Project 18

Design and Implementation of a Modular Biometric Authentication System

TDDC03 Projects, Spring 2007

Ignacio Porres
ignpo864@student.liu.se

Victor Martinez Rubio,
vicma863@student.liu.se

Supervisor: Christoph Schuba

Ignacio Porres
ignpo864@student.liu.se

Victor Martinez Rubio,
vicma863@student.liu.se

Design and Implementation of a Modular Biometric Authentication System

Abstract

Biometrics authentication is an actual topic in information society. It consists of a set of methods used to identify a person from his unique characteristics. This paper will present an overview of biometric authentication system. As a proof of concept, the design and implementation of a generic biometric authentication system will be exhibited.

It serves as the basics for any kind of biometric recognition algorithms created in upcoming projects of the course as well as for evaluating any kind of biometric input.

The design is based on six main components of the typical system architecture: the portal, central controlling unit, input unit, feature extraction module and the biometric matching algorithm. The implementation is done with Java, an object-oriented programming language.

A simple prototype has been developed, which takes only a password as input since it would be similar to any biometric input receives as a bit stream.

Key words
Biometric, Authentication, Verification, Identification.

1. Introduction

Humans always have had the preoccupation of developing systems for personal identification, systems that tell us that a person is the one he or she claims to be.

There are several systems such as signature, retinal, fingerprints and voice recognitions. Each one needs a specific algorithm more or less complex depending on the type of the biometric.

Development of the Internet and in general by the telecommunication and information systems in the last years, together with the reduction in price and the massive construction of computers and sensors, has caused an increasing interest in systems that allow the establishment of the identity of an individual in automated form. As a direct consequence of this fact, the biometry, science dedicated to the identification of individuals from an anatomical characteristic or a characteristic of its behavior, has been transformed into a growing area.

In this paper, we will describe how a biometric system works, how its components interact and relate between them. Clear definitions are stated about the two main objectives of a biometric system: identification and verification. The background theory is followed by a brief description of the problem we are solving. Subsequently, the paper will present the existing research about this case and will denote the differences existing in our system. As a final point will be shown the methodology used while developing the system. The design discussion and the conclusions with the results obtained will also be presented with a simple demonstration of our system.

2. Background

2.1. Biometric definitions

Biometrics (ancient Greek: bios ="life", metron ="measure") is the study of methods for uniquely recognizing humans based upon one or more intrinsic physical or behavioral traits.[1]

Required characteristics: These requirements are used as an indicator to validate or contradict a specific biometric characteristic.

- Universality: every person has this biometric characteristic.
- Uniqueness: the probability of two persons with an equal biometric characteristic is very low.
- Permanence: the characteristic does not change over time.
- Collectability: the characteristic has to be able to be measured.

Multi-modal biometric: The use of several biometrics. It can be by an AND or OR combination. The availability is increased with the OR and the accuracy with the AND.

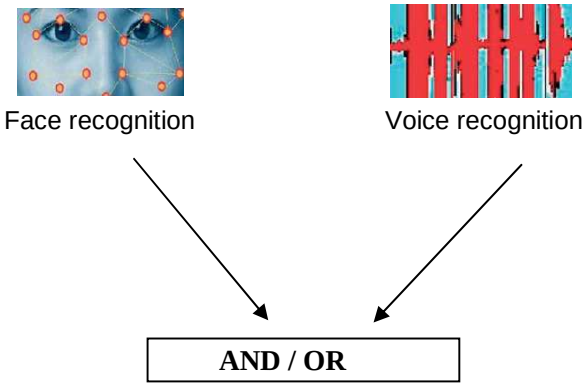


Figure 1. Multimodal system

2.1. Biometric System Components

Portal: Its purpose is to protect certain assets. An example of a portal is the gate at an entrance of a building. If the user has been successfully authenticated and is authorized to access an object then access is granted.

Central controlling unit: receives the authentication request, controls the biometric authentication process and returns the result of user authentication.

Input device. The aim of the input device is biometric data acquisition. During the acquisition process, the user's liveness and quality of the sample will be verified.

