FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Development of a Generic Fingerprint Authentication Framework

Andre Filipe Tavares

Report of Project/Dissertation

Master in Informatics and Computing Engineering

Supervisor: Antonio Miguel Pontes Pimenta Monteiro (Professor)

3rd March, 2009

Development of a Generic Fingerprint Authentication Framework

Andre Filipe Tavares

Report of Project/Dissertation

Master in Informatics and Computing Engineering

March, 2009

Confidencial

Abstract

In any line of business it is impossible not to depend on third-party solutions and the obvious drawback is that we are then limited to the environments these support. It is precisely the case of the ALERT® software suite, which relies on Digital Persona's fingerprint authentication technology. This project arose from the urgent need to eliminate this dependency, enabling support for different environments and equipment incompatible with the current solution.

Early on the research about the subject, fingerprint authentication, it was concluded that it follows the exact same process as any other type of biometric authentication. Thus, it makes perfect sense the approach of a generic biometric framework, where the fingerprint module will fit. This way, other types of biometric technology can be easily added to the Framework. After evaluating available market solutions for both the biometric framework and the fingerprint module, a decision was made of developing a generic biometric authentication framework internally and using the current available fingerprint technology, Digital Persona, as a module of the framework.

The biometric framework was then designed and implemented. An API was defined in order to allow changes to its core without breaking the applications that depend on it and to make it modular enough to support not only different biometric technologies but, for each technology, different vendors. This way, the application now doesn't depend on a specific technology/vendor. Finally, the current available fingerprint technology was fitted in the biometric framework.

Tests of this solution on the several environments it is expected to run were successful.

Resumo

Em qualquer ramo de negócio é impossível não depender de soluções de terceiros e, deste modo, estar limitado aos ambientes que estas suportam. É precisamente o que se passa com o software ALERT®, que depende de tecnologia de autenticação por impressão digital da Digital Persona. Este projecto surge da necessidade urgente de eliminar esta dependência, alargando o suporte a diversos ambientes e equipamentos que esta não suporta.

Cedo na pesquisa sobre a temática de autenticação por impressão digital se conclui que esta segue o mesmo processo que qualquer outro tipo de autenticação por biometria. Desta maneira, faz todo o sentido a abordagem de uma framework genérica de autenticação por biometria, onde o módulo de impressão digital será inserido. Assim, outros tipos de autenticação por biometria passarão a ser passíveis de ser suportados. Após avaliar as soluções de mercado existentes quer para a framework de biometria, quer para o módulo de impressão digital, é tomada a decisão de desenvolver uma framework genérica de autenticação por biometria internamente e utilizar a tecnologia de autenticação por impressão digital existente, da Digital Persona, como módulo da mesma.

A framework é, então, desenhada e implementada. Uma API é definida por forma a permitir que alterações na Framework não tenham impacto na aplicação que dela depende e a tornem modular o suficiente para permitir não só o suporte de diferentes tipos de tecnologias de autenticação por biometria mas igualmente de diferentes fornecedores para cada uma. Desta maneira, a aplicação deixa de depender numa tecnologia / fornecedor específico. Por fim, a solução actual de autenticação por impressão digital é adicionada à framework de biometria.

Os testes à framework de biometria nos ambientes em que é esperado que esta corra foram realizados com sucesso.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

In any line of business it is impossible not to depend on others. The software industry is no different. In the development of any product there are always dependencies of features that aren't directly related to it and it is unrealistic to believe that, while theoretically possible, it is feasible to develop them internally.

Thus, most of the time, the most logical option is to adopt third-party solutions. It is important, however, to isolate these dependencies as much as possible to make sure that the final product is never put in jeopardy.

More than a mere financial matter, this is a strategic one. IT is rich in heterogeneous environments and, in most cases, it is impossible to control where the software will run.

The adoption of solutions that comply with international standards is perhaps the best way to achieve this goal. The problem arises when there are no specific standards or these aren't widely adopted.

That is precisely the case of biometric authentication technology.

1.2 Context

This project arose from an invitation by the ALERT Life Sciences Computing, S.A. (ALERT®) to the student André Filipe Tavares, finalist of the Integrated Master in Informatics and Computation Engineering from the Faculty of Engineering of the University of Porto, under scope of the project/dissertation course.

The ALERT Group is dedicated to the development, distribution and implementation of the healthcare software ALERT®, conceived to enable paper free medical environments. With headquarters in Vila Nova de Gaia, Portugal, the parent company began operations in December 1999 and the group has now a multidisciplinary team of over 600 employees. In 2008, more than 50% of turnover resulted from contracts signed outside the country [ALE08].

1.3 Project

Fingerprint authentication is a vital component of the ALERT® software suite. Together with a Personal Identification Number (PIN), is the way users authenticate themselves in the clinical environment.

The verification process starts with the acquisition of the user's fingerprint by means of specialized equipment, a fingerprint reader. Once captured, specialized algorithms identify the points of interest which then will be used to generate a template. This is a unique digital representation of the fingerprint and is, by law, the only information that can be stored and transferred through the network. Then, the template is sent to the server for comparison with the stored reference template for that same user. Complex algorithms compare both fingerprints but, since the returned value is never 100% accurate, a minimum acceptance value must be set. If the comparison is successful, the user is authenticated on the application.

Implemented in 2002, the first adopted software/hardware solution was Digital Persona's. This is a proprietary and closed solution and only compatible with Windows 32 bits. Thus, the feature extraction algorithms only work with Digital Persona's fingerprint readers and the generated templates are represented in a closed proprietary format. Therefore, this solution isn't compatible with other fingerprint readers nor can these templates be used by other algorithms than Digital Persona's. This is a typical case of vendor lock-in.

At the hardware level, the original choice for Digital Persona's fingerprint readers is maintained, mainly due to its high level of reliability and durability.

At the software level, it became necessary to run the application on Linux and 64-bit Windows servers. To solve the compatibility with Linux, another third-party solution that, albeit also proprietary, supported the readers used, was adopted. MegaMatcher was the choice. Relatively to 64-bit Windows systems the adopted solution is to run a 32-bit Windows server solely for fingerprint comparison.

The current client infrastructure is still limited to 32-bit Windows platforms and the Internet Explorer (ActiveX) web browser.

This way, there are currently two distinct and incompatible fingerprint authentication development environments that still do not cover a broad range of possible use case scenarios.

With the growing presence of ALERT® in several distinct international markets, it becomes increasingly difficult to control the environments in which the application will run and the equipment used. It is therefore essential to eliminate the dependence on this solution that limits the use of the application with other fingerprint readers and restricts it to the Windows platform (on the client side of the application).

It is the aim of this project to find a unique solution for this problem. A solution which, ideally, is compatible with any operating system/architecture and any web browser, without breaking compatibility with templates previously generated with Digital Persona and compatible with its fingerprint readers, currently used in all implementations of ALERT® around the world.

1.4 Goals

For this project were defined as goals the development of two documents:

- Market Solutions Analysis, including readers and supported environments, capital gains and non attainable requirements;
- Architecture Description Document about the solution to implement;

As a result of the implementation of the solution, it is also expected a software library with the following requirements:

- Support for a large number of fingerprint readers;
- Creation of a standard API responsible for the interoperability with the ALERT® software;
- Support for different operating systems: Windows (XP/Vista/others), Linux/Unix;
- Support for the Macintosh platform (optional);
- Compatibility with Single Sign On Kerberos mechanisms and SAML (optional).

1.5 Document Structure

This document is divided in six distinct chapters:

- Chapter 1: The general purpose of this project is explained, its context and objectives.
- Chapter 2: This chapter describes the state of the art. Based on the background and objectives described in Chapter 1, research was done to assess which solutions already exist on the market and which will have to be developed. The result of that research is exposed here.

- Chapter 3: Based on the conclusions taken from Chapter 2 and considering the objectives stated on Chapter 1, this chapter proposes an architectural solution for the problem.
- Chapter 4: This chapter describes the design and implementation of the solution devised on Chapter 3.
- Chapter 5: Tests of the implemented solution are described in this chapter.
- Chapter 6: This final chapter presents the overall conclusions that were taken in the course of this project and proposes future improvements.

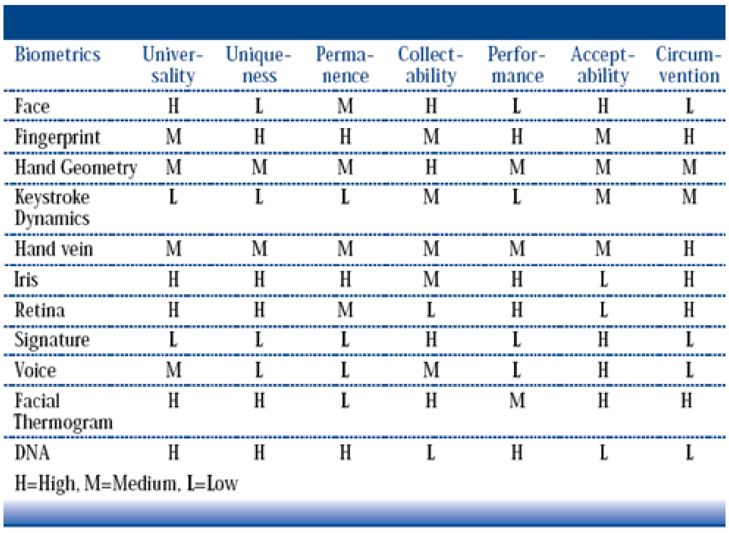
Chapter 2

State of the Art

2.1 Biometric Identification

All biometric identification systems are based in the same principle: each human being is unique and, as such, has unique characteristics. These can be divided in two main groups: physical and behavioral. The first are associated with specific parts of the human body and, the second, derive from the person's behavior. Typical examples of biometric systems based on physical characteristics are fingerprint and iris recognition. Signature and speech recognition are two widely used behavioral biometric data.

Unlike other types of identification, biometric data is less likely to be duplicated or lost. Its usage has direct impact in reducing fraud and enhancing user convenience. The figure 2.1 was extracted from "The 123 of Biometric Technology" [Yun03] and shows a comparison amongst various biometric technologies.



Biometrics	Universality	Uniqueness	Permanence	Collectability	Performance	Acceptability	Circumvention
Face	H	L	M	H	L	H	L
Fingerprint	M	H	H	M	H	M	H
Hand Geometry	M	M	M	H	M	M	M
Keystroke Dynamics	L	L	L	M	L	M	M
Hand vein	M	M	M	M	M	M	H
Iris	H	H	H	M	H	L	H
Retina	H	H	M	L	H	L	H
Signature	L	L	L	H	L	H	L
Voice	M	L	L	M	L	H	L
Facial Thermogram	H	H	L	H	M	H	H
DNA	H	H	H	L	H	L	L

H=High, M=Medium, L=Low

Figure 2.1: Comparison of Biometric Technologies

Thus, it is perceived why the preference in the adoption of such systems as a way of improving the forms of authentication.

The biometrical identification process is the same for every type of biometric data and, as seen in the figure 2.2, is summed up in three steps: Capture, Extraction and Matching [Rob05]. This way, first, specific hardware sensors capture a biometric sample. Then, the information is pre-processed and a specific algorithm extracts its features and generates a template, which is a synthesis of all the characteristics extracted from the source, in the optimal size to allow for adequate identifiability. Finally, it can either be stored as a reference template or compared with another by means of a matching algorithm.

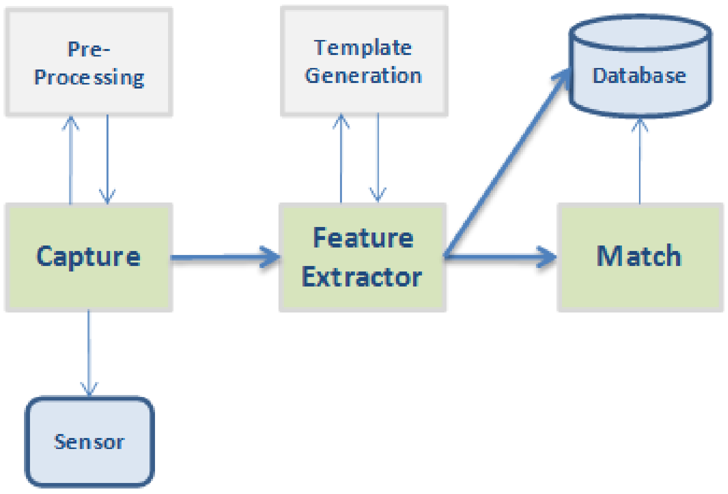


Figure 2.2: Biometrical Identification Process

2.2 Fingerprint

Fingerprint analysis is used in criminal identification since 1986 and is commonly accepted as being the oldest method of biometric identification [Yun03]. Traditionally, fingerprints were modeled in paper by means of the ink-and-roll procedure and comparisons done manually. In the 60s, this process was automated with the valuable help of computer systems. Even today, it is the preferred method used by police forces all over the world to identify criminals.

Among the different biometrics, fingerprints have the right balance of qualities including distinctiveness, persistence, accuracy, throughput, size and cost of readers, maturity of technology and convenience of use, making it the dominant biometric technology in commercial applications [Dig07].

To acquire a fingerprint image, a fingerprint scanner is needed. Although there are several variants, the most commonly used are the optical and the capacitance ones. Both return a digital image of the finger, despite acquiring it in distinct ways. Optical scanners have a Charge Coupled Device (CCD), the same light sensor used by digital cameras. Thus, the way they acquire the fingerprint image is analogous to how a digital camera takes a picture. Capacitive scanners use electrical current to generate the images. Its sensors are made up of one or more semiconductor chips containing an array of tiny cells, each containing two conductor plates that act as capacitors. The surface of the finger acts as a third capacitor plate, affecting the ability to store charge of each tiny capacitor and the overall result can be translated into a picture of the finger [How08]. As with every other technology, none of these approaches is perfect. Optical scanners might not be able to distinguish between the finger itself and an image; Capacitive scanners might be fooled by a mold of the finger. This way, it is common to combine biometric information with conventional identification methods, such as a password. Improvements to the fingerprint readers are also being made. There are readers that, in addition to acquiring the fingerprint image, also check for a pulse.

When matching two fingerprints, there are two main technical approaches: minutiae matching and pattern matching. Not only are the fingerprint templates from the second approach 2-3 times larger than in the first, but also these can be used to reconstruct the original fingerprint image [Yun03]. In general, the templates generated from the first approach are not reversible and an original image cannot be generated or reverse engineered. This is an important feature in the protection of privacy and the maintenance of security [Rob05]. Thus, the minutiae approach is followed by most fingerprint extraction and matching algorithms and is, inclusively, used in Automated Fingerprint Identification Systems (AFIS) used by forensic applications around the world. It is also accepted as valid evidence in a court of law. Minutiae are points of interest in a fingerprint. From the fifty two types listed, seven are usually used by human experts and only two by automated systems, namely bifurcations (a ridge splitting in two) and ridge endings [Jai04]. A typical fingerprint image may produce between 15 and 50 minutiae points, depending on the portion of the image captured [otSCPtSotFo05].

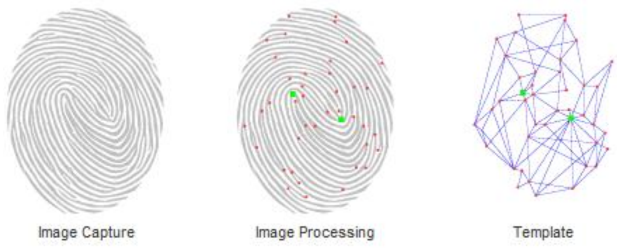


Figure 2.3: Source: Digital Persona

To verify the identity of a user by automatically extracting minutiae from his or her fingerprint image, a fingerprint recognition algorithm is required. The fingerprint recognition algorithm is composed of two main technologies: image processing technology that captures the characteristics of the corresponding fingerprint by having the image undergoing several stages, and matching algorithm technology that authenticates the identity by comparing feature data comprised of minutiae with reference templates stored in an Identity Management System (IMS).

A major difference between traditional authentication systems and biometric ones regards to how the credentials are validated. Given that a slight change in terms of finger shape or angle is enough to produce templates that do not match 100%, a valid person can be rejected and vice-versa. This way, ratios were developed to evaluate the probabilities of the two cases: False Acceptance Rate (FAR) and False Rejection Rate (FRR) [RFL09]. It is up to the application that makes use of biometrics to decide which values fit the purpose. For criminal investigations, for example, a lower value of FRR would be desirable.

2.3 Standards

Since the two main goals of this project are to achieve a platform independent infrastructure and enable future integration of other kind of biometric authentication, it is important to comply, if possible, with existing standards.

A standard Biometric API would allow seamless integration not only with other kind of biometric systems but also with different providers within a specific type, in this case, fingerprint.

Fingerprint template standards would ensure compatibility within different providers of fingerprinting technology without the increased overhead of having to deal with legacy issues when switching from one provider to another.

2.3.1 Biometrics API

BioAPI version 2.0 is specified in ISO/IEC 19784-1 and is a key part of the International Standards that support systems that perform biometric enrollment and verification (or identification). It defines an Application Programming Interface (API) that brings platform and device independence to application programmers and biometric service providers [Bio08].

It is the defacto standard and a result of a combined effort towards an industry standard biometric API. There are other generic biometric APIs that exist merged with BioAPI (or are based on it) in this effort, such as:

- BAPI
- HA-API

- IBM AIS API
- Intel HRS

BAPI was developed by I/O Software in 1998 to be operating system and hardware independent, while maintaining a consistent user interface. In December 1998, I/O Software joined the BioAPI Consortium and the BAPI specification was integrated as the lower level of BioAPI specification. In May 2002, Microsoft acquired BAPI technology with a view to integrating BAPI into Windows operating systems and applications. As of Windows 7, it is available in the form of Windows Biometric Framework (WBF).

HA-API was developed by NRI (National Registry Inc, later Saflink) in 1997 through an US Department of Defense contract and sponsored by the NSA and the Biometric Consortium. It was a simple high-level API focusing on the easy use and integration of multiple biometrics and placed in the public domain. It merged with BioAPI in March 1999.

IBM's AIS API was recently submitted to BioAPI and Intel HRS is heavily based on BioAPI.

Despite most of the biggest biometric service providers being members of the BioAPI Consortium, its adoption is scarce and very few products support it.

