UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA

PROYECTO FIN DE CARRERA

EVALUACIÓN DE CONFORMIDAD CON ISO/IEC 19784-1 PARA ALGORITMOS DE IDENTIFICACIÓN BIOMÉTRICA

INGENIERÍA TÉCNICA INDUSTRIAL
(ELECTRÓNICA INDUSTRIAL)

AUTOR: José Campo Buzón

DIRECTOR: Raúl Alonso Moreno
TUTOR: Raúl Sánchez Reillo

Febrero de 2009

PRÓLOGO

El texto que a continuación se extiende es el resultado de la práctica más individual y trabajosa que he realizado a lo largo de mis años en la universidad. La labor llevada a cabo cierra una etapa de formación en la que he desarrollado (y espero demostrarlo con el trabajo que aquí presento) algunas de las habilidades necesarias para enfrentarme al trabajo que hoy en día puede hacer un ingeniero de mis características en la vida laboral. Su realización no ha estado exenta de complicaciones y problemas, lo cual tras el momento inicial de preocupación sirve para curtirte en las distintas dificultades que pueden surgir en cualquier momento de tu labor siendo o no ajenas a ti. Su repercusión en tu trabajo te lleva a saber salir de ellos como sea, y te hace mejorar. En toda esta labor ha sido fundamental el apoyo de la gente de mi entorno, familia y amigos, y desde luego el departamento de Tecnología electrónica de la universidad Carlos III, en especial el director de mi proyecto, Raúl Alonso.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

CAPITULO 2

CAPITULO 3

CAPITULO 4

CAPITULO 5

CAPITULO 6

GLOSARIO DE ABREVIACIONES

API – Application Programming Interface

BCS – BioAPI Conformity Statement

BDB – Biometric Data Block

BFP – BioAPI Function Provider

BIR – Biometric Information Record

BSP – Biometric Service Provider

CBEFF – Common Biometric Exchange

CTS – conformance Test suite

FPI – Function Provider Interface

GUI – Graphical User Interface

ID – Identity/Identification/Identifier

IUT – Implementation Under Test

PID – Product ID

SB – Security Block

SBH – Standard Biometric Header

SPI – Service Provider Interface

UUID – Universally Unique Identifier

CAPÍTULO 1

Introducción

El presente capítulo hace a continuación un esfuerzo por relatar brevemente el trabajo que se desarrolla en el proyecto.

Se trata de enmarcar el ámbito en el cual su aplicación es de relevancia y se relatan los procedimientos y herramientas utilizadas para alcanzar los objetivos que también son explicados, intentando poner al lector en situación, para seguir de la manera adecuada el proceso de nuestro desarrollo.

Por último, se ofrece una relación estructurada de la distribución que conforma el documento

1.1 ÁMBITO DEL PROYECTO

El actual texto desarrolla el proyecto: "**Evaluación de conformidad con ISO/IEC 19784-1 para algoritmos de identificación biométrica**". El departamento de Tecnología Electrónica de la Universidad Carlos III de Madrid ha sido el lugar de su concepción y desarrollo. La tutoría del mismo le corresponde al Dr. Raúl Sánchez Reillo. La dirección a D. Raúl Alonso Moreno. Y la realización a D. José Campo Buzón.

Como toda tecnología que busca una implantación segura, fiable, y eficaz, la biometría necesita de la estandarización. Esa estandarización ha de tener a su vez maneras seguras, fiables, y eficaces de probarse. Esta será la tarea que podemos proclamar principal objetivo del proyecto.

Numerosos proyectos del ámbito de la Biometría son llevados a cabo en el departamento de Tecnología Electrónica, de modo que la realización del presente trabajo se ha apoyado en herramientas en las que otras personas ya han estado o están trabajando. La aplicación desarrollada que se explica en este documento se ha probado con los BSP generados por el Grupo Universitario de Tecnologías de la Identificación (GUTI), sobre modalidades biométricas basadas en Iris y Huella, comprobando que se cumple la norma ISO/IEC 19784-1 que regula la especificación Biometric Application Programming Interface (en adelante BioAPI) en su versión 2.0

1.2 OBJETIVOS

El objetivo del proyecto es desarrollar una aplicación con la que, dado un algoritmo de identificación biométrica, se evalúe si cumple con el API biométrico definido en ISO/IEC 19784-1. Ese algoritmo de identificación biométrica en este caso serán dos, que corresponden a los BSP mencionados en el apartado anterior. Los BSP están basados en: 1) reconocimiento de la huella dactilar 2) reconocimiento de iris ocular. Dado el incremento actual de este tipo de técnicas, es cada vez más necesario poder comprobar de un modo rápido y fiable que cumplen con los estándares que internacionalmente se estipulan. El cumplimiento de dicha norma garantiza una seguridad y facilidad de adaptación de aplicaciones distintas y en lugares distintos. La mayor parte del trabajo, pues, se ha centrado en el desarrollo del algoritmo que consigue comunicarse con los BSP y chequea en que medida cumplen con las normas de conformidad.

1.3 HERRAMIENTAS

La idea básica que se debe tener en la cabeza para seguir el desarrollo del proyecto es la de una aplicación informática que se implementa de modo que el usuario de la misma pueda comprobar con ella el grado de conformidad de su BSP con la norma.

El estándar de modelos de pruebas de conformidad esta declarado de modo amplio y conciso en la norma ISO/IEC 24709-2 (Conformance Testing for BioAPI-Test Assertions for Biometric Service Providers) la cual debemos de citar como primera herramienta para la consecución del proyecto ya que por decirlo así, ahí están dictadas las reglas de cómo llevar a cabo estas pruebas de conformidad de que hablamos. Como se explicará mas adelante el esquema básico de funcionamiento de las aplicaciones biométricas que cumplen con el estándar BioAPI es:

Aplicación <-->Framework <--> BSP

Donde una determinada aplicación biométrica se comunica con un proveedor de servicios biométricos (BSP) a través de una interfaz definida denominada Framework de BioAPI que proporcionan un marco común de comunicación.

Aunque la aplicación que se ha desarrollado actúa en gran medida como aplicación y Framework (por los motivos que se explicarán más adelante) a los ojos de los BSP, para llevar a cabo el desarrollo se ha utilizado un Framework que interactúa entre una aplicación biométrica y los BSP. Estos últimos que han sido proporcionados por el departamento de Tecnología Electrónica son evidentemente también una herramienta pues sobre ellos se prueba que la aplicación funciona correctamente, y a la vez pasan a formar parte de ella.

Por último citamos dos herramientas que son: el programa soporte en que hacemos el trabajo que es el Borland Developer Studio 2006 el cual nos permite trabajar en C++ realizando una programación orientada a objetos. Y un pequeño programa ya desarrollado por Raúl Alonso Moreno cuya función es cargar Los BSP en el sistema en que estemos trabajando y cuya inicial mecánica de funcionamiento ha sido un apoyo sobre el que se ha levantado el diseño de la aplicación de este proyecto.

A continuación se expone una breve reseña de cada una de las herramientas:

Norma ISO/IEC 19784-1: Estándar Internacional que proporciona un modelo genérico biométrico de alto nivel de autenticación que satisface a la mayor parte de formas de tecnología biométrica.

Framework de BioAPI: implementado en Visual C++, Este Framework es el único disponible en estos momentos bajo la especificación de BioAPI 2.O y está desarrollado por la división BioFoundry®, perteneciente a la empresa OSS Nokalva. Se encarga de interactuar entre la aplicación y el proveedor de servicios biométricos (BSP). Será de utilidad para el desarrollo de la aplicación pero será sustituido en gran medida por la propia aplicación para la realización de las pruebas de conformidad

BSP: Proveedor de servicios biométricos desarrollado en Builder 2006, que será implementado bajo Test (IUT). Proporcionados por el departamento de Tecnología electrónica de la Universidad Carlos III de Madrid.

Borlan Developer Studio 2006: desarrollador de la compañía Borland que utilizaremos para crear la aplicación programada en C++, basándonos en una programación orientada a objetos (P.O.O)

1.4 ESTRUCTURA DE LA MEMORIA

- Capítulo 1: Introducción y planteamiento de los objetivos.

- Capítulo 2: Sistemas de identificación biométrica. Definición y técnicas. Breve descripción de los 2 sistemas que han sido sometidos a las pruebas de conformidad

- Capítulo 3: Estado actual y evolución de la estandarización, así como su aplicación a la biometría.

- Capítulo 4: BioAPI, introducción a la norma y se explicación de su arquitectura.

- Capítulo 5: Test de conformidad. Pasos a seguir para la construcción de un BSP Testing application.

- Capítulo 6: Pruebas y desarrollo. Interacción entre el BSP Testing application y el BSP, resultados obtenidos y explicación del funcionamiento del programa. Exposición de la interfaz.

- Capítulo 7: Conclusiones y líneas futuras de investigación.

- Bibliografía de referencia del proyecto y costes que éste ha tenido.

CAPÍTULO 2

Sistemas de Identificación Biométrica

Dado que el objetivo del proyecto es desarrollar una aplicación con la que, tomado un algoritmo de identificación biométrica, se evalúe si cumple con el API biométrico definido en ISO/IEC 19784-1 parece conveniente entender antes que es un sistema de identificación biométrica y en primer lugar, definir el concepto de biometría. El contexto y las aplicaciones de esta tecnología serán desgranadas a continuación en el grado que se considera necesario.

2.1 INTRODUCCIÓN

El avance de las nuevas tecnologías de la información y comunicación junto con los progresos fundamentalmente de la electrónica digital ha propiciado que la autenticación y verificación basadas en características físicas (existen desde siempre y sin darnos cuenta son las que más utilizamos) puedan ser llevadas a cabo de una manera rápida, eficaz y segura; lo que en el mundo de hoy en día (compras por Internet, operaciones de todo tipo a distancia, múltiples controles de seguridad) hace de estos sistemas una herramienta cada vez mas demandada y con un futuro alentador.

Algunos ejemplos de aplicaciones ya en uso y de entidades o compañías que lo utilizan son por ejemplo el Bank of America (donde ya se han implementado sistemas de reconocimiento por huella y del iris para hacer frente a las grandes pérdidas debidas en parte a la poca seguridad que presentan los sistemas utilizados hasta ahora), Pasaportes y DNI´s electrónicos para pasar los controles de aduanas, Toshiba (pionero en sacar al mercado portátiles con reconocimiento de huella dactilar)

2.2 BIOMETRÍA

Según la Real Academia de la Lengua Española la definición de biometría es: "Estudio mensurativo o estadístico de los fenómenos o procesos biológicos". Pero para la definición de esta palabra en el ámbito técnico que nos ocupa se considera más exacto o completo decir que "es una tecnología de seguridad basada en el reconocimiento de una **característica física e intransferible** de las personas". Algunas de estas características, que denominaremos biométricas pueden ser encontradas en los seres humanos en lugares tales como: Iris, geometría de la mano, poros de la piel, voz...

A continuación se expone un cuadro con las principales características biométricas [API07]:

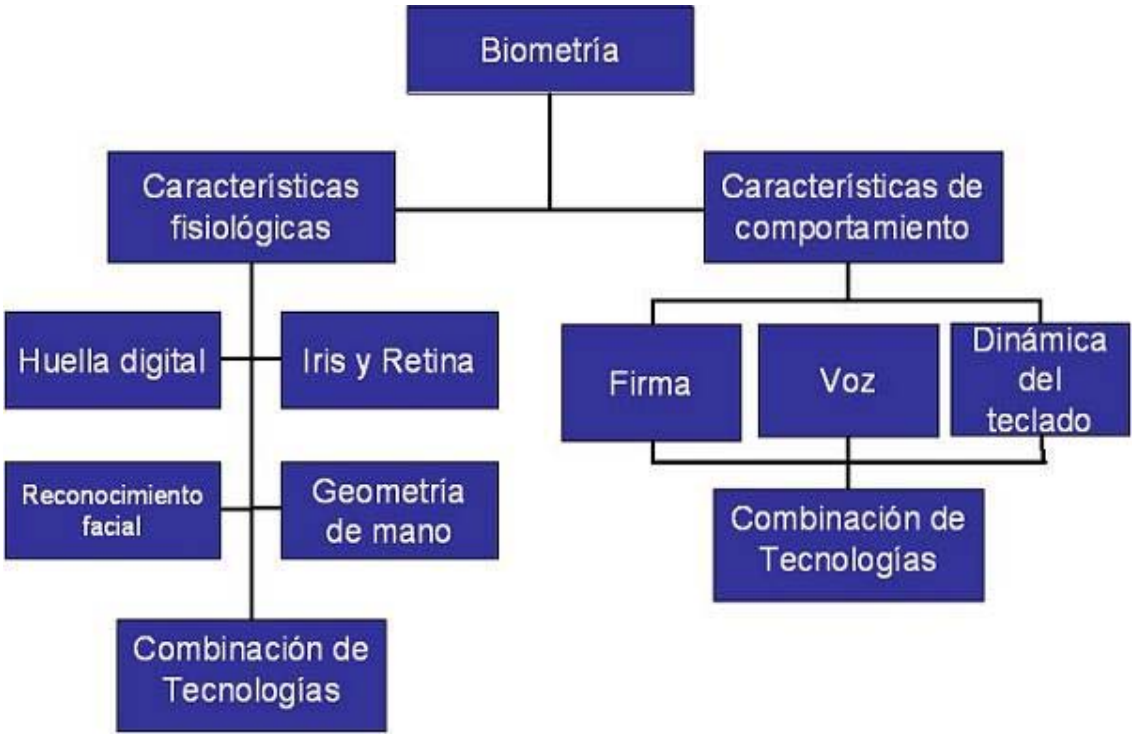


Figura 2.1)
Esquema de características biométricas

Los sistemas de lectura biométricos incluyen un sistema de captación y un software biométrico que interpreta la muestra física y la transforma en una secuencia numérica. Cuando posteriormente se necesite reconocer a una persona se recurre a esa secuencia numérica que se ha guardado, en una primera fase de registro, y se compara con la que el usuario proporciona en la fase de verificación.

2.3 ORIGENES

Si bien hoy en día existen varios sistemas biométricos automatizados, las técnicas que utilizan están basadas en conceptos que fueron concebidos hace cientos, quizá miles, de años atrás.

Uno de los ejemplos más básicos de una característica física que es utilizada para reconocimiento de una persona es el rostro. Desde épocas muy remotas, los humanos han utilizado el rostro de sus semejantes para diferenciar individuos familiares o que eran conocidos de los desconocidos. Muchas son las referencias de personas, que en la antigüedad, han sido identificados por diversas características físicas y morfológicas como cicatrices, medidas, color de los ojos, tamaño de la dentadura, etc.

Se han descubierto antiguas cuevas con pinturas que se supone fueron realizadas por hombres prehistóricos que vivieron en ellas. Alrededor de dichas pinturas se hallaron numerosas impresiones de manos que actúan como las firmas que identifican a sus creadores. Se han encontrado evidencias de que alrededor de 500 AC las transacciones comerciales entre los babilonios eran registradas en unas pastillas de arcilla que incluían la impresión de la huella digital.

Joao de Barros, un explorador y escritor español, escribió que los primeros mercaderes chinos incluían la impresión de las huellas digitales en sus transacciones de negocios. Asimismo, los padres chinos utilizaban las impresiones de dedos y pies para diferenciar a sus hijos de otros. En los comienzos de la historia egipcia, los comerciantes eran identificados por sus características físicas para diferenciar entre los comerciantes de confianza y conocida reputación y transacciones exitosas previas, de aquellos que recién se iniciaban en los negocios.

A mediados de 1800, con el gran crecimiento de las ciudades y la población surgen nuevas técnicas de identificación demandadas por la justicia, la cual, en ese momento, pretendía imponer castigos más severos a los infractores reincidentes y más leves a aquellos que infringían la ley por primera vez. Esto requería de un sistema que pudiera medir y registrar distintos rasgos que identificaran a los infractores. El primero de los dos métodos utilizados fue el sistema Bertillon el cual se basaba en la medición de varios parámetros físicos. Dichos parámetros eran volcados en unas tarjetas que luego se ordenaban según la altura, el largo de los brazos y algunos otros parámetros. El otro método fue el uso de las huellas digitales por parte de los departamentos de policía.

Si bien estos métodos surgieron en Sudamérica y en Asia, a fines de 1800, Edward Henry, inspector general de la policía de Bengala, India, ideó un método para registrar las huellas digitales que ofrecía la posibilidad de recuperar un registro de la forma en que lo hacía el método Bertillon pero basado en un rasgo físico mucho más individual y personal: las huellas

digitales. Este método, llamado Sistema Henry y algunas variaciones del mismo es el que se utiliza hoy en día para clasificar las huellas digitales.

Los sistemas biométricos modernos comienzan a surgir en la segunda mitad del siglo XX junto con el desarrollo de los sistemas computarizados. En los años 90 se produce una gran explosión en este campo, creando tecnologías masivas, más económicas y al alcance de la mano de mayor cantidad de usuarios, lo que introduce a los sistemas biométricos en un sinfín de aplicaciones que utilizamos día a día.

Línea de tiempo

Fecha	Descripción
6000 A.C.	Almacenamiento de huellas digitales, usado por asirios, babilónicos, japoneses y chinos.
Siglo XIV	Chinos estampaban las huellas de manos y pies de los niños para identificarlos
1686	Malpighi identificó diferencias en los patrones de huellas digitales
1823	Purkine, identificó la naturaleza única de las huellas digitales
1858	Hershel crea el primer registro de huellas palmares de empleados
1870	Bertillon desarrolla el sistema de antropometría descriptiva
28-oct-1880	Faulds publica el artículo "On the Skin-Furrows of the Hand"
1882 -1890	La policía de Francia utiliza la técnica desarrollada por Bertillon
1-sep-1891	Se empieza a utilizar el método de Juan Vucetich en Argentina
1892	Francis Galton publica el libro "Finger Prints"
1892	Se identifica por primera vez por la huella digital a una asesina
1896	La policía de Bengal implementa el sistema de huella digital
1900	Scotland Yard adopta el sistema de huellas digitales de Henry
1902	Denmark Hill en el Reino Unido es conectado con la escena del crimen
1903	El departamento de policía de New York empieza los archivos de huellas digitales

1903	Colapsa el sistema Bertillon
1905-1908	Se implementa el sistema de huellas digitales en las Fuerzas Militares de EEUU
1918	Locard establece 12 detalles Galton como mínimo para identificación positiva de una persona.
Sep-1935	Se publica el artículo "A new Scientific Method of Identification"
1936	Burch propone el concepto de los patrones de iris para reconocimiento
1955	Se publica el artículo "The fundus Oculi in monozigotic twins: Report of six pairs of identical twins"
1960	Publicación modelo de los componentes fisiológicos de la producción del discurso acústico.
9-mar-1963	Publicación articulo "automatic Comparison of Finger-Ridge Patterns"

2.4 RASGOS Y TÉCNICAS DE LA BIOMETRÍA

Las características biométricas de las personas serán aptas para su uso como patrón para la identificación biométrica siempre que satisfagan los siguientes requisitos [FER06]: *Universalidad* (el rasgo biométrico debe estar presente en todos los individuos), *Singularidad* (debe asegurar su precisión para discriminar entre dos individuos), *Estabilidad* (no puede cambiar con el tiempo o las condiciones ambientales), *Cuantificabilidad* (se debe poder medir en términos cuantitativos para poder procesar la información capturada),*Aceptabilidad* (los usuarios del programa deben colaborar en su funcionamiento), *Rendimiento* (se requiere un nivel de exactitud muy alto para que las identificaciones sean fiables y eficientes), *Resistencia a fraudes* (establecimiento de un nivel al cual el sistema es capaz de resistir ante técnicas fraudulentas)

2.5 SISTEMAS DE IDENTIFICACIÓN

Cada sistema biométrico utiliza una cierta clase de interfaz, un sensor o mecanismo de captura determinado y un software específico. La identificación biométrica experimenta una aceptación creciente debido a la reducción de los

costos de los dispositivos y a su alta confiabilidad. Por ello, no se restringe su uso a aplicaciones de alta seguridad, como bancos e instalaciones gubernamentales, sino que también se extiende a las empresas, para el control de clientes y empleados y en el acceso a oficinas y plantas comerciales e industriales. Aunque la lista sería interminable, algunas de las aplicaciones de la identificación mediante sistemas biométricos de distinto tipo serían los servicios públicos, servicios policiales, penitenciarios, instituciones de salud, permisos de conducir, inmigración, registro de armas, controles de acceso, tiempo y asistencia, seguridad de redes informáticas, comercio electrónico, educación, etc.

