. The getter/setter methods affect the private data members. Calling saveTo() causes the values stored in the private data members to be written to the Properties object of the corresponding BSUser object. Conversely, calling loadFrom() initializes the private data members from the Properties object of the corresponding BSUser object.

3.2.7 BSAction

See Figure 14(b) for a UML diagram of this class. This class works in the same manner as BSFace.
3.2.8 Properties file

All data (except for the actual face images) needed by Boss-Spy is stored in a Java .properties file called boss_spy.properties. The syntax of this file is as follows:

```
line=user.<user_id>.<user_prop>=<value>
value=<string>
user_id=0...n
user_prop=user_name|first_name|last_name|passwd_hash|<pref>|<face>
pref=pref.<pref_key>
pref_key=<string>
face=face.<face_id>.<face_key>
face_id=0...n
face_key=name|img|<action>
action=action.<action_id>.<action_key>
action_id=0...n
action_key=name|type|arg
```

Here is a stripped down sample boss_spy.properties file to clarify the above syntax for the reader:

```
user.2.user_name=paul
user.2.first_name=Paul
user.2.last_name=Melici
user.2.passwd_hash=e621a8394319eb0bb3aa676008c0a93f
user.2.pref.sensitivity=5
user.2.face.5.name=John
user.2.face.5.img=john.png
user.2.face.5.action.0.name=Play sound.wav
user.2.face.5.action.0.type=play_sound
user.2.face.5.action.0.arg=sound.wav
```

3.2.9 Backend interface

The command line syntax for starting the backend is as follows:

```bash
14
```
Once started, the GUI and the backend communicate via standard input/output. The backend accepts the following commands:

- `debug <0...10>` - sets the debug level for the backend. The higher the debug level, the more verbose output will be.
- `quit` - causes the backend to exit.

The backend sends the following messages via standard output:

- `+READY` - sent after initialization is completed successfully.
- `+OK` - this is sent in response to a successful change in the debug level
- `+FOUND <FACE_IMG>` - sent when FACE_IMG is detected and recognized by the algorithm.
- `## <DEBUG_LEVEL>) <DEBUG_MSG>` - debug output is delivered in this manner
- `+WARN <WARN_MSG>` - used to notify the GUI of a non-fatal error.
- `+ERR <ERR_MSG>` - used to notify the GUI of an error.

To make the above more concrete here is an example of what running the backend from the command line might look like:

```
paul@some_unix_box:~$ ./backend 1 5 john.jpg paul.jpg
+READY
## 1) Face detected... attempting recognition.
+FOUND john.jpg
quit
paul@some_unix_box:~$
```

In the above example all text is printed by the backend, except for the `quit` at the end. This was typed by the user to exit the program.

But perhaps john.jpg doesn’t exist. Let’s see what the session would look like in this scenario:

```
paul@some_unix_box:~$ ./backend 0 5 john.jpg paul.jpg
+ERR FATAL: Face image not found: john.jpg
paul@some_unix_box:~$
```

### 3.3 Computation Unit

Figure [17] is a UML class diagram of the computation unit of BOSS-SPY. The primary class is `ComputationUnit`, which is called, as a binary, from the graphical user interface. The class begins the search for target individuals and takes the following parameters:

1. The debug level
2. The algorithm sensitivity
3. The list of target images
The \textit{ComputationUnit} class creates an instance of the generic camera class in order to receive images from the video feed. Our design is such that this generic class will serve as a virtual class that will be inherited by implementations that work over specific hardware links with the camera. For our implementation, we will be developing a camera class that will receive images over an IEEE 1394, or firewire connection.

In order to process images from the camera, the \textit{ComputationUnit} class creates instances of the \textit{FaceDetect} and \textit{FaceRecognition} classes. The \textit{FaceDetect} class has a single method, which takes an image and localizes a face. The class itself uses Intel's OpenCV computer vision library. Lastly, the \textit{FaceRecognition} takes an image, and a list of targets and returns which, if any, of the targets are in the image.

4 Behavioral Model

4.1 General Behavioral Model

Figure 16 includes a UML sequence diagram illustrating a typical user session with the BOSS-SPY system. Initially, a user logs into the system through the GUI. If the login was successful then the user is notified. The GUI then creates an instance of the computation unit. Once started, the computation unit opens up communication with the camera. While running, the computation unit passes images from the camera, as well as messages about algorithm performance, back to the GUI. The computation unit runs until one of the targets has been identified, in which case the instances halts and the GUI performs the action specified by the user.

4.2 Computation Unit

Figure 17 contains a state diagram that illustrates the control and data flow in the computation unit. After an instance of the computation is created (denoted by the filled in black circle on the left), the computation unit begins and waits for an image from the camera. Once an image is received, two concurrent threads are created. The first sends the image to the GUI and then halts. The second processes the image, but first localizing any faces, and then searching for any targets. If the recognition algorithm successfully detects a face then the GUI is notified, otherwise the process begins again with a new image from the camera.

Figure 15: Computation unit
Figure 16: UML Sequence diagram illustrating a typical user scenario

Figure 17: UML state diagram illustrating the computation unit behavior
5 Future Directions

The modular design of the Boss-Spy system lends itself to concurrent development. Implementation of our design will be centered around three efforts to independently develop a graphical user interface, computation unit on test data and camera communication. Once this has been accomplished the algorithms in the computation unit will be combined with the camera interface. Once the computation unit has been thoroughly tested, it will be combined with the graphical user interface to create a complete system.