Feature extraction module processes the biometric data. The output of the module is a set of extracted features suitable for the matching algorithm. During the feature extraction process, the module will also evaluate the quality of the input biometric data.

Storage of biometric templates. This will typically be some kind of a database. Biometric templates can also be stored on a user-held medium (e.g., smartcard). In that case, a link between the user and her biometric template must exist (e.g., in the form of an attribute certificate).

The biometric matching algorithm compares the current biometric features with the stored template. The desired security threshold may be a parameter of the matching process. In this case, the result of the matching will be a yes/no answer. Otherwise, a score representing the similarity between the template and the current biometric sample is returned. The central unit then makes the yes/no decision.

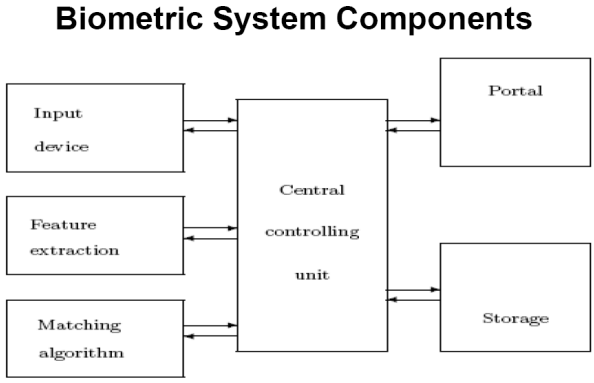


Figure 2. Biometric System Components

2.2. Enrollment

The enrollment module acquires and stores the originating data of the biometric with the intention of comparing this data with the proportionate one later incoming to the system. The tasks executed by the enrollment module are made possible thanks to the action of the biometric reader and the feature extractor. The first one is in charge of collecting data relative to the chosen biometric indicator and to give a representation of this in a digital format. The second extracts represented characteristic of the indicator from the output of the reader. The previous set of characteristics will be stored in a central database as a template. That template is the representative information of the biometric indicator and it will be used in the different parts of the system.

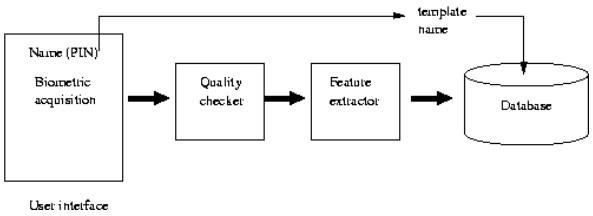


Figure 3. Creating a template for a user

2.3 Verification

The user provides the system with its identity and its biometric characteristic. It is compared with what is stored in the data base. Finally, it is confirmed if the user is the one he is claiming to be.

The modules are as follows: first, the biometric characteristic reader that converts the input to a digital format, with the quality check that ascertains good quality of the input. Then, the feature extractor compresses the data and gives a template format to be compared with the template of the expected person using the matcher.

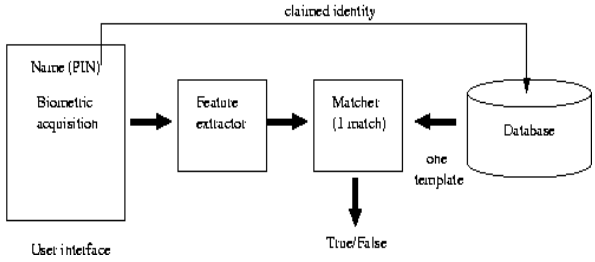


Figure 4. Verification

2.4. Identification

The target of the system is to identify a person taking its biometric characteristic and comparing it with the database to determinate who is the owner of this characteristic.

The same modules as in the Verification task are used. The difference is that all the templates in the database are needed since the user does not claim to be someone, unlike the verification process.