2.3.2 Fingerprint Template Standards

As stated above, the minutiae approach is widely adopted by most of the feature extraction and matching algorithms for fingerprints. However, since there are differences in how each algorithm implements this method, detecting and analyzing the points of interest, no two algorithms can be expected to yield the same template from a given fingerprint. Thus, a template generated by one algorithm might not match with one generated by another [otSCPtSotFo05].

To tackle this problem and facilitate the interoperability of different fingerprint identification systems, standards for the fingerprint minutiae templates were devised. Currently, there are two: INCITS 378 and ISO/IEC 19794-2. There is even a program of the American National Institute of Standards and Technology (NIST) to coordinate efforts in order to improve the performance and interoperability between implementations of these standards, the Minutiae Interoperability Exchange Test (MINEX) [Nat08b]. Thus, templates generated by algorithms that have passed the MINEX tests have a high probability of being compatible.

2.4 Vendors

In the search of already available market solutions for the Biometrics middleware and the Fingerprinting layer we only took in consideration those who met our requirements: BioAPI 2.0 compliance for the first and MINEX compliance for the latter.

In the comparison analysis we also considered Digital Persona because, despite not meeting the above requirements, it is the current fingerprint reader and software provider in ALERT® solutions and is necessary to support it for legacy issues.

2.4.1 Biometric Middleware

The three major BioAPI Middleware vendors are BioBex, BioFoundry and ImageWare.

All three support a variety of programming languages, such as Java, which indeed allows for platform independence. BioBex is also referenced as a perfect candidate for the biometrics layer of a Java Authentication Framework by Sun [Nag].

The drawback is the expensive and complex license models that all have. Even though it eliminates the dependence of a specific Operating System, for example, it adds the dependence on a license model, besides the added monetary costs.

2.4.2 Fingerprint Solutions

Although there are well defined standards to facilitate and promote interoperability between different providers of fingerprint identification technology, the reality is quite different. This is a very competitive market with distinct solutions for very specific scenarios and niches, where each company seeks to maintain its hegemony through closed and proprietary solutions (vendor lock-in). Many choose to implement proprietary formats and not to support neither other vendor's technology or international standards.

Since the goal is to achieve future interoperability between Extraction and Matching algorithms, the only considered providers were those in the MINEX Compliant List [Nat08a]. From the entire list, these are the ones with consumer available products:

- Aware
- BIO-Key
- Innovatrics
- L-1 Identity Solutions
- MegaMatcher
- Sagem Morpho
- SecuGen

- Sonda
- Suprema

Even though they might support template standards, a great deal of fingerprint providers still enforce lock-in by limiting the use of their algorithms with their devices or even their own software. That is the case of L-1 Identity Solutions, Sagem Morpho, SecuGen and Sonda.

Due to the lack of a Java API, convenient for platform independence, nor some kind of Linux support, Suprema was also eliminated

Aware was also eliminated, in this case due to the complex licensing scheme which required an usb dongle per client that, in a web scenario for example, wouldn't be viable. Although MegaMatcher also has an undesirable licensing scheme, it has to be considered since it is already being used and, as it happens with Digital Persona, has to be considered for legacy issues.

2.5 Fingerprint Vendor Analysis

For the fingerprint module we then consider four vendors:

- BIO-Key
- Digital Persona
- Innovatrics
- MegaMatcher

2.5.1 Fingerprint Reader Support

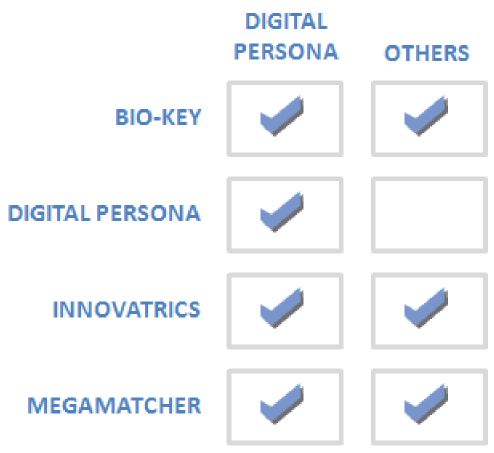
Since practically every fingerprint reader deployed to date by ALERT® is Digital Persona, it is imperative that our solution supports these. All four providers support Digital Persona's Fingerprint Readers out-of-the-box.

Only Digital Persona doesn't offer support for other readers, as we can see in figure 2.4.

There is, however, the possibility of extracting the fingerprint image from a Digital Persona reader using its SDK and then "feeding" it to the algorithms used for processing/matching.

2.5.2 Image Input

Image input capability is crucial for fingerprint reader independence. As we can see in figure 2.5, the only provider that doesn't allow this is Digital Persona.



	DIGITAL PERSONA	OTHERS
BIO-KEY	✓	✓
DIGITAL PERSONA	✓	
INNOVATRICS	✓	✓
MEGAMATCHER	✓	✓

Figure 2.4: Fingerprint Reader Support

Before inputting the image it would be interesting to access its quality, since it affects directly the overall performance: better quality images mean better performance (and lower False Acceptance Rate) and vice-versa. The NIST Fingerprint Image Quality (NFIQ) is practically the standard algorithm for doing just that, but any other kind of quality assessment is good.

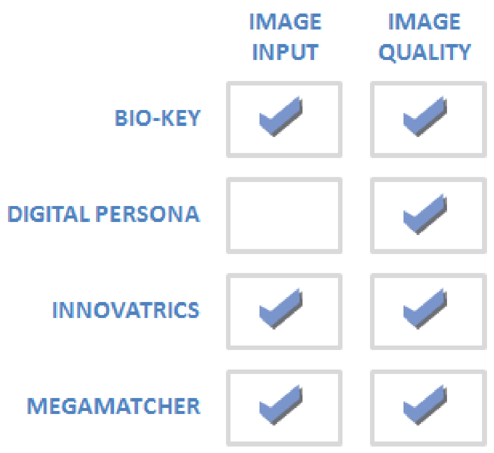


	IMAGE INPUT	IMAGE QUALITY
BIO-KEY	✓	✓
DIGITAL PERSONA		✓
INNOVATRICS	✓	✓
MEGAMATCHER	✓	✓

Figure 2.5: Image Input and Quality Assessment Capabilities

Even if we could somehow provide an externally acquired image to the Digital Persona SDK for feature extraction / template matching, the EULA doesn't allow the SDK to be used with other hardware besides Digital Persona's.

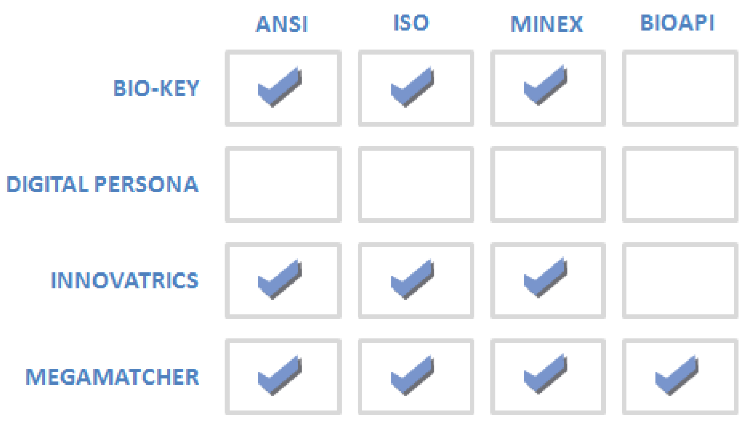
2.5.3 Supported Standards

In order to assure future compatibility and interoperability of Fingerprint Templates it is highly desirable that these comply with certain standards. Also, using standard templates enables effortless future developments related with extracting/matching algorithms as it removes the need to support legacy templates/algorithms.

The ANSI378 and ISO19794-2 are Fingerprint Minutiae Standards and MINEX is a program of the National Institute of Standards and Technology (NIST) which coordinates efforts aimed at improving the performance and interoperability of core implementations of the INCITS 378 and ISO/IEC 19794-2 fingerprint minutiae standards.

The Biometric Application Programming Interface (BioAPI) is part of the International Standards and defines interfaces between modules that enable software from multiple vendors to be integrated together. Ideally, it would be interesting to support BioAPI. In practice, its adoption is scarce.

In figure 2.6 we can which standards are supported by each vendor.



	ANSI	ISO	MINEX	BIOAPI
BIO-KEY	✓	✓	✓	
DIGITAL PERSONA				
INNOVATRICS	✓	✓	✓	
MEGAMATCHER	✓	✓	✓	✓

Figure 2.6: Supported Standards

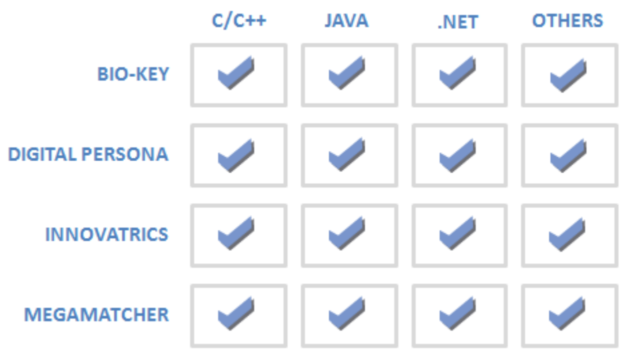
Again, Digital Persona doesn't meet the requirement being that it doesn't support any fingerprint minutiae standard.

2.5.4 API Programming Languages

Since we are aiming at platform independence, the best way to achieve it is using Java. However, it doesn't mean that the algorithms themselves work in the same platforms the Java SDK does.

The programming languages supported by each vendor are illustrated in figure 2.7.

BIO-Key and Innovatrics algorithms are coded with cross platform in mind, which means that they are willing to provide the API in virtually any language desired.



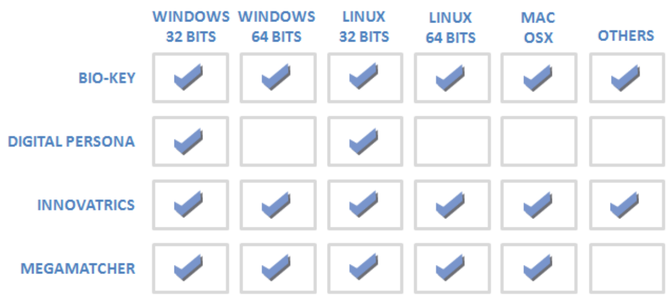
	C/C++	JAVA	.NET	OTHERS
BIO-KEY	✓	✓	✓	✓
DIGITAL PERSONA	✓	✓	✓	✓
INNOVATRICS	✓	✓	✓	✓
MEGAMATCHER	✓	✓	✓	✓

Figure 2.7: Supported API Programming Languages

2.5.5 Supported OSes

The ultimate goal is to achieve platform independence, but at least Windows (32/64bits) and Linux (32/64bits) must be supported. Possibly, in the worst case, Linux versions can be used in other UNIX-certified OSes using binary compatibility tools, therefore enabling its use in Mac OSX, for example, which is also a requirement.

The supported operating systems by each vendor are illustrated in figure 2.8.



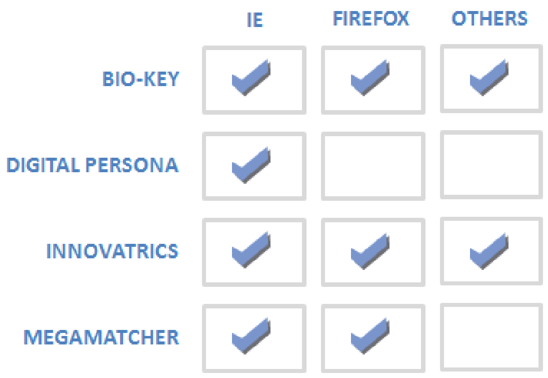
	WINDOWS 32 BITS	WINDOWS 64 BITS	LINUX 32 BITS	LINUX 64 BITS	MAC OSX	OTHERS
BIO-KEY	✓	✓	✓	✓	✓	✓
DIGITAL PERSONA	✓		✓			
INNOVATRICS	✓	✓	✓	✓	✓	✓
MEGAMATCHER	✓	✓	✓	✓	✓	

Figure 2.8: Supported Operating Systems

The best solutions regarding platform independence of the algorithms are BIO-Key and Innovatrics, whose algorithms are coded with cross platform in mind and are willing to provide us with binaries for practically every platform we desire.

2.5.6 Supported Browsers

Again, with multiplatform in mind, Microsoft Internet Explorer and Mozilla Firefox support is required and only Digital Persona doesn't offer it for both, as we can see in figure 2.9.



	IE	FIREFOX	OTHERS
BIO-KEY	✓	✓	✓
DIGITAL PERSONA	✓		
INNOVATRICS	✓	✓	✓
MEGAMATCHER	✓	✓	

Figure 2.9: Supported Web Browsers

Again, being true cross platform, the BIO-Key and Innovatrics solutions will most likely run in every scenario, being that BIO-Key has indeed guaranteed it to work not only in IE and Firefox, but also Netscape and Safari.

2.5.7 Pricing and License Models

The desired licensing scheme is one that doesn't require licensing servers, usb dongles and such. An annual, royalty free, fee is the preferred choice.

BIO-Key and Innovatrics annual fee is around $100.000 USD, but is subject to negotiation. BIO-Key licensing model doesn't involve any kind of licenses, it's based on trust, and they already predict an annual, royalty free fee. Innovatrics didn't give a final answer on this matter. In the event of ending the agreement, with Innovatrics, it only affects future deployments. No answer on this from BIO-Key.

Digital Persona's SDK is free, but can only be used with Digital Persona Fingerprint Readers.

MegaMatcher has an annual fee of $80.000 and a complex licensing software that has revealed to be unstable in ALERT®'s USA Datacenter.

2.5.8 Feedback

The best feedback came from Innovatrics and BIO-Key, who early in our talks offered themselves to call us on the phone. Both promptly answered our questions and were willing to partnership with us, being open to negotiating prices and licensing schemes and making their solutions meet our needs (like compiling it for specific OSes).

BIO-Key has a great relationship with several Fingerprint Reader Manufacturers which may reveal itself fruitful in achieving reader independence since it may enable us a privileged communication channel with them. This might mean easy access to drivers and

APIs to allow easy integration of readers with our biometrics framework and compatibility with specific platforms (Mac OS?).

Digital Persona's core business is selling their readers and so isn't very open to meet our needs. Being that one main goal of this new infrastructure is reader independence it doesn't make sense for Digital Persona to help us in order to do so.

Communicating with MegaMatcher has revealed itself to be very difficult as they tend to take very long to answer e-mails.

2.6 Conclusions

The main purpose of this project is to build a platform independent fingerprint authentication framework that doesn't break backwards compatibility and is scalable. When talking about scalability, we cannot discard the fact that it makes sense to include it in a biometrics framework.

All the considered biometric middleware solutions are expensive and add undesired complexity to the authentication framework where it will be integrated. On top of that, the BioAPI standard adoption is minimal and there is the issue of supporting legacy fingerprinting technology. Therefore, it makes more sense to build a custom biometrics middleware that tightly integrates with the existing authentication framework and with a publicly available API in order to allow for vendors to develop ALERT-compatible biometrics technology.

Developing this custom biometrics middleware will allow keeping legacy systems running and pave the way to easily and seamlessly integrate new fingerprinting technology and even other kinds of biometrics with the product.

Adopting a MINEX compliant fingerprint technology is the way to eliminate future dependence of specific vendor's algorithms, but given the high costs of adopting this solution and the need to, first and foremost, support the current technology (Digital Persona) and enable it to satisfy current needs, for now only this will be integrated in the biometric framework.

Chapter 3

Architecture Description

3.1 Architecturally Relevant Requirements

3.1.1 Key Use Cases

In the end-user point of view, this framework will serve only two purposes: to register its biometrical data for future reference and to authenticate itself using previously stored biometric data.

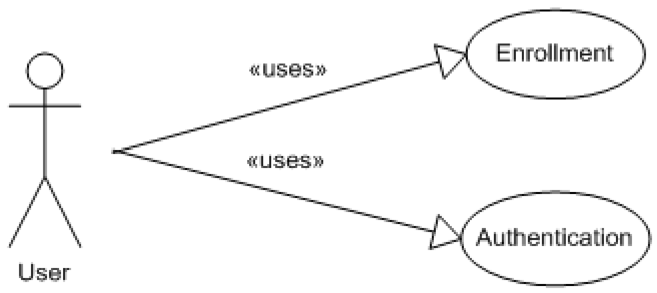


Figure 3.1: Use Cases

Therefore, as we can see in figure 3.1, these are the two use cases.

3.1.2 Non-Functional Requirements and Constraints

The purpose of this framework is to be platform independent, scalable and backwards compatible. Therefore and foremost, it must support Digital Persona's fingerprint readers and templates.

A biometrics API must be defined to easily allow future integration of other forms of biometric authentication and must enable not only the integration of different biometric technologies but also multiple vendors from one such.

Regarding Operating Systems, it must run in Windows and Linux (both 32 and 64 bits versions of them) on the server side and Windows and Linux (32 bits only) in the client side.

On the browser version, it must be able to run on Microsoft Internet Explorer, Mozilla Firefox and Google Chrome.

Regarding the user databases, the framework must interface with Microsoft Active Directory and OpenLDAP.

Fingerprint images must never be stored nor transferred through the network.

3.2 Context View

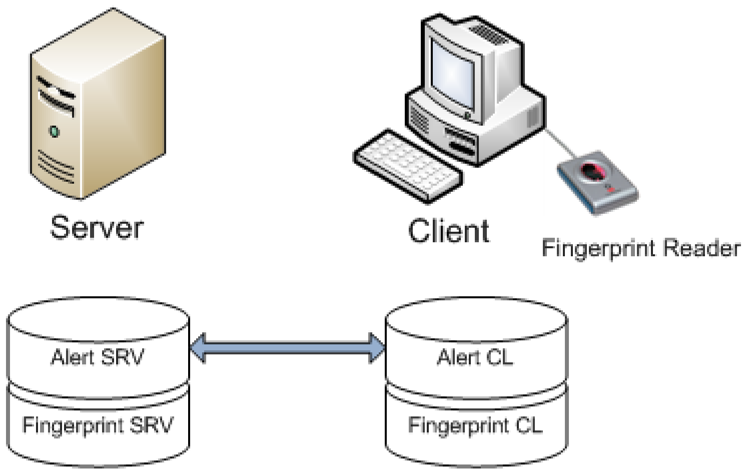


Figure 3.2: Context View

The client must be able to acquire a fingerprint image from the fingerprint reader and extract its features in order to generate a fingerprint template. It must be able to communicate with the authentication server for enrolment and verification.