En este caso los BSP que el actual proyecto evaluara con respecto a la norma se corresponden con los siguientes sistemas biométricos

2.5.1. Identificación por huellas dactilares [DIS08]

Son las formas caprichosas que adopta la piel que cubre las yemas de los dedos. Están constituidas por rugosidades que forman salientes y depresiones.

Las salientes se denominan crestas papilares y las depresiones surcos interpapilares. En las crestas se encuentran las glándulas sudoríparas. El sudor que éstas producen contiene aceite, que se retiene en los surcos de la huella, de tal manera que cuando el dedo hace contacto con una superficie, queda un residuo de ésta, lo cual produce un facsímil o negativo de la huella.

Son únicas e irrepetibles aún en gemelos idénticos, debido a que su diseño no está determinado estrictamente por el código genético, sino por pequeñas variables en las concentraciones del factor del crecimiento y en las hormonas localizadas dentro de los tejidos. Cabe señalar que en un mismo individuo la huella de cada uno de sus dedos es diferente.

Clasificación

Los patrones de huellas digitales están divididos en 4 tipos principales, todos ellos matemáticamente detectables. Esta clasificación es útil al momento de la verificación en la identificación electrónica, ya que el sistema sólo busca en la base de datos del grupo correspondiente.

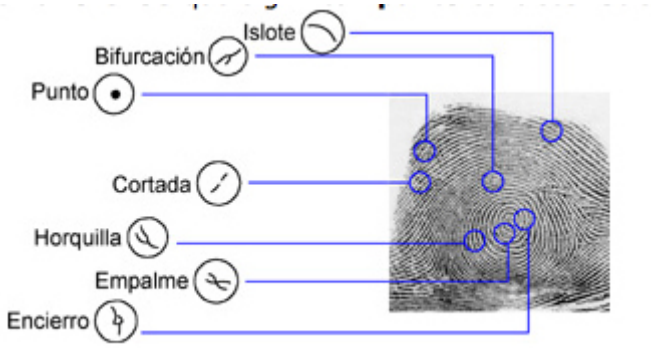


Figura 2.2)*Esquema de la huella dactilar*

En la figura aparecen 8 puntos característicos que hay en un dedo, éstos se repiten indistintamente para formar entre 60 y 120 (por ejemplo 10 horquillas 12 empalmes 15 islotes, etc.). A estos puntos también se llaman minutas, o minucias, término utilizado en la medicina forense que significa "punto característico".

Procedimiento

Con este conjunto de puntos, el software biométrico de huella digital genera un modelo en dos dimensiones, según se muestra en el ejemplo, mismo que se almacena en una base de datos, con la debida referencia a la persona que ha sido objeto del estudio.

Para ello, la ubicación de cada punto característico o minucia se representa mediante una combinación de números (x.y) dentro de un plano cartesiano, los cuales sirven como base para crear un conjunto de vectores que se obtienen al unir las minucias entre sí mediante rectas cuyo ángulo y dirección generan el trazo de un prisma de configuración única e irrepetible.

Para llevar a cabo el proceso inverso o verificación dactilar, se utilizan estos mismos vectores, no las imágenes capturadas por el sensor.

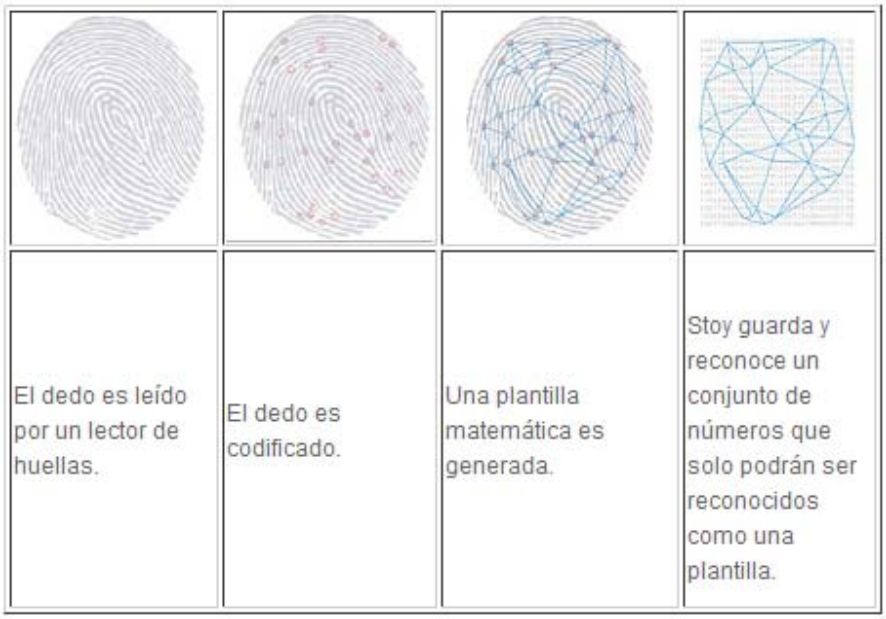


Figura 2.3)

Ejemplo del proceso de captura de una huella con un algoritmo comercial llamado stoy

Dispositivo para identificación

El Sistema de Identificación Automatizada de Huellas Dactilares, tiene un índice de seguridad del 99.9% ya que verifica la identidad de una persona, basada en las características de sus huellas digitales.

Para tratar los datos de la huella se utiliza un algoritmo que permite asociar la huella que se desea identificar, con otras de similares características, almacenadas en la base de datos.

Existen dos maneras distintas usar los datos de una huella o de otros sistemas biométricos.

1. Verificación o autenticación ¿Es la persona quien dice ser?

Dependiendo del sistema implementado se suele pedir un código, una lectura de tarjeta y comparar esa huella almacenada con la huella puesta en el lector. La verificación es un proceso un poco más molesto por que se le pide una información o una acción adicional al usuario, (si bien es de los más aceptados por la población) pero como ventaja tiene que es más rápido y más seguro.

2. Identificación. ¿Quién es la persona?

En este proceso directamente se compara una huella capturada contra todas las que están almacenadas en el ordenador, es un proceso algo más lento, pero la interacción con el usuario es mínima.

2.5.2. Identificación por reconocimiento de iris [BAR08]

El reconocimiento de iris es el proceso de reconocer a una persona analizando el patrón del iris. El método automatizado de reconocimiento de iris es relativamente joven, existiendo en patente solamente desde 1994. El iris es un músculo dentro del ojo que regula el tamaño de la pupila, controlando la cantidad de luz que entra en el ojo. Es la porción coloreada del ojo basando su color en la cantidad del pigmento Melatonina dentro del músculo. Un estudio cercano del mismo muestra un conjunto de rasgos característicos, como pueden ser estrías, anillos, surcos, texturas, etc.

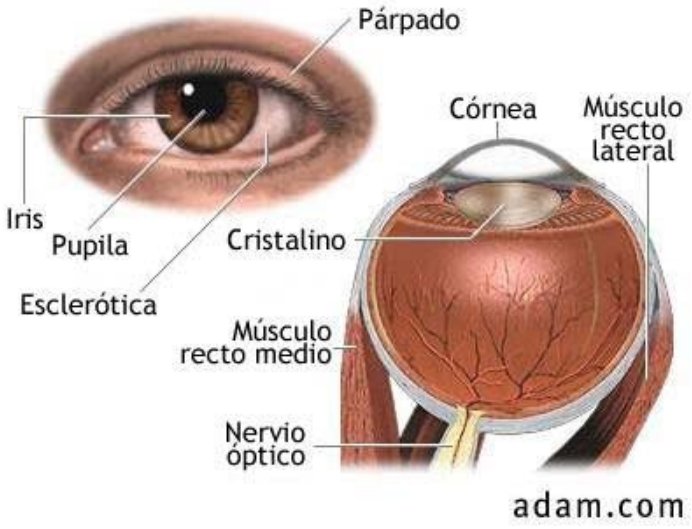


Figura 2.4) globo ocular

El sistema adquiere una imagen del iris y transforma las características anteriormente mencionadas en patrones numéricos, que se contrastan con los previamente almacenados. Concretamente, una cámara de reconocimiento de iris toma una fotografía del mismo. Las cámaras cumplen los estándares internacionales de iluminación segura, y utilizan un método de iluminación de longitud de onda cercana al infrarrojo que es escasamente visible y muy seguro. La imagen del ojo es primeramente procesada por un programa que localiza el iris. Luego, un programa codifica los patrones del ojo creando un código para la secuencia de texturas y rasgos característicos del iris.

Las principales características que le hacen tan útil para su uso en la biometría son:

- Esta protegido de agentes externos gracias a la cornea, la cual no solo protege sino que lo hace visible al exterior.

- El patrón del iris se mantiene sin cambios, es decir, es estable a lo largo del tiempo gracias precisamente a la protección que le confiere la cornea. Por lo tanto, el patrón que se almacenó inicialmente puede ser utilizado durante toda la vida.

- Presencia de pequeñas variaciones en su tamaño tanto con cambios en la luminosidad como cuando se ve sometido a una iluminación fija, lo cual permite que sea un sencillo mecanismo para la detección de "sujeto vivo". Esto hace que ofrezca gran dificultad a la hora de posibles falsificaciones.

- Los patrones del iris son más complejos y aleatorios que otros patrones biométricos, lo cual ofrece un método de alta precisión para la autenticación individual de cada uno de los usuarios con una tasa de error por falsa aceptación es inferior a uno sobre 1.2 millones.

- Su intento de falsificación implicaría la necesidad de realizar intervenciones quirúrgicas que podría dañar seriamente la capacidad de visión del sujeto.

- Sus patrones no están determinados genéticamente, por lo que incluso el ojo izquierdo y el derecho de un mismo individuo son diferentes. Incluso, según indican determinados estudios, ni los hermanos gemelos poseen patrones de iris similares.

2.6 FASES DE UN SISTEMA BIOMETRICO

Los pasos básicos en un sistema biométrico genérico son:

▪ **Registro**: captura de una o más muestras biométricas que se utilizan como identidad biométrica del ciudadano para posteriores procesos de identificación

▪ **Extracción de plantillas biométricas (procesado)**: extracción del modelo matemático que caracteriza al individuo, comúnmente denominado como "vector de características".

▪ **Almacenamiento** de la plantilla biométrica en un repositorio
Centralizado: base de datos central con información biométrica de los ciudadanos. Posibilita tanto identificación como la verificación de identidad pero depende de las comunicaciones
Distribuido: el propio documento de identificación. Posibilita sólo la verificación pero no depende de las comunicaciones

▪ **Verificación**: proceso de comparación de la platilla biométrica almacenada con la que se acaba de capturar

Para exponer de una forma mas detallada como lleva a cabo su trabajo un sistema de identificación biométrica, vamos a utilizar como ejemplo las etapas en la Verificación Dactilar de un sistema comercial

Figura 2.5)
Aparatos de recogida de una huella dactilar

Registro: Captura la imagen de la impresión dactilar mediante un lector en vivo con posibilidad o no de huella rodada. Existe la opción de captura sobre una impresión entintada usando un escáner plano

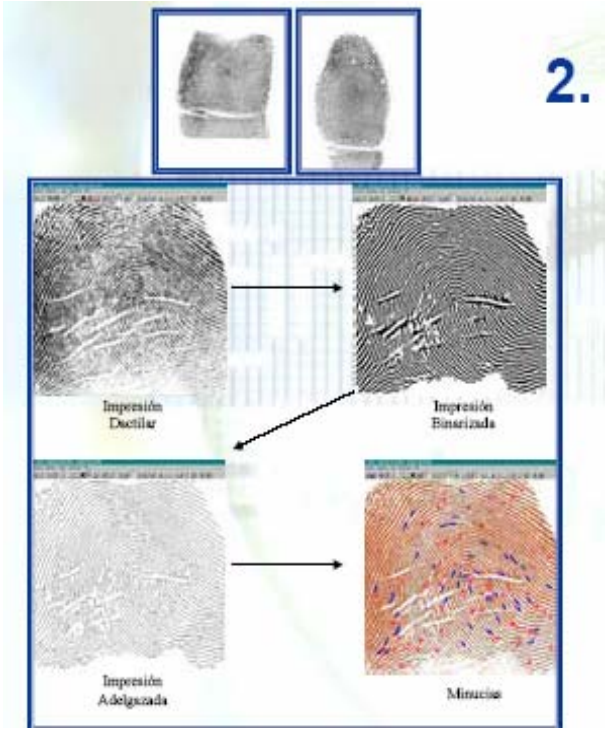


Figura 2.6) Procesado de huella

Procesado: extracción de puntos característicos o minucias (típico de 30 a 100 puntos). El procesado se componen de las siguientes etapas:

▪ Análisis de características básicas: direcciones de crestas, zonas de sobre/sub-presión, etc.

▪ Binarización

▪ Adelgazamiento

▪ Detección de puntos

(Detalle del procesado)

Figura 2.7) Detalle del procesado de una huella

(Detalle de las minucias)

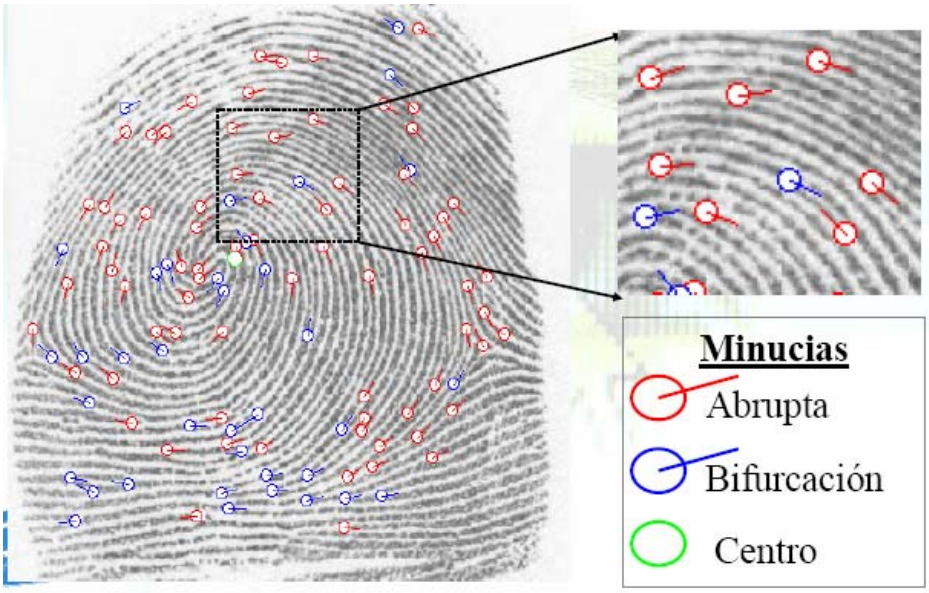


Figura 2.8) minucias

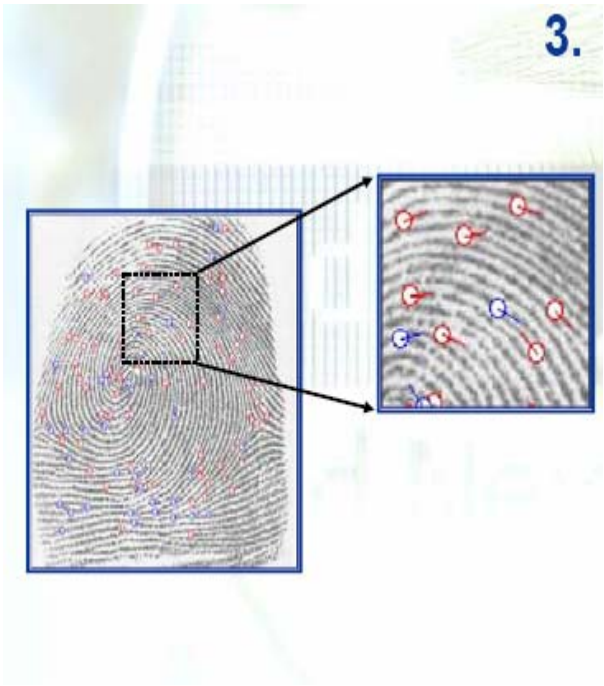


Figura 2.9)
Almacenamiento de una huella

Almacenamiento: baja necesidad de espacio de almacenamiento (típico < 500 Bytes). Típicamente se almacenan de 2 a 4 impresiones, aunque hay proyectos en los que se almacenan las 10 (visa Schengen)

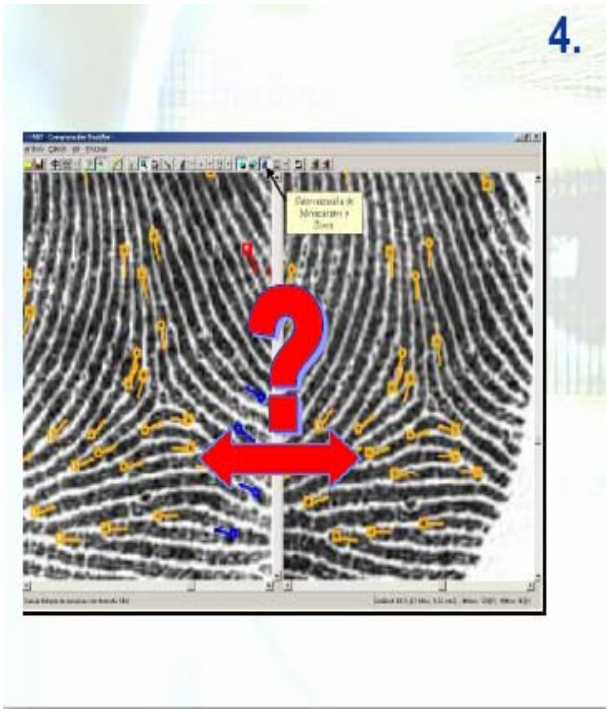


Figura 2.10) Verificación de una huella

Verificación: Procedimiento matemático por el cual se determinan similitudes y diferencias entre los puntos característicos de dos impresiones dactilares. Se tiene en cuenta giros, deformaciones elásticas, cicatrices, etc. Dependiendo de un umbral que se fije, el cotejo da un resultado positivo o negativo.

Problemas a resolver en la verificación:

- Aparición de Minucias Falsas.
- Desaparición de Minucias Verdaderas.
- Giro de la Impresión Dactilar.
- Diferente Área de Superposición.
- Presión Diferente.
- Etc...

Veamos también de forma superficial un ejemplo de reconocimiento por iris:

Antes de que ocurra el reconocimiento de iris, se localiza el iris usando características del punto de referencia. Estas características del punto de referencia y la forma distinta del iris permiten digitalización de la imagen, el aislamiento de la característica, y la extracción. La localización del iris es un paso importante en el reconocimiento del iris porque, si está hecho incorrectamente, el ruido resultante (pestañas, reflexiones, pupilas, y párpados) en la imagen puede conducir al bajo rendimiento

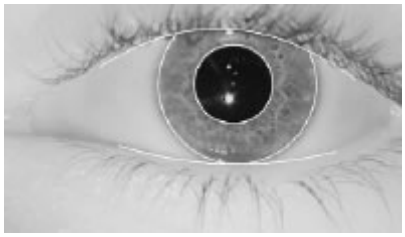


Figura 2.11)
Los contornos blancos indican la localización
de los límites del iris y del párpado

La digitalización de imagen del iris requiere el uso de una cámara fotográfica digital de la alta calidad. Las cámaras fotográficas comerciales de iris actuales utilizan comúnmente la luz infrarroja para iluminar el iris sin causar daño o malestar al sujeto. Al digitalizar la imagen del iris, haciendo uso de los filtros de Gabor se filtran y trazan los segmentos del iris en los fasores (vectores).