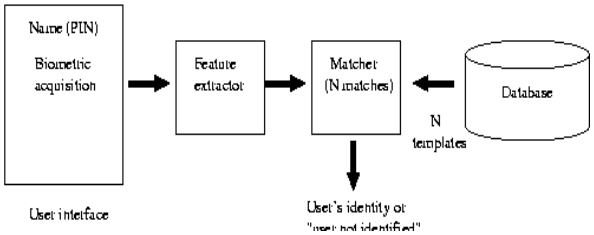


Figure 5. Identification

3. Problem Statement

A system that can be used as a generic biometric authentication system will be designed and implemented. It must be capable of handling user enrollment, simple and multi-modal user identification, and support the dynamic adjustment of its usability and performance metrics.

The system could use any biometric input device because it uses a bit stream. Its interface is defined as realistically as possible in order to make it easy to use and understand for other programmers when trying to expand or just adding new features such as a new matching algorithm.

Furthermore, the system will be verified with a simple example consisting of password validation.

4. Related work

With the apparition of the API (Application Programming Interface), the programmer has a very useful tool for the creation of a generic biometric system.

A standard biometric API helps the user to easily rebuild a biometric system if he only wants to change some feature or just change the type of biometric.

There are several programs related to the work done as a result of this paper. One important document that has been found is the BIOApi specification that shows the programmer the best practices and standards for developing a biometric system.

The consortium named BIOApi has defined the relation between the biometric and API. It is formed by Compaq, IBM, Identicator, Microsoft, Miros and Novell.

“The BIOApi Specification is intended to provide a high-level generic biometric authentication model; one suited for any form of biometric technology. It covers the basic functions of Enrollment, Verification, and Identification, and includes a database interface to allow a biometric service provider (BSP) to manage the Identification population for optimum performance. It also provides primitives that allow the application to manage the capture of samples on a client, and the Enrollment, Verification, and Identification on a server.” [2]

The differences most easily observed when analyzing the system demonstrated in this paper are that it is a simplified system. It is also an API with the basic functions of a biometric system (Authentication, Identification and Enrollment) but the build classes are simpler yet perfectly valid for a generic system. Several inputs also connected to the same server can be used, being concurrent as occurs in the BIOApi application.

Another competing product is BioWebAuthentication, which is a platform that allows the acquisition of a database of biometrics. It is also based on Java and it is focused on web authentication. It includes a client application to be used with any software or device using the BIOApi standard. In addition, the system has a basic face recognition algorithm, still being developed.

As can be seen, the standard BIOApi is behind almost every biometric system. It can be found in many programs, most of them open source.

5. Methodology

To build the system, Java technology has been chosen. From this can be obtained all the features needed by the object-oriented paradigm.

The developed system is a modular framework; its purpose is to build reusable components for the development of several biometric systems. Hence, the object-oriented paradigm is used, applying abstract classes, inheritance, modularity, encapsulation, polymorphism and interfaces.

Abstract: this characteristic is implemented in the BAST class and ClientT class.

Interfaces: it has been modeled in the DatabaseT class; the biometric system has a database which is used for saving each template. The system is database-agnostic, so it can be used with many different databases.

The serialization has been implemented in the messages exchange between the server and the client with sockets. It is showed in BiometricMessage class.

Encapsulation is used in ValueT class to protect the data and its access.

The inheritance is implemented for each kind of biometric system with BAST class.

One of the main points in the program is the scalability that will make future work easily extensible.

Furthermore, care has been put in creating a distributed system. The framework uses the client-server model; it can be executed in several machines where there might be one or more clients at the same time. The server is the BAS and the client is the biometric input device.

The communication between the server and the client has been implemented with sockets. These sockets are not secure. No kind of message encryption exists.

The system is concurrent; the BAS can receive several queries from different input devices.

6. Design

The relations between each class and interface are shown in the class diagram of the BAS framework modelled. [See Figure 1 Appendix]

In this diagram, there are three main classes: BAST, ServerT and ClientT. The first one is related to the DataBaseT and the Server class has a specific BAST instance. The second represents the client connection; it is like an interface between the biometric input device and the Servert. The communication between the Server and Client is made via BiometricMessage and Response classes. The BiometricMessage class is using the Request and the kind of biometric task (Enrollment, Verification or Identification). It is used when the client sends a message to the server. On the other hand, the message from the server to the client is the ResponseT class. The BiometricType class contains each kind of error or confirmation of successful function in the system.