The server's role is to interface with the fingerprint database in order to store and retrieve reference templates in order to enrol and verify users. To do so, it must be able to communicate with the clients. The matching of two fingerprints is done by the server and the result returned to the client.

The Client/Server communication must be done by TCP/IP and support encryption protocols, such as MD5, SHA1, HTTPS and others, since it is desirable that the template never travels the network unencrypted for security reasons. Data is exchanged between the ALERT client application and the ALERT Java Gateway Server.

An illustration of this client/server paradigm is found on figure 3.2.

3.3 Logical View

3.3.1 Logical Model

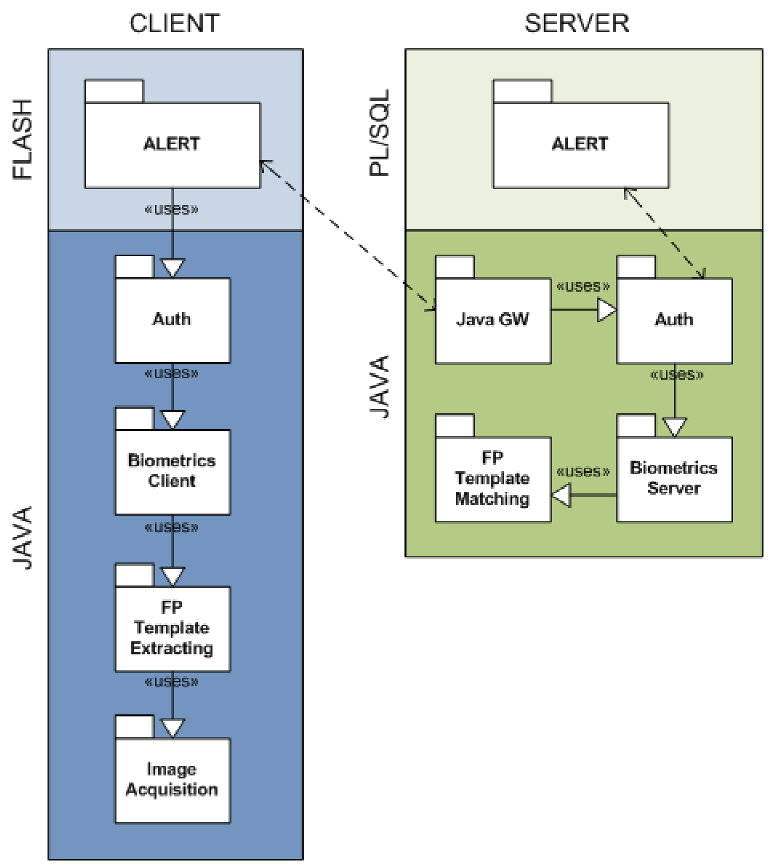


Figure 3.3: Logical Model

3.3.1.1 Client

The fingerprinting logic is included in the biometrics module that is itself included in the authentication module. Therefore, when the ALERT client wishes to perform user enrolment or verification, it relies on the authentication module to do so.

The authentication module then calls the biometrics client that uses the image acquisition module to acquire the fingerprint and the extraction module to extract the fingerprint features and generate a template.

Both the Image Acquisition and the Extraction modules implement an API in order to support various sensor and algorithm drivers, as seen on figure 3.4.

This way we can support legacy systems and pave the way for seamless future updates and developments without breaking the application.

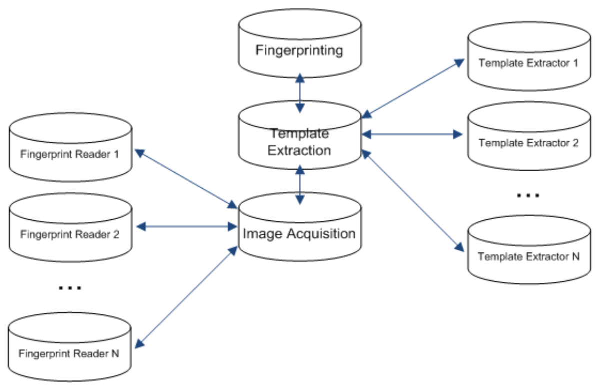


Figure 3.4: Acquisition and Extraction Modules

3.3.1.2 Client / Server Communication

The client communicates with the server through the ALERT Client and the Java Gateway, respectively.

The acquired fingerprint sample is sent to the server to be stored, in the case of enrolment, or matched with the reference template in order to perform user verification.

3.3.1.3 Server

On the server, the Java Gateway is responsible for handling the communication with the client and forwarding fingerprint matching requests to the authentication server module.

The authentication module accesses the user database to retrieve the reference template and passes it on to the biometrics server module for matching.

Like on the client, an API is defined for the Matching algorithm in order to support several algorithm drivers, as we can see in figure 3.5.

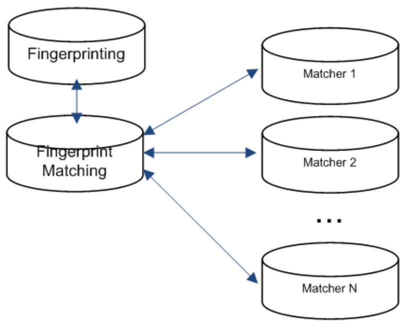


Figure 3.5: Matching Module

This allows for multiple vendor support and, therefore, multiple template formats such as legacy ones.

3.4 Dependency Model

The fingerprint authentication framework has external dependencies that have direct impact in its workings. They are:

- Fingerprint Reader Device Drivers and SDK
- Template Extraction and Matching Algorithms
- Authentication Modules
- Java Gateway
- Fingerprint Database

Relatively to the fingerprint reader, first it has to have drivers for the operating system where it is wanted to run and then an SDK that allows it to be integrated with the framework by means of writing the necessary drivers to interface with its fingerprint acquisition API.

It is unviable to write extraction and matching algorithms and, therefore, it is always necessary to have at least one algorithms provider compatible with all deployed devices for this framework to work. Like with the fingerprint reader, there is always the dependency in the operating systems that these support which, in its turn, limits the operating systems where the entire framework will be able to run in.

The entire framework's client / server communications are handled by the authentication modules and the Java Gateway. The biometrics framework must be integrated in the authentication module and the latter must be extended to support some specific needs of the first.

3.5 Behavioural View

3.5.1 Scenario & Collaboration Models

When enrolling a user, the ALERT® application makes a call to the Authentication Module. It then relies on the Biometrics Client to handle the capturing of the biometric data and generation of the reference template. After generating the template, it is returned by the Biometrics Client to the Authentication Module that then sends it to the server via the Java Gateway. Finally, after receiving the request for user enrollment and the respective reference template, the Java Gateway sends it to the Authentication Module to handle its storage.

This scenario is illustrated in figure 3.6, showing all the modules, layers and calls involved.

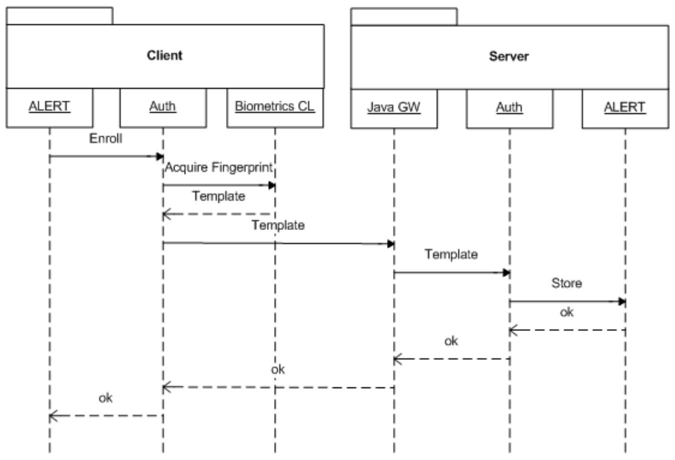


Figure 3.6: Fingerprint Enrollment Process

When authenticating a user, the ALERT® application makes a call to the Authentication Module, this time with the purpose of verifying the user's biometric data. Again, the Authentication Module relies on the Biometrics Module to handle the biometric data acquisition. It is then sent to the server, via de Java Gateway, to be matched against a stored reference template for that same user. The Java Gateway forwards the request to the Authentication Module that tries to match the received biometric data with the one it has previously fetched from the database. The result of the matching is returned to the Java Gateway that then relays it to the Authentication Module on the client. According to the result, the Authentication Module then authenticates, or not, the user in the ALERT® application.

This scenario is illustrated in the sequence diagram on figure 3.7

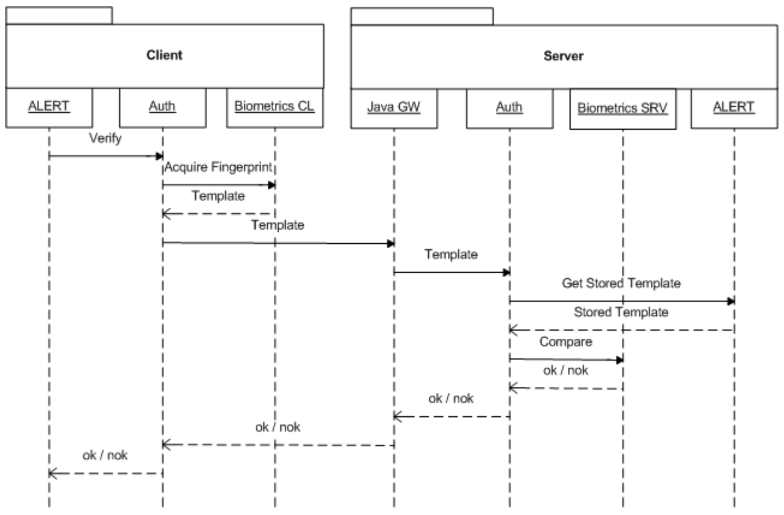


Figure 3.7: Fingerprint Verification Process

3.5.2 State Models

When enrolling a user's fingerprint, usually four captured samples are needed. This way, it is expected that the user scans the finger being enrolled four times. After doing this, the samples quality is assessed and, if satisfactory, a template with the purpose of enrollment is generated using them. The states of this procedure are illustrated in figure 3.8.

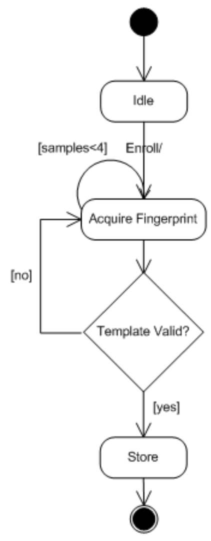


Figure 3.8: Fingerprint Enrollment States

When verifying a user's fingerprint, first a sample is captured. It is then matched against the stored reference template. If it doesn't match, the user can scan the finger again and repeat the process. The states needed in this procedure can be seen in figure 3.9.

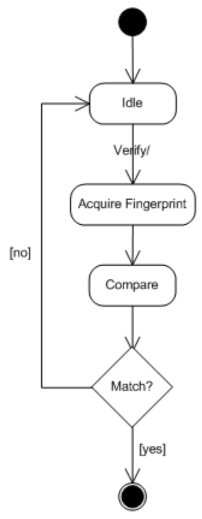


Figure 3.9: Fingerprint Verification States

3.6 Deployment View

In figure 3.10, it is shown how the biometric "functional entities" described in the logical view are deployed onto implementation entities. This way, in the client side we find the biometric data acquisition and feature extraction part of the framework and, in the server side, all the matching logic.

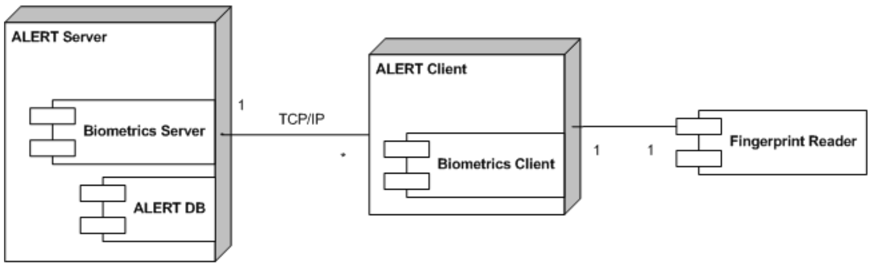


Figure 3.10: Deployment Diagram of the ALERT Biometric Framework

A server can support several clients, communicating to each one by a TCP/IP channel.

3.7 Architectural Design Principles & Patterns

This project arose from the inability of the previous solution to evolve, satisfying the current needs of the application that depends on it. Thus, the aim is to not only provide it with the features needed in the immediate future, but rather make it scalable. Accordingly, the demand is to develop a Framework in the true sense of the word, an abstraction supported by a well defined API that allows easy inclusion and modification of modules without breaking compatibility with the application that depends on it.

To support this abstraction, two software patterns should be used: interfaces and factories. This way, the vital components of the Biometrics Framework must be based on interfaces that define the methods that the modules have to implement. Thus, this will be the API the application will use, allowing the whole logic behind it to be changed without impact. Similarly, it should be transparent to the application the instantiation of the module to be used. Thus, for each interface, a factory must be defined and implement a static method that the application will use. It will then dynamically and transparently instantiate the specific module. As is the case with the interfaces, this allows changes to the internal logic of factories without affecting the main application.

Besides the abstraction itself, it is important to produce code that is understandable, simple and flexible. Only then can the application be truly scalable. Making changes is generally easy when you know what needs changing and if learning what the code does is time consuming per se, when the code is cluttered it not only takes even more time, but accidentally breaking it is easy [Bec08].

There isn't a perfect style of development, but there are common principles that should be always taken in consideration like structuring code so that it can be easily changed without affecting the sum. In order to do so, code duplication should be avoided (Do Not Repeat Yourself – DRY) [HT99]. Logic and data should be packaged together, since changes in one most of the times imply changes in the other and keeping them together increases chances that the consequences remain local.

To easily and rapidly assess the impact of changes in the code, automated tests should be used. Not only do they eliminate human error when testing, but speed up development and debugging and facilitate platform compatibility tests.

Chapter 4

Software Design and Implementation

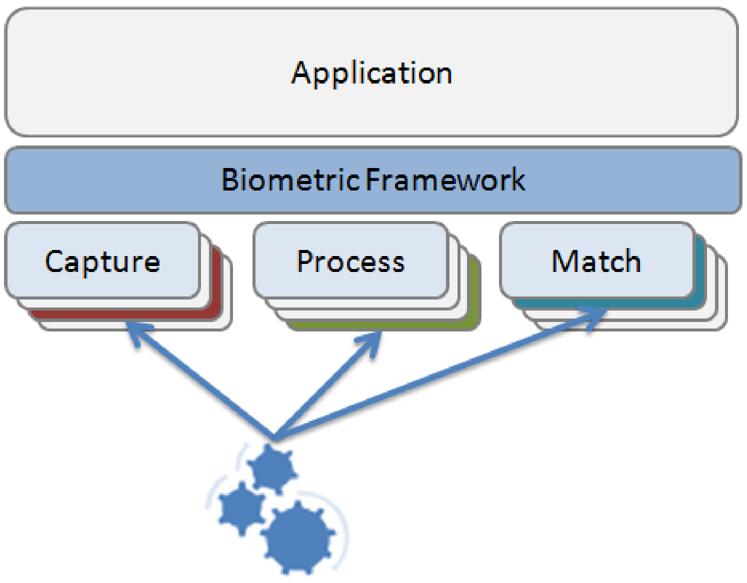


Figure 4.1: Biometric Framework

4.1 Technologies

The programming language used in implementing the core of the Biometric Framework is Java. Not only it is the language in which the Authentication Framework is written, which allows seamless integration with it, but being portability the main characteristic of Java, it suits the premise of a platform agnostic framework.

Although the extraction and matching algorithms provided by Digital Persona are written in C/C++, there is a SDK included with a Java API, so, there isn't the need to implement a Java Native Interface wrapper and the integration with the biometric framework is trivial. There is, though, an issue with running them in a 64-bit Java VM. The libraries are compiled in 32-bit and linking 64-bit executables with them cannot be done without some sort of Inter Process Communication (IPC) scheme. This was done resorting to the Java Remote Method Invocation (RMI), the Java equivalent of Remote Procedure Calls (RPC).

For the integration of the biometric framework with directory servers, the necessary modules were written for the authentication server using the Java Naming and Directory Interface (JNDI) and schema extensions devised in order to allow biometric information to be added to the user model of the two most popular directory servers: Active Directory and OpenLDAP.

During the development of the framework, several tools were used. The main ones were:

- Subversion (SVN) – Version Control System
- Eclipse – Java Integrated Development Environment
- Apache Directory Studio – LDAP Browser and Directory Client

4.2 Biometric Framework

As it was said before, fingerprint authentication follows the same procedure as any other kind of biometric based authentication technology: capture, extraction and matching. Therefore, these will be the core of the biometric framework and the API that the authentication server will use. Then, modules for the biometric technology / vendor will have to implement this API in order to interface with the framework. This way, it can support virtually every kind of biometric technology and, in any given technology, an unlimited number of providers. By configuration, we select which technologies will be used, specifying the sensors and the algorithms for processing and matching, as illustrated in figure 4.1.

4.2.1 Class Diagram

4.2.2 Biometric Capture

The Biometric Capture class is the interface that must be implemented by each biometric sensor. It defines the method "capture" which must return a BiometricData instance with the captured sample.