Estos fasores incluyen la información sobre la orientación y la frecuencia espacial (el "qué" de la imagen) y la posición de estas áreas (el "dónde" de la imagen). Esta información se utiliza para obtener los vectores de caracteristicas (Figuras 2.13 y 2.14).

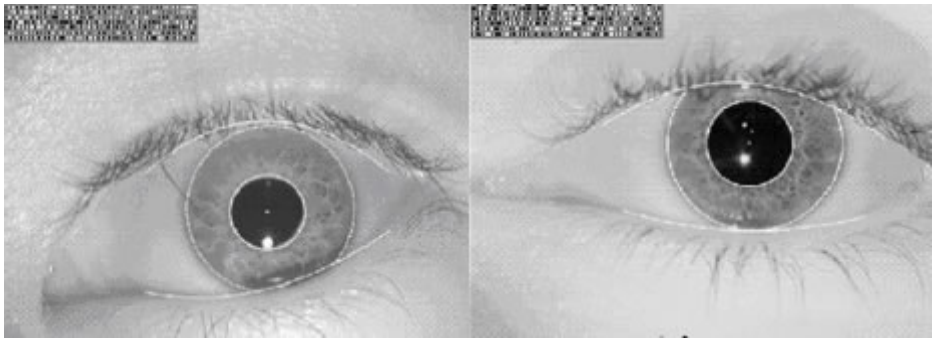


Figura 2.12) Iris Localizados Y vectores de características

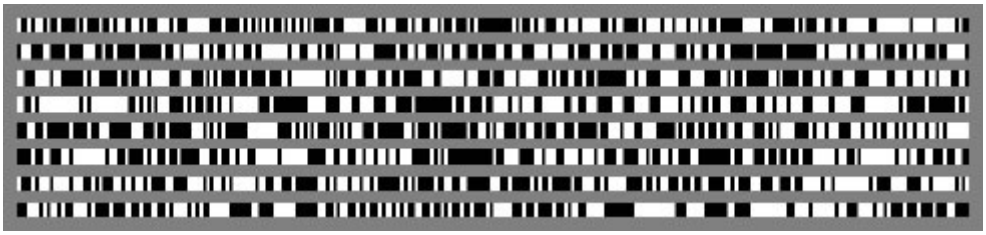


Figura 2.13) Ejemplo ilustrado de un vector de características (IrisCode®).

Los patrones del diafragma son descriptos por un vector de caracteristicas usando la información de la fase recogida en los fasores. La fase no es afectada por el contraste, la ganancia de la cámara fotográfica, o los niveles de la iluminación. Esta fase característica de un iris se suele describir en la mayor parte de los sistemas actuales usando 256 bytes de datos mediante un sistema de coordinada polar. También se incluyen en la descripción del iris los bytes de control que se utilizan para excluir las pestañas, las reflexiones, y otros datos indeseados. Para realizar el reconocimiento, se comparan dos vectores de características. La suma de diferencias entre dos - la Distancia de Hamming (HD) - se utiliza como prueba de independencia estadística entre los vectores. Si el HD indica que menos de un tercio de los bytes en los vectores de características son diferentes, el vector falla en la prueba de importancia estadística, indicando que son del mismo iris. Por lo tanto, el concepto dominante para el reconocimiento de iris es fallido a la prueba de la independencia estadística.

2.7 FUNCIONAMIENTO; RENDIMIENTO Y SUPLANTACION

En un sistema de Biometría típico, la persona se registra en el sistema cuando una o más de sus características físicas y de conducta son obtenidas, procesada por un algoritmo biométrico, e introducida en una base de datos, o en un dispositivo tipo token. Idealmente cuando se verifica con éxito, implica que sus características concuerdan por encima de un umbral establecido; (en función de la modalidad) cuando entra, casi todas sus características concuerdan. Entonces cuando alguna otra persona intenta identificarse, esto no sucede y el sistema no le permite el acceso. Las tecnologías actuales tienen tasas de error que varían ampliamente (desde valores bajos como el 60%, hasta altos como el 99,9%), veamos a continuación dos aspectos interesantes [WIK08]:

2.7.1 Rendimiento

El rendimiento de una medida biométrica se define generalmente en términos de tasa de falso positivo (*False Acceptance Rate* o FAR), la tasa de falso negativo (*False NonMatch Rate* o FNMR, también *False Rejection Rate* o FRR), y el fallo de tasa de alistamiento (*Failure-to-enroll Rate*, FTR o FER).

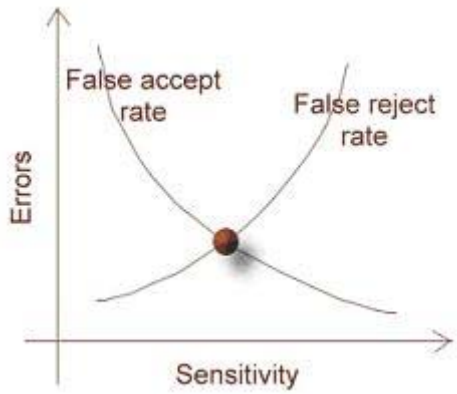


Figura 2.14) Rendimiento biométrico

En los sistemas biométricos reales el FAR y el FRR pueden transformarse en los demás cambiando cierto parámetro. Una de las medidas más comunes de los sistemas biométricos reales es la tasa en la que el ajuste en el cual acepta y rechaza los errores es igual: la tasa de error igual (*Equal Error Rate* o EER),

también conocida como la tasa de error de cruce (*Cross-over Error Rate* o CER). Cuanto más bajo es el EER o el CER, se considera que el sistema es más exacto.

Las tasas de error anunciadas implican a veces elementos idiosincrásicos o subjetivos. Por ejemplo, un fabricante de sistemas biométricos fijó el umbral de aceptación alto, para reducir al mínimo las falsas aceptaciones; en la práctica, se permitían tres intentos, por lo que un falso rechazo se contaba sólo si los tres intentos resultaban fallidos (por ejemplo escritura, habla, etc.), las opiniones pueden variar sobre qué constituye un falso rechazo. Si entro a un sistema de verificación de firmas usando mi inicial y apellido, ¿puedo decir legítimamente que se trata de un falso rechazo cuando rechace mi nombre y apellido?

A pesar de estas dudas, los sistemas biométricos tienen un potencial para identificar a individuos con un grado de certeza muy alto. La prueba forense del ADN goza de un grado particularmente alto de confianza pública actualmente (ca. 2004) y la tecnología está orientándose al reconocimiento del iris, que tiene la capacidad de diferenciar entre dos individuos con un ADN idéntico.

2.7.2 Suplantación

Lo que hasta ahora parecía solo posible en las películas de cine, empieza a ser estudiado como una posible amenaza: engañar los sistemas de identificación biométricos mediante suplantación.

La suplantación biométrica no es ninguna novedad. De hecho, cualquiera que haya visto películas donde intervienen sistemas informáticos casi seguro que ha visto algún ejemplo. Desde los más clásicos (cortar el dedo para acceder a los sistemas protegidos mediante huella digital o grabar la contraseña en una cinta magnetofónica) a los más 'sofisticados' de utilizar una lentilla para imitar el iris.

En el pasado algunas de estas 'hazañas cinematográficas' han sido realizadas en entornos de laboratorio. Ya hace años se demostró como muchos sistemas de reconocimiento de huella digital podían ser fácilmente suplantados (afortunadamente no era necesario cortar el dedo del usuario; un molde del dedo realizado con gelatina era más que suficiente).

No obstante, el último aviso sobre la posibilidad de realizar una suplantación biométrica ha venido de la laboratorio de biometría de Deloitte & Touch: no es necesario el dedo, basta con disponer de la huella digital, algo que solemos dejar en fielmente cualquier sitio. Para protegerse ante esta amenaza,

algunos sistemas lectores verifican que la huella suministrada tiene pulso, evitando así la lectura de dedos y huellas artificiales.

Por ahora son únicamente pruebas de concepto a nivel de laboratorio: recoger una huella digital y utilizarla para suplantar a un usuario a través de un lector biométrico. No obstante, cabe recordar que los sistemas biométricos no son habitualmente utilizados por lo que los incentivos para los atacantes son reducidos. Ahora bien, la utilización de un sistema de protección biométrico también suele ser un indicador que allí hay algo de valor.

La lección importante que debe aprenderse de esto es que la biometría trata con un elemento que no es secreto: hay elementos visibles que pueden ser registrados (cara, iris...); la voz puede grabarse; por allí donde pasamos solemos dejar muestras de ADN y huellas digitales. En consecuencia, la biometría es un mal sustituto de los sistemas de identificación, pero puede ser útil en sistemas de autenticación de doble factor: no sólo es necesario demostrar que el dedo es nuestro, también deberemos asegurar que sabemos algo, como una contraseña o un PIN.

2.8 CONCLUSIONES

Como hemos visto en las explicaciones anteriores, los sistemas biométricos están en alza por su seguridad, sencillez de uso, y fiabilidad. Su esquema se puede resumir en dos pasos.

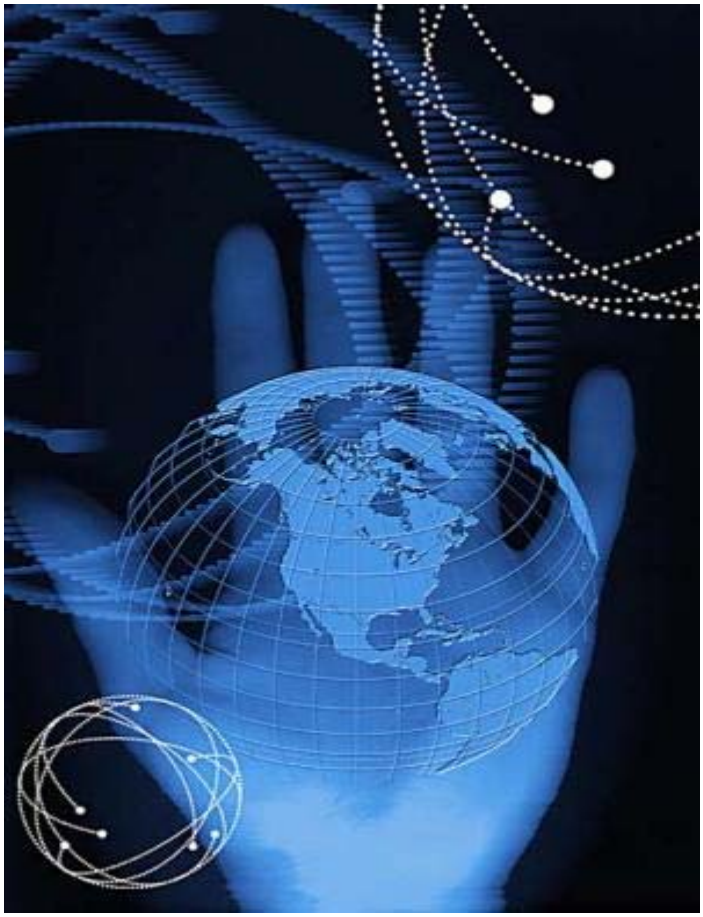
El primer paso consiste en el registro del usuario en el sistema. Durante este proceso, el sistema extrae el rasgo característico con que trabaje el sistema de identificación, y lo procesa para creando el llamado patrón. Dicho patrón debe ser guardado en una base de datos, una tarjeta inteligente o en algún otro dispositivo del cual pueda ser extraído en cualquier momento que se requiera.

El segundo paso depende de si se necesita verificar o identificar. En el caso de verificación, la persona le informa al sistema cual es su identidad ya sea presentando una tarjeta de identificación o introduciendo alguna clave, mientras que si se trata de una identificación, el sistema debe comparar el vector de características extraído con un conjunto de patrones que se han de tener almacenados; con estas comparaciones determinará la identidad de la persona.

La biometría es un campo muy interesante y excitante que ha crecido exponencialmente en los últimos años y especialmente desde el 2001, tras los

ataques terroristas del 11 de septiembre. El estado actual de la investigación en este campo permite augurar un futuro a corto/medio plazo en el que estos sistemas están tomando un papel relevante en lo que a seguridad se refiere y es sabido que un incremento en el uso trae consigo una reducción en los costes. Paralelamente a esta reducción se está produciendo un incremento sustancial de la precisión. La gran variedad de características físicas únicas que nuestro cuerpo nos da nos permitirá vivir en un mundo sin passwords donde la clave que te proporciona un acceso o un permiso la llevas en tu propio cuerpo.

CAPÍTULO 3



Biometría y Estándares

El uso de la biometría como técnica generalmente reconocida para la identificación de personas, puede datar su inicio a finales del siglo XIX. En esas fechas, la Identificación Biométrica tenía una componente casi exclusivamente forense, y una aplicación fundamentalmente jurídico/policial asociada a técnicas de verificación personal a partir del estudio de las impresiones dactilares. A finales del siglo XX, los avances de los sistemas informáticos consiguen que la Identificación Biométrica se expanda a otros entornos lo que conlleva un aumento claro de la necesidad de imponer un estándar para hacer posible la compatibilidad.

3.1 INTRODUCCIÓN

A lo largo del tiempo, las soluciones creadas para realizar las aplicaciones que utilizan la Biometría, han sido propietarias y han carecido de cualquier tipo de componente estandarizado o normalizado. De esta forma, tanto los dispositivos, como las bases de datos y los interfaces, eran distintos para cada suministrador tecnológico, imposibilitando cualquier tipo de interoperabilidad entre sistemas. Sólo en algunos aspectos de los sistemas biométricos jurídico/policiales basados en técnicas de identificación dactilar, se disponían de componentes normalizados aunque sin llegar a tener consideración de estándares de *"jure"* internacionales.

En 1926, algunos organismos nacionales de normalización fundaron la Federación Internacional de Asociaciones Nacionales de Normalización (**ISA**), con la finalidad de promover el comercio internacional a través de la estandarización de los procesos de producción y los productos La ISA, que ceso sus actividades en 1942, puede considerarse el precedente inmediato del actual Organismo Internacional de Normalización (**ISO**). Esta nueva organización, ISO, comenzó oficialmente sus actividades el 23 de Febrero de 1947 En la actualidad, 90 países forman parte de ISO y cuenta con 190 comités técnicos de muy distintas materias.

Figura 3.0) un logo de ISO

Fue a finales de los años 90 cuando se empieza a ver la necesidad de crear interfaces comunes, así como formatos de datos conocidos. De ahí surgen iniciativas de carácter privado y sectorial, que impulsan determinadas tentativas de estándares de facto. Sin embargo, a consecuencia de los eventos del 11 de Septiembre de 2001, se empuja esa necesidad, y sobre todo, el hecho de que los acuerdos sean de índole mundial. Esto motiva que el 20 de Agosto de 2002, se cree un Subcomité dedicado a la Identificación Biométrica, dentro del Comité Conjunto ISO/IEC sobre Tecnologías de la Información (JTC1). Desde esa fecha, el número de países miembros de ese Subcomité ha ido aumentando,

y se ha ido trabajando en sacar a la luz estándares internacionales que cubran todos los ámbitos de la Identificación Biométrica.. En el próximo apartado se va a dar a conocer los objetivos y estructura del Subcomité Internacional, para posteriormente pasar a un apartado dedicado a la enumeración de los trabajos realizados hasta la fecha.

3.2 ESTRUCTURA GENERAL DE LA ISO

La Organización ISO está compuesta por tres tipos de miembros [ISO08]:

Miembros natos → hace referencia al "cuerpo nacional de estandarización mas representativo del país", por lo tanto habrá uno por país.

Miembros correspondientes → de los organismos de países en vías de desarrollo y que todavía no poseen un comité nacional de normalización. No toman parte activa en el proceso de normalización pero están puntualmente informados acerca de los trabajos que les interesen.

Miembros suscritos → países con reducidas economías a los que se les exige el pago de tasas menores que a los correspondientes.

La ISO fundamentalmente está conformada por:

Asamblea General: Está constituida por un grupo de Delegados que son nombrados por los Organismos Miembros. Esta Asamblea General debe reunirse por lo menos cada 3 años y durante su sesión cada miembro tiene derecho a emitir un sólo voto por cada uno de los acuerdos emanados.

Consejo: Es un organismo que esta constituido por un Presidente y por las representaciones de 18 organismos, que duran en su cargo tres años y cuyas funciones principales son las de vigilar que el trabajo que se lleva a cabo se realice dentro de las disposiciones que se encuentran en los Estatutos y en las Reglas de Procedimiento de la Organización. Con el propósito de realizar en forma eficaz sus funciones, el Consejo ha creado los siguientes órganos:

Junta Directiva: Ayuda al Consejo a estudiar asuntos de administración y organización que pudieran surgir entre las reuniones del Consejo y toma medidas en nombre del Consejo para la designación de Presidentes de Comités Técnicas.

Junta Técnica: Asesora al Consejo en todos los asuntos tocantes a la organización, coordinación y planeación del trabajo técnico de la ISO. Revisa y aprueba títulos y alcances de Comités Técnicos individuales para garantizar la mayor coordinación y evitar hasta donde sea posible la duplicidad de trabajos, examina recomendaciones apropiadas al Consejo, actúa, si es necesario, dentro del sistema de la política previa de decisiones del consejo, recomienda el establecimiento o eliminación de Divisiones Técnicas.

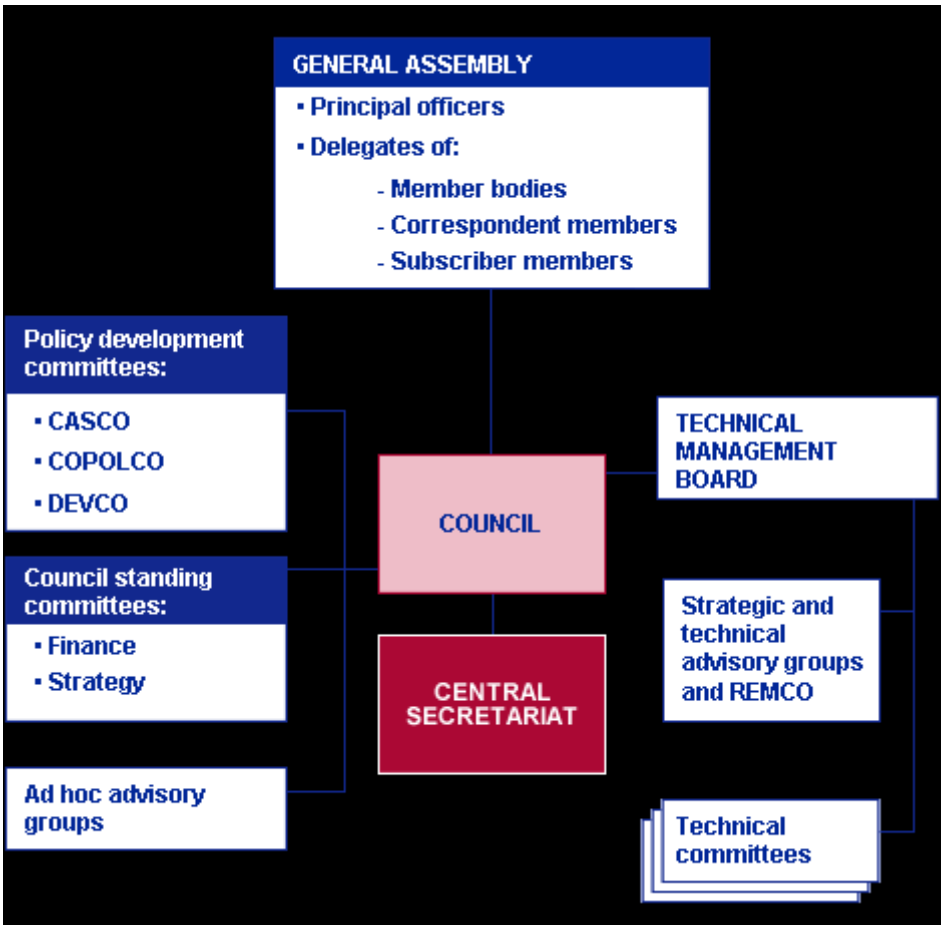


Figura 3.1) estructura de ISO

CASCO(Comité para el Aseguramiento de la Conformidad): Estudia medios para el aseguramiento de la conformidad de producto, procesos, servicios y sistemas de calidad con las normas apropiadas u otras especificaciones técnicas, prepara guías para pruebas, inspección y certificación de productos, procesos, y servicios y aseguramiento de sistemas de calidad, laboratorios de ensayos,

organismos de inspección, certificación para su operación y aceptación. Promueve el reconocimiento y aceptación mutua de sistemas nacionales y regionales de aseguramiento de conformidad con normas internacionales para los ensayos, inspección, certificación y actividades relacionadas.

