A multiplatform Java wrapper for the BioAPI framework

Elisardo González-Agulla *, Enrique Otero-Muras, Carmen García-Mateo, José L. Alba-Castro

GTS — University of Vigo, Spain

Received 4 May 2006; received in revised form 31 October 2007; accepted 18 November 2007
Available online 3 December 2007

Abstract

We present a solution for the development of multiplatform and web-oriented Java applications for biometric authentication based on the BioAPI framework. Our proposal is a single Java Native Interface wrapper that is compatible with the BioAPI instantiations of Windows and Linux/Unix operating systems. Following a study of existing wrappers, we extended the open-source Linux/Unix wrapper to include Windows support. In order to show the utility of this multiplatform Java wrapper, we discuss some of its possible uses and present a real-life application. © 2007 Elsevier B.V. All rights reserved.

Keywords: Biometrics; BioAPI; Java; Web authentication

1. Introduction

Biometrics, which can be defined as the automatic recognition of a claimed identity using certain physiological or behavioural traits associated with the person [1], is an emerging technology that holds great promise for the future. Biometrics are gaining ground over traditional methods such as the use of passwords or PINs in the field of personal authentication because they link events to a particular individual (a password or token may be used by someone else than the authorized user), are convenient (there is nothing to carry or remember), accurate (they provide for positive authentication), can provide an audit trail, and are becoming socially acceptable and inexpensive. One of the major early problems that biometrics faced was the lack of interoperability between different software applications and devices developed by different vendors.

The BioAPI Consortium was founded to develop a biometric Application Programming Interface (API) that would bring platform and device independence to application programmers and Biometric Service Providers (BSPs). The BioAPI Consortium developed a specification and reference implementation for a standardized API (BioAPI 1.1) [2] that is compatible with a wide range of biometric application programs and a broad spectrum of biometric technologies [3]. The BioAPI Specification defines an open system standard application program interface (API) which allows software applications to communicate with a broad range of biometric technologies. The BioAPI 2.0 standard is a completely new version developed by the international standards committee for biometrics within ISO (ISO/IEC JTC1 SC37). The international version (ISO/IEC 19784-1 [4]) offers several improvements over the ANSI standard version (BioAPI 1.1).

All the software instantiations of the BioAPI specification are currently implemented in C/C+ and run primarily on a Windows (Win32) platform although versions for Linux/Unix and other operating systems are becoming available.

In order to access the BioAPI framework in C/C+ from a Java environment, a Java wrapper based on Java Native Interface (JNI) technology can be used. To the best of our knowledge, there are currently two different JNI-based BioAPI wrappers available: the BioAPI wrapper for Windows developed by Gens Software [5] and an open-source wrapper for Linux/Unix called JBioAPI [6]. This second wrapper constituted the starting point for the work presented in this paper.
Our study focused on extending the JBioAPI wrapper for the Windows platform by creating a single Java Native Interface wrapper for the BioAPI framework that would allow BioAPI modules installed in either Windows or Linux/Unix OS to be managed from any Java application. It is, as such, intended to facilitate the development of multiplatform and web-oriented Java applications for BioAPI-based biometric authentication.

In order to demonstrate the functionality of this unified JNI wrapper we developed a multiplatform application for distributed biometric authentication in a web environment called BioWebAuth [7]. BioWebAuth (Biometrics for Web Authentication) is an open-source project that provides single sign-on web authentication based on BioAPI-compliant biometric software or devices. It can be found at SourceForge under LGPL license.

The rest of the paper is organized as follows. Section 2 introduces the BioAPI model, and pays special attention to client/server architectures. Section 3 presents our approach to the development of multiplatform Java BioAPI-compliant applications: a unified Java Native Interface wrapper for the BioAPI framework. Section 4 comments on the uses of the unified Java BioAPI wrapper for the simple development of web-oriented biometric applications, and presents an example of a multiplatform Java-based application for biometric authentication in a web environment. Section 5 discusses the relevance of our approach to the standards and Section 6 outlines our conclusions.

2. The BioAPI model

The BioAPI standard [2] uses the term template to refer to the biometric enrollment data for a user. The template must be matched (within a specified tolerance) by samples taken from the user that needs to be authenticated.

The term Biometric Identification Record (BIR) refers to any biometric data that is returned to the application, including raw data, intermediate data, processed samples ready for verification or identification, as well as enrollment data. Typically, the only data stored persistently by the application is the BIR generated for enrollment (i.e., the template).

There are three principal high-level abstraction functions in the API:

1. Enroll: Samples are captured from a device, processed into a usable form from which a template is constructed, and returned to the application.
2. Verify: One or more samples are captured, processed into a usable form, and then matched against the claimed identity template. The results of the comparison are returned.
3. Identify: One or more samples are captured, processed into a usable form, and matched against a set of templates. A list is returned showing how close the samples compare against the top candidates in the set.