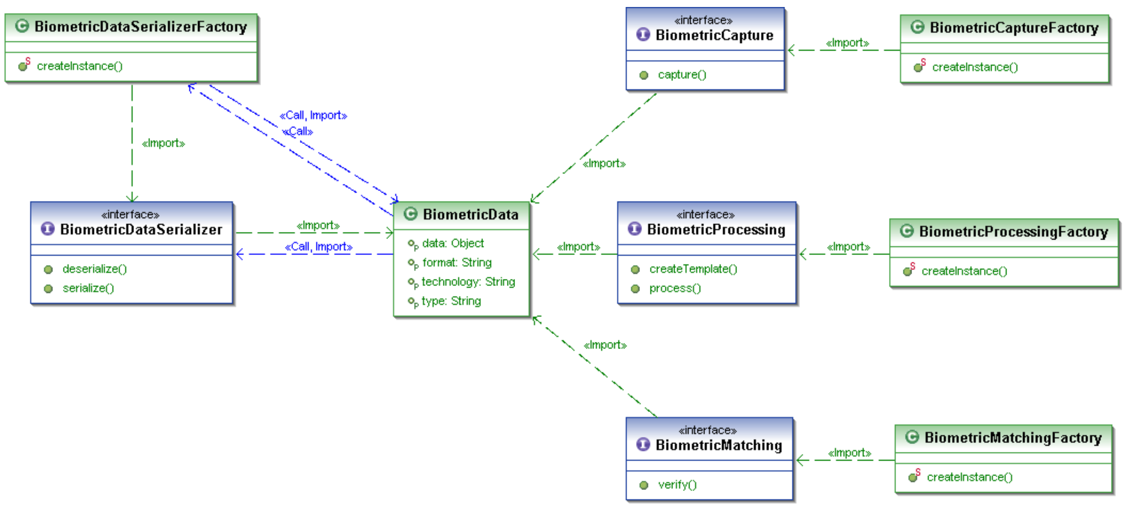


Figure 4.2: Biometric Framework Core Classes

Biometric sensors usually perform quality assessment of the captured sample. It is up to each module implementing the capture interface to handle these specifics and throw an exception if it is the case.

4.2.3 Biometric Capture Factory

The Biometric Capture Factory class is responsible for instantiating the biometric sensor module. According to the values passed by configuration, the static method "createInstance" returns the respective instance of the sensor's module that implements the Biometric Capture interface.

4.2.4 Biometric Processing

The Biometric Processing class is the interface that must be implemented by each biometric extraction algorithm and is responsible for the feature extraction of biometric samples. Feature extraction can have two distinct purposes: enrollment and verification.

For the first scenario, the "createTemplate" method is defined and accepts an array of captured BiometricData, since it takes several samples to generate a reference template and this number varies for each specific technology / vendor, and returns a template for the purpose of enrollment. In the second scenario, the method "process" is defined. It accepts a single captured BiometricData and returns a template for matching with a reference template.

Since the BiometricData that the methods receive by parameter is a generic container that may contain data that the specific module can't handle, it is each module's responsibility to verify it and throw an exception if it is the case. In the case of "createTemplate", which accepts an array of captured samples, it is also each module's responsibility to check if the number of samples is enough to generate a template.

Some algorithms provide tools to assess the quality of the generated template. This is important since it affects the overall performance of the matching algorithms. Given that this varies from provider to provider, it's up to each module to handle the specifics of it.

4.2.5 Biometric Processing Factory

The Biometric Processing Factory class handles the instantiation of the biometric extraction algorithm passed by configuration. It has a static method "createInstance" that returns the respective instance of the algorithm's module that implements the Biometric Processing interface.

4.2.6 Biometric Matching

The Biometric Matching class is the interface that must be implemented by each biometric matching algorithm. It defines the method "match" that accepts two biometric templates for comparison and returns true or false whether the matching was successful or not.

The first biometric data passed by parameter must be a processed template with the purpose of comparison and, the second, one with the purpose of enrollment. Not only must the modules check for this consistency, but also other kind of unexpected data that might be passed as a parameter, since the BiometricData is a generic container that can hold several kinds of data.

When comparing two fingerprints, usually there is a False Acceptance Rate (FAR) and a False Rejection Rate (FRR) value associated which defines when it is accepted as a match. Some algorithms allow it to be configured, hence it is up to each module to handle these specifics.

4.2.7 Biometric Matching Factory

This Biometric Matching Factory class handles the instantiation of the biometric matching algorithm passed by configuration. It has a static method "createInstance" that returns the respective instance of the algorithm's module that implements the Biometric Matching interface.

4.2.8 Biometric Data

The Biometric Data class defines the single, generic container for all biometric data and is used by the three core classes of the framework. This way, it can hold not only biometric data relative to the different steps of the biometric authentication process, but also from different technologies and technology vendors. As it can be seen in figure 4.3, besides the data object itself (data), it has additional information describing it (type, technology and format).

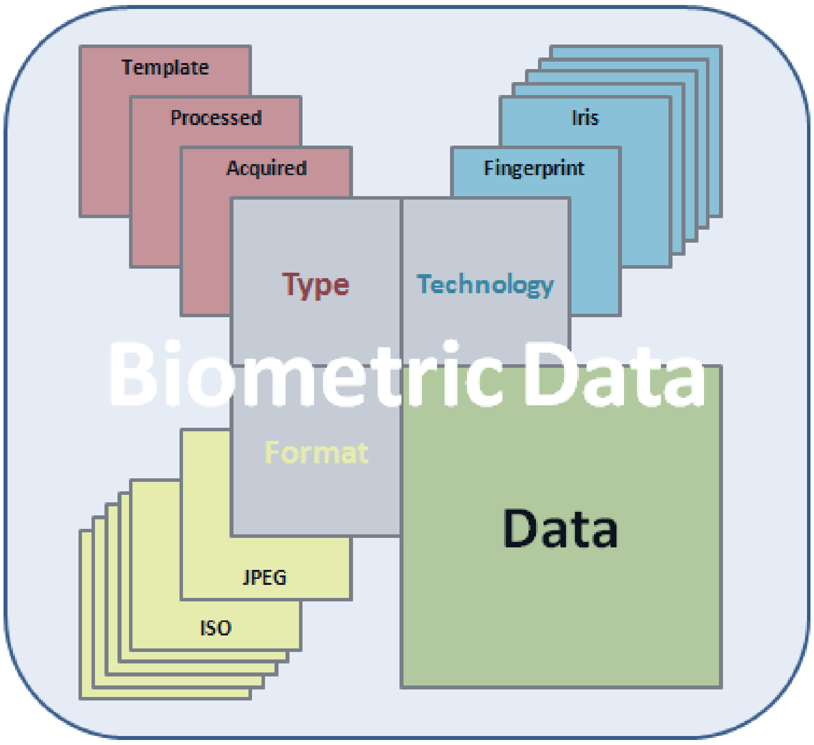


Figure 4.3: Biometric Data Structure

- ***Type***: States the purpose of the biometric data and can assume one of three values, according to it: acquired, processed or template. The first is relative to a captured biometric sample and the other two are templates resulting of the feature extraction of an acquired sample. The first template (processed) is used for verification and the latter (template) is used for enrollment.
- ***Technology***: This field refers to the biometric technology from which the data resulted. Therefore, it might be fingerprint, iris, voice or any other kind of supported biometric technology. This value must be one from a list of available technologies.
- ***Format***: Information about the format in which the data is represented is hold here. It can also be a variety of data types, ranging from plain jpeg images to proprietary biometric formats. This value must be one from a list of supported data formats.
- ***Data***: This field holds the biometric data object itself.

In order for this data to be interoperable with other pieces of software, transferable through the network and stored in any given type of storage service, it must be possible to parse it to several different formats. The adopted solution was to make the biometric data class serializable since there are several available libraries that allow seamless parsing of a serializable object to a variety of different formats, such as the popular Extensible Markup Language (XML). Since only the templates can be stored and communicated through the network, only biometric data holding these will be serializable. For each biometric technology vendor, a module that implements the biometric data serialization interface must be written for it.

A biometric data serializer defines the methods that the biometric data structure requires when serializing objects. The issue with making the biometric data serializable is with the biometric template data itself since all the other attributes are Java Strings and, therefore, serializable by nature. The template data is a generic Java Object since it can hold a variety of vendor specific objects and, therefore, there isn't a guarantee that all are serializable. This way, it isn't enough for the biometric data class to implement the Java Serializable Interface. It must override the writeObject and readObject methods and relies on the biometric data serializer to do so.

4.2.9 Biometric Data Serializer

The Biometric Data Serializer interface must be implemented for each biometric technology vendor since it defines the methods that the BiometricData requires when serializing objects. This way, it defines the methods "serialize" and "deserialize". The first accepts a BiometricData as parameter and returns a byte array corresponding to the serialized "data" field of the BiometricData and, the second, accepts a BiometricData with an empty "data" field and a byte array corresponding to the serialized biometric data and returns the same BiometricData but with the "data" field filled with the deserialized object.

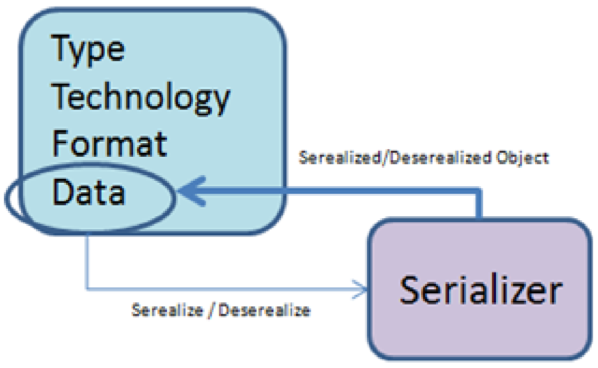


Figure 4.4: Biometric Data Serializer

4.2.10 Biometric Data Serializer Factory

The Biometric Data Serializer Factory class handles the instantiation of the biometric data serializer. It defines a static method "createInstance" that accepts a BiometricData as a parameter and returns an instance of the module that implements the BiometricDataSerializer interface for it.

4.2.11 Conventions

The modules for the Capture, Processing, Matching and Biometric Data Serialization are organized in packages by technology (fingerprint, iris, etc) and then, in each technology package, one folder for each of the above holds the modules for each vendor, as seen on figure 4.5.

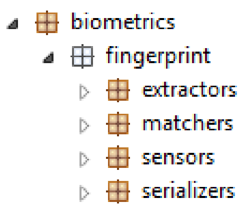


Figure 4.5: Package Naming Conventions

The factories that dynamically instantiate the modules depend on this structure and the naming convention of the classes that follows:

- `<BiometricTechnology>Sensor<TechnologyVendor>.java`
- `<BiometricTechnology>Extractor<TechnologyVendor>.java`
- `<BiometricTechnology>Matcher<TechnologyVendor>.java`
- `<BiometricTechnology>Serializer<TechnologyVendor>.java`

4.2.12 Configurations

Since the framework supports several biometric technologies and technology vendors, it is necessary to configure it to load the specific modules for the sensor, extraction algorithm and matching algorithm. The Capture, Processing and Matching factories then will use this configuration to instantiate them.

This way, the configuration must include the following parameters:

- Biometric Technology
- Biometric Sensor

- Biometric Extraction Algorithm
- Biometric Matching Algorithm

The values that can be set are defined in a file with all available technologies, sensors, extraction and matching algorithms since they must match the naming convention of the packages and classes of the respective modules in order for these to be instantiated dynamically at runtime by the factories.

It would be desirable to set values for the False Acceptance Rate (FAR) but some vendors might not have this option, which is the case of Digital Persona. This way, it is possible to add the desired value of the FAR in the configuration, but it's up to each Biometric Matching module to implement it according to its capabilities.

As with the FAR values, there might be a number of optional module specific parameters that can be configured. This way, it is important to document these so that, when configuring the framework, these are correctly set.

These configurations are made using the Authentication Server's Configurator. It is an abstraction intended to provide a single, common configuration scheme for all the components of it, as illustrated on figure 4.6.

This configuration scheme is independent of storage medium, therefore allowing for the configurations to be loaded from different places. In this stage, only the ConfigMemory was implemented, which means that the configurations had to be hard coded. An example of a configuration in order to use Digital Persona's fingerprint technology is illustrated in figure 4.7.

As it was said, besides the required parameters (as seen above), there can be countless specific configurations for each module. An example of passing the FAR value is illustrated in figure 4.8.

4.3 Fingerprint Module

Since on of the main goals of this project is to support fingerprint authentication using Digital Persona's technology, the necessary modules were written in order for it to be supported by the biometrics framework. Since Digital Persona provides a Java SDK, writing its modules was pretty straightforward.

4.3.1 Biometric Framework Integration

Following the conventions defined for the modules, first a "fingerprint" package with four folders (sensors, extractors, matchers and serializers) was created. Then the necessary modules for the framework to be able to use Digital Persona's fingerprint reader and extraction and matching algorithms were written and added using the naming convention

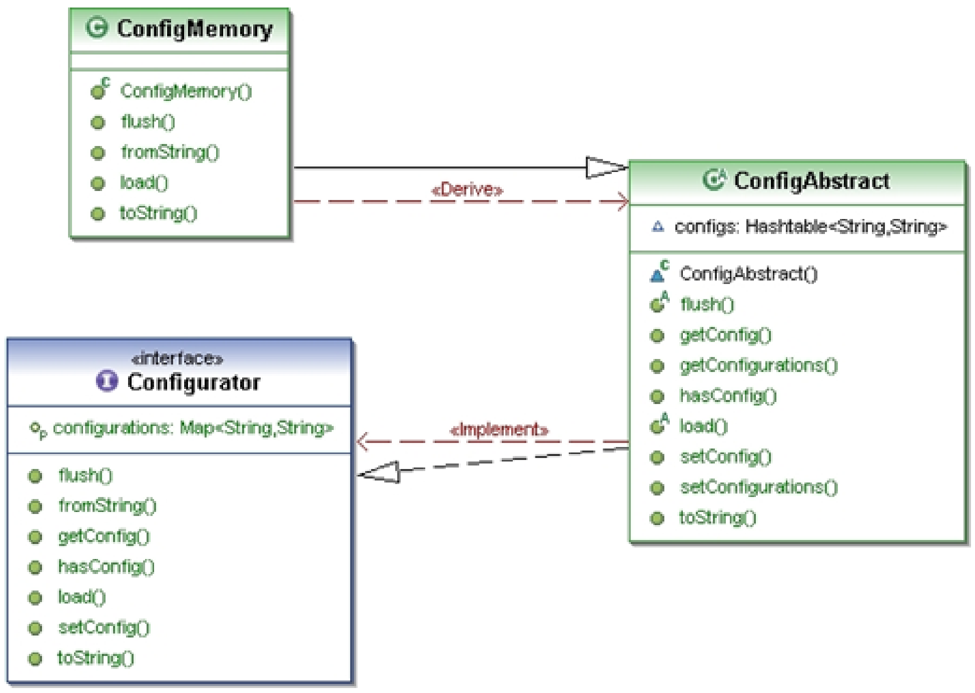


Figure 4.6: Authentication Server Configurator Classes

```
Configurator config = new ConfigMemory();
config.setConfig("SECURITY_AUTH_BIOMETRIC_Technology", "fingerprint");
config.setConfig("SECURITY_AUTH_BIOMETRIC_Sensor", "DigitalPersona");
config.setConfig("SECURITY_AUTH_BIOMETRIC_Extractor", "DigitalPersona");
config.setConfig("SECURITY_AUTH_BIOMETRIC_Matcher", "DigitalPersona");
```

Figure 4.7: Configuration Example for Digital Persona

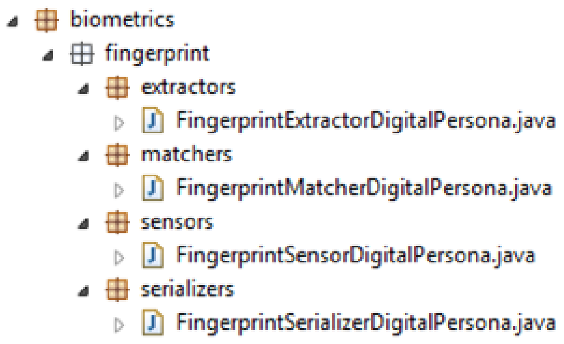
```
config.setConfig("SECURITY_AUTH_BIOMETRIC_FAR_Value", "HIGH_SECURITY_FAR");
```

Figure 4.8: Configuration Example for FAR value

`Fingerprint<Sensor/Extractor/Matcher>DigitalPersona.`

Finally, the serializer was added (FingerprintSerializerDigitalPersona).

In figure 4.9 it is shown the modules in the respective packages.



```
biometrics
  fingerprint
    extractors
      FingerprintExtractorDigitalPersona.java
    matchers
      FingerprintMatcherDigitalPersona.java
    sensors
      FingerprintSensorDigitalPersona.java
    serializers
      FingerprintSerializerDigitalPersona.java
```

Figure 4.9: Digital Persona Modules

The integration was seamless and Digital Persona's fingerprint technology was now available to the biometric framework.