COPOLCO(Comité para Políticas del Consumidor): Estudia los medios para ayudar al consumidor a beneficiarse con la Normalización Nacional e Internacional.

DEVCO(Comité de Desarrollo): Identifica las necesidades y analiza las propuestas de países en vías de desarrollo en campos de la normalización (Control de Calidad, Metrología, Certificación, etc.) y los apoya para solucionar dichas necesidades.

INFCO(Comité de Información): Promueve los objetivos establecidos en la Constitución de ISONET (Red de Información de la ISO), ayuda en la armonización de las actividades de los centros de información sobre normas, regulaciones técnicas y asuntos relacionados, fomenta el uso de Normas Internacionales en el trabajo de los Centros Individuales de Información y del sistema de trabajo en conjunto, estimula el intercambio de conocimientos y experiencias entre los centros y fomenta el entrenamiento de personal para la información internacional. Asesora al Consejo en lo antes mencionado y en otros asuntos relacionados con la recopilación, almacenamiento, recuperación, aplicación y difusión de información técnica y científica sobre normalización.

REMCO: (Comité sobre Materiales de Referencia) Establece definiciones, categorías, niveles y clasificación de materiales de referencia que emplea la ISO, formula el criterio que deberá aplicarse para la selección de fuentes que se mencionan en los documentos de la ISO, propone, hasta donde sea posible, las medidas a tomarse sobre materiales de referencia, requeridos por los trabajos técnicos de la ISO y atiende asuntos de su competencia que surjan con relación a otras organizaciones internacionales y asesora al Consejo sobre medidas a tomarse.

STACO(Comité Permanente para el Estudio de los Principios de la Normalización):Elabora e informa sobre los métodos para la identificación de necesidades de normalización y para la selección de prioridades, incluyendo métodos para medir los efectos de la normalización. Elabora la clasificación de los diferentes tipos de normas, las definiciones básicas para la normalización y los principios para la preparación de las normas, así como los métodos de adiestramiento en el campo de la normalización.

COMITES TECNICOS DE LA ISO: El trabajo técnico de la ISO se lleva a cabo a través de los Comités Técnicos (TC). Cada Comité puede establecer Subcomités (SC) y Grupos de Trabajo (WG) para cubrir las diferentes áreas de su campo de especialización. Los Comités Técnicos tienen números asignados siguiendo el orden progresivo en el que fueron creados, empezando por el ISO-TC-1 creado en 1947, hasta el ISO-TC-218 creado en 1998. Cuando un Comité técnico es disuelto su número no es asignado a otro nuevo comité, de tal forma que actualmente existe un listado de 218 comités técnicos de los cuáles 186 se encuentran en funciones. Los organismos miembros que deciden tomar parte activa en el trabajo del Comité Técnico o Subcomité se designan con el nombre de "Miembros Participantes" (P) de dicho Comité o Subcomité. Los países que solamente desean estar enterados del trabajo que realizan los Comités Técnicos o Subcomités se registran como "Miembros Observadores" (O). La mayor parte del trabajo técnico se lleva a cabo a través de correspondencia. Solamente cuando es completamente justificable se convoca a reunión internacional. Cada año se circulan alrededor de 10, 000 documentos de trabajo. Los organismos miembros que deciden tomar el carácter de "miembro P " tienen los siguientes derechos y obligaciones:

Derechos:

- Tener voz y voto durante las reuniones de la Asamblea General
- Integrar y participar en los Comités Técnicos que se constituyan, para dar cumplimiento a los objetivos de la ISO.
- Recibir los documentos oficiales del Secretariado Central de la ISO.
- Emitir comentarios y observaciones a los documentos técnicos.

Obligaciones:

- Cumplir con las Directrices de la ISO/IEC y con las decisiones que emanan de la Asamblea y el Consejo.
- Asistir a las Reuniones de la Asamblea y del Consejo, cuando se participe como miembro de este ultimo.
- Votar, en los casos en los que corresponda, pudiendo abstenerse de hacerlo.
- Pagar en término la cuota que establezca el Consejo de la ISO.

3.3 ETAPAS DE UN ESTANDAR

Debemos tener en cuenta que Los estándares antes de comenzar a ser aplicados pasan por una serie de etapas que verifican que van a estipular las guías correctas para todos aquellos trabajos que deban ceñirse a ellos. Las fases que se pueden considerar son las siguientes: **1) Necesidades** (se realiza un estudio de viabilidad técnico-económica y se analizan las necesidades) **2) Planificación colectiva** (se realiza un programa de trabajo) **3) Borrador** (un comité técnico coordina a expertos que lo elaboran) **4) Aprobación final** (todos los miembros que han formado parte del equipo de trabajo han de estar de acuerdo en la forma final del estándar) **5) Comprobación** (debe ser de interés general, sin crear objeciones importantes) **6) Aprobación** (tras revisar el texto definitivo se puede publicar) **7) Revisión** (seguimiento continuo para adaptarlo a los nuevos avances)

3.4 CONTENIDO Y CLASIFICACIÓN DE LOS ESTANDARES

El contenido de los estándares puede variar según sus características, objeto y medio al que vaya dirigido, sin embargo siempre se cumplen una serie de aspectos. En el trabajo que este documento abarca, los estándares son el segundo protagonista junto con los sistemas de identificación biométrica y por ello vamos a conocer más sobre ellos.

3.4.1 Caracteristicas

- **Cubren un amplio abanico de disciplinas:** Los estándares describen todos los aspectos técnicos, económicos y sociales de la actividad humana. Asimismo engloban todas las disciplinas básicas de la sociedad (lenguaje, matemáticas, física, ingeniería, etc.)

- **Son coherentes y consistentes:** Esto se consigue gracias a que son desarrollados por comités técnicos al amparo de un organismo especializado, consiguiendo así resolver discrepancias entre diferentes áreas de actividad y sectores empresariales.

- **Surgen como resultado de la participación:** Son aprobados por consenso de todas las partes involucradas: productores, usuarios, laboratorios, autoridades públicas, consumidores, etc.

- **El proceso de desarrollo es activo:** Se trata de crear estándares basados en al experiencia previa llegando a un compromiso entre la tecnología más avanzada y las limitaciones económicas.

- **Están actualizados:** Son revisados periódicamente para llevar un avance paralelo al de la tecnología y progreso social.

- **Reconocidos nacional e internacionalmente:** Los estándares son documentos reconocidos regional, nacional o internacionalmente.

- **Accesibles a todo el mundo:** Son de libre consulta y compra sin restricciones.

- **Uso voluntario:** A pesar de que en algunos casos es obligatorio, en general no es obligatorio su uso.

Dependiendo del ámbito de aplicación al que estén destinados, en función de su contenido o por su forma de aplicación podemos dar la clasificación siguiente a los estándares

3.4.1 Clasificación

Por el ámbito de aplicación

-Nacionales (Tabla 1, Capítulo Anexos)

-Normas para el sector industrial

-Normas para la empresa

-Normas para organismos nacionales

-Internacional (Tabla 2, Capítulo Anexos)

Por el contenido

-Científico

-Definiciones de magnitudes, unidades y símbolos.

-Designaciones de la simbología matemática.

-Designaciones de notaciones científicas.

-Industrial

-Normas de calidad

-Normas dimensionales

-Normas orgánicas

-Normas de trabajo

Por la forma de aplicación

-Obligatorias

-Voluntarias

3.5 ESTANDARIZACION Y BIOMETRIA

En los últimos años se ha notado una preocupación creciente por las organizaciones reguladoras respecto a elaborar estándares relativos al uso de técnicas biométricas en el ambiente informático. Esta preocupación es reflejo del creciente interés industrial por este ámbito tecnológico, y a los múltiples beneficios que su uso aporta. No obstante ello, aún la estandarización continua siendo deficiente y como resultado de ello, los proveedores de soluciones biométricas continúan suministrando interfaces de software propietarios para sus productos, lo que dificulta a las empresas el cambio de producto o vendedor.

Un estándar biométrico cumple con las características generales de los estándares, especificando a su vez una serie de características propias como son:

1. Formato para el intercambio de datos biométricos.

2. Un formato común de archivo que proporciona independencia de la plataforma utilizada, así como una clara separación entre la lógica de procesamiento y la de presentación.

3. Aplicación de interfaces de programación y de perfiles.

4. Definiciones de medida y cálculo del rendimiento.

5. Métodos para evaluar el rendimiento.

6. Requisitos para realizar informes de resultados.

A nivel mundial el principal organismo que coordina las actividades de estandarización biométrica es el Sub-Comité 37 (SC37) del Joint Technical Committee on Information Technology (ISO/IEC JTC1), del International Organization for Standardization (ISO) y el International Electrotechnical Commission (IEC).

En Estados Unidos desempeñan un papel similar el Comité Técnico M1 del INCITS (InterNational Committee for Information Technology Standards), el National Institute of Standards and Technology (NIST) y el American National Standards Institute (ANSI).

Figura 3.2) Logo del Joint Technical Committee

Existen además otros organismos no gubernamentales impulsando iniciativas en materias biométricas tales como: Biometrics Consortium, International Biometrics Groups y BioAPI. Este último se estableció en Estados Unidos en 1998 compuesto por las empresas Bioscrypt, Compaq, Iridiam, Infineon, NIST, Saflink y Unisis. El Consorcio BioAPI desarrolló conjuntamente con otros consorcios y asociaciones, un estándar que promoviera la conexión entre los dispositivos biométricos y los diferentes tipos de programas de aplicación, además de promover el crecimiento de los mercados biométricos.

Los estándares más importantes son: **Estándar ANSI X.9.84** creado en 2001, por la ANSI (American National Standards Institute) y actualizado en 2003, define las condiciones de los sistemas biométricos para la industria de servicios financieros haciendo referencia a la transmisión y almacenamiento seguro de información biométrica, y a la seguridad del hardware asociado.

Estándar ANSI / INCITS 358 Estándar creado en 2002 por ANSI y BioApi Consortium, que presenta una interfaz de programación de aplicación que garantiza que los productos y sistemas que cumplen este estándar son interoperables entre sí.

Estándar NISTIR 6529 También conocido como CBEFF (Common Biometric Exchange File Format) es un estándar creado en 1999 por NIST y Biometrics Consortium que propone un formato estandarizado (estructura lógica de archivos de datos) para el intercambio de información biométrica y que a día de hoy se encuentra estandarizado bajo el SC37.

CAPÍTULO 4

BioAPI

El consorcio BioAPI fue fundado para desarrollar un interfaz de programación de aplicaciones biométricas (API) que aporta la plataforma y dispositivos independientes para programadores de aplicaciones y proveedores de servicios biométricos. En esta plataforma se sustenta todo el trabajo realizado en este proyecto, habiendo guiado su desarrollo con las herramientas y directrices que proporciona la versión 2.0 del consorcio, la cual es a día de hoy la última que se puede utilizar.

4.1 INTRODUCCIÓN

El Consorcio es un grupo de más de 120 empresas y organizaciones que tienen un interés común en promover el crecimiento del mercado de la biometría. El pliego de condiciones y de referencia para la aplicación normalizada de un API que es compatible con una amplia gama de programas de aplicación biométrica y un amplio espectro de tecnologías biométricas.

La especificación de BioAPI define un estándar abierto de programación de interfaces de aplicación (API) que permita a las aplicaciones comunicarse con cualquier tipo de tecnologías biométricas. La norma internacional BioAPI 2.0 fue desarrollada, por el organismo ISO/IEC JTC1 SC37, dando lugar a el estándar ISO/IEC 19784-1. Esta versión contiene un número de simplificaciones y mejoras importantes respecto a la versión 1.1.

La principal diferencia de ambas versiones es el modelo de componentes., en BioAPI 2.0 existen tres elementos principales: la aplicación, los BSPs y el Framework situado entre ellos, esta estructura es igual en BioAPI 1.1, sin embargo, mientras que en la versión 2.0 existe una capa adicional por "debajo" de los BSPs, que son los BFP´s (que pueden realizar parte de la funcionalidad de los BSP, realizando así una división del trabajo entre ambos componentes) en la versión 1.1 esta división no se da.

Por lo tanto, se puede decir que las grandes diferencias con BioAPI 1.1 son: el modelo de componentes, la incorporación de sesiones, unidades y dispositivos y la existencia, de un registro.

4.2 HISTORIA Y EVOLUCIÓN DE BioAPI

La primera vez que el Consorcio BioAPI anunció su formación y el intento de desarrollar un estándar biométrico fue en Abril de 1998. Al final de este año este grupo había desarrollado una arquitectura API multinivel y empezó a definir todos sus componentes asociados.

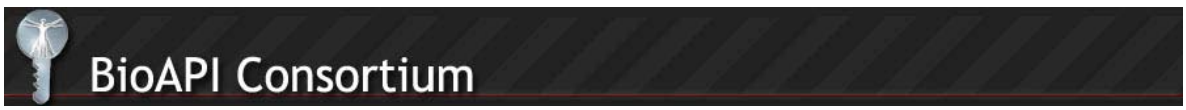


Figura 4.0): Consorcio BioAPI

En 1999, el Laboratorio de Información Tecnológica del NIST (National Institute of Standard and Technology) y el Consorcio Biométrico de los Estados

Unidos, anunciaron su unificación mediante una conferencia en la cual el grupo de trabajo Human Authentication API (HA-API), que había publicado un API biométrico de alto nivel en 1997, aceptó colaborar con el Consorcio BioAPI, lo cual provocó una reestructuración de la organización. Tras esta reestructuración el consorcio aumento sus esfuerzos para definir la arquitectura del API biométrico, así como consolidar su estructura interna y aumentar sus operaciones. Estos esfuerzos empezaron a mediados de 1999 dado lugar a la versión 1.0 de la especificación en Marzo del 2000, mientras que la implementación de referencia vio la luz en Septiembre del mismo año.

La versión 1.1, tanto la especificación como la implementación de referencia, fue publicada en Marzo de 2001. Finalmente, como fruto de la colaboración entre ISO e IEC en el JTC1 SC 37, tal y como se ha comentado en capítulos precedentes (figura 4-1), se presentó la versión 2.0 de BioAPI.

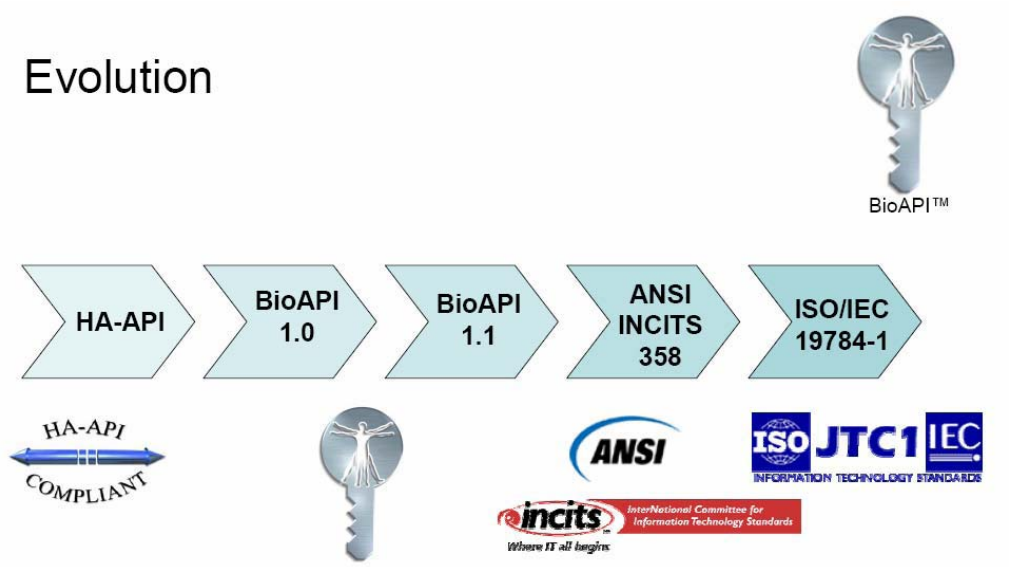


Figura 4.1): Evolución de BioAPI

4.3 PLATAFORMA DE DESARROLLO.

La implementación de referencia es un paquete software que contiene la instanciación del Framework de BioAPI. Este Framework soporta las funciones definidas en la especificación API (por el cual una aplicación interactua con el Framework) e invoca a las que se encuentran en el SPI (interfaz utilizado para comunicarse con el BSP). Aunque como veremos mas adelante la aplicación desarrollada en este proyecto sustituirá al Framework en la comunicación con los BSPs, para la elaboración de la misma es de gran utilidad. El Framework

también gestiona el mantenimiento de los BSPs y la redirección de las llamadas de las funciones del API a las del SPI (del BSP apropiado).

Lo situación óptima en el caso que nos ocupa sería una implementación de referencia pública y de código abierto, de esta forma se aceleraría la adopción del estándar. Desgraciadamente, por el momento, la única implementación de referencia de BioAPI 2.0 esta desarrollada por la división ***The BioFoundry®*** de ***OSS Nokalva.*** Su paquete consiste en:

BioAPI 2.0 Framework: Se proporciona el archivo (.exe) que instala el Framework en el sistema.

Plantilla para BSP: Implementación, en Visual C++ .NET de la plantilla básica para la elaboración de un BSP. Esta plantilla ha sido la utilizada para el desarrollo del BSP creado para este proyecto.

Ejemplo sencillo de BSP: Un sencillo ejemplo de BSP llamado "password BSP", también realizado en Visual C++.

Ejemplo de aplicación: Un ejemplo, en Visual C++, de una aplicación que utiliza el BSP anterior.

Documentación básica de usuario.

4.4 ARQUITECTURA BioAPI

4.4.1 El modelo API/SPI.

La arquitectura de BioAPI, tal y como se ha venido introduciendo, incorpora un modelo API/SPI (que definen las interfaces disponibles) tal y como se muestra en la figura 4.1. Un sistema BioAPI consiste en una serie de componentes que proporcionan unas interfaces normalizadas.