The high-level abstraction constitutes a very simple means of developing stand-alone biometric applications based on BioAPI. However, the processing of the biometric data, from the capture of raw samples up to the matching against a template, may be accomplished in many stages. The API has been defined to allow the biometric developer the maximum freedom in the placement of the processing involved, and allows the processing to be shared between a client machine (which has the biometric device attached), and a server machine. For this purpose, there are four low-level primitive functions in the API which, when used in sequence, can accomplish the same results as the high-level abstractions, and allow the development of client/server implementations.

- Capture: Capture is always executed on the client machine. One or more samples are acquired (either for Enrollment, Verification or Identification). The Capture function is allowed to perform as much processing on the samples as it sees fit, and may, in fact, for verification or identification, complete the construction of the BIR. If processing is incomplete, Capture returns an “intermediate” BIR, indicating that the Process function needs to be called. If processing is complete, Capture returns a “processed” BIR; indicating that the Process function does not need to be called. The application specifies the purpose for which the samples are intended, giving the BSP the...
opportunity to do special processing. This purpose is recorded in the header of the constructed BIR.

- Process: The “processing algorithms” must be available on the server, but may also be available on the client. The Process function is intended to provide the processing of samples necessary for the purpose of verification or identification (not enrollment). It always takes an “intermediate” BIR as input, and may complete the processing of the biometric data into “final” form suitable for its intended purpose. On the client, if it completes the processing, it returns a “processed” BIR; otherwise it returns an “intermediate” BIR, indicating that Process needs to be called on the Server. The server will always complete processing, and always return a “processed” BIR. The application can always choose to defer processing to the server machine, but may try to save bandwidth (and server horsepower) by calling Process on the client.

- Match: Performs the actual comparison between the “processed” BIR and one template (VerifyMatch), or between the “processed” BIR and a set of templates (IdentifyMatch). The support for IdentifyMatch is optional, but the supported Match functions are always available on the server, and may be available on the client.

- CreateTemplate: CreateTemplate is provided to perform the processing of samples for the construction of an enrollment template. CreateTemplate always takes an “intermediate” BIR as input, and constructs a template (i.e., a “processed” BIR with the recorded purpose of either “enroll_verify” and/or “enroll_identify”). Optionally, CreateTemplate can take an old template and create a new template, which is the adaptation of the old template using the new biometric samples in the “intermediate” BIR.

Fig. 1 shows the BioAPI model, and highlights the differences between the high-level abstractions and the low-level primitives.

The low-level BioAPI primitives provide the developer with flexibility for distributing the biometric processing between a client machine and a server. Fig. 2 shows the diagram of a client/server architecture for biometric verification based on BioAPI primitives.

The BioAPI model, however, has a number of shortcomings in terms of distributed behaviour. For one, the developer has to decide which middleware and flow control system to implement for communication between the client and the server. Accordingly, there are a number of initiatives underway that are attempting to resolve this problem. Specifically, the Biometric Interworking Protocol (BIP) standard (ISO/IEC 24708) is being designed to extend BioAPI 2.0 to enable an application running on a particular machine to use a biometric module running on a different machine as though it were running locally. Unfortunately, the BIP standard is not available yet and BioAPI 2.0 is not sufficiently supported as things currently stand.

3. A multiplatform Java Native Interface wrapper for the BioAPI framework

There are several reasons for trying to port the BioAPI framework to Java:

- Java is a simple high-level object-oriented language designed primarily on the basis of software engineering principles. It also provides a simpler means of developing maintainable software applications than do more powerful yet more complex C/C+ languages.
- As has already been mentioned, the BioAPI is an operating system dependent interface, which means that the development of multiplatform BioAPI-compliant applications is not straightforward.
- Porting the BioAPI to a Java environment would facilitate the development of web-oriented biometric applications.

![Fig. 2. The BioAPI model in a client/server environment: Use of low-level primitives.](image-url)
because of the numerous Java-based web-oriented technologies that exist.

In view of the above, a pure Java instantiation of the BioAPI is desirable, and it would have the additional advantage of facilitating the extension of biometric applications to mobile devices.

As mentioned briefly in Section 1, there is an INCITS proposal underway for the standardization of a BioAPI Java specification. However, this will not offer a solution in the short term. By September 2007, the progress report [8] said that the Reference Implementation was 95% completed. Moreover, the problem of interoperability between the new pure Java BioAPI framework and the existing C/C+ implementations of BSPs will not be solved a priori. Accordingly, we present a necessary functional solution that is immediately available.