The specific biometric system is implemented in the next diagram [See figure 2 Appendix]. In this diagram, there are four main classes in the framework: BAST, SocketServer, ClientT and DatabaseT, which are implemented for each specific biometric system. The main component is the BAST class which has defined specific functions for each kind of biometric system: feature extractor, quality checker and algorithm matcher. The general functions such as enrollment, verification and identification are used independently of the type of biometric.

The ClienT class is an interface between the biometric input device and the biometric authentication system.

7. Discussion

As it is said in the methodology and the design part above, the main points of a good program have been followed in order to make future uses of the system easier. It may be developed with a few more classes but the system would not be as clear as it is now. A balance must be found between the number of classes and clarity. A bad relation may incur a problem of non-scalability, which is

one of the main points that have been looked up in the developed of the system.

8. Conclusion

The results obtained with the simple demo are not based on a real biometric, but since it is a generic system, a binary input will work as a “real” biometric input with its own algorithm.

References

[1] **Wikipedia**. (2007, April). [Online]. Available: *http://www.wikipedia.org*

[2] **BioAPI Consortium**. (2007, April). [Online]. Available: *http://www.bioapi.org/index.html*

[3] **Building Biometric Authentication for J2EE, Web, and Enterprise Applications**. ***Sun Microsystems, Inc***. (2007, April 10). [Online]. Available: *http://developers.sun.com/prodtech/identserver/reference/techart/bioauthentication.html*

[4] **Liveness detection in biometric systems**. ***Biometrics Info***. (2007, April 10). [Online]. Available: *http://www.biometricsinfo.org/whitepaper1.htm*

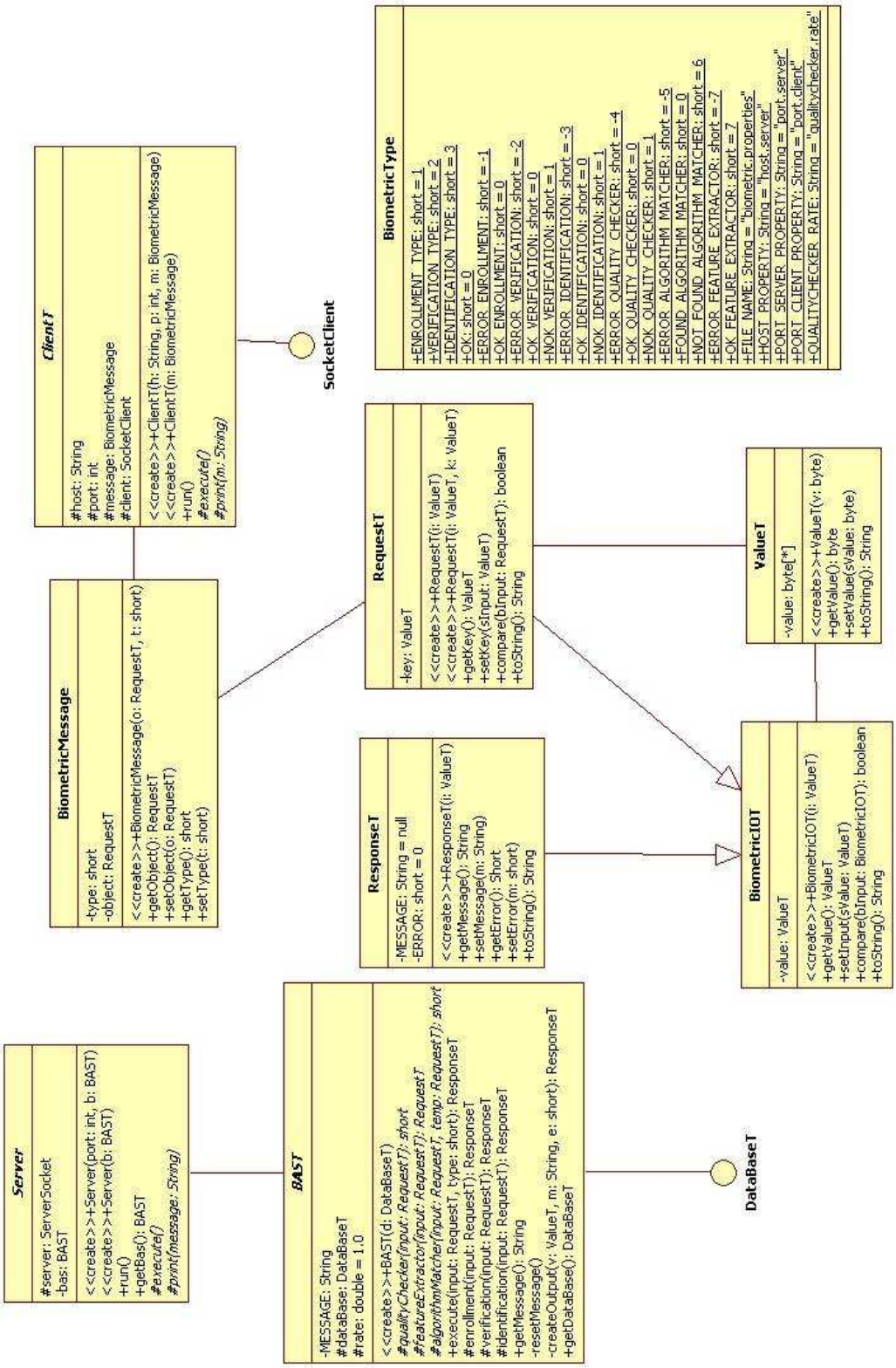
[5] **Biometric SDK**. ***SourceForge .net***. (2007, April). [Online]. Available: *http://sourceforge.net/projects/biometricsdk*