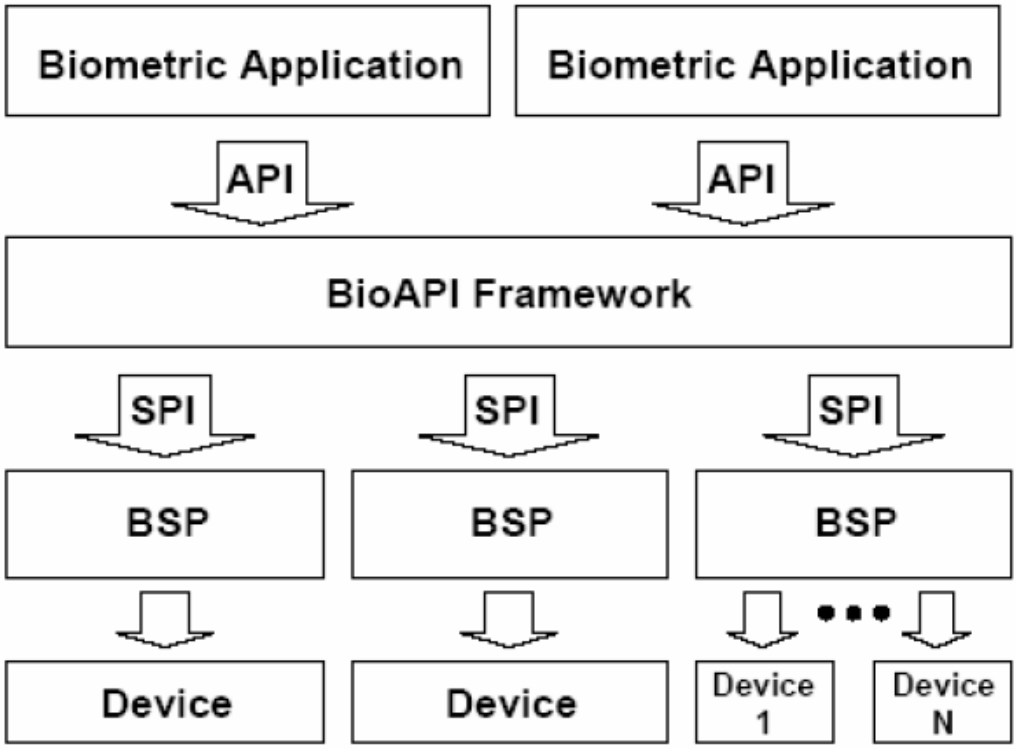


Figura 4.2) Arquitectura de BioAPI
(donde BSP: Biometric Service Provider, BFP: Biometric Function Provider y SPI: Service Provider Interface.)

El API especifica la interfaz entre el BioAPI Framework y la aplicación biométrica. La aplicación es la que invoca a las funciones especificadas en el API, dichas funciones son soportadas por el Framework el cual se encargará de realizar las llamadas oportunas al BSP. El SPI, por su parte, proporciona la interfaz entre el Framework de BioAPI y los distintos BSPs. En este caso es el Framework el encargado de invocar a las funciones descritas en la especificación del SPI.

El Framework gestiona los BSPs instalados en el sistema, así como las llamadas realizadas por una determinada función del API a una función del SPI de un BSP concreto. Una aplicación puede acceder a las funciones de un BSP solo después de que éste haya sido debidamente cargado (*load*) e incorporado (*attach*) al sistema. De igual forma cuando una aplicación ya no requiere el uso del BSP, éste ha de ser correctamente retirado (*detach*) y descargado (*unload*) del sistema. Es preciso indicar que una aplicación puede cargar más de un BSP al mismo tiempo. Del mismo modo, un BSP puede ser incorporado a más de una aplicación al mismo tiempo.

Esta arquitectura presenta una serie de beneficios:

1. Fácil sustitución de tecnologías biométricas.

2. Uso de la tecnología para múltiples aplicaciones.

3. Fácil integración gracias al uso de una misma interfaz.

4. Rápido desarrollo de las aplicaciones y reducción de costes.

4.4.2 BSP

El proyecto que desarrolla este documento basa su principal actividad en evaluar si los BSP cumplen con la norma, por ello vamos a explicar más detalladamente qué son y cómo funcionan los BSP. Los BSP (Biometric Service Provider) encapsulan el hardware o software que implementen los BioAPI Units (abstracción de un dispositivo biométrico) para posteriormente presentarlas a la aplicación que las requiera.

Esta encapsulación ha provocado que se cambie el concepto de *"device"* utilizado en BioAPI 1.1 a *"unit"* en la versión actual.

Actualmente hay definidas las siguientes categorías de BioAPI Unit:

• **Sensor unit:** Representan sensores, los cuales son normalmente dispositivos físicos, capaces de capturar muestras biométricas de un determinado sujeto.

• **Archive unit:** Representan un camino a una base de datos almacenada en algún medio, bien sean tarjetas, bases de datos relacionales, o cualquier otro medio de almacenamiento.

• **Matching algorithm unit:** representan algoritmos de comparación de muestras biométricas con patrones biométricos.

• **Processing algorithm unit:** representan algoritmos de procesado biométrico que transforman datos "intermedios" a datos "procesados".

Es común, aunque no obligatorio, que las dos primeras unidades estén asociadas con componentes hardware mientras que las dos últimas sean piezas de software que no suelen tener ningún tipo de hardware asociado.

Como se ha indicado anteriormente un BioAPI Unit puede ser manejado internamente por el propio BSP, o puede estar asociado a un BioAPI Function

Provider (BFP), el cual se comunica con el BSP por medio del BioAPI Function Provider Interface (FPI) tal y como se indicará mas adelante.

Un BFP puede manejar múltiples BioAPI Units de una determinada categoría (pero nunca más de uno para una misma sesión y para una aplicación dada). Al igual que ocurría con los BioAPI Units los BFPs se dividen en:

- Sensor BFP
- Archive BFP
- Matching algorithm BFP
- Processing algorithm BFP

La responsabilidad de estos elementos es equivalente a la de los BioAPI Units. La interfaz (BioAPI FPI) para cada una de las categorías de BFP está definida en otra parte del estándar. Todas las FPIs tienen acceso a uno o más (pero nunca más de un por sesión) BioAPI Units del BFP seleccionado.

Un BSP que soporte múltiples BioAPI Units (directamente o a través de BFPs), ha de soportar la interfaz SPI que permite a la aplicación seleccionar un determinado BioAPI Unit (uno de cada categoría) para una sesión. Si la aplicación no indica el BioAPI Unit que quiere alcanzar será el BSP el encargado de seleccionar el que considere más apropiado. Si en la implementación del BSP se exige soportar una determinada categoría de BioAPI Unit, entonces el BSP ha de ser capaz de manejar dicha unidad directamente o de interactuar con el correspondiente BFP mediante el FPI asociado.

Hay funciones, a parte de las operaciones biométricas comunes, del API de BioAPI y las correspondientes funciones SPI (incluso FPI), que permiten a la aplicación tanto enviar como recibir información de control y estado respectivamente a (o desde) un BioAPI Unit haciendo uso de estas interfaces (BioAPI API y SPI). Los parámetros de las funciones de control no están estandarizados por lo tanto si no son soportados por el BSP o BFP se producirá un error. Cuando un BSP establece una sesión, al menos ha de seleccionarse un BioAPI Unit de una determinada categoría.

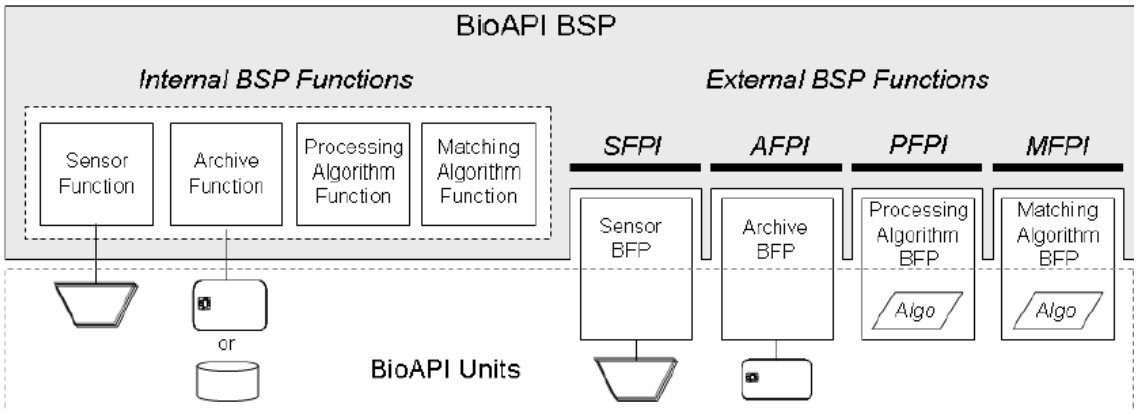


Figura 4.3) Estructura de un BSP

4.4.3 El Registro de BioAPI

Uno de los componentes mas importantes de BioAPI es el Registro del sistema, este registro contiene información útil tanto de los BSPs instalados en el sistema como de los BFPs que se estén utilizando. En el modelo de BioAPI existe un único BioAPI Framework y un único componente de registro asociado para el sistema biométrico.

A pesar de la teoría, en un ordenador real pueden coexistir múltiples registros compartidos por un mismo código de BioAPI Framework o incluso por distintas versiones de éste. Sin embargo uno de los requisitos de BioAPI es que no debería haber interferencias entre diferentes sistemas biométricos implementados en un mismo ordenador. De esta forma se evitan posibles errores. Determinadas funciones del Framework devuelven información que se almacena en el registro, a esta información puede acceder la aplicación y puede ser de los siguientes tipos:

1. Información sobre el mismo BioAPI Framework.
2. Información detallada sobre los BSPs instalados.
3. Información detallada sobre los BFPs instalados. Esta información puede ser también obtenida por los BSP mediante el uso de funciones *Callback [Funciones Callback. Anexo capítulo 9].*

La información anterior es relativa al Framework, sin embargo una aplicación también puede obtener información de un BSP específico mediante el uso de funciones que son pasadas a través del SPI. Esta información puede contener:

1. Detalles de todos los BFPs que son soportados por el BSP seleccionado.
2. Detalles de todas los BioAPI Units que se encuentran en "estado insertado" y son accesibles a través del BSP (o el correspondiente BFP).

Esta información la puede obtener una aplicación haciendo llamadas a una serie de funciones:

1. BioAPI_Init:
    a. Proporciona información del propio BioAPI Framework.
    b. Detalles de todos los BSPs instalados.
    c. Detalles de todos los BFPs instalados.
2. BioAPI_BSPLoad:
    a. Información de todos los BFPs que son soportados por cada BSP.
    b. Detalles de todos los BioAPI Units que están en "estado insertado" y que son accesibles desde cada BSP. Esto se puede hacer a través del identificador de BSP (BSP UUID) o, en su caso, del BFP correspondiente.

4.4.4 Instalación y desinstalación de BSP y BFP

A la hora de instalar un BSP o BFP, es preciso tener un identificador único (UUID) que identifique, de manera inequívoca, a cada componente. De esta forma, si se intentara instalar un BSP o BFP con el mismo identificador que uno que ya esté instalado en el sistema, se produciría un error. Sin embargo, si un mismo BSP o BFP se instala en distintos sistemas biométricos deberían (aunque no es obligatorio) tener el mismo UUID en cada uno de ellos.

No existen mecanismos para informar a un BSP instalado (se encuentre o no en uso) de la instalación o desinstalación de nuevas BFPs. Para tener

actualizado este dato, los BSP han de recurrir a una función callback (BioSPI_BFP_ENUMERATION_HANDLER). La desinstalación de BFPs provoca que los BioAPI Units (hardware y software) asociados a ella se desinstalen del sistema, por eso el control de esta parte lo tiene que manejar la propia implementación del BSP, dado que el Framework no tiene ninguna función estandarizada para ello. A continuación se expone un esquema general del funcionamiento de todos estos elementos en BioAPI 2.0

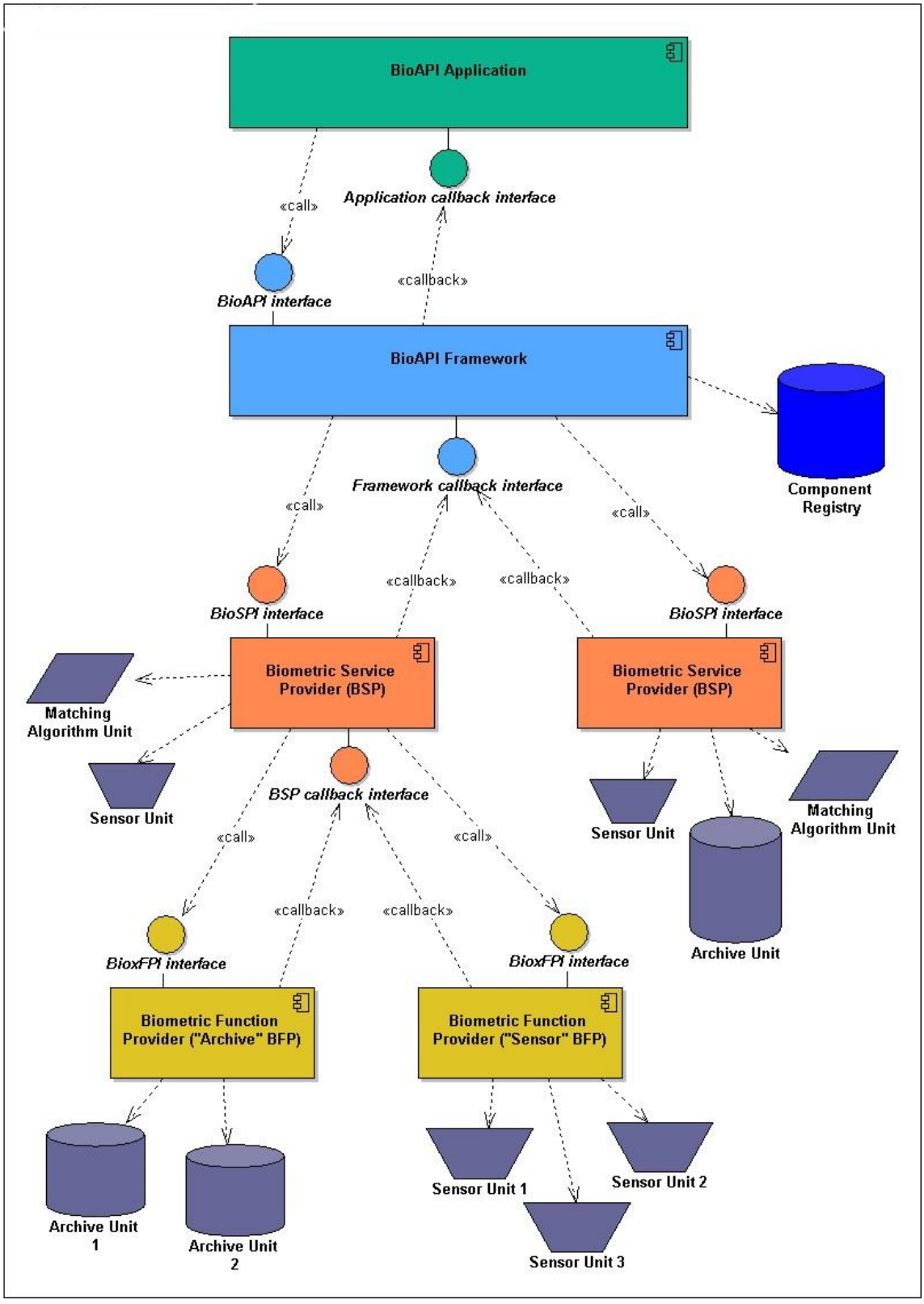


Figura 4.4) esquema BioAPI 2.0

CAPÍTULO 5

Test de conformidad

Es la parte protagonista de este proyecto. Para llevar a cabo el diseño de una aplicación que compruebe que un sistema cumple con la norma, según la ISO y BioAPI, se han estipulado una serie de pautas y métodos a seguir por parte de la persona o del equipo de personas que lleven a cabo la elaboración de dicha aplicación y han quedado reflejadas en el documento referencia para el desarrollo del presente trabajo: ISO/IEC 24709.

5.1 ISO/IEC 24709

La ISO (la Organización Internacional para la Estandarización) y IEC (la Comisión Internacional Electrotécnica) forman el sistema especializado para la estandarización mundial. Los cuerpos nacionales que son los miembros de ISO o IEC participan en el desarrollo de Normas Internacionales por comités técnicos establecidos por la organización respectiva para tratar con los campos particulares de actividad técnica. La ISO y comités IEC técnicos colaboran en los campos de interés mutuo. Otras organizaciones internacionales, gubernamentales y no gubernamentales, en el enlace con la ISO e IEC, también participan con el trabajo. En el campo de tecnología de la información, ISO e IEC han establecido un comité conjunto técnico, ISO/IEC JTC 1.

Figura 5.0) logotipo del comité

La tarea principal del comité conjunto técnico es la de preparar Normas Internacionales. Estas normas son el resultado de adecuar el tema que corresponda a las reglas dadas en las Directivas ISO/IEC. Las Normas Internacionales adoptadas por el comité conjunto técnico son difundidas a cuerpos nacionales para la votación. La publicación como un Estándar Internacional requiere la aprobación por al menos el 75 % de los cuerpos nacionales que emiten un voto.

ISO/IEC 24709 se desarrolla en cuatro partes bajo el tirulo genérico de "BioAPI Conformance Testing":

- Parte 1: métodos y procedimientos
- Parte2: declaraciones de Test para BSP
- Parte 3: declaraciones de Test para BioAPI Framework
- Parte 4: declaraciones de Test para Aplicaciones biométricas

Para nuestro proyecto hemos necesitado recurrir a la primera y segunda parte. La **Parte 1** define una metodología de prueba de conformidad. Esto especifica tres modelos de Test que permiten las pruebas de conformidad de cada uno de los componentes BioAPI siguientes: Aplicación, Framework, o BSP. Así mismo se especifica una lengua de validación de evaluaciones que es usada para la definición de declaraciones de prueba.

Para diseñar un modelo de pruebas son necesarias una serie de directrices informativas y por ello también son incluidas en la primera parte de la ISO/IEC 24709. Estas directrices identifican los tipos de actividades, responsabilidades, servicios, y la documentación recomendada para conducir la evaluación de conformidad y la certificación de puestas en práctica BioAPI-conformant.

En la **parte 2** se define un número de declaraciones de prueba escritas en la lengua de validación. Estas declaraciones permiten a un usuario probar la conformidad a ISO/IEC 19784-1 (BioAPI 2.0) de cualquier proveedor de servicio biométrico (BSP) que busque una implementación acorde con el estándar internacional. Básicamente son los pasos que el código desarrollado debe seguir para poder probar la conformidad en cada una de las funciones del BSP que el usuario elija. Las declaraciones además son organizadas según subclases de conformidad

5.2 LENGUAJE Y ELEMENTOS BASICOS

5.2.1 Lenguaje de validación de evaluaciones

El lenguaje de validación de evaluaciones es una parte integral de la metodología de Tests de conformidad. Las puestas en práctica de la conformidad BioAPI tienen la necesidad de usar todas las declaraciones proporcionadas en estas partes y sólo estas. El lenguaje de validación de evaluaciones tiene una sintaxis basada en W3C XML 1.0

Ejemplo de declaración:

EXAMPLE (non-normative)

```
<assertion name="CreateTemplate1" model="BSPTesting">
<description>
Test the BioSPI_CreateTemplate function of a BSP.
The UUID and version of the BSP must be provided
as input to the Test.
</description>
<input name="_uuid"/>
<input name="_version"/>
<invoke activity="CreateTemplate"
package="7346D660-1583-13D0-A3A5-00C0FFD756E3">
<input name="BSPUuid" var="_uuid"/>
<input name="BSPVersion" var="_version"/>
<input name="deviceIDOrNull" value="0"/>
<input name="inserttimeouttime" value="15000"/>
<input name="sourcepresenttimeouttime" value="10000"/>
<input name="capturetimeouttime" value="20000"/>
</invoke>
<bind activity="EventHandler" function="BioAPI_EventHandler"/>

</assertion>
```

5.2.2 Elementos importantes

Antes que nada deberíamos conocer ciertos elementos clave para la comprensión del funcionamiento de la aplicación y como interactúan entre si los elementos:

SPI

El interfaz de proveedor de servicios (SPI) es el interfaz de programa que un BSP presentará para trabajar con el Framework BioAPI. En general, el SPI consiste en un mapeo (exacto) de la llamada que una aplicación biométrica hace al Framework para añadir una sesión, es decir para comenzar una nueva operación con un dato biométrico. Las convenciones siguientes han sido adoptadas:

- Donde una función del SPI tiene parámetros que son idénticos a aquellos de una función API del mismo nombre, el SPI no da explicación alguna de sus atributos

- Cuando los parámetros difieran se dará una explicación completa referida al SPI.