4.3.2 Fingerprint Extractor Digital Persona

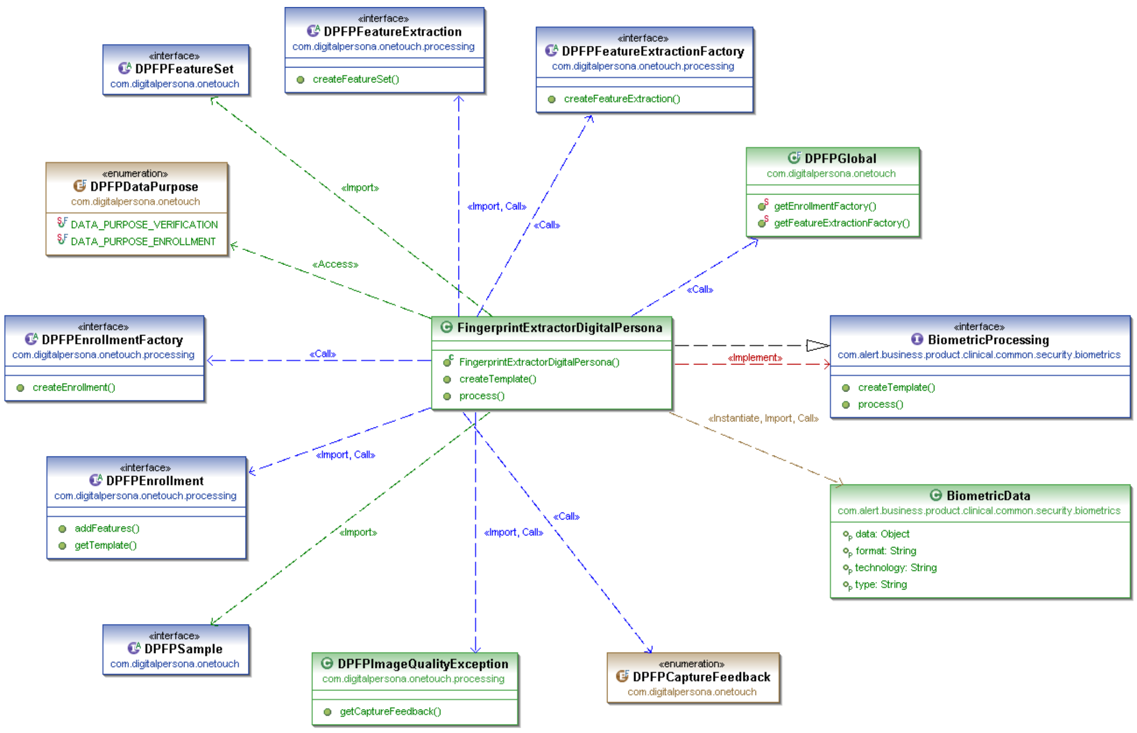

«interface» DPFPFeatureSet com.digitalpersona.onetouch
«interface» DPFPFeatureExtraction com.digitalpersona.onetouch.processing createFeatureSet()
«interface» DPFPFeatureExtractionFactory com.digitalpersona.onetouch.processing createFeatureExtraction()
«enumeration» DPFPDataPurpose com.digitalpersona.onetouch DATA_PURPOSE_VERIFICATION DATA_PURPOSE_ENROLLMENT
DPFPGlobal com.digitalpersona.onetouch getEnrollmentFactory() getFeatureExtractionFactory()
«Import»
«Import, Call»
«Call»
«Access»
«Call»
«interface» DPFPEnrollmentFactory com.digitalpersona.onetouch.processing createEnrollment()
«Call»
FingerprintExtractorDigitalPersona FingerprintExtractorDigitalPersona() createTemplate() process()
«Implement»
«interface» BiometricProcessing com.alert.business.product.clinical.common.security.biometrics createTemplate() process()
«Instantiate, Import, Call»
«interface» DPFPEnrollment com.digitalpersona.onetouch.processing addFeatures() getTemplate()
«Import, Call»
BiometricData com.alert.business.product.clinical.common.security.biometrics data: Object format: String technology: String type: String
«Import»
«Import, Call»
«Call»
«interface» DPFPSample com.digitalpersona.onetouch
DPFPImageQualityException com.digitalpersona.onetouch.processing getCaptureFeedback()
«enumeration» DPFPCaptureFeedback com.digitalpersona.onetouch

Figure 4.10: Digital Persona Extractor Module

This is the implementation of the Biometric Processing interface using Digital Persona's Java SDK.

The "process" method accepts a BiometricData with a Digital Persona proprietary captured sample object (DPFPSample) and returns a BiometricData with an again proprietary template object (DPFPFeatureSet). To extract the features from the sample, the "createFeatureSet" from the class DPFPFeatureExtraction was used with the parameter DATA_PURPOSE_VERIFICATION in order to specify the type of template desired.

In the case of "createTemplate", four BiometricData objects with captured samples (DPFPSample) are needed in order to create a template with the purpose of enrollment. The number of samples is specified by Digital Persona. Then, features will be extracted from each one with the parameter DATA_PURPOSE_ENROLLMENT and added to the template. Finally, a BiometricData with the generated template, in a Digital Persona proprietary format (DPFPTemplate), is returned.

In both cases, when extracting the features from the fingerprint, an exception is thrown if the image quality is poor and not sufficient to complete the task. Quality assessment is done transparently by the Digital Persona's SDK and isn't configurable.

4.3.3 Fingerprint Matcher Digital Persona

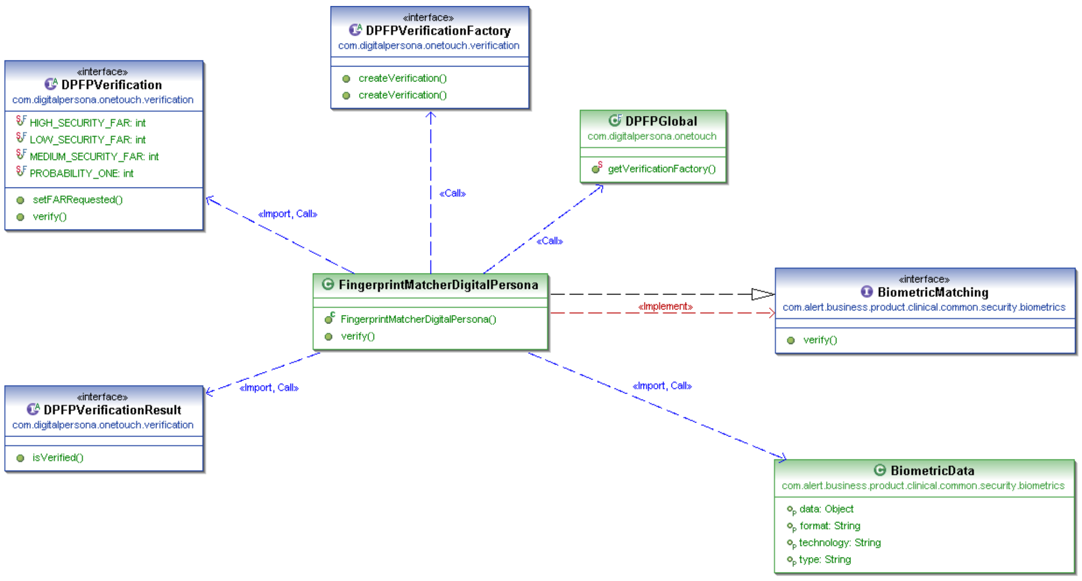


Figure 4.11: Digital Persona Matcher Module

This is the implementation of the Biometric Matching interface using Digital Persona's Java SDK. The "verify" method accepts two BiometricData objects with Digital Persona proprietary fingerprint templates for comparison. The first must be one with the purpose of verification (DPFPFeatureSet) and the latter a reference template (DPFPTemplate). Before matching them, a value for the FAR is set (setFARRequested). If none is configured, a default value of HIGH_SECURITY_FAR is assumed. Finally, "true" is returned if the matching is successful and "false" if not.

4.3.4 Fingerprint Sensor Digital Persona

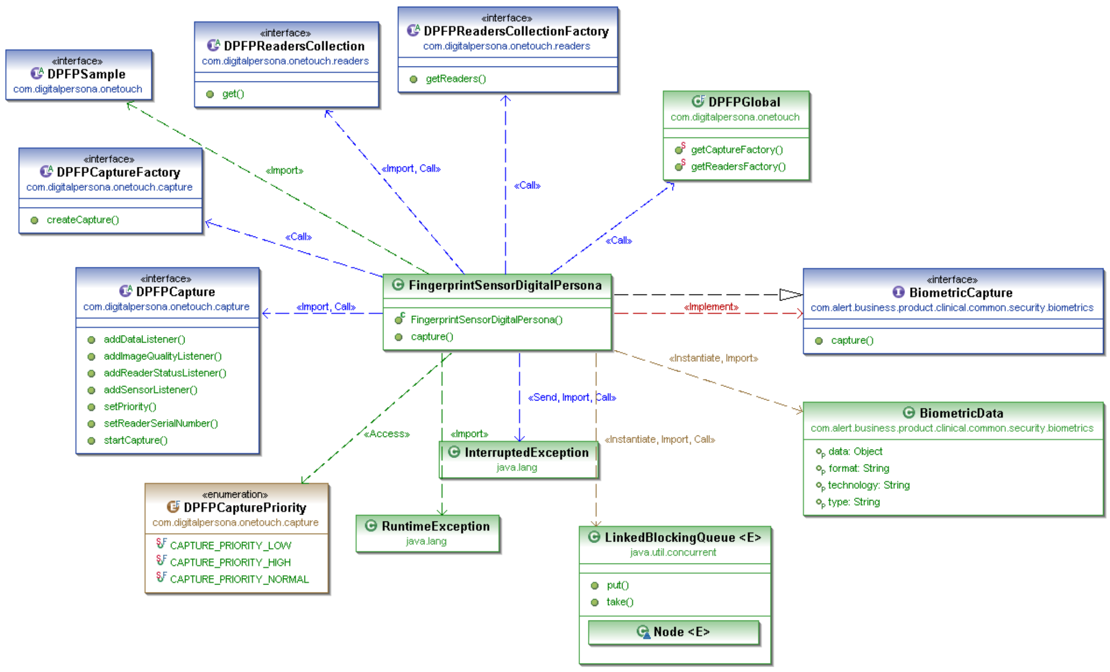
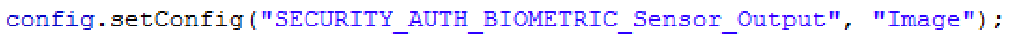

«interface» DPFPSample
com.digitalpersona.onetouch
«interface» DPFPReadersCollection
com.digitalpersona.onetouch.readers
get()
«interface» DPFPReadersCollectionFactory
com.digitalpersona.onetouch.readers
getReaders()
DPFPGlobal
com.digitalpersona.onetouch
getCaptureFactory()
getReadersFactory()
«interface» DPFPCaptureFactory
com.digitalpersona.onetouch.capture
createCapture()
«Import»
«Import, Call»
«Call»
«Call»
«Call»
«interface» DPFPCapture
com.digitalpersona.onetouch.capture
addDataListener()
addImageQualityListener()
addReaderStatusListener()
addSensorListener()
setPriority()
setReaderSerialNumber()
startCapture()
«Import, Call»
FingerprintSensorDigitalPersona
FingerprintSensorDigitalPersona()
capture()
«Implement»
«interface» BiometricCapture
com.alert.business.product.clinical.common.security.biometrics
capture()
«Instantiate, Import»
«Send, Import, Call»
«Access»
«Import»
«Instantiate, Import, Call»
BiometricData
com.alert.business.product.clinical.common.security.biometrics
data: Object
format: String
technology: String
type: String
InterruptedException
java.lang
«enumeration» DPFPCapturePriority
com.digitalpersona.onetouch.capture
CAPTURE_PRIORITY_LOW
CAPTURE_PRIORITY_HIGH
CAPTURE_PRIORITY_NORMAL
RuntimeException
java.lang
LinkedBlockingQueue <E>
java.util.concurrent
put()
take()
Node <E>

Figure 4.12: Digital Persona Sensor Module

This is the implementation of the Biometric Capture interface using Digital Persona's Java SDK and implements the method "capture" that returns a BiometricData with a captured fingerprint sample in a Digital Persona proprietary format (DPFPSample). First, it checks for any available readers supported by Digital Persona (DPFPReadersCollectionFactory) and, by default, chooses the first one to be used for capturing the fingerprint sample and waits for user input. Then, when a fingerprint image is acquired, image quality is assessed and, if good, a BiometricData with the captured sample returned in the Digital Persona proprietary format (DPFPSample). This is the default behaviour for the Digital Persona module, but there is also the possibility to configure it to return an image of the fingerprint instead of the Digital Persona proprietary sample object. This enables the use of its readers with other algorithms that allow fingerprint image input. When configuring the biometric framework to use the Digital Persona sensor module, the parameter illustrated in figure 4.13 has to be passed.



```
config.setConfig("SECURITY_AUTH_BIOMETRIC_Sensor_Output", "Image");
```

Figure 4.13: Digital Persona Sensor Configuration

This way, the method "capture" returns a biometric data with the image of the fingerprint. The image is obtained by converting the DPFPSample to an image using the Digital Persona API's DPFPSampleConversion class.

4.3.5 Fingerprint Serializer Digital Persona

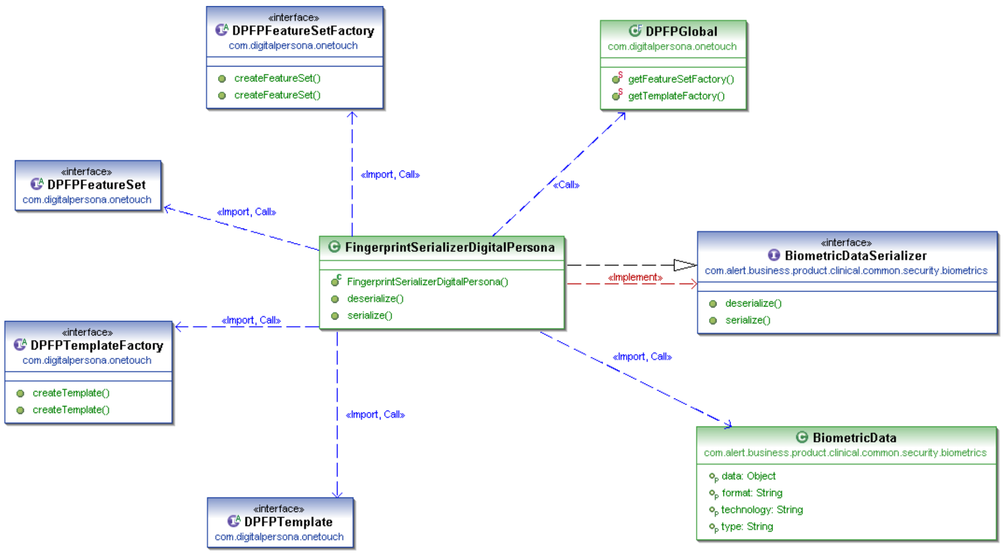


Figure 4.14: Digital Persona Template Serializer Module

This class implements the Biometric Data Serializer using the Digital Persona's SDK and, this way, allows for BiometricData objects containing Digital Persona proprietary templates to be serializable (DPFPFeatureSet and DPFPTemplate). Even though objects from these classes have methods to serialize them, they aren't serializable by nature.

When serializing or deserializing, first it is checked if the biometric data is a template (either for verification or enrollment) since only biometric templates are allowed to be stored or transferred through the network.

The "serialize" method accepts a BiometricData with a Digital Persona proprietary template and returns a byte array with the serialized template. The "deserialize" method takes a BiometricData with an empty "data" field and a byte array containing the serialized template as parameters. It then returns the same BiometricData but with the deserialized template in the "data" field.

4.3.6 Running in 64 bits

Unfortunately, the algorithm's libraries aren't written in Java and the only binaries provided are for Windows and Linux (both 32 bits) and supporting these environments in 64 bits is imperative. Linking of 64bit code with 32bit libraries is impossible without using some kind of Inter Process Communication (IPC) [TN06].

Since all x86_64 processors support native 32bit mode, running the Java Virtual Machine in 32 bit mode would allow the framework to link to the 32 bit libraries. Despite this fact, it is intended for it to run in full 64 bit mode. Windows handles this issue transparently and running the VM in 64 bit mode didn't cause any hassles with the 32bit Dlls. In Linux there is the possibility of installing compatibility libraries, but it varies from distribution to distribution and it is not guaranteed it will work and other linux-binary-compatible Unices might not have this layer.

Since it is desirable to have a simple and (possibly) universal approach, the IPC workaround was chosen. The matching process is executed in a different VM, running in 32bit mode, and IPC communication is done by Java RMI. This way, the framework can run in 64 bit mode.

A general architecture of this solution is illustrated in figure 4.15.

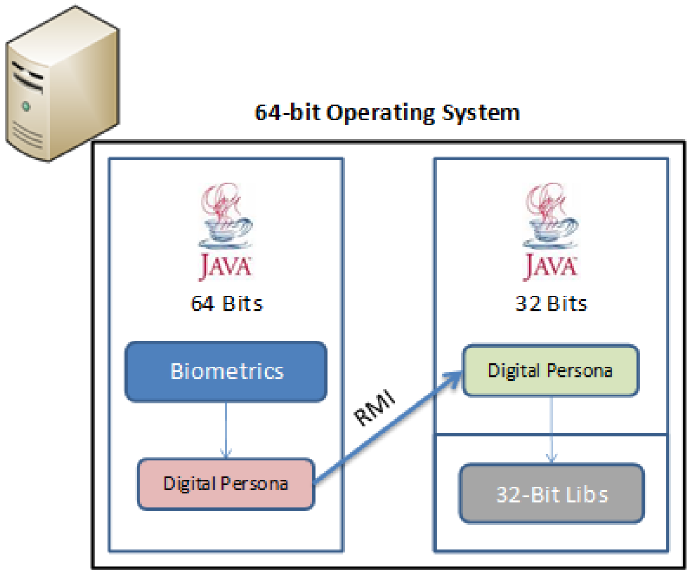


Figure 4.15: Digital Persona 64-bit Architecture

This solution is very simple and effective and didn't require any changes to the framework's structure thanks to the interface approach. The application makes use of the same exact methods that rely on the Digital Persona's libraries and all the IPC logic is handled transparently by the Digital Persona module. When on a 64bit platform, all methods that use these libraries make a remote method call to the ones being run in the 32-bit Java VM.

The parameter illustrated in figure 4.16 must be set in the configuration in order for the Digital Persona modules to use this approach.

```
config.setConfig("SECURITY_AUTH_BIOMETRIC_CPUArch", "64");
```

Figure 4.16: Digital Persona 64-bit Configuration

4.4 Integration with the Authentication Server

In order to use the biometric framework as an authentication method it had to be integrated with the authentication server since it is responsible for the interface with the application, client / server communication and the user database management.