3.1. Existing Java Native Interface wrappers for the BioAPI framework

In order to access the BioAPI framework in C/C+ from within the Java environment, a Java wrapper based on the Java Native Interface technology can be used.

Java Native Interface (JNI) is a standard programming interface for writing Java native methods and embedding the Java virtual machine into native applications. The primary goal is binary compatibility of native method libraries across all Java virtual machine implementations on a given platform.

To the best of our knowledge, there are two different Java Native Interface wrappers currently available for the BioAPI framework: the JNI BioAPI wrapper for Windows developed by Gens Software and JBioAPI, an open-source JNI BioAPI wrapper for Linux/Unix. Both of these wrappers are based on the BioAPI 1.1 specification.

3.2. A multiplatform wrapper based on the open-source JBioAPI project

The main problem with the existing original JNI BioAPI wrappers for Windows and Linux/Unix is that they do not have the same interface, and this complicates the development of multiplatform BioAPI-compliant applications. Moreover, both of the original wrapper interfaces allow access only to the high-level abstractions of the BioAPI model presented in Section 1: enroll, verify and identify tasks, and, as already explained in the same section, it is necessary to have access to the low-level primitives (capture, process, createTemplate and match) in order to achieve distributed behaviour at the application level.

We have developed an enhanced version of the JBioAPI project by generating a BioAPI Java wrapper for Windows that uses the same interface as Linux/Unix. Furthermore, we have modified the interface in order to permit access to the low-level primitives and thereby enable the application to support distributed behaviour. This extends JBioAPI and allows the development of Java applications that are compatible with both Windows and Linux/Unix platforms.

We used the source code of the BioAPI 1.1 Reference Implementation for Windows [2] to develop the extension of the JBioAPI wrapper for Windows. This source code is available as a Visual Studio 6.0 project. For the generation of the JNI code we employed the SWIG (Simplified Wrapper and Interface Generator) tool [9].

Both JBioAPI and our JBioAPI extension with Windows support are licensed by the GNU General Public License and are publicly available on the Internet as open-source projects [6, 10].

4. Uses of the extended Java JBioAPI wrapper

Our extension of the Java JBioAPI wrapper is designed to facilitate the development of multiplatform web-oriented biometric applications.

It includes the Java classes needed by Java developers to access the underlying BioAPI framework using Java code, and a dynamic linking library that executes the native calls to the BioAPI installed on the system.

The native library is obviously operating system dependent. Compiled versions for the Win32 platform (jbioapi.dll), and Linux Debian (libjbioapi.so) are provided. Since the source code is included, the compilation for other Unix/POSIX operating systems such as Solaris is straightforward.

The Java classes, however, are not operating system dependent. They are divided into two different packages: one which includes the classes with the low-level JNI calls to the native library, and another which includes the classes that implement the high-level interface the Java developer will use. These high-level classes are the following:

- **BiometricsFramework**: Offers a way to read/write a BIR from/to the hard disk and to select a BSP using the corresponding BSP and device identifiers (uuid and deviceID).
- **BiometricServiceProvider**: Allows access to high-level abstractions (enroll, verify and identify methods) and biometric primitives (capture, createTemplate, process, verifyMatch and identifyMatch methods).
- **BiometricLoginModule**: Designed for use within the Java Authentication and Authorization Service (JAAS).

The use of the extended Java JBioAPI wrapper opens multiple possibilities for porting the BioAPI to a web environment. Several scenarios are possible, including client–server configurations for biometric authentication.

- **Calling the client-side BioAPI**: This means that if the client application is a Java Applet, a Java Application, or a Java Web Start Application, we can perform calls to the BioAPI Framework through the JBioAPI wrapper. Of these three possibilities, we recommend the use of Java Web Start because it allows the JBioAPI wrapper to be embedded inside the application (meaning that it does not need to be installed in the operating system).
- **Calling the server-side BioAPI**: With this technique, BioAPI support can be offered to J2EE web applications.
J2EE (Java Platform, Enterprise Edition) is the industry standard for developing portable, robust, scalable and secure server-side Java applications. This includes several web technologies, such as Servlets, Enterprise Java Beans and Web Services.

There are several good reasons why biometric processing and matching should take place on the server-side.

1. The algorithms will execute in a more secure environment, making tampering or reverse-engineering more difficult.
2. The client machine may not have sufficient power to run complex algorithms. Network distributed biometrics have the potential of extending biometric applications to mobile devices, which usually have very limited resources.
3. Upgrading of biometric algorithms for state-of-the-art performance in verification can be done by replacing the software on the server-side in a way that is transparent to the user.
4. Identification over large populations can only reasonably be done on a server.