- No todas las funciones de API tienen una función correspondiente SPI; aquellas funciones que no la tienen, son manejadas completamente por el Framework

Framework Callback Interface

Hay que tener en cuenta que nuestra aplicación será vista por el BSP como un Framework, es decir, a efectos prácticos hará directamente las llamadas que normalmente hace el Framework para intercomunicar aplicación y BSP dejando la comunicación como se representa en el siguiente esquema.

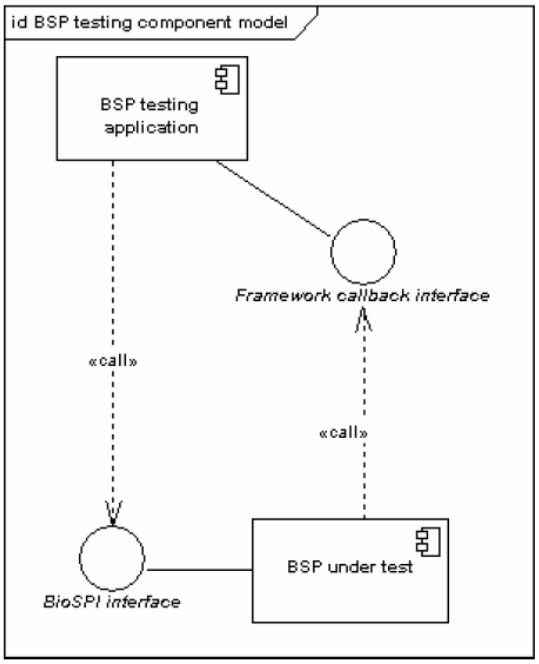


Figure 4 — Conformance testing model for BioAPI BSPs.

Figura 5.1) *Esquema del funcionamiento de la aplicación según la norma*

Debido a esto es importante señalar que se ha tenido que añadir al proyecto en Borland la librería correspondiente a las funciones BioSPI del BSP que necesitan estar definidas en algún punto de la aplicación. Esto es debido a que ahora será la aplicación quien haga las llamadas al interfaz BioSPI y no BioAPI

(quien utilizaria al Framework de intermediario) excepto en casos puntuales que se explicará mas adelante.

El Framework callback interface realiza la función de interface auxiliar que admite la recepción de la siguiente información de los BSP's:

a) notificaciones de eventos relativos a BioAPI units;

b) datos "streaming" enviados durante operación llevada a cabo por BSP o BFP(en nombre de un BSP)

c) "GUI state information" enviados durante operación llevada a cabo por BSP o por un BFP(en nombre de un BSP)

El Framework callback interface consiste en las siguientes funciones:

- **BioSPI_EventHandler**
- **BioSPI_GUI_STATE_CALLBACK**
- **BioSPI_GUI_STREAMING_CALLBACK**
- **BioSPI_BFP_ENUMERATION_HANDLER**
- **BioSPI_MEMORY_FREE_HANDLER**

BSP Testing application

En la **figura 5.1** se puede observar que la aplicación se denomina "BSP Testing application". Según la norma y como ya hemos apuntado someramente, En el modelo de Test de conformidad para los BSP de BioAPI un componente especial de Test llamado "BSP Testing application" debe reemplazar a la aplicación normal y al Framework normal.

BSP under Test

Es el proveedor de servicios biométricos que será puesto bajo la supervisión de la aplicación para pasar las pruebas de conformidad. En nuestro caso como ya hemos apuntado en otros apartados serán tres los posibles BSP probados: huella dactilar, geometría de la mano, reconocimiento por iris.

5.3 CONFORMIDAD DE BioAPI BSP´s

Para ser conforme con la especificación BioAPI, los BSP´s deberán poner en práctica funciones obligatorias para su subclase de conformidad, como se subraya mas abajo. Estos, reclaman la conformidad para una de las subclases de conformidad especificadas en la cláusula 1.1 de la norma

Los BSPs aceptarán todos los parámetros de entrada válidos y devolverán salidas válidas. Así mismo aquellos atributos o capacidades consideradas opcionales como la devoluciones opcionales de valores no deben ser probados respecto de la conformidad, pero en caso de ser puestos en práctica se hará conforme a las exigencias de especificación. En el proceso de instalación de los BSP´s se realizará la recopilación de todas las entradas de registro de componentes requeridas. Los BSPs poseerán un UUID válido y único que es asociado con un producto específico BSP y su versión.

La tabla de la **Figura 5.2** es un resumen de exigencias de conformidad BSP según la subclase de BSP a que pertenece. Proporcionan detalles en las subcláusulas siguientes.

5.3.1 Subclase de BSP

Dependiendo de las funciones que realicen los BSP son de una subclase o de otra. A continuación se exponen las distintas divisiones que la norma hace y que aparecen en el cuadro anterior:

1 BioAPI Conformant Verification BSPs

Aquellos capaces de llevar a cabo una comparación de 1:1 (o autenticación) pero no de 1:N

2 BioAPI Conformant Identification BSPs

Aquellos BSP´s capaces de llevar a cabo tanto una comparación de 1:1 como una identificación 1:N (o identificación)

3 BioAPI Conformant Capture BSPs

BioAPI Capture BSPs son aquellos que proveen de un interfaz a uno o mas sensores biométricos y devuelven BIRs intermedios que pueden ser usados

por otros BSP´s (para mas información ver cláusulas A.4.4 y A.4.5). No proporcionan la capacidad de procesar o comparar el dato que captura.

4. BioAPI Conformant Verification Engines

Estos motores de verificación de conformidad son BSP´s que contienen el procesado biométrico y los algoritmos de comparación 1:1 pero que no realizan la captura biométrica. Son típicamente usados junto con otros BSP que guían la captura

5. BioAPI Conformant Identification Engines

BSP con los mismos requisitos del anterior, con la siguiente excepción:

NOTA: (esta excepción hace referencia los parámetros específicos de las funciones: *BioSPI_CreateTemplate* y *BioSPI_Process* que deben tomar los valores indicados)

BioSPI_CreateTemplate debe también aceptar un *CapturedBIR* con un *Purpose* value of BioAPI_PURPOSE_ENROLL_FOR_IDENTIFICATION_ONLY y *BioSPI_Process* debe aceptar un *CapturedBIR* con un *Purpose* value de BioAPI_PURPOSE_IDENTIFY así como el añadido de la siguiente función: *BioSPI_IdentifyMatch*

La subclase de BSP con que trabaja la aplicación del proyecto que nos ocupa corresponde a la número 4: **BioAPI Conformant Verification Engines.**

Table A.1 – BSP Conformance Sub-Classes

Function	Verification BSP	Identification BSP	Capture BSP	Verification Engine	Identification Engine
Component Management Functions					
BioSPI_BSPLoad	X	X	X	X	X
BioSPI_BSPUnload	X	X	X	X	X
BioSPI_BSPAttach	X	X	X	X	X
BioSPI_BSPDetach	X	X	X	X	X
BioSPI_QueryUnits	X	X	X		
BioSPI_QueryBFPs					
BioSPI_ControlUnit					
Handle Functions					
BioSPI_FreeBIRHandle	X	X	X	X	X
BioSPI_GetBIRFromHandle	X	X	X	X	X
BioSPI_GetHeaderFromHandle	X	X	X	X	X
Callback and Event Functions					
BioSPI_EnableEvents	X	X	X	X	X
BioSPI_SetGUICallbacks					
Biometric Functions					
BioSPI_Capture			X		
BioSPI_CreateTemplate				X	X
BioSPI_Process				X	X
BioSPI_ProcessWithAuxBIR					
BioSPI_VerifyMatch				X	X
BioSPI_IdentifyMatch					X
BioSPI_Enroll	X	X			
BioSPI_Verify	X	X			
BioSPI_Identify		X			
BioSPI_Import					
BioSPI_PresetIdentifyPopulation					
Database Functions					
BioSPI_DbOpen					
BioSPI_DbClose					
BioSPI_DbCreate					
BioSPI_DbDelete					
BioSPI_DbSetMarker					

Figura 5.2)exigencias de conformidad según subclase de BSP

Function	Verification BSP	Identification BSP	Capture BSP	Verification Engine	Identification Engine
BioSPI_DbFreeMarker					
BioSPI_DbStoreBIR					
BioSPI_DbGetBIR					
BioSPI_DbGetNextBIR					
BioSPI_DbDeleteBIR					
BioAPI Unit Functions					
BioSPI_SetPowerMode					
BioSPI_SetIndicatorStatus					
BioSPI_GetIndicatorStatus					
BioSPI_CalibrateSensor					
Utility Functions					
BioSPI_Cancel	X	X	X	X	X
BioSPI_Free	X	X	X	X	X

Figura 5.2) exigencias de conformidad según subclase de BSP

5.4 REQUISITOS, BCS y CTS

Existe una declaración de aptitudes conforme a la norma BioAPI. Esta declaración debe ser cumplimentada por el que suministre la implementación bajo Test antes de las pruebas de conformidad, y recibe el nombre de BioAPI Conformity Statement (BCS). Como mínimo la BCS debe contener una lista detallada de los elementos, obligatorios, opcionales, y condicionales implementados en la IUT.

Para alcanzar pruebas de conformidad creíbles, el resultado de ejecutar un caso de prueba sobre una IUT debería ser el mismo siempre que sea realizado. Mientras no sea posible ejecutar una suite de prueba completa y observar que los resultados de prueba son idénticos a aquellos obtenidos en las otras ocasiones, cada esfuerzo debería buscar el asegurar la repetibilidad de resultados de prueba y reducir al mínimo la posibilidad que un caso de prueba produzca resultados diferentes sobre condiciones iguales

Elementos a Testar: 3 tipos

- Elementos obligatorios: siempre han de Testarse
- Elementos condicionales: han de Testarse si las condiciones expuestas en las especificaciones se aplican

- Elementos optativos: pueden ser seleccionados para adaptar la implementación y solo se Testan si se seleccionan

En el proyecto que nos ocupa la tarea principal es la de probar la conformidad y para ello debe seguirse una serie de pasos y deben reunirse algunos documentos y elementos que resultan imprescindibles para emitir un análisis de conformidad adecuado con las normas internacionales. Cada puesta en práctica de método de prueba incluirá:

1. La "Herramienta de conformidad" (CTS), incluyendo la documentación de la suite de prueba, describiendo categorías de prueba, objetivos para cada prueba individual, instrucciones sobre como ejecutar la suite de prueba, y los resultados esperados de ejecutar las pruebas individuales. El CTS será capaz de ejecutar los casos de prueba, capturando los resultados devueltos, evaluándolos y relatándolos.

2. Casos documentados de prueba, que asegurarán suficientemente la conformidad con el estándar. Los casos serán formalmente representados en forma de declaraciones prueba que usan la lengua de validación.

3. El Procedimiento de Conformidad que identificará y definirá todas las actividades necesarias para la preparación de Pruebas de Conformidad, realizará las pruebas, y relatará los resultados. El procedimiento será lo suficientemente detallado de modo que las pruebas de un dado IUT puedan ser repetidas sin el cambio significativo de resultados.

5.4.1 Etapas generales

Para llevar a cabo a verificación de conformidad de los BSP´s se ha de analizar la especificación BioAPI. En resumen a los pasos del apartado anterior, todo método aplicado debe incluir:

Documentación, categorías, objetivos, instrucciones de uso, resultados esperados, contraste de resultados…. y debe pasar por varias etapas:

1. Analizar la especificación BioAPI, y sacar los casos y declaraciones documentados

2. El CTS estará formado por ejecutables para los Tests

3. IUT será sometida a pruebas y se comprobará con resultados esperados

4. Se evaluaran los Test con criterios de fallo/acierto

Según refleja literalmente la norma:

El proceso de pruebas de conformidad implica tres fases:

a) La preparación para pruebas, que incluye el análisis del BCS, la preparación del Plan de Prueba de Conformidad, la selección y la configuración de la CTS, y la preparación del IUT.

b) La ejecución de prueba, que incluye la ejecución del CTS y los resultados observados. Los resultados de pruebas de conformidad se aplicarán sólo al IUT y ambiente para el cual las pruebas son realizadas.

c) La producción de un Informe de prueba, que incluye la grabación de todos los acontecimientos ocurridos durante la ejecución de cada caso de prueba, incluyendo todos los resultados de prueba y veredictos de prueba, así como la descripción del ambiente de prueba (el sistema operativo, la configuración de hardware, etc.).

NOTA: El apartado c) no se ha desarrollado en su totalidad en la aplicación, la cual se limita a dar una breve descripción de cada caso de prueba y su resultado

5.4.2 Requisitos de las pruebas de conformidad

Según estipula la **parte 1** de la norma **ISO/IEC FDIS 24709-1:2006(E):**

1 *"BioAPI conformance Test suite debe soportar uno o mas modelos de pruebas de conformidad. (ver 6.2) y debe poder ejecutar cualquier declaración valida de Test para el modelo de Test que soporta y que están escritas en la lengua de validación. "*

En nuestro caso únicamente se soporta un modelo: BioAPI Conformant Verification Engines.

2 *"BioAPI conformance Test suite debe poder verificar la corrección sintáctica de cualquier paquete (ver 7.1.6) que contenga declaraciones o actividades (o ambas)*

para cualquier modelo de Test de conformidad incluyendo modelos de Test que la implementación no soporta "

Los paquetes están incluidos en formato XML y en la lengua de validación para que el usuario pueda constatar si son conformes a la norma, pero no se realiza ninguna comprobación automática por parte del programa.

3 *"Por cada modelo de Test soportado, BioAPI conformance Test suite debe poder llevar a cabo las acciones necesarias para interactuar con una implementación bajo Test (IUT), haciendo llamadas a las funciones del estándar BioAPI expuestas por la IUT y recibiendo llamadas a las funciones de esta"*

4 *"BioAPI conformance Test suite debe producir un Test log (ver clausula 11) y un Test report (ver clausula 12) por cada implementación Testada."*

La aplicación desarrollada genera un Test report automáticamente, pero el log no se ha desarrollado en esta versión del programa. Queda abierto a futuras ampliaciones.

5 *"si un BioAPI conformance Test suite es incapaz de llevar a cabo un Test, este resultado debe ser grabado en el Test report en estos términos, antes que como de no conformidad de la IUT."*

Como corresponde con lo señalado en los paquetes de cada caso de prueba, estos casos quedan recogidos en el Test report como "undecided"

6 *"BioAPI conformance Test suite debe proporcionar el medio para un usuario para entrar en todos los datos necesarios como input de la prueba"*

Los inputs disponibles para el usuario, son el modo, los casos de prueba (en el modo manual) y la elección entre dos BSP

5.4.3 Descripción de la CTS

Esta herramienta consiste en un conjunto de pruebas, denominados "casos". Una serie de datos asociados a dichas pruebas y los procedimientos para realizarlas. Estas pruebas serán validadas según la especificación del estandar BioAPI.

El procedimiento de prueba describe como la ejecución de la CTS debe ser llevada a cabo y las directrices para el probador. Además, el procedimiento debería ser suficientemente detallado de modo que la validación de un caso dado en una IUT pueda ser repetida sin el cambio de resultados de prueba.

La CTS no puede requerir los parámetros que son opcionales en un estándar, pero esta podría incluir pruebas para aquellos. La CTS tiene una estructura jerárquica, en la cual un nivel importante es el caso de prueba. Cada caso de prueba normalmente tiene un solo objetivo de prueba, como la de verificación de que el IUT tiene una cierta capacidad requerida o expone un cierto comportamiento requerido. Cada caso de prueba consiste en lo siguiente:

a) Una descripción del objetivo de la prueba (p. ej., ¿que está siendo probado?, las condiciones, exigencias, o las capacidades que deben ser comprobadas por una determinada prueba)

b) Los criterios "pasarás/fallarás" para el caso de prueba

c) Una referencia a la cláusula en el estándar de especificación BioAPI de lo cual el caso de prueba es sacada

Cada caso de prueba proporciona el objetivo, resultados repetibles, inequívocos, y exactos de prueba. Cada caso de prueba, cuando es ejecutado sobre el IUT, generará una condición pasaré/fallaré. Asociado con cada grupo de prueba puede haber un objetivo de grupo de prueba. Los casos de prueba pueden ser también fraccionados en módulos usando subdivisiones llamadas pasos de prueba. En nuestro caso esta opción no se ha tomado.

El resultado del Test es la serie de los acontecimientos que ocurrieron durante la ejecución de un caso de prueba. Esto incluye toda la entrada y la salida del IUT en el punto de control y observación.

Un resultado previsto de Test es el que ha sido definido por el caso de prueba, p. ej., los eventos que ocurrieron durante la ejecución del caso de prueba. Un resultado previsto de Test siempre causa la asignación de un veredicto de Test que será uno de los siguientes:

Pasado: quiere decir que el resultado observado proporciona evidencia de conformidad respecto de los requisitos para ese caso particular

Fallido: quiere decir que el resultado observado demuestra inconformismo en lo que concierne a al menos una de las exigencias de

conformidad sobre las cuales el objetivo es enfocado o contiene al menos un evento inválido de prueba.

Indeciso: que el resultado observado de prueba es tal que ni un pase ni un veredicto de suspenso puede dar.

Resultado imprevisto de prueba: es el que no ha sido identificado por el Test; p. ej., los acontecimientos que ocurrieron durante la ejecución del Test no emparejaron ninguna secuencia de los acontecimientos definidos en el caso de prueba. Un resultado imprevisto de prueba debe conllevar la grabación de un error o una terminación de caso anormal de prueba para el Test. Un error es registrado si es detectado en el caso de prueba sí mismo o en su realización, p. ej., un caso de error en un Test ejecutable.

Una notación estandarizada de prueba es usada para la especificación de todas las declaraciones. La notación asume que la estructura interna o el código original de un IUT no están disponibles al probador. A continuación se muestra el Archivo XML de referencia para un desarrollo futuro de los log de los Test.