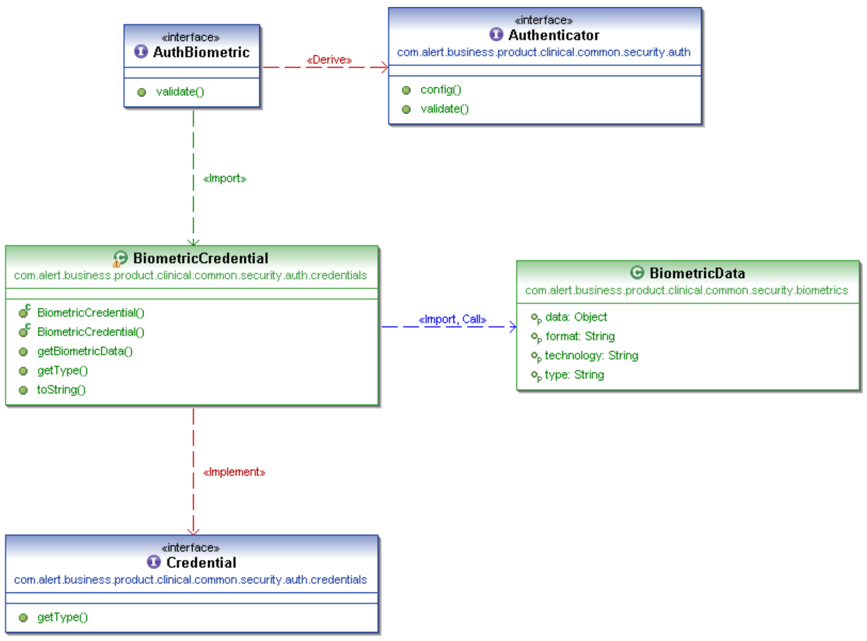


Figure 4.17: Biometric Credential and Authentication Modules

As illustrated in figure 4.17, AuthBiometric and BiometricCredential were created. The first relies on the biometric framework API and contains the authentication logic: fetches the stored reference template, matches it against the acquired sample and returns “true” or “false”, according to the matching result. The second is a simple wrapper for the biometric data that implements the authentication server’s credential interface in order for it to be used as such.

Finally, for each Identity Management System (IMS), like LDAP, a module must be written in order for the AuthBiometric to be able to fetch the user’s biometric data transparently.

4.4.1 Directory Servers

One of the main goals of the project was to be able to perform biometric authentication using popular directory servers based on the LDAP protocol, such as OpenLDAP and Microsoft Active Directory. In order to do so, a schema extension had to be devised since neither contemplate this kind of user data, nor is there a specific RFC (or any other kind of commonly adopted standard) for storing user biometric information in a directory service.

4.4.1.1 Directory Services

Directory services store, organize and provide access to the information held in organizations directories and directories are public or private resource lists containing names, locations and other identifying information. Their main advantage in Identity Management Systems (IMS) is the consolidation of existing services in a single directory, accessible by various vendors. This way, data redundancy is reduced and therefore so is the administrative overhead needed to maintain data [Car03]. Not only does this enable vendor interoperability but also seamless scalability.

The *defacto* standard for directory services is X.500. Introduced in 1988 by the International Organization for Standardization (ISO), it defines the protocols and the information model for an application and network platform agnostic directory service. The LDAP protocol was developed as a lightweight alternative to X.500 and was widely adopted in the advent of the Internet thanks to the integration with the TCP/IP protocol and its simple API. Today, the most popular directory servers are OpenLDAP, which is derived from the original University of Michigan reference LDAP implementation, and Microsoft's Active Directory that, albeit being LDAP compliant [adl03], cannot be considered as a true LDAP server.

Since it's common to find LDAP deployed in organizations, such as hospitals, as the central IMS, it is desirable for any application that requires authentication to integrate with these.

4.4.1.2 Extending the Schema

In LDAP, information is represented in a very specific logical form. It is represented as entries and these belong to one or more object classes. Each object class is defined by a set of attributes which consist in a type and one or more values. Therefore, object class and attribute type definitions make up the schema [Don03].

Since the whole purpose of using LDAP is interoperability, mapping the biometric data in plain text (string) is preferable to storing Java Objects. Reference to the type of biometric data (captured, processed, etc) is not necessary, since the only kind that is stored is templates. Therefore, as illustrated in figure 4.18, an object is created to hold the biometric information and four attributes to represent it: one for the biometric characteristic, one for the biometric technology type, one for the biometric template data format and one for the biometric template data itself. The biometric template data is represented in Base64.

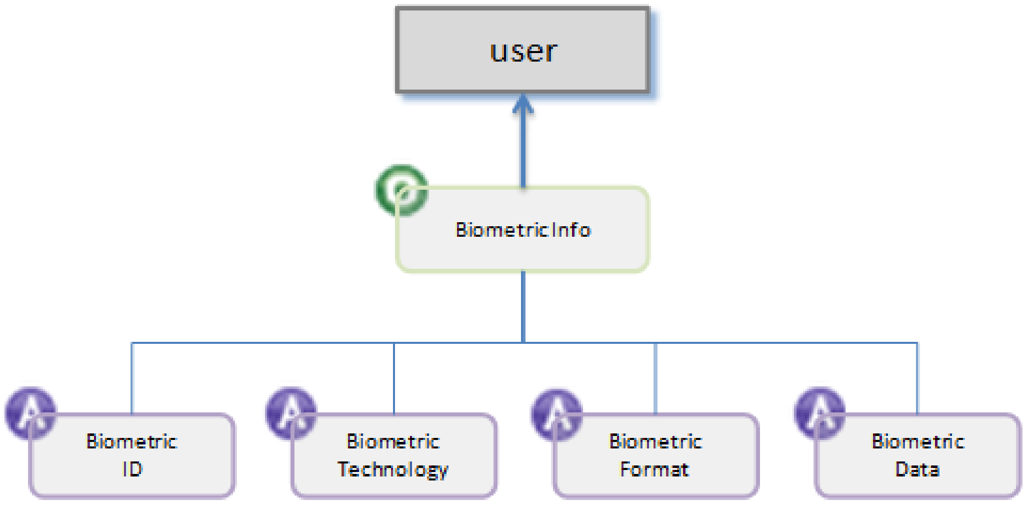


Figure 4.18: Biometric Schema Diagram

Given that the purpose of extending the schema is to allow the possibility of storing user biometric data not only in new deployments but mainly in existing ones, integrating seamlessly with them, it is desired that the biometric info class acts as an auxiliary for the one that handles the user's information.

Each schema element is identified by a globally unique Object Identifier (OID). The most common OIDs usually belong to the private enterprise numbers allocated by the Internet Assigned Numbers Authority (IANA) under the 1.3.6.1.4.1 namespace. The preferred way to obtain a root OID is to request one from an International Standards Organization (ISO) Name Registration Authority, which usually involves paying a fee but it's a one-time action that grants us the property of a unique root OID that we can administer ourselves. An easier, faster and cheaper way of obtaining an OID is from Microsoft. They provide a script that generates on-the-fly unique OID numbers assigned under the Microsoft OID number space 1.2.840.113556.1.800x, where x is a unique number assigned to the organization. It is recommended to divide it in two categories and two only: one for the classes and one for the attributes [Mic09d].

After acquiring an OID from Microsoft, it was divided following the previous guidelines as illustrated in figure 4.19.

Naming of the Attributes and Classes also follows an uniqueness principle and, therefore, either the company name should be registered with the IANA or prefixed with "x-" to place in the "private use" namespace [Ope08]. Microsoft inclusively suggests that

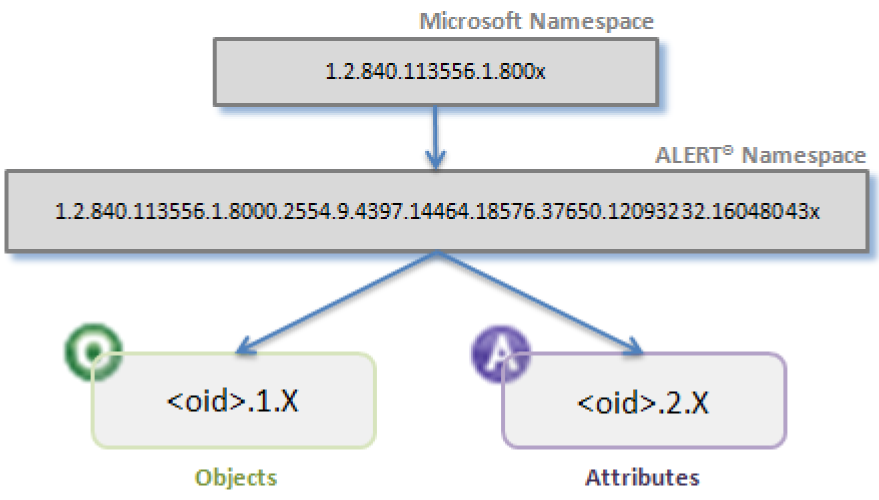


Figure 4.19: ALERT OID Structure

the prefix should be followed by the company name, ensuring not only their uniqueness company-wise but also facilitate browsing the schema, since all company objects are displayed consecutively [Mic09c]. This way the previously specified object and attributes will be named in the following way:

- x-alert-Biometric-Info
- x-alert-Biometric-ID
- x-alert-Biometric-Technology
- x-alert-Biometric-Template-Format
- x-alert-Biometric-Template-Data

Writing schema extensions for Active Directory and OpenLDAP are very different processes.

4.4.1.3 Active Directory

Adding, modifying and deleting Active Directory (AD) objects is a complex process and once the schema has been extended with the new objects (classes and attributes), they cannot be deleted. Therefore, any change to the production Active Directory schema requires a lot of planning and must be done carefully [Mal08].

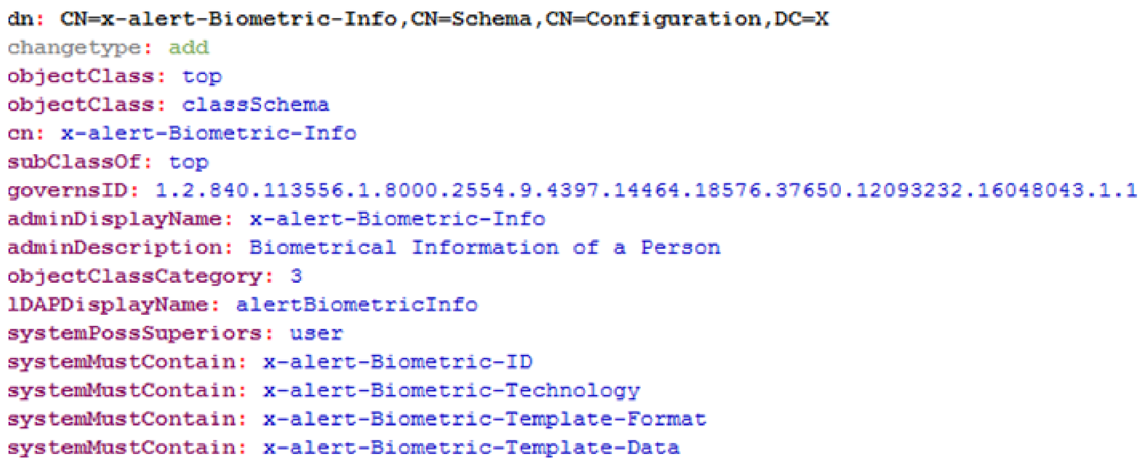
A common strategy for testing AD schema extensions before deploying in production servers is using Microsoft's Active Directory Lightweight Directory Service (ADLDS) in Windows Server 2008 (or Active Directory Application Mode, ADAM, in Windows Server 2003). ADLDS does not require the deployment of domains or domain controllers, multiple instances can run concurrently on a single computer and provides the same functionality as Active Directory. This way, while developing the schema, changes can be “reverted” by simply deleting the ADLDS instance and creating a fresh new one.

Active Directory Domain Services support four mechanisms for extending the AD Schema:

- LDAP Data Interchange Format (LDIF) Scripts
- Comma-Separated Value (CSV) Scripts
- Programmatically
- Using the User Interface

Microsoft suggests the use of the first [Mic09e], so, a LDIF file is written with the schema extension specification following Microsoft’s principles for defining new Classes and Attributes [Mic09b, Mic09a].

The class holding the biometric information is auxiliary (“objectClassCategory 3”) and should only be instantiated in the “user” class making this its parent class (“systemPossSuperiors” that, after added to the AD, cannot be changed). It is a subclass of “top” (“subClassOf”) since it doesn’t inherit from any class and must contain all four biometric attributes (id, technology, template format and data).

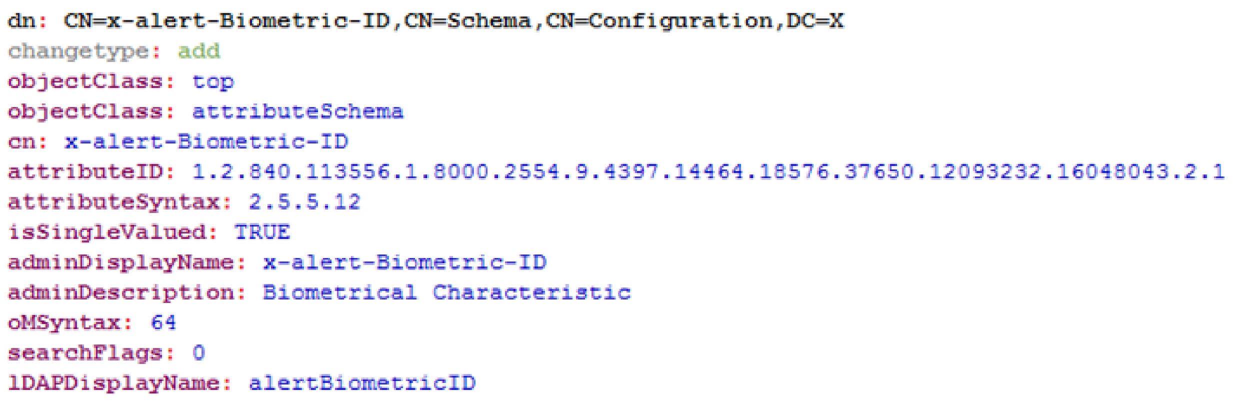


```
dn: CN=x-alert-Biometric-Info,CN=Schema,CN=Configuration,DC=X
changetype: add
objectClass: top
objectClass: classSchema
cn: x-alert-Biometric-Info
subClassOf: top
governsID: 1.2.840.113556.1.8000.2554.9.4397.14464.18576.37650.12093232.16048043.1.1
adminDisplayName: x-alert-Biometric-Info
adminDescription: Biometrical Information of a Person
objectClassCategory: 3
lDAPDisplayName: alertBiometricInfo
systemPossSuperiors: user
systemMustContain: x-alert-Biometric-ID
systemMustContain: x-alert-Biometric-Technology
systemMustContain: x-alert-Biometric-Template-Format
systemMustContain: x-alert-Biometric-Template-Data
```

Figure 4.20: AD Schema Extension: BiometricInfo Object

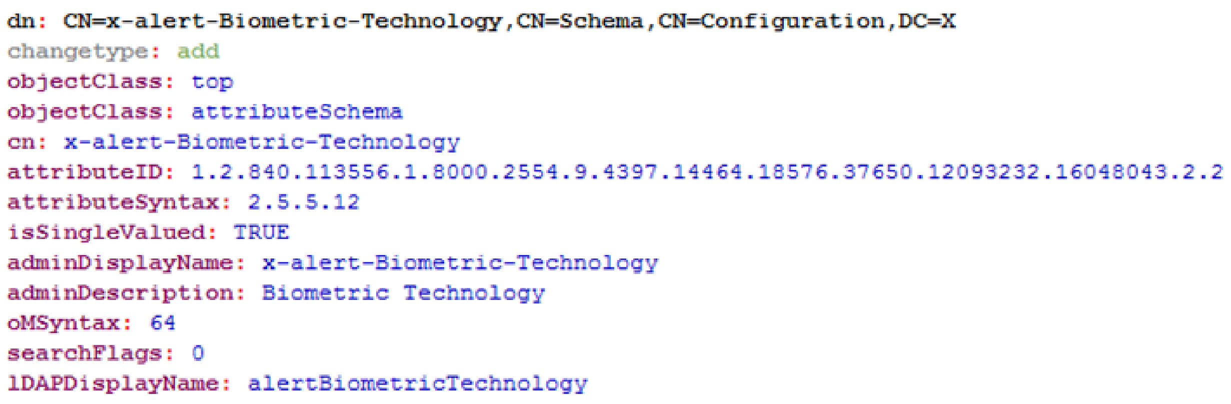
As stated before, all the four biometric attributes are strings (“attributeSyntax 2.5.5.12” and “oMSyntax 64”). Each attribute must also have only one value (“isSingleValued TRUE”). The ldif code containing these four attributes is illustrated in figure 4.21, figure 4.22, figure 4.23 and figure 4.24.

Then, the “user” class has to be modified, as illustrated in figure 4.25 to be able to instantiate the “biometric info” class.