[6] **BioWebAuthentication**. ***Vigo University***. (2007, April). [Online]. Available: *https://www.webapps.gts.tsc.uvigo.es/biowebauth/action/common/setlocaleindex?lang=es*

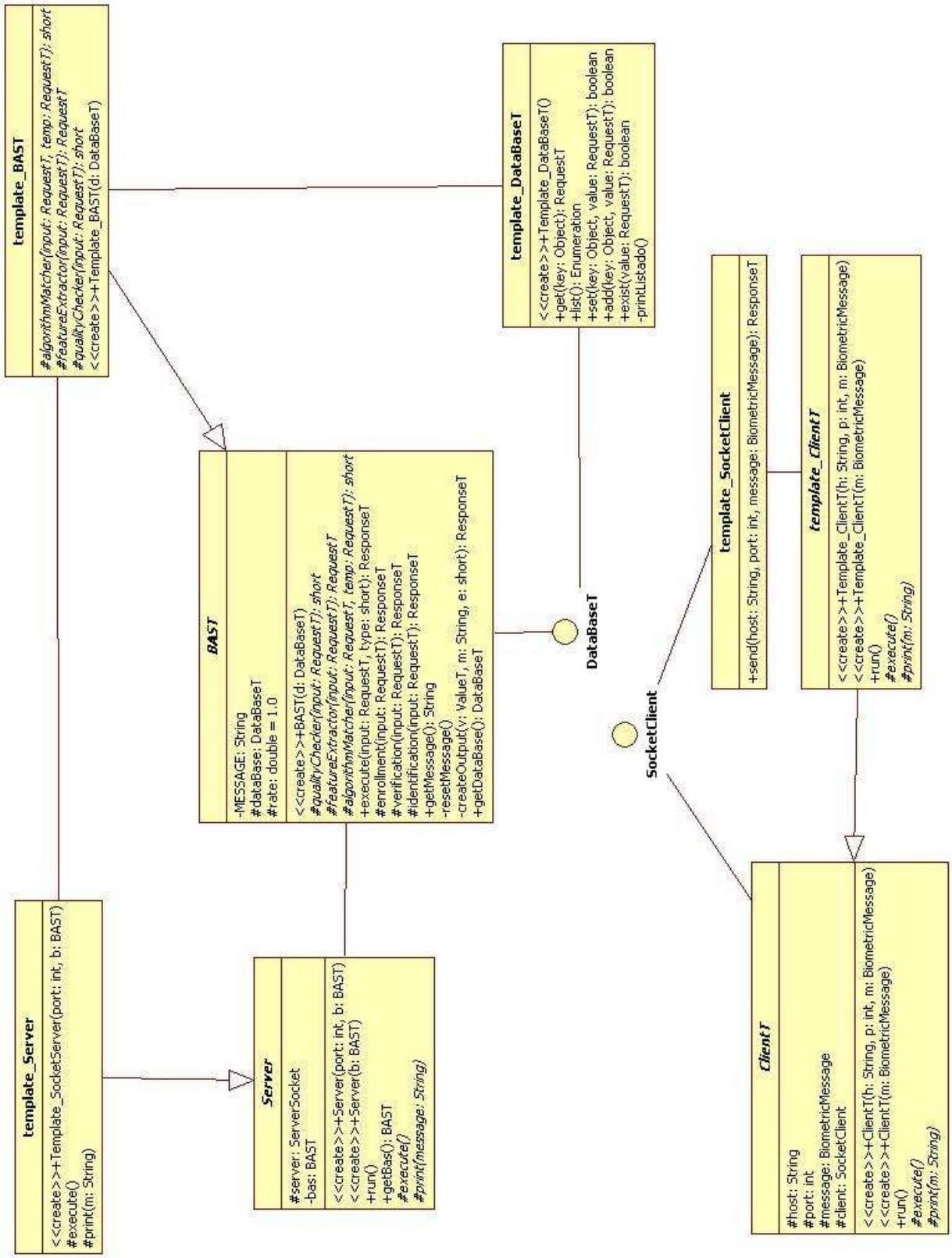
[7] **The Biometrics Resource Center Website**. (2007, April). [Online]. *Available: http://www.itl.nist.gov/div893/biometrics*

[11] U. Uludag, S. Pankanti, S. Prabhakar and K. Jain, “Biometric Cryptosystems: Issues and Challenges” *Michigan State University, Biometrics Researchs*, 2004.