Annex C
(normative)
XML Schema for the Test log

```
<?XML version='1.0' encoding="utf-8"?>
<xs:schema XMLns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="conformance_Test_log" type="conformance_Test_log"/>
<xs:complexType name="conformance_Test_log">
<xs:sequence>
<xs:element name="TestingLaboratory" type="Contact"/>
<xs:element name="Vendor" type="Contact"/>
<xs:element name="Biometric_Product" type="Biometric_Product"/>
<xs:element name="CTS_ID" type="xs:string"/>
<xs:element name="Test_assertion" type="Test_assertion" />
<xs:group ref="func_inline_response_group"
minOccurs="0" maxOccurs="unbounded" />
</xs:sequence>
<xs:attribute name="date_time" type="xs:string" use="required"/>
<xs:attribute name="standard" type="xs:string" />
</xs:complexType>
<xs:group name="func_inline_response_group">
<xs:choice>
<xs:element name="function" type="function"/>
<xs:element name="inline_response" type="inline_response" />
</xs:choice>
```

```
</xs:group>
<xs:complexType name="Contact">
<xs:sequence>
<xs:element name="Name" type="xs:string"/>
<xs:element name="Address" type="Address" />
<xs:element name="Phone" type="xs:string"/>
<xs:element name="Email" type="xs:string" minOccurs="0" />
<xs:element name="Url" type="xs:string" minOccurs="0" />
</xs:sequence>
</xs:complexType>
<xs:complexType name="Address">
<xs:sequence>
<xs:element name="Street" type="xs:string"/>
<xs:element name="City" type="xs:string" />
<xs:element name="StateOrProvince" type="xs:string" minOccurs="0"/>
<xs:element name="ZipOrPostalCode" type="xs:string" />
<xs:element name="Country" type="xs:string" />
</xs:sequence>
</xs:complexType>
<xs:complexType name="Biometric_Product">
<xs:sequence>
<xs:element name="Description" type="xs:string"/>
</xs:sequence>
<xs:attribute name="Name" type="xs:string" use="required" />
<xs:attribute name="SerialNo" type="xs:string" use="required"/>
</xs:complexType>
<xs:complexType name="Test_assertion">
<xs:sequence>
<xs:element name="package_name" type="xs:string" />
<xs:element name="assertion_name" type="xs:string" />
<xs:element name="description" type="xs:string" />
<xs:element name="input" type="inputOutputType"
minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="inputOutputType">
<xs:attribute name="name" type="xs:NCName" use="required"/>
<xs:attribute name="value" type="xs:string" use="required"/>
</xs:complexType>
<xs:complexType name="function">
<xs:sequence>
<xs:element name="name" type="xs:string" />
<xs:element name="input" type="inputOutputType"
minOccurs="0" maxOccurs="unbounded" />
<xs:element name="output" type="inputOutputType"
minOccurs="0" maxOccurs="unbounded" />
<xs:element name="return" type="return" />
</xs:sequence>
<xs:attribute name="dir" type="calldirection" default="outgoing"/>
</xs:complexType>
<xs:simpleType name="calldirection">
<xs:restriction base="xs:token">
<xs:enumeration value="outgoing"/>
<xs:enumeration value="incoming"/>
</xs:restriction>
</xs:simpleType>
<xs:complexType name="return">
```

```
<xs:attribute name="value" type="xs:string" use="required"/>
</xs:complexType>
<xs:complexType name="inline_response">
<xs:sequence>
<xs:element name="asserted_condition" type="xs:string"/>
<xs:element name="conformance">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:enumeration value="error" />
<xs:enumeration value="pass"/>
<xs:enumeration value="fail"/>
<xs:enumeration value="undecided"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
</xs:sequence>
</xs:complexType>
        </xs:schema>
```

CAPÍTULO 6

Pruebas y desarrollo

Una vez presentada la parte fundamental del sistema, el "BSP Testing aplication", vamos a detallar los pasos para crearla y explicar la manera en que la norma rige el proceso. En este sexto capítulo se abordarán los criterios de diseño e implementación utilizados y se incluirá un manual de usuario que explique de forma clara y concisa el uso y funcionamiento de la aplicación.

6.1 Estructura de la aplicación

Como ya se ha mencionado en capítulos anteriores el funcionamiento del sistema que utilizamos lleva el siguiente orden: Aplicación→Framework→ BSP. Pero para realizar pruebas de conformidad a un BSP lo que se busca es realizar la comunicación entre Aplicación y BSP sin utilizar el Framework. Esto es la mayor singularidad de este proyecto, pues el Framework es el encargado de gestionar todo lo necesario para que la comunicación sea satisfactoria. En la figura 6.0 observamos la secuencia que las funciones siguen para llevar a cabo una operación biométrica

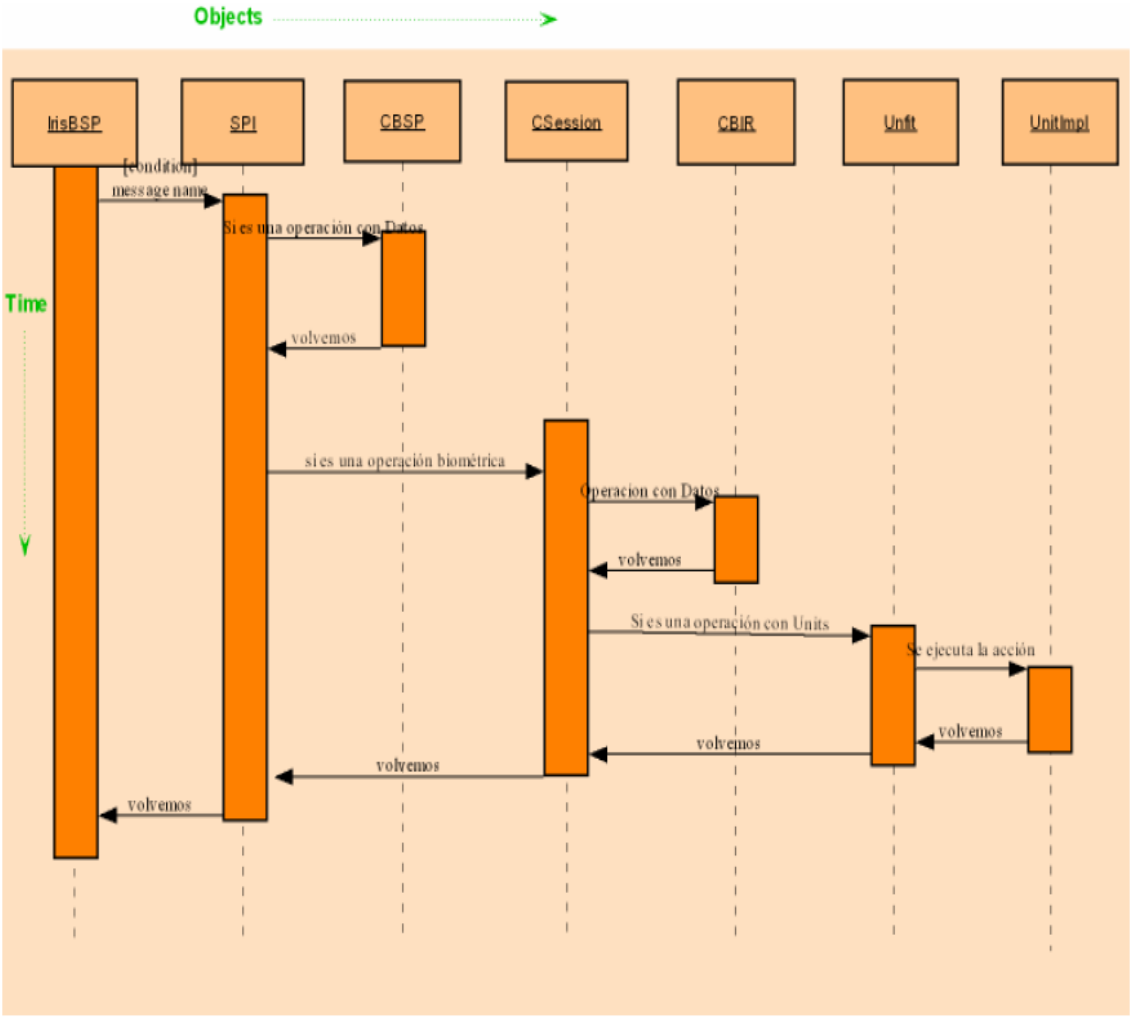


Figura 6.0) secuencia de ejecución de una función Biométrica

- Vamos a intentar exponer nuestro programa según su estructura y función.

6.1.1 Registrar BSP

Nuestra aplicación puede cargar o descargar un BSP en función del que se quiera Testar. La carga de un BSP se corresponde con el código de la figura 6.1

```
void __fastcall TForm1::BotonRegistrarClick(TObject *Sender)
{

    OpenDialogBSP->Execute();
//  char* bsppath = (char*)malloc(maxPath);
    bsppath = OpenDialogBSP->FileName.c_str();
    if(insertarBSP(bsppath) == true){
        MessageBox(NULL, "BSP cargado correctamente", "INFO", MB_OK);
        ActualizarListaBSP();
    }
    else{
        MessageBox(NULL, "El BSP no se ha cargado", "ERROR", MB_OK);
    }

}
```

Figura 6.1) codigo registrar BSP

Las posibilidades que ofrece la programación orientada a objetos con Borland C++ nos permiten desarrollar esta sencilla función. Si existe una ruta (bsppath) seleccionada por el usuario, se llama a la función insertarBSP y se realiza la carga en el sistema como vemos a continuación:

```
bool TForm1::insertarBSP(char *path){

lib= LoadLibrary(path);

//Si falla la carga de la libreria mostramos un mensaje y devolvemos false.

if (NULL == lib)
{
return false;
}


// obtenemos las funciones exportadas para la lectura/liberación del esquema del BSP

BioSPIRI_BSPGetSchema_PTR pGetSchema = (BioSPIRI_BSPGetSchema_PTR)
GetProcAddress(lib, "BioSPIRI_BSPGetSchema");

BioSPI_Free_PTR pFree = (BioSPI_Free_PTR) GetProcAddress(lib, "BioSPI_Free");
```

```
if (NULL == pGetSchema)

{

MessageBox(NULL, "Error: El BSP no exporta su esquema. Fallo de registro", "ERROR",
MB_OK);

FreeLibrary(lib);

return false;

}

// leemos el esquema

                                                                    BioAPI_BSP_SCHEMA
bspSchema;// = pGetSchema[0];

                                                                BioAPI_INSTALL_ERROR
Error;

                                                                    if (BioAPI_OK !=
pGetSchema(&bspSchema))
                                                                    {

                                                                MessageBox(NULL, "Error
en BioSPIRI_BSPGetSchema. Fallo de registro", "ERROR", MB_OK);
                                                                        FreeLibrary(lib);
                                                                        return false;
                                                                    }

// establecemos el path (se convierte de Windows LPSTR a BioAPI UTF8)

int MaxBytes=maxPath;
wchar_t * Dest;
wchar_t * DestWchar;
char * DestUtf8 = (char*)malloc(MaxBytes*sizeof(char));
int wide_char =  sizeof(Dest)*MaxBytes+1;
Dest = (wchar_t*)malloc(maxPath*sizeof(wchar_t)+1);
AnsiString Source = AnsiString(path);
DestWchar=StringToWideChar (Source, Dest, wide_char);
UnicodeToUtf8(DestUtf8, DestWchar, wide_char*sizeof(char));
bspSchema.Path=DestUtf8;
BioAPI_RETURN valorDevuelto;

// registrar el BSP

if (BioAPI_OK !=BioAPI_Util_InstallBSP(BioAPI_INSTALL_ACTION_INSTALL, &Error,
&bspSchema))

//BioAPI_Util_InstallBSP(BioAPI_INSTALL_ACTION_INSTALL, &Error, &bspSchema);

//if(BioAPI_OK != valorDevuelto)

{
MessageBox(NULL, "Error al registrar el BSP", "ERROR", MB_OK);
return false;
}

// liberar los miembros del esquema
```

```
                                                    if (pFree)
                                                    if
(bspSchema.BSPSupportedFormats)

                                                pFree(bspSchema.BSPSu
pportedFormats);

// descargar BSP DLL
                                                    FreeLibrary(lib);
                                                    //liberar memoria
                                                    free(DestUtf8);
                                                    free(Dest);

//Si la carga ha tenido exito devolvemos true.
                                                return true;
}
```

Una vez expuesta esta función es importante señalar que ya aparece aquí la formula que es utilizada para establecer la comunicación entre la aplicación y el BSP llamando directamente al interfaz SPI. En la parte inicial del código podemos ver:

```
// obtenemos las funciones exportadas para la lectura/liberación del esquema del BSP

BioSPIRI_BSPGetSchema_PTR pGetSchema = (BioSPIRI_BSPGetSchema_PTR)
GetProcAddress(lib, "BioSPIRI_BSPGetSchema");

BioSPI_Free_PTR pFree = (BioSPI_Free_PTR) GetProcAddress(lib, "BioSPI_Free");
```

Es decir para poder utilizar en nuestro código la función BioSPIRI_BSPGetSchema. Nos servimos básicamente de dos cosas. La función **GetProcAddress** que importará a nuestro sistema la función como una librería. Y **BioSPIRI_BSPGetSchema_PTR** que funciona como un puntero a una función. (Figura 6.2)

```
DLLSPI BioAPI_RETURN BioAPI  BioSPIRI_BSPGetSchema (
    OUT BioAPI_BSP_SCHEMA* pBSPSchema);

typedef BioAPI_RETURN (BioAPI * BioSPIRI_BSPGetSchema_PTR) (
    OUT BioAPI_BSP_SCHEMA* pBSPSchema);
```

Figura 6.2) declaración de BioSPIRI_BSPGetSchema_PTR

Esto implica que para cada función que queramos utilizar en nuestro código y que tenga que se directamente llamada al código del BSP sin el Framework, deberá ser previamente cargada de este modo en el sistema.

6.1.2 Eliminar BSP

Se llama a la función Eliminar BSP que gestiona la descarga de la manera que se observa en la figura 6.3. La función BioAPI_Util_InstallBSP es la que proporciona la información acerca de los BSP están instalados o no en el sistema. La función ActualizarListaBSP simplemente refresca lo que se muestra por pantalla con los cambios

```
// Eliminar un BSP de  la lista
void TForm1::EliminarBSP() {
    // get selection
    int nSeleccionado = ListaBSP->ItemIndex;

    if (LB_ERR == nSeleccionado){
        MessageBox(NULL, "Seleccione un BSP de la lista", "ERROR", MB_OK);
        return;
    }

    BioAPI_INSTALL_ERROR Error;

    // borrar (unregister)
    if (BioAPI_OK != BioAPI_Util_InstallBSP(BioAPI_INSTALL_ACTION_UNINSTALL,
    &Error, &m_pBSPs[nSeleccionado]))
        MessageBox(NULL, "Error al eliminar el BSP", "ERROR", MB_OK);
    else
        MessageBox(NULL, "BSP correctamente eliminado", "INFO", MB_OK);

    ActualizarListaBSP();
}
```

Figura 6.3) función EliminarBSP

6.1.3 Preparación necesaria para las pruebas

El método de comunicación que se ha comentado en la función *cargar BSP* no ha resultado del todo satisfactorio ya que no se ha conseguido que funcione correctamente con La función BioSPI_BSPLoad. En este caso puntual parece ser que es necesario un mayor control ya que el Load es el que inicia todo el proceso en la comunicación con un BSP y el Framework realiza un mayor trabajo en él sin que esté exactamente estudiado todavía, al menos con la documentación de que disponemos a estas alturas. Esto ha provocado que

excepciones relativas problemas de memoria saltasen e impidiesen el correcto funcionamiento de las pruebas. Por ello para poder desarrollar la aplicación que era el principal objetivo se optó por dejar el control de esta llamada en manos del Framework, con una excepción.

La primera prueba que se realiza: *//8.3 Assertion 1a - BioSPI_BSPLoad_InvalidUUID//* donde se introduce un UUID del BSP invalido como es: *const BioAPI_UUID *BSPUuid= {00000000-0000-0000-0000-000000000000};* y que debido a que el UUID no es correcto probablemente no inicie los procesos que generan el error. A continuación se muestra el código que corresponde con las funciones necesarias de ***InicializarBioAPI()*** y ***Bio_Load(BioAPI_UUID *uuid***

```
bool InicializarBioAPI(){

                                                    bool ret = false;
                                                    BioAPI_RETURN rc;
                                                    int fugamem;
//Variable auxiliar para evitar posibles fugas de memoria.

                                                    // Inicializamos el
Framework de bioAPI y contrastamos que la version es la
                                                    // misma que la instalada
en el sistema.
                                                    rc =
BioAPI_Init(BIOAPI_REQUIRED_VERSION);
                                                    if (rc != BioAPI_OK)
                                                        ret = false;

                                                    //Obtenemos la
informacion de los BSP instalados en el sistema. En m_pBSPs tendremos un
                                                    // puntero a un array con
los esquemas de los BSP instalados, y en m_nBSPsInstalled
                                                    // el numero de BSP
instalados.
                                                    rc =
BioAPI_EnumBSPs(&m_pBSPs,&m_nBSPsInstalled);
                                                    if (rc != BioAPI_OK)
                                                      ret  =false;

                                                    ret =true; //todo ha ido OK
                                                    return ret;

}

typedef struct SensorDetectionContext {
                                                    BioAPI_UNIT_ID unitID;
                                                    BioAPI_BOOL
foundSensor;
} SensorDetectionContext;

BioAPI_RETURN BioAPI SensorDetectionEventHandler(
 const BioAPI_UUID *BSPUuid,
 BioAPI_UNIT_ID UnitID,
```

```
 void* AppNotifyCallbackCtx,
 const BioAPI_UNIT_SCHEMA *UnitSchema,
 BioAPI_EVENT EventType);

BioAPI_RETURN BioAPI SensorDetectionEventHandler(

 const BioAPI_UUID *BSPUuid,
 BioAPI_UNIT_ID UnitID,
 void* AppNotifyCallbackCtx,
 const BioAPI_UNIT_SCHEMA *UnitSchema,
 BioAPI_EVENT EventType) {
 SensorDetectionContext *context = (SensorDetectionContext *) AppNotifyCallbackCtx;

                                                    if (EventType ==
BioAPI_NOTIFY_INSERT &&
                                                        UnitSchema != NULL
&&
                                                        UnitSchema-
>UnitCategory ==BioAPI_CATEGORY_SENSOR &&
                                                        context->foundSensor)
{
                                                        /* uses the first sensor
found */
                                                                    context-
>unitID = UnitID;
                                                                    context-
>foundSensor = BioAPI_TRUE;
                                                         }
                                                    return BioAPI_OK;

}
bool Bio_Load(BioAPI_UUID *uuid){

                                                    BioAPI_RETURN
bioReturn;
                                                    BioAPI_EventHandler
m_EventHandler;
                                                    SensorDetectionContext
context;
                                                    context.unitID = 0;
                                                    context.foundSensor =
BioAPI_FALSE;

                                                    m_EventHandler =
SensorDetectionEventHandler;

                                                    bioReturn =
BioAPI_BSPLoad(uuid, NULL, &context);


                                                    if(BioAPI_OK !=
bioReturn)
                                                    {
                                                        return 1;
                                                    }

                                                    return 0;}
```

Es importante reseñar que las funciones básicas para que el Framework lleve el control de las llamadas a la función BioSPI_BSPLoad son ***BioAPI_RETURN BioAPI_Init(BioAPI_VERSION Version)*** y ***BioAPI_RETURN BioAPI BioAPI_Terminate (void)***. La primera inicializa el Framework y verifica que la versión de BioAPI que solicita la aplicación es compatible con la que esta instalada en el sistema. La segunda finaliza el uso del Framework por parte de la aplicación. De esta manera el Framework borra todos los estados y recursos internos asociados con dicha aplicación.

Como BioAPI_Terminate() no es llamada dentro del Bio_Load ni de InicializarBioAPI esta llamada se realiza en cada uno de los casos de prueba, mientras que BioAPI_Init(BioAPI_VERSION Version) queda gestionada junto con los demás recursos necesarios con una única llamada al InicializarBioAPI() también en cada caso de prueba. Todas las demás llamadas a funciones del SPI que se realicen ya no serán gestionadas por el Framework.