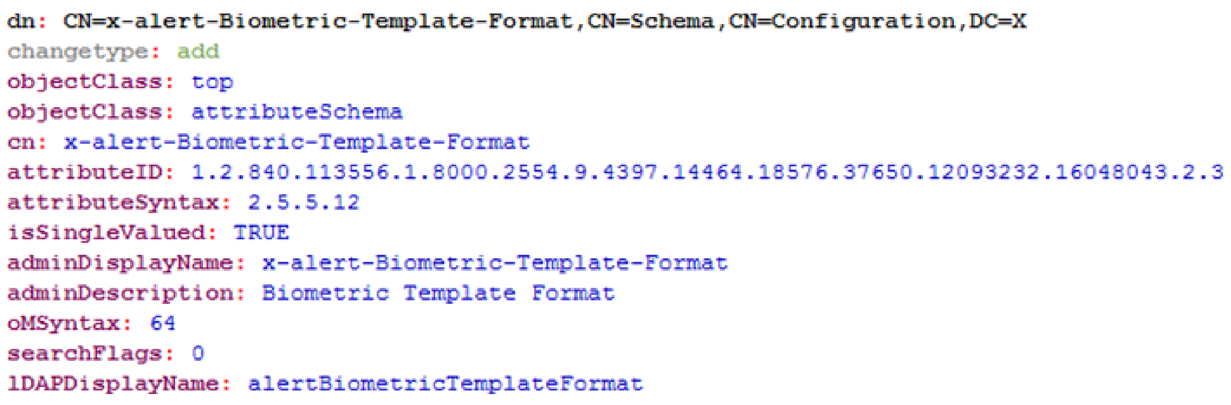
```
dn: CN=x-alert-Biometric-ID,CN=Schema,CN=Configuration,DC=X
changetype: add
objectClass: top
objectClass: attributeSchema
cn: x-alert-Biometric-ID
attributeID: 1.2.840.113556.1.8000.2554.9.4397.14464.18576.37650.12093232.16048043.2.1
attributeSyntax: 2.5.5.12
isSingleValued: TRUE
adminDisplayName: x-alert-Biometric-ID
adminDescription: Biometrical Characteristic
oMSyntax: 64
searchFlags: 0
lDAPDisplayName: alertBiometricID
```

Figure 4.21: AD Schema Extension: BiometricID Attribute



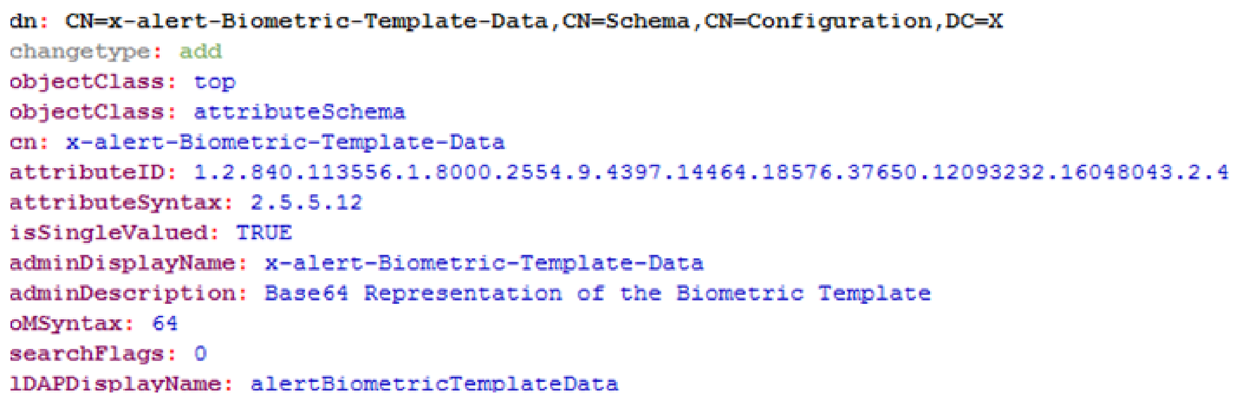
```
dn: CN=x-alert-Biometric-Technology,CN=Schema,CN=Configuration,DC=X
changetype: add
objectClass: top
objectClass: attributeSchema
cn: x-alert-Biometric-Technology
attributeID: 1.2.840.113556.1.8000.2554.9.4397.14464.18576.37650.12093232.16048043.2.2
attributeSyntax: 2.5.5.12
isSingleValued: TRUE
adminDisplayName: x-alert-Biometric-Technology
adminDescription: Biometric Technology
oMSyntax: 64
searchFlags: 0
lDAPDisplayName: alertBiometricTechnology
```

Figure 4.22: AD Schema Extension: BiometricTechnology Attribute



```
dn: CN=x-alert-Biometric-Template-Format,CN=Schema,CN=Configuration,DC=X
changetype: add
objectClass: top
objectClass: attributeSchema
cn: x-alert-Biometric-Template-Format
attributeID: 1.2.840.113556.1.8000.2554.9.4397.14464.18576.37650.12093232.16048043.2.3
attributeSyntax: 2.5.5.12
isSingleValued: TRUE
adminDisplayName: x-alert-Biometric-Template-Format
adminDescription: Biometric Template Format
oMSyntax: 64
searchFlags: 0
lDAPDisplayName: alertBiometricTemplateFormat
```

Figure 4.23: AD Schema Extension: BiometricFormat Attribute



```
dn: CN=x-alert-Biometric-Template-Data,CN=Schema,CN=Configuration,DC=X
changetype: add
objectClass: top
objectClass: attributeSchema
cn: x-alert-Biometric-Template-Data
attributeID: 1.2.840.113556.1.8000.2554.9.4397.14464.18576.37650.12093232.16048043.2.4
attributeSyntax: 2.5.5.12
isSingleValued: TRUE
adminDisplayName: x-alert-Biometric-Template-Data
adminDescription: Base64 Representation of the Biometric Template
oMSyntax: 64
searchFlags: 0
lDAPDisplayName: alertBiometricTemplateData
```

Figure 4.24: AD Schema Extension: BiometricData Attribute

Finally, the schema is extended by importing the file with the LDIFDE Tool. Now, biometric information about a user can be added in the Active Directory.

4.4.1.4 OpenLDAP

In OpenLDAP, extending the schema is pretty straightforward. The principles followed before for creating the classes and attributes are also applied here. The difference is in the way the server's default schema is extended.

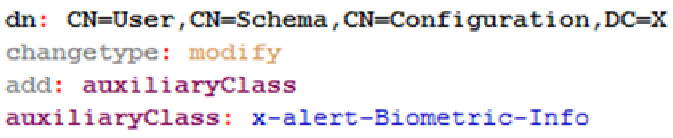
Unlike Active Directory, schemas are defined in files and not in the directory tree itself. Therefore, our custom schema is contained in one file and it is then included in the server's configuration file to be loaded when it starts. The schema follows the LDAP schema RFC 2252, section 4.4 (figure 4.26) for the classes and section 4.2 (figure 4.27) for the attributes.

The resulting schema file, equivalent to the schema extension defined previously for Active Directory, is specified for OpenLDAP as it is shown in figure 4.28.

4.5 Java Applet

Another goal of the project was for the biometric framework to support web browsers. In theory, a Java application can be run as a Java Applet without any major modifications to the code and have full access to the machine it is running (if the user agrees to), thus having access to the fingerprint readers and local extraction/matching libraries if needed (which is the case of the Digital Persona's module). Since displaying the applet varies from browser to browser and, again, we are looking at a single, scalable solution, a single html file handles this issue using JavaScript, as illustrated on figure 4.29.

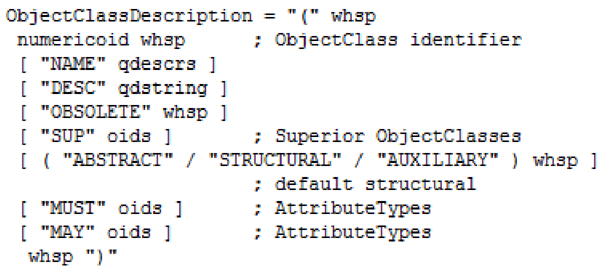
The only issue when using the biometric framework through the applet came with the Digital Persona's module since it relies on JNI. Only applications and signed applets can



```
dn: CN=User,CN=Schema,CN=Configuration,DC=X
changetype: modify
add: auxiliaryClass
auxiliaryClass: x-alert-Biometric-Info
```

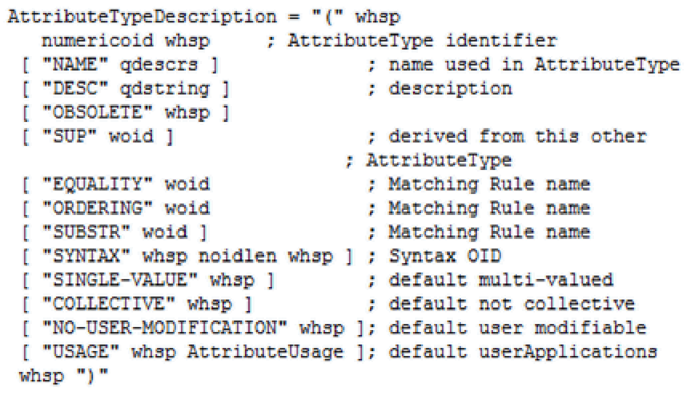
Figure 4.25: AD Schema Extension: Modifying User Class

invoke the JNI and, therefore, the applet and all the jars it depends on have to be digitally signed in order for it to work. The Java SDK provides a tool to do this: jarsigner.



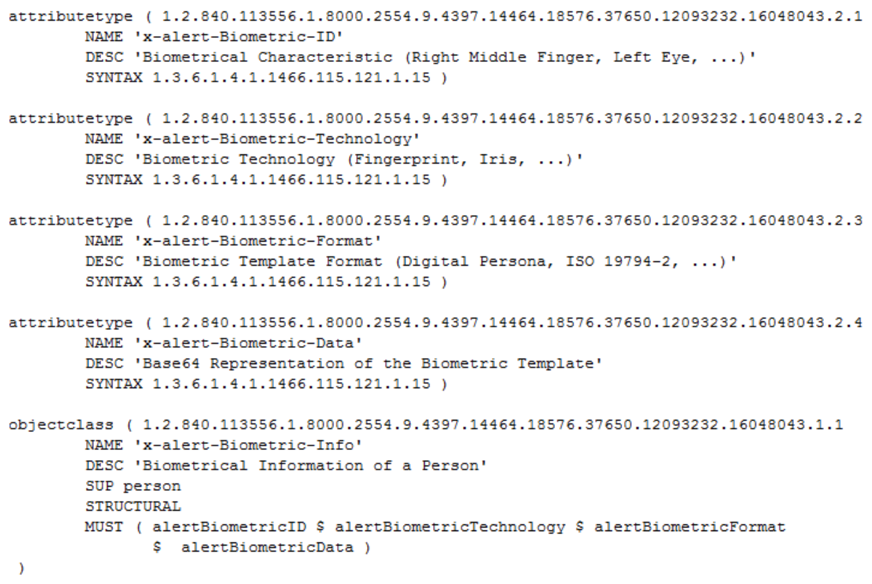
```
ObjectClassDescription = "(" whsp
 numericoid whsp      ; ObjectClass identifier
 [ "NAME" qdescrs ]
 [ "DESC" qdstring ]
 [ "OBSOLETE" whsp ]
 [ "SUP" oids ]        ; Superior ObjectClasses
 [ ( "ABSTRACT" / "STRUCTURAL" / "AUXILIARY" ) whsp ]
                       ; default structural
 [ "MUST" oids ]       ; AttributeTypes
 [ "MAY" oids ]        ; AttributeTypes
  whsp ")"
```

Figure 4.26: RFC 2252 - Object Class



```
AttributeTypeDescription = "(" whsp
   numericoid whsp     ; AttributeType identifier
 [ "NAME" qdescrs ]             ; name used in AttributeType
 [ "DESC" qdstring ]            ; description
 [ "OBSOLETE" whsp ]
 [ "SUP" woid ]                 ; derived from this other
                              ; AttributeType
 [ "EQUALITY" woid              ; Matching Rule name
 [ "ORDERING" woid              ; Matching Rule name
 [ "SUBSTR" woid ]              ; Matching Rule name
 [ "SYNTAX" whsp noidlen whsp ] ; Syntax OID
 [ "SINGLE-VALUE" whsp ]        ; default multi-valued
 [ "COLLECTIVE" whsp ]          ; default not collective
 [ "NO-USER-MODIFICATION" whsp ]; default user modifiable
 [ "USAGE" whsp AttributeUsage ]; default userApplications
 whsp ")"
```

Figure 4.27: RFC 2252 - Attribute Class



```
attributetype ( 1.2.840.113556.1.8000.2554.9.4397.14464.18576.37650.12093232.16048043.2.1
        NAME 'x-alert-Biometric-ID'
        DESC 'Biometrical Characteristic (Right Middle Finger, Left Eye, ...)'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

attributetype ( 1.2.840.113556.1.8000.2554.9.4397.14464.18576.37650.12093232.16048043.2.2
        NAME 'x-alert-Biometric-Technology'
        DESC 'Biometric Technology (Fingerprint, Iris, ...)'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

attributetype ( 1.2.840.113556.1.8000.2554.9.4397.14464.18576.37650.12093232.16048043.2.3
        NAME 'x-alert-Biometric-Format'
        DESC 'Biometric Template Format (Digital Persona, ISO 19794-2, ...)'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

attributetype ( 1.2.840.113556.1.8000.2554.9.4397.14464.18576.37650.12093232.16048043.2.4
        NAME 'x-alert-Biometric-Data'
        DESC 'Base64 Representation of the Biometric Template'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

objectclass ( 1.2.840.113556.1.8000.2554.9.4397.14464.18576.37650.12093232.16048043.1.1
        NAME 'x-alert-Biometric-Info'
        DESC 'Biometrical Information of a Person'
        SUP person
        STRUCTURAL
        MUST ( alertBiometricID $ alertBiometricTechnology $ alertBiometricFormat
               $  alertBiometricData )
 )
```

Figure 4.28: OpenLDAP Schema File

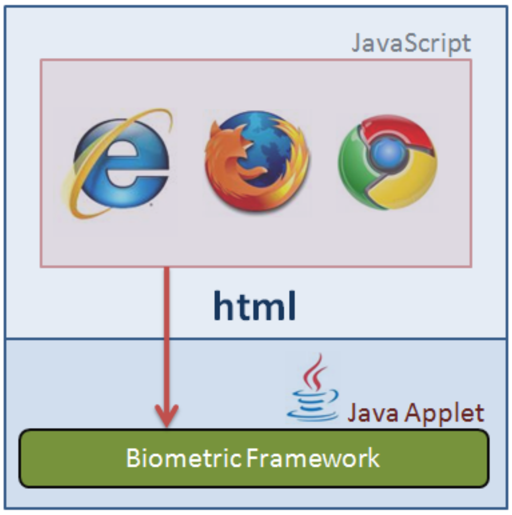


Figure 4.29: Java Applet Diagram

Chapter 5

Testing

The main goal of this project was to design a biometric framework that, first and foremost, supported Digital Persona's fingerprint authentication technology and ran, at least, in Windows and Linux (and their 64bit counterparts) and in Internet Explorer and Mozilla Firefox. Given the increased popularity of the Mac OSX operating system and the Safari and Google Chrome web browsers, they were also tested for compatibility.

Since the core of the framework is built in Java, it is safe to assume that it will run in every operating system and web browser that have a Java Virtual Machine. The same cannot be said about the modules that depend on proprietary technology that might not be able to run in the same environments.

5.1 JUnit

JUnit test cases were used to easily assess the compatibility of the biometric framework and, in this case, the Digital Persona fingerprint modules with the various operating systems. If the tests pass, it is safe to assume that they are compatible.

Unit testing is a black box automated test scenario, so, there cannot be any input/output. This way, we have to save previously captured fingerprints and accept them as valid. Not only it isn't possible to test the capture process, but also the feature extraction since only the biometric templates are serializable and, therefore, can be saved. So, it is only possible to test the template serialization/deserialization and the matching of two fingerprints. Regarding the serialization of the templates, two distinct scenarios are tested: using the serializer module directly and by writing/reading a biometric data object (which relies on the first to do so).

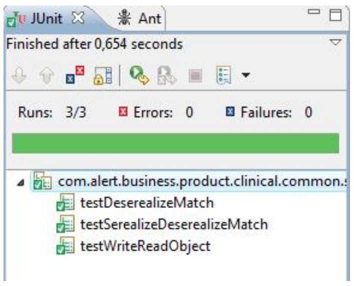
After capturing one fingerprint template for verification and other for enrollment from two distinct fingers, three test cases were written:

- testDeserealizeMatch: The four templates are deserialized using the serializer module directly and matched against each other. If the templates from the same finger match and from different fingers do not match, the test is successful.

- testSerealizeDeserealizeMatch: This test is similar to the previous but the templates are first deserialized and serialized again.

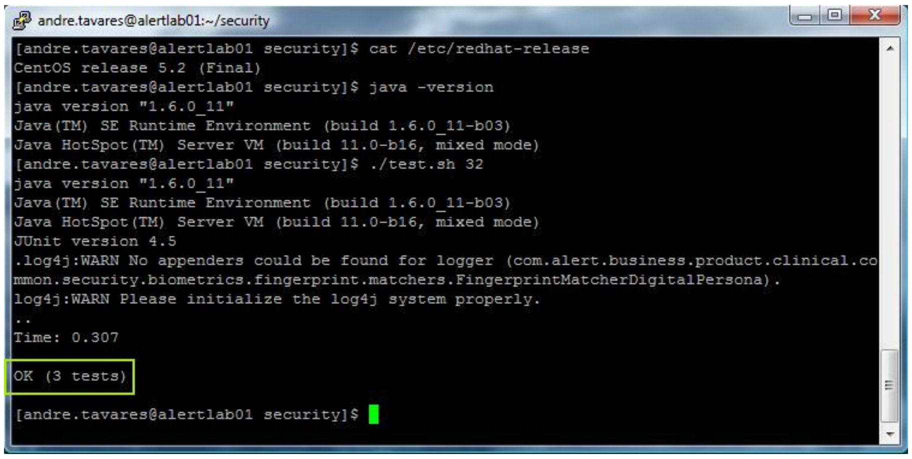
- testWriteReadObject: This test is similar to the second but, instead of using the serializer module directly, it is the biometric data object that is serialized and then deserialized.

An output of a successful test in Eclipse looks like figure 5.1 and, in the command line, like figure 5.2.



```
Finished after 0,654 seconds
Runs: 3/3    Errors: 0    Failures: 0
com.alert.business.product.clinical.common.s
    testDeserealizeMatch
    testSerealizeDeserealizeMatch
    testWriteReadObject
```

Figure 5.1: JUnit Success Output - Eclipse



```
[andre.tavares@alertlab01 security]$ cat /etc/redhat-release
CentOS release 5.2 (Final)
[andre.tavares@alertlab01 security]$ java -version
java version "1.6.0_11"
Java(TM) SE Runtime Environment (build 1.6.0_11-b03)
Java HotSpot(TM) Server VM (build 11.0-b16, mixed mode)
[andre.tavares@alertlab01 security]$ ./test.sh 32
java version "1.6.0_11"
Java(TM) SE Runtime Environment (build 1.6.0_11-b03)
Java HotSpot(TM) Server VM (build 11.0-b16, mixed mode)
JUnit version 4.5
.log4j:WARN No appenders could be found for logger (com.alert.business.product.clinical.co
mmon.security.biometrics.fingerprint.matchers.FingerprintMatcherDigitalPersona).
log4j:WARN Please initialize the log4j system properly.
..
Time: 0.307

OK (3 tests)

[andre.tavares@alertlab01 security]$
```

Figure 5.2: JUnit Success Output - Command Line

When testing compatibility with an operating system it doesn't suffice to run these tests and see if they pass. Not passing the tests doesn't directly imply that they aren't compatible. Some specific configurations, such as library paths, might be needed and these tests also allow easy detection of common errors such as this.