4.1. An example of use in a web environment

In order to show the utility of the JBioAPI wrapper we present our application for server-side biometric authentication in a web environment [11].

In our system, the biometric client is a Java Web Start application, and the server has a J2EE Web Application deployed in an open-source JBoss server, using a MySQL database. Both applications use the extended JBioAPI wrapper, described previously, for managing the BioAPI calls. All these technologies allow the overall system to be implemented in a multiplatform environment.

Obviously, the BioAPI framework must be installed on both the authentication server and the client machine. Currently, our system supports operating without a BioAPI installed on the client side. In this case, the client application is responsible for performing the capture process ad-hoc, outside of the BioAPI framework, and building a BIR compatible with the server-side BSP responsible for performing the verification.

The overall diagram is presented in Fig. 3. Sample acquisition is executed locally and the enrollment and verification tasks are executed as remote calls to a Web Service. We have used Web Services technology as middleware for improving the interoperability of our framework.

Our system performs multimodal biometric verification by integrating our own implementations of BSPs for face-based [12] and voice-based [13] user verification.

We have tested the system with machines under Windows XP and Linux Debian OS.

5. Relevance to standards

The utility of the developed JBioAPI library consist of allowing to build multiplatform (Windows/Linux/Unix) Java applications able to interoperate with a BSP written in C/C+ for BioAPI 1.2 framework. The same methodology followed to create this library can be easily applied for the recently released BioAPI 2.0. We have focused our work on extending the Michael.R.Crusoe’s open-source project JBioAPI 1.2 (a Java BioAPI wrapper for Linux/Unix), from which we are now also members [10], by including Windows support and allowing access to the low-level primitives of the BioAPI model in order to achieve distributed behaviour at the application level. This fact allows us to use, in a client–server mode, BSPs that originally were developed for local use.

There is a current proposal, led by Purdue University and OSS Nokalva [14], for the standardization of a BioAPI Java specification that is semantically equivalent to the existing C API specified in ISO/IEC 19784-1 and amended by ISO/IEC...
19784-1 Amd. 1. The initial public review of the Reference Implementation is still not ready. Furthermore, the proposal is focused on the development of a pure Java instantiation of the BioAPI, and will not specify the use of existing BioAPI C/C++ modules with a Java BioAPI application (although it will not prevent such uses).

Accordingly, the absence of reference implementations for the proposed new standards further validates our approach as it offers an immediate and adaptable solution that will help to develop multiplatform web-oriented biometric applications.

6. Conclusions

We have presented a multiplatform Java Native Interface wrapper for the BioAPI framework. Our work successfully addresses the lack of compatibility between the different Java wrappers that exist for Windows and Linux/Unix. Moreover, we have demonstrated the utility of this unified wrapper by developing a multiplatform Java-based application for biometric authentication in a web environment.

An example of real-life application which makes use of JBioAPI for accessing the underlying BioAPI v1.2 framework, is BioWebAuth, an open-source Java framework that provides single sign-on web authentication based on BioAPI-compliant biometric software or devices. It can be found at SourceForge under LGPL license.

Acknowledgments

This work has been partially funded by the Spanish Ministry of Education and Culture (project PRESA TEC2005-07212), by the Xunta de Galicia (project PGIDIT05PXIC32202PM), and by the European Network of Excellence BioSecure.

References


Elisardo González-Agulla received the telecommunications engineering degree from the Universidad de Vigo, Spain, in 2002, where he is currently pursuing a Ph.D. degree in the field of biometrics for web applications. His research interests include biometric standards and computer vision.

Enrique Otero-Muras received the telecommunications engineering degree from the Universidad de Vigo, Spain, in 2005, where he is currently pursuing a Ph.D. degree in the field of biometrics for web applications. His research interests include biometric standards.

Carmen García-Mateo received the M.Sc. and Ph.D. degrees (Hons.) in telecommunications engineering from the Universidad Politécnica de Madrid, Spain, in 1987 and 1993 respectively. Her research interests include speech, and speaker recognition, dialogue systems and biometric applications. She has been the leader of a number of R&D projects and published papers on these topics. She is a Full Professor in the field of Discrete Signal processing at the Universidad de Vigo.

José Luis Alba-Castro received the M.Sc. and Ph.D. degrees (Hons.) in telecommunications engineering from the Universidad de Santiago, Spain, in 1990 and Universidad de Vigo, Spain, in 1997. His research interests include computer vision, statistical pattern recognition, automatic speech, and speaker recognition and image-based biometrics. He has been the leader of several R&D projects and published papers on these topics. He is an Associate Professor of Discrete Signal processing, Pattern Recognition, Image Processing and Biometrics at the Universidad de Vigo.