Appendix 1


Server
#server: ServerSocket
-bas: BAST
<<create>>+Server(port: int, b: BAST)
<<create>>+Server(b: BAST)
+run()
+getBas(): BAST
#execute()
#print(message: String)
BAST
-MESSAGE: String
#dataBase: DataBaseT
#rate: double = 1.0
<<create>>+BAST(d: DataBaseT)
#qualityChecker(input: RequestT): short
#featureExtractor(input: RequestT): RequestT
#algorithmMatcher(input: RequestT, temp: RequestT): short
+execute(input: RequestT, type: short): ResponseT
#enrollment(input: RequestT): ResponseT
#verification(input: RequestT): ResponseT
#identification(input: RequestT): ResponseT
+getMessage(): String
-resetMessage()
-createOutput(v: ValueT, m: String, e: short): ResponseT
+getDataBase(): DataBaseT
DataBaseT
BiometricMessage
-type: short
-object: RequestT
<<create>>+BiometricMessage(o: RequestT, t: short)
+getObject(): RequestT
+setObject(o: RequestT)
+getType(): short
+setType(t: short)
ClientT
#host: String
#port: int
#message: BiometricMessage
#client: SocketClient
<<create>>+ClientT(h: String, p: int, m: BiometricMessage)
<<create>>+ClientT(m: BiometricMessage)
+run()
#execute()
#print(m: String)
SocketClient
RequestT
-key: ValueT
<<create>>+RequestT(i: ValueT)
<<create>>+RequestT(i: ValueT, k: ValueT)
+getKey(): ValueT
+setKey(sInput: ValueT)
+compare(bInput: RequestT): boolean
+toString(): String
ResponseT
-MESSAGE: String = null
-ERROR: short = 0
<<create>>+ResponseT(i: ValueT)
+getMessage(): String
+setMessage(m: String)
+getError(): Short
+setError(m: short)
+toString(): String
BiometricIOT
-value: ValueT
<<create>>+BiometricIOT(i: ValueT)
+getValue(): ValueT
+setInput(sValue: ValueT)
+compare(bInput: BiometricIOT): boolean
+toString(): String
ValueT
-value: byte[*]
<<create>>+ValueT(v: byte)
+getValue(): byte
+setValue(sValue: byte)
+toString(): String
BiometricType
+ENROLLMENT_TYPE: short = 1
+VERIFICATION_TYPE: short = 2
+IDENTIFICATION_TYPE: short = 3
+OK: short = 0
+ERROR_ENROLLMENT: short = -1
+OK_ENROLLMENT: short = 0
+ERROR_VERIFICATION: short = -2
+OK_VERIFICATION: short = 0
+NOK_VERIFICATION: short = 1
+ERROR_IDENTIFICATION: short = -3
+OK_IDENTIFICATION: short = 0
+NOK_IDENTIFICATION: short = 1
+ERROR_QUALITY_CHECKER: short = -4