5.2 Operating Systems

Regarding operating systems, tests were conducted in the following:

- Windows Vista (32 bits)
- Windows Server 2003 (64 bits)
- Ubuntu Linux 8.10 (32 bits)
- CentOS 5.2 (64 bits)
- Mac OSX 10.5 Leopard (64 bits)

The only OS where the tests weren't successful was Mac OSX. Despite being UNIX certified, its binaries aren't in the ELF format but Mach-O. This makes it impossible for the Digital Persona's Linux libraries, which are precisely in ELF, to work with it.

5.3 Browser and LDAP

To test the browser applet, a real world scenario was devised. As illustrated in figure 5.3, the applet is served by a production Apache web server and connects to an authentication server running in a Windows Server 2003 (64bits) in order to perform fingerprint verification, which then connects to an OpenLDAP server running on a CentOS 5.2 64bits machine.

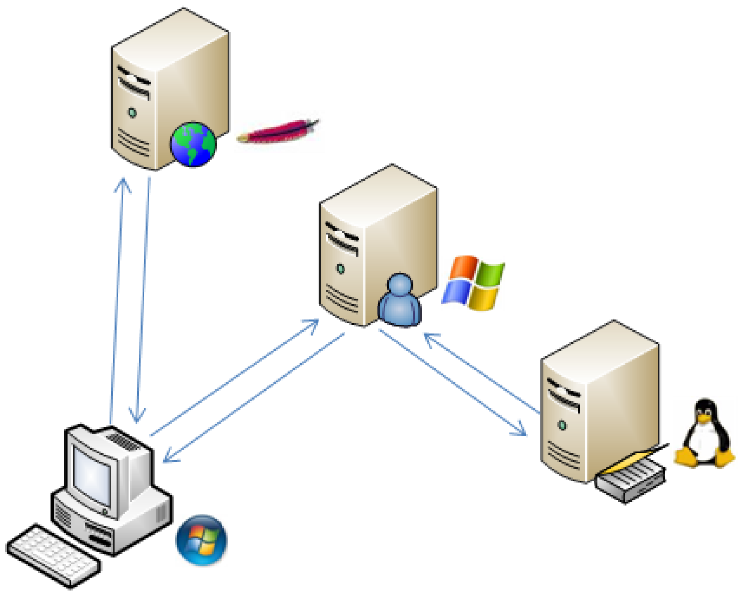
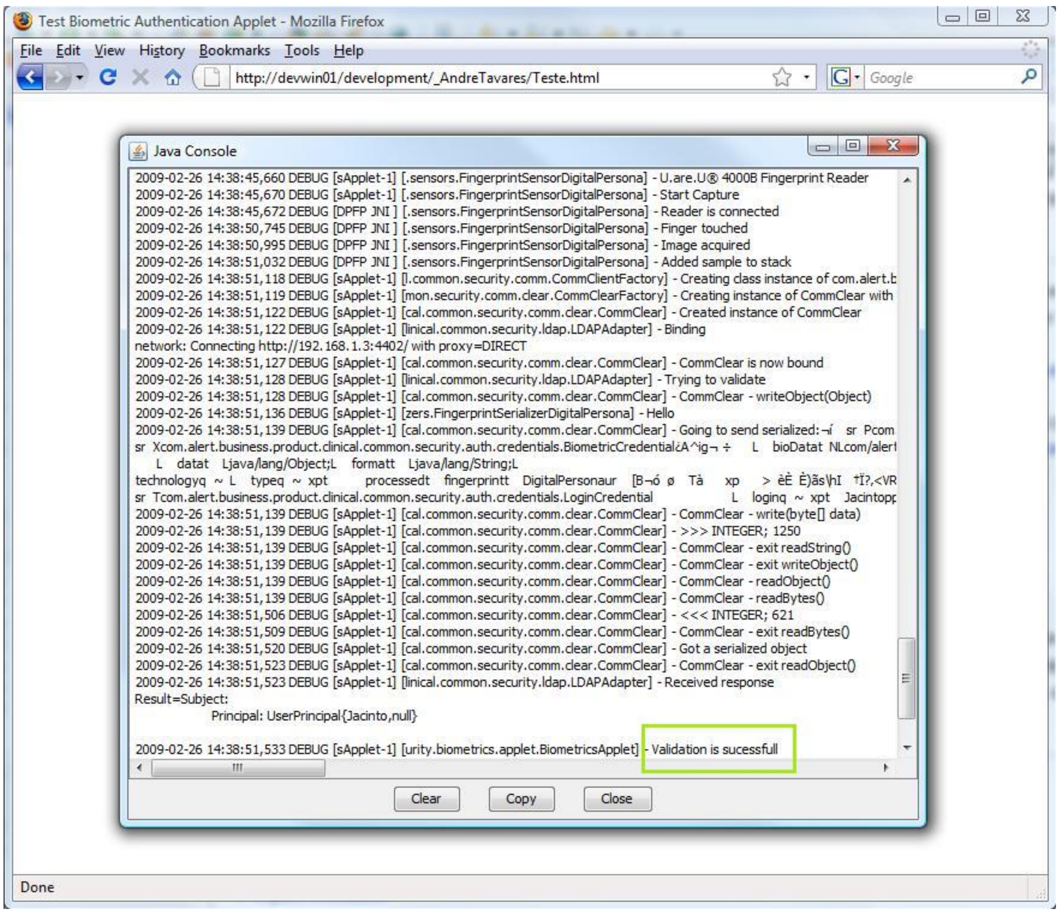


Figure 5.3: Test Setup

A successful fingerprint verification in the Java Console looks like figure 5.4.


Test Biometric Authentication Applet - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://devwin01/development/_AndreTavares/Teste.html
Google
Java Console
Clear
Copy
Close
Done

```
2009-02-26 14:38:45,660 DEBUG [sApplet-1] [.sensors.FingerprintSensorDigitalPersona] - U.are.U® 4000B Fingerprint Reader
2009-02-26 14:38:45,670 DEBUG [sApplet-1] [.sensors.FingerprintSensorDigitalPersona] - Start Capture
2009-02-26 14:38:45,672 DEBUG [DPFP JNI ] [.sensors.FingerprintSensorDigitalPersona] - Reader is connected
2009-02-26 14:38:50,745 DEBUG [DPFP JNI ] [.sensors.FingerprintSensorDigitalPersona] - Finger touched
2009-02-26 14:38:50,995 DEBUG [DPFP JNI ] [.sensors.FingerprintSensorDigitalPersona] - Image acquired
2009-02-26 14:38:51,032 DEBUG [DPFP JNI ] [.sensors.FingerprintSensorDigitalPersona] - Added sample to stack
2009-02-26 14:38:51,118 DEBUG [sApplet-1] [l.common.security.comm.CommClientFactory] - Creating class instance of com.alert.b
2009-02-26 14:38:51,119 DEBUG [sApplet-1] [mon.security.comm.clear.CommClearFactory] - Creating instance of CommClear with
2009-02-26 14:38:51,122 DEBUG [sApplet-1] [cal.common.security.comm.clear.CommClear] - Created instance of CommClear
2009-02-26 14:38:51,122 DEBUG [sApplet-1] [linical.common.security.ldap.LDAPAdapter] - Binding
network: Connecting http://192.168.1.3:4402/ with proxy=DIRECT
2009-02-26 14:38:51,127 DEBUG [sApplet-1] [cal.common.security.comm.clear.CommClear] - CommClear is now bound
2009-02-26 14:38:51,128 DEBUG [sApplet-1] [linical.common.security.ldap.LDAPAdapter] - Trying to validate
2009-02-26 14:38:51,128 DEBUG [sApplet-1] [cal.common.security.comm.clear.CommClear] - CommClear - writeObject(Object)
2009-02-26 14:38:51,136 DEBUG [sApplet-1] [zers.FingerprintSerializerDigitalPersona] - Hello
2009-02-26 14:38:51,139 DEBUG [sApplet-1] [cal.common.security.comm.clear.CommClear] - Going to send serialized: ¬í   sr  Pcom
sr  Xcom.alert.business.product.clinical.common.security.auth.credentials.BiometricCredential¿A^ig¬ ÷     L   bioDatat  NLcom/alert
    L   datat   Ljava/lang/Object;L    formatt    Ljava/lang/String;L
technologyq  ~ L   typeq  ~ xpt       processedt   fingerprintt   DigitalPersonaur    [B¬ó  ø   Tà     xp     > èÈ  È)ãs\hI   †Ï?,<VR
sr  Tcom.alert.business.product.clinical.common.security.auth.credentials.LoginCredential              L   loginq  ~ xpt   Jacintopp
2009-02-26 14:38:51,139 DEBUG [sApplet-1] [cal.common.security.comm.clear.CommClear] - CommClear - write(byte[] data)
2009-02-26 14:38:51,139 DEBUG [sApplet-1] [cal.common.security.comm.clear.CommClear] - >>> INTEGER; 1250
2009-02-26 14:38:51,139 DEBUG [sApplet-1] [cal.common.security.comm.clear.CommClear] - CommClear - exit readString()
2009-02-26 14:38:51,139 DEBUG [sApplet-1] [cal.common.security.comm.clear.CommClear] - CommClear - exit writeObject()
2009-02-26 14:38:51,139 DEBUG [sApplet-1] [cal.common.security.comm.clear.CommClear] - CommClear - readObject()
2009-02-26 14:38:51,139 DEBUG [sApplet-1] [cal.common.security.comm.clear.CommClear] - CommClear - readBytes()
2009-02-26 14:38:51,506 DEBUG [sApplet-1] [cal.common.security.comm.clear.CommClear] - <<< INTEGER; 621
2009-02-26 14:38:51,509 DEBUG [sApplet-1] [cal.common.security.comm.clear.CommClear] - CommClear - exit readBytes()
2009-02-26 14:38:51,520 DEBUG [sApplet-1] [cal.common.security.comm.clear.CommClear] - Got a serialized object
2009-02-26 14:38:51,523 DEBUG [sApplet-1] [cal.common.security.comm.clear.CommClear] - CommClear - exit readObject()
2009-02-26 14:38:51,523 DEBUG [sApplet-1] [linical.common.security.ldap.LDAPAdapter] - Received response
Result=Subject:
            Principal: UserPrincipal{Jacinto,null}

2009-02-26 14:38:51,533 DEBUG [sApplet-1] [urity.biometrics.applet.BiometricsApplet] - Validation is sucessfull
```

Figure 5.4: Applet Testing - Firefox

The applet was tested in the following browsers:

- Apple Safari 4
- Google Chrome 1
- Microsoft Internet Explorer 7
- Mozilla Firefox 3

The tests were successful in all four. Not only can we conclude the browser compatibility but also that the biometric framework was able to use a LDAP server as repository for biometric information about a user.

5.4 Deployment

In figure 5.5 is shown an example of a computer running the ALERT(r) application and the fingerprint reader used for user authentication.

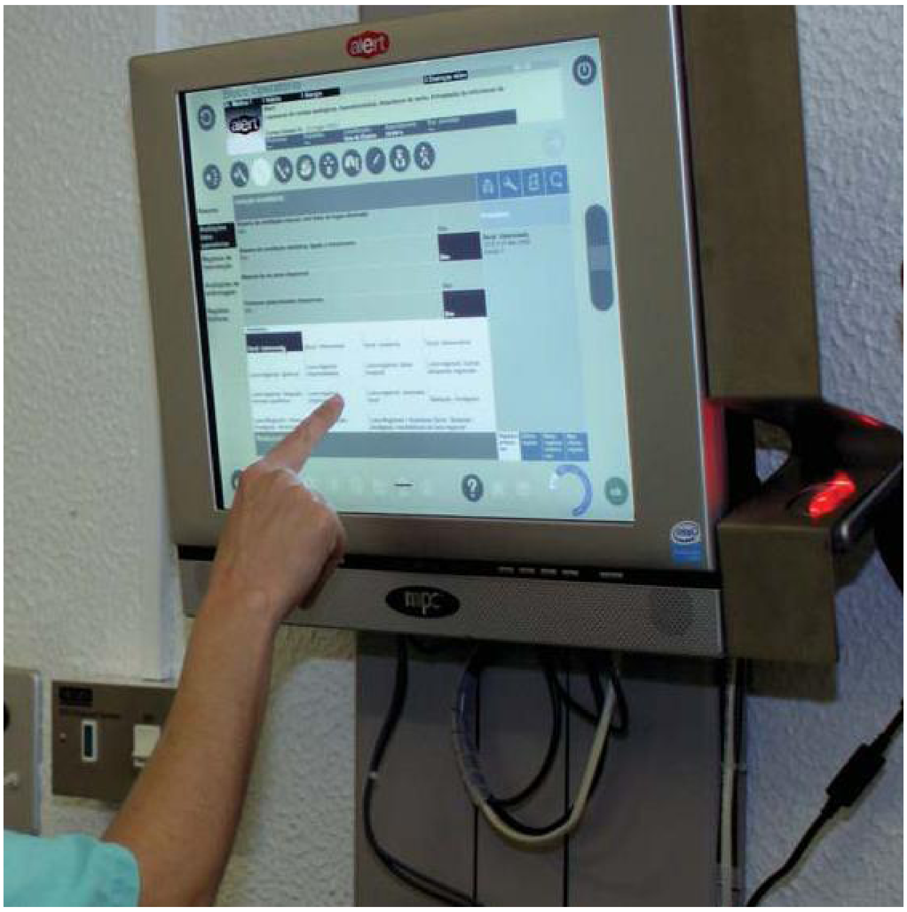


Figure 5.5: Computer Running ALERT(r) Software

Chapter 6

Conclusions

Before the development of this framework, not only was the ALERT(r) software dependant on one specific biometric technology, fingerprint, but also one provider. As it was said in Chapter 1, it is impossible not to depend on others, but the question here is how deep one lets that dependence go. An application should be layered into logical components that interact with each other to provide the overall functionality, eliminating direct dependencies between them and, thus, enabling seamless changes to each one to be done without breaking functionality.

Moving the biometric authentication logic out of the core of the application and defining a standard immutable API for these to communicate, allows for any desired developments at the biometrics level to be done without breaking the application. This way, it stops being dependant on a specific provider. By taking this abstraction one step further and isolating the biometric process logic, it then becomes possible to support different biometric technologies and, for each one, different vendors.

This is what was achieved with the developed biometrics framework. The ALERT(r) software now supports not only different fingerprint authentication technologies, but different biometric technologies.

Adopting a single vendor that complied with international standards would be the ideal scenario, mainly because it would also eliminate the overhead of having to support different incompatible technologies. Unfortunately, all the solutions that meet this requirement are still very expensive.

Of the two fingerprint technology providers currently in use, only Digital Persona was implemented, since it is represents more than 90% of the application deployments. It now supports the required operating system (64-bit Windows and Linux) and browser (Internet Explorer, Firefox and Chrome) environments. The obvious next step is to add support for MegaMatcher.

The framework is still only a proof of concept and further developments have to be done to put it in production, such as exception handling, the ability to store user biometric information (only fetch was implemented) and its configuration internals. It would also be interesting to test it in other Unices, such as OpenSolaris.

References

[adl03] Active directory ldap compliance. Technical report, Microsoft Corporation, October 2003.

[ALE08] ALERT Life Sciences Computing, S.A. Company Presentation, 2008. http://www.alert.pt/.

[Bec08] Kent Beck. *Implementation Patterns*. Addison-Wesley, First edition, 2008.

[Bio08] BioAPI Consortium. Bioapi 2.0 version description, 2008. http://www.bioapi.org/.

[Car03] Gerald Carter. *LDAP System Administration*. O'Reilly, 2003.

[Dig07] Digital Persona, Inc. Guide to fingerprint recognition. Technical report, July 2007.

[Don03] Clayton Donley. *LDAP Programming, Management and Integration*. Manning, 2003.

[How08] HowStuffWorks, Inc. How fingerprint scanners work, 2008. http://computer.howstuffworks.com/fingerprint-scanner1.htm.

[HT99] Andrew Hunt and David Thomas. *The Pragmatic Programmer*. Addison-Wesley, 1999.

[Jai04] Anil K. Jain. Fingerprint recognition. Technical report, Michigan State University, January 2004.

[Mal08] Vikas Malhotra. Extending the active directory schema. *TechNet Magazine*, May 2008.

[Mic09a] Microsoft Corporation. Defining a new attribute, 2009. http://msdn.microsoft.com/en-us/library/ms675883(VS.85).aspx.

[Mic09b] Microsoft Corporation. Defining a new class, 2009. http://msdn.microsoft.com/en-us/library/ms675884.aspx.

[Mic09c] Microsoft Corporation. Naming attributes and classes, 2009. http://msdn.microsoft.com/en-us/library/ms677603(VS.85).aspx.

[Mic09d] Microsoft Corporation. Obtaining an object identifier from microsoft, 2009. http://msdn.microsoft.com/en-us/library/ms677620(VS.85).aspx.

[Mic09e] Microsoft Corporation. Supported installation mechanisms, 2009. http://msdn.microsoft.com/en-us/library/ms677966(VS.85).aspx.

[Nag] Ramesh Nagappan. Stronger authentication with biometric sso using opensso enterprise and biobextm. Technical report, Sun Microsystems. http://www.coresecuritypatterns.com/blogs/wp-content/uploads/2009/01/egov-opensso-biobex-rameshnagappan.pdf.

[Nat08a] National Institute of Standards and Technology. Minex compliant feature extractors, 2008. http://fingerprint.nist.gov/minex/QPL.html.

[Nat08b] National Institute of Standards and Technology. Minutiae interoperability exchange test, 2008. http://fingerprint.nist.gov/minex/.

[Ope08] OpenLDAP Foundation. Schema specification, 2008. http://www.openldap.org/doc/admin24/schema.html.

[otSCPtSotFo05] Report on the Study Conducted Pursuant to Section 157 of the Fair and Accurate Credit Transactions Act of 2003. The use of technology to combat identity theft. Technical report, The Department of Treasury, February 2005.

[RFL09] RFLOGICS. Overview of the biometrics system, 2009. http://www.rflogicsinc.com/Technology/Biometrics.htm.

[Rob05] Chris Roberts. Biometrics. Technical report, November 2005.

[TN06] C. Timossi and H. Nishimura. Epics sca clients on the .net x64 platform. Technical report, 2006.

[Yun03] Yau Wei Yun. The ‘123’ of biometric technology, 2003. http://www.cp.su.ac.th/~rawitat/teaching/forensicit06/coursefiles/files/biometric.pdf.