+OK_QUALITY_CHECKER: short = 0
+NOK_QUALITY_CHECKER: short = 1
+ERROR_ALGORITHM_MATCHER: short = -5
+FOUND_ALGORITHM_MATCHER: short = 0
+NOT_FOUND_ALGORITHM_MATCHER: short = 6
+ERROR_FEATURE_EXTRACTOR: short = -7
+OK_FEATURE_EXTRACTOR: short = 7
+FILE_NAME: String = "biometric.properties"
+HOST_PROPERTY: String = "host.server"
+PORT_SERVER_PROPERTY: String = "port.server"
+PORT_CLIENT_PROPERTY: String = "port.client"
+QUALITYCHECKER_RATE: String = "qualitycheckerrate"

Appendix 2


template_BAST
#algorithmMatcher(input: RequestT, temp: RequestT): short
#featureExtractor(input: RequestT): RequestT
#qualityChecker(input: RequestT): short
<<create>>+Template_BAST(d: DataBaseT)
template_DataBaseT
<<create>>+Template_DataBaseT()
+get(key: Object): RequestT
+list(): Enumeration
+set(key: Object, value: RequestT): boolean
+add(key: Object, value: RequestT): boolean
+exist(value: RequestT): boolean
-printListado()
BAST
-MESSAGE: String
#dataBase: DataBaseT
#rate: double = 1.0
<<create>>+BAST(d: DataBaseT)
#qualityChecker(input: RequestT): short
#featureExtractor(input: RequestT): RequestT
#algorithmMatcher(input: RequestT, temp: RequestT): short
+execute(input: RequestT, type: short): ResponseT
#enrollment(input: RequestT): ResponseT
#verification(input: RequestT): ResponseT
#identification(input: RequestT): ResponseT
+getMessage(): String
-resetMessage()
-createOutput(v: ValueT, m: String, e: short): ResponseT
+getDataBase(): DataBaseT
DataBaseT
template_SocketClient
+send(host: String, port: int, message: BiometricMessage): ResponseT
SocketClient
template_ClientT
<<create>>+Template_ClientT(h: String, p: int, m: BiometricMessage)
<<create>>+Template_ClientT(m: BiometricMessage)
+run()
#execute()
#print(m: String)
template_Server
<<create>>+Template_SocketServer(port: int, b: BAST)
#execute()
#print(m: String)
Server
#server: ServerSocket
-bas: BAST
<<create>>+Server(port: int, b: BAST)
<<create>>+Server(b: BAST)
+run()
+getBas(): BAST
#execute()
#print(message: String)
ClientT
#host: String
#port: int
#message: BiometricMessage
#client: SocketClient
<<create>>+ClientT(h: String, p: int, m: BiometricMessage)
<<create>>+ClientT(m: BiometricMessage)
+run()
#execute()
#print(m: String)