Casos de prueba:

Dado que nuestra subclase de BSP pertenece al modelo nº 4. BioAPI Conformant Verification Engines. Según la norma estipula deben ser sometidas a pruebas de conformidad las siguientes funciones numeradas en paquetes ("packages") que contienen la descripción de la prueba y su desarrollo en documento XML.

ejemplo:

1a *BioSPI_BSPLoad_InvalidUUID* 020e90c8-0c19-1085-ab54-0002a5d5fd2e

1a = numero de aserción
BioSPI_BSPLoad_InvalidUUID = nombre de la aserción
020e90c8-0c19-1085-ab54-0002a5d5fd2e = package

1a *BioSPI_BSPLoad_InvalidUUID* 020e90c8-0c19-1085-ab54-0002a5d5fd2e
1b *BioSPI_BSPLoad_ValidParam* 01f6c6f0-0c19-1085-97fe-0002a5d5fd2e

2a *BioSPI_BSPUnload_ValidParam* 01661010-0c22-1085-8688-0002a5d5fd2e
2b *BioSPI_BSPUnload_InvalidUUID* 01c2e5b0-0c3b-1085-b31d-0002a5d5fd2e
2c *BioSPI_BSPUnload_UnmatchedLoad* 02f6c618-0c23-1085-ba89-0002a5d5fd2e
2d *BioSPI_BSPUnload_Confirm* 03daf040-0c3b-1085-a9fd-0002a5d5fd2e

3a *BioSPI_BSPAttach_ValidParam* 00ae6488-0c3d-1085-9912-0002a5d5fd2e
3b *BioSPI_BSPAttach_InvalidUUID* 049cc170-0c5f-1085-981f-0002a5d5fd2e
3c *BioSPI_BSPAttach_InvalidVersion* 0052ac10-0c60-1085-9883-0002a5d5fd2e
3d *BioSPI_BSPAttach_InvalidBSPHandle* 03826830-0c57-1085-bfb0-0002a5d5fd2e

4a *BioSPI_BSPDetach_ValidParam* 00e0d2b0-0c7a-1085-b8ac-0002a5d5fd2e
4b *BioSPI_BSPDetach_InvalidBSPHandle* 0434c458-0c79-1085-9f2c-0002a5d5fd2e
4c *BioSPI_BSPDetach_Confirm* 002e7e58-0c78-1085-9e1d-0002a5d5fd2e

5a *BioSPI_FreeBIRHandle_ValidParam* 0280a7d0-0c80-1085-a9a0-0002a5d5fd2e

5b *BioSPI_FreeBIRHandle_InvalidBSPHandle* 047aed48-0c80-1085-898b-0002a5d5fd2e
5c *BioSPI_FreeBIRHandle_InvalidBIRHandle* 018e6c18-0c9c-1085-afdf-0002a5d5fd2e

6a *BioSPI_GetBIRFromHandle_ValidParam* 0460b658-0cb4-1085-a304-0002a5d5fd2e
6b *BioSPI_GetBIRFromHandle_InvalidBSPHandle* 02445668-0cc5-1085-a3ac-0002a5d5fd2e
6c *BioSPI_GetBIRFromHandle_InvalidBIRHandle* 0194a9c0-0cc7-1085-8780-0002a5d5fd2e
7a *BioSPI_GetHeaderFromHandle_ValidParam* 027a7db0-0cc7-1085-9391-0002a5d5fd2e
7b *BioSPI_GetHeaderFromHandle_InvalidBSPHandle* 057e0d38-0ccd-1085-83b8-0002a5d5fd2e
7c *BioSPI_GetHeaderFromHandle_InvalidBIRHandle* 02195e68-0cce-1085-a46f-0002a5d5fd2e
7d *BioSPI_GetHeaderFromHandle_BIRHandleNotFreed* 01cc0988-0ccf-1085-a367-0002a5d5fd2e
8a *BioSPI_EnableEvents_ValidParam* 0333f628-0ccf-1085-aceb-0002a5d5fd2e
8b *BioSPI_EnableEvents_InvalidBSPHandle* 04ed0838-0ccf-1085-b64e-0002a5d5fd2e

10a *BioSPI_CreateTemplate_PayloadSupported* 04a01118-0cf9-1085-96d4-0002a5d5fd2e
10b *BioSPI_CreateTemplate_BIRHeaderQuality* 00b5c728-0cfb-1085-8969-0002a5d5fd2e
10c *BioSPI_CreateTemplate_OutputBIRDataType* 0193c730-0cf9-1085-b0a3-0002a5d5fd2e
10d *BioSPI_CreateTemplate_OutputBIRPurpose* 03dbdaa0-0cf2-1085-99ed-0002a5d5fd2e
10e *BioSPI_CreateTemplate_InputBIRDataType*6d543ea0-2ce9-11d9-9669-0800200c9a66
10f *BioSPI_CreateTemplate_Inconsistent_Purpose* 28ec1620-e995-11d9-b1d1-0002a5d5c51b

11a *BioSPI_Process_ValidParam* 4ec34700-e9a0-11d9-8fc8-0002a5d5c51b
11b *BioSPI_Process_BIRHeaderQuality* 211668e0-e9a6-11d9-bcc8-0002a5d5c51b
11c *BioSPI_Process_OutputBIRPurpose* e1bb4f20-ed61-11d9-9344-0002a5d5c51b
11d *BioSPI_Process_BuildsProcessedBIR* fce6540-ed66-11d9-9618-0002a5d5c51b
11e *BioSPI_Process_InputBIRDataType* 3cf96080-ed6b-11d9-9acf-0002a5d5c51b

12a *BioSPI_VerifyMatch_ValidParam* 688aad60-ee30-11d9-a62c-0002a5d5c51b
12b *BioSPI_VerifyMatch_Payload* 692ebe20-ee47-11d9-bd34-0002a5d5c51b
12c *BioSPI_VerifyMatch_Inconsistent_Purpose* 9108ec70-2e9b-11d9-9669-0800200c9a66

Estas aserciones de prueba están incluidas en documentos XML recogidos en la carpeta aserciones que se incluye junto con la aplicación, y se pueden encontrar en la norma en la Parte 2: **Test Assertions for Biometric Service Providers.**

6.1.4 BCS

Respecto a la **BCS**, en nuestro caso los elementos opcionales, por norma general han sido Inicializados a NULL y están señalados como OPTIONAL en su declaración. Elementos condicionales no han sido utilizados. Como se vio en el capítulo 5 Para alcanzar pruebas de conformidad creíbles, el resultado de ejecutar un caso de prueba sobre una IUT debería ser el mismo siempre que sea realizado. Se comprobó que en nuestra aplicación la llamada a ciertas funciones no siempre devolvía los mismos resultados teniendo los mismos elementos de entrada y por ello se puede observar en el código frecuentes llamadas a la función *sleep()* . Básicamente el error se produce por un fallo de los tiempos de ejecución de la herramienta de desarrollo. Lo que conseguimos con esta función

fue ralentizar esos tiempos de manera que el código pasase por todas sus líneas sin excepción y así se solucionó el problema.

A continuación se detalla el código inicial previo a las pruebas donde se importan las funciones y se declaran los elementos necesarios para cada aserción. Las funciones no es necesario importarlas más que una vez, por lo tanto aparecen la primera vez que la función en cuestión vaya a ser utilizada y posteriormente tan solo habrá que llamarlas a través del puntero. Están señaladas en negrita, así como los elementos que han sido necesarios declarar para cada función. Una parte importante es la de rellenar el BIR que ha sido necesario para poder hacer llamadas a ciertas funciones. El código esta separado por el número de aserción en que aparece.

```
//DECLARACIONES NECESARIAS PARA TODAS//
//LAS PRUEBAS //


//1a

BioSPI_BSPLoad_PTR pLoad = (BioSPI_BSPLoad_PTR)GetProcAddress(lib,
"BioSPI_BSPLoad");

const BioAPI_UUID *BSPUuid= {00000000-0000-0000-0000-000000000000};
BioSPIRI_EVENTHANDLER BioAPINotifyCallback=
(BioSPIRI_EVENTHANDLER)malloc(sizeof(BioSPIRI_EVENTHANDLER));
BioSPI_BFP_ENUMERATION_HANDLER BFPEnumerationHandler;
BioSPI_MEMORY_FREE_HANDLER MemoryFreeHandler=
(BioSPI_MEMORY_FREE_HANDLER)malloc(sizeof(BioSPI_MEMORY_FREE_HA
NDLER));

//2a

BioSPI_BSPUnload_PTR pUnLoad = (BioSPI_BSPUnload_PTR)
GetProcAddress(lib, "BioSPI_BSPUnload");

//2d

BioSPI_BSPAttach_PTR pAttach = (BioSPI_BSPAttach_PTR)
GetProcAddress(lib, "BioSPI_BSPAttach");

BioAPI_VERSION Version=
(BioAPI_VERSION)(BIOAPI_REQUIRED_VERSION);
const BioAPI_UNIT_LIST_ELEMENT
*UnitList=(BioAPI_UNIT_LIST_ELEMENT*)
malloc(sizeof(BioAPI_UNIT_LIST_ELEMENT));
```

```
uint32_t NumUnits={0};
BioAPI_HANDLE BSPHandle={1};

//3d

BioAPI_HANDLE BSPHandleMalo={-23450};

//4a

BioSPI_BSPDetach_PTR pDetach = (BioSPI_BSPDetach_PTR)
GetProcAddress(lib, "BioSPI_BSPDetach");

//4c

BioSPI_Enroll_PTR pEnroll = (BioSPI_Enroll_PTR) GetProcAddress(lib,
"BioSPI_Enroll");

IN BioAPI_BIR_PURPOSE Purpose=(BioAPI_BIR_PURPOSE)
(BioAPI_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY);
IN BioAPI_BIR_SUBTYPE Subtype={0};
IN const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputFormat=
(BioAPI_BIR_BIOMETRIC_DATA_FORMAT*)malloc(sizeof(BioAPI_BIR_BIOME
TRIC_DATA_FORMAT));
IN OPTIONAL const BioAPI_INPUT_BIR *ReferenceTemplate=
(BioAPI_INPUT_BIR *)malloc(sizeof(BioAPI_INPUT_BIR));
OUT BioAPI_BIR_HANDLE * NewTemplate=(BioAPI_BIR_HANDLE*)
malloc(sizeof(BioAPI_BIR_HANDLE));
IN OPTIONAL BioAPI_DATA *Payload=(BioAPI_DATA*)
malloc(sizeof(BioAPI_DATA));
IN int32_t Timeout={15000};
OUT OPTIONAL BioAPI_BIR_HANDLE * AuditData=
(BioAPI_BIR_HANDLE*) malloc(sizeof(BioAPI_BIR_HANDLE));
OUT OPTIONAL BioAPI_UUID *TemplateUUID=(BioAPI_UUID*)
malloc(sizeof(BioAPI_UUID));

//5a

BioSPI_FreeBIRHandle_PTR pFreeBIRHandle = (BioSPI_FreeBIRHandle_PTR)
GetProcAddress(lib, "BioSPI_FreeBIRHandle");

BioSPI_GetBIRFromHandle_PTR pGetBIR =(BioSPI_GetBIRFromHandle_PTR)
GetProcAddress(lib,"BioSPI_GetBIRFromHandle");

BioAPI_BIR_HANDLE Handle=(BioAPI_BIR_HANDLE)
malloc(sizeof(BioAPI_BIR_HANDLE));
OUT BioAPI_BIR *BIR= (BioAPI_BIR *)malloc(sizeof(BioAPI_BIR));


//7a
```

```
BioSPI_GetHeaderFromHandle_PTR  pGetHeaderFH=
(BioSPI_GetHeaderFromHandle_PTR)GetProcAddress(lib,"BioSPI_GetHeaderF
romHandle");

OUT BioAPI_BIR_HEADER *Header=(BioAPI_BIR_HEADER *)
malloc(sizeof(BioAPI_BIR_HEADER));

//8a

BioSPI_EnableEvents_PTR pEnableEvents=
(BioSPI_EnableEvents_PTR)GetProcAddress(lib,"BioSPI_EnableEvents");

// probamos por ejemplo con la mascara BioAPI_NOTIFY_INSERT_BIT
IN BioAPI_EVENT_MASK Events={0x00000001};
BioAPI_UNIT_ID UnitID=(BioAPI_UNIT_ID)malloc(sizeof(BioAPI_UNIT_ID));
const BioAPI_UNIT_SCHEMA *UnitSchema=(BioAPI_UNIT_SCHEMA *)
malloc(sizeof(BioAPI_UNIT_SCHEMA));

//10a

BioSPI_Capture_PTR pCapture=
(BioSPI_Capture_PTR)GetProcAddress(lib,"BioSPI_Capture");

BioSPI_CreateTemplate_PTR pCreateTemplate=
(BioSPI_CreateTemplate_PTR)GetProcAddress(lib,"BioSPI_CreateTemplate");


OUT BioAPI_BIR_HANDLE *CapturedBIR1=(BioAPI_BIR_HANDLE *)
malloc(sizeof(BioAPI_BIR_HANDLE));
//OUT OPTIONAL BioAPI_BIR_HANDLE *AuditData --> NULL;
BioAPI_INPUT_BIR *CapturedBIR=(BioAPI_INPUT_BIR*)
malloc(sizeof(BioAPI_INPUT_BIR));
```

Importante rellenamos el BIR

```
BioAPI_BIR *BIR_TEMPLATE = (BioAPI_BIR *)malloc(sizeof(BioAPI_BIR));
CapturedBIR->InputBIR.BIR = BIR_TEMPLATE;
CapturedBIR->Form = 3;
BioAPI_DTG dtg = {BioAPIRI_NO_DATE_AVAILABLE,
BioAPIRI_NO_TIME_AVAILABLE};
BioAPI_BIR_BIOMETRIC_DATA_FORMAT g_Formats[] =
{{SAMPLE_FORMAT_OWNER, SAMPLE_FORMAT_TYPE}};

BIR_TEMPLATE->BiometricData.Data = "prueba";
```

```
BIR_TEMPLATE->BiometricData.Length = 6;
BIR_TEMPLATE->SecurityBlock.Data = "NULL";
BIR_TEMPLATE->SecurityBlock.Length = 4;
BIR_TEMPLATE->Header.HeaderVersion = BIOAPI_REQUIRED_VERSION;
BIR_TEMPLATE->Header.Type = BioAPI_BIR_DATA_TYPE_INTERMEDIATE;
BIR_TEMPLATE->Header.Format = g_Formats[0];
BIR_TEMPLATE->Header.Quality = BioAPIRI_QUALITY_UNSUPPORTED;
BIR_TEMPLATE->Header.Purpose = Purpose;
BIR_TEMPLATE->Header.FactorsMask =BioAPI_TYPE_IRIS;
BIR_TEMPLATE->Header.ProductID.ProductOwner = 3;
BIR_TEMPLATE->Header.ProductID.ProductType = 1;
BIR_TEMPLATE->Header.Subtype = BioAPI_NO_SUBTYPE_AVAILABLE;
BIR_TEMPLATE->Header.CreationDTG = dtg;
BIR_TEMPLATE->Header.SBFormat.SecurityFormatOwner=0;
BIR_TEMPLATE->Header.SBFormat.SecurityFormatType=0;


//10b

BioAPI_BIR_HANDLE Handle2=
(BioAPI_BIR_HANDLE)malloc(sizeof(BioAPI_BIR_HANDLE));

//10f

IN BioAPI_BIR_PURPOSE PurposeMalo=
(BioAPI_BIR_PURPOSE)(BioAPI_PURPOSE_VERIFY);

//10d

OUT BioAPI_BIR_HEADER *HeaderA=(BioAPI_BIR_HEADER *)
malloc(sizeof(BioAPI_BIR_HEADER));
OUT BioAPI_BIR_HEADER *HeaderB=(BioAPI_BIR_HEADER *)
malloc(sizeof(BioAPI_BIR_HEADER));

//11a

BioSPI_Process_PTR pProcess=(BioSPI_Process_PTR)GetProcAddress(lib,
"BioSPI_Process");

OUT BioAPI_BIR_HANDLE *ProcessedBIR=(BioAPI_BIR_HANDLE *)
malloc(sizeof(BioAPI_BIR_HANDLE));
OUT BioAPI_BIR_HANDLE *ProcessedBIR2=(BioAPI_BIR_HANDLE *)
malloc(sizeof(BioAPI_BIR_HANDLE));

//12a

BioSPI_VerifyMatch_PTR pVeriMatch=
(BioSPI_VerifyMatch_PTR)GetProcAddress(lib,"BioSPI_VerifyMatch");

IN BioAPI_FMR MaxFMRRequested=(BioAPI_FMR)malloc(sizeof(BioAPI_FMR));
```

```
OUT BioAPI_BOOL *Result=(BioAPI_BOOL *)malloc(sizeof(BioAPI_BOOL));
OUT BioAPI_FMR *FMRAchieved=(BioAPI_FMR *)malloc(sizeof(BioAPI_FMR));
IN const BioAPI_INPUT_BIR *ProcessedBIR12a=(BioAPI_INPUT_BIR *)
malloc(sizeof(BioAPI_INPUT_BIR));
IN const BioAPI_INPUT_BIR *ReferenceTemplate12a=(BioAPI_INPUT_BIR *)
malloc(sizeof(BioAPI_INPUT_BIR));

//12b

OUT BioAPI_BIR_HANDLE *ProcessedBIR12b=(BioAPI_BIR_HANDLE
*)malloc(sizeof(BioAPI_BIR_HANDLE));
IN const BioAPI_INPUT_BIR *ProcessedBIR12b2=(BioAPI_INPUT_BIR
*)malloc(sizeof(BioAPI_INPUT_BIR));

//12c

OUT BioAPI_BIR_HANDLE * NewTemplate2=(BioAPI_BIR_HANDLE
*)malloc(sizeof(BioAPI_BIR_HANDLE));
OUT OPTIONAL BioAPI_UUID * TemplateUUID2=(BioAPI_UUID
*)malloc(sizeof(BioAPI_UUID));
OUT OPTIONAL BioAPI_BIR_HANDLE * AuditData2=(BioAPI_BIR_HANDLE
*)malloc(sizeof(BioAPI_BIR_HANDLE));
IN OPTIONAL const BioAPI_INPUT_BIR * ReferenceTemplate2=
(BioAPI_INPUT_BIR *)malloc(sizeof(BioAPI_INPUT_BIR));
```

6.1.5 CTS

A continuación vamos a exponer los datos que se corresponderían con la CTS, es decir: los resultados esperados, descripción del objetivo y criterios y referencia a la cláusula, en cada prueba.

La suite de prueba se ejecuta como ya se ha explicado: se carga el BSP deseado en el sistema, y se elige realizar un Test automático que pasara todas las pruebas o manual donde elegiremos cual de ellas queremos realizar. Una vez hecho esto los resultados serán expuestos en el Test report y en caso de querer volver a realizar un Test no hay más que volver a seleccionar las características del modo en que hemos especificado y que mas adelante mostraremos gráficamente sobre las imágenes de la aplicación.

Los dos BSP utilizados para las pruebas ofrecen idénticos resultados por ello los datos representados a continuación son validos para ambos algoritmos.

- **8.3 Assertion 1a - BioSPI_BSPLoad_InvalidUUID**

Descripción: esta aserción prueba si llamando a la función BioSPI_BSPLoad con parámetros inválidos de entrada (invalid UUID) devuelve BioAPIERR_H_FRAMEWORK_INVALID_UUID
Resultado esperado: invalid UUID
Resultado obtenido: FAILED
Descripción de resultado: devuelve invalid input pointer.

- **8.4 Assertion 1b - BioSPI_BSPLoad_ValidParam**

Descripción: esta aserción prueba si llamando a la función BioSPI_BSPLoad con parámetros válidos de entrada devuelve BioAPI_OK.
Resultado esperado: BioAPI_OK.
Resultado obtenido: PASSED
Descripción de resultado: Devuelve BioAPI_OK

- **8.5 Assertion 2a - *BioSPI*_BSPUnload_ValidParam**

Descripción: esta aserción prueba si llamando a la función BioSPI_BSPUnLoad con parámetros válidos de entrada devuelve BioAPI_OK.
Resultado esperado: BioAPI_OK.
Resultado obtenido: PASSED
Descripción de resultado: Devuelve BioAPI_OK

- **8.6 Assertion 2b - BioSPI_BSPUnload_InvalidUUID**

Descripción: esta aserción prueba si llamando a la función BioSPI_BSPUnLoad con parámetros inválidos de entrada (invalid UUID) devuelve BioAPIERR_INVALID_UUID
Resultado esperado: BioAPIERR_INVALID_UUID
Resultado obtenido: FAILED
Descripción de resultado: devuelve invalid imput pointer.

- **8.7 Assertion 2c - BioSPI_BSPUnload_UnmatchedLoad**

Descripción: esta aserción prueba si llamando a la función BioSPI_BSPUnLoad sin haber hecho previamente la llamada al Load devuelve BioAPIERR_BSP_NOT_LOADED.
Resultado esperado: invalid UUID
Resultado obtenido: PASSED
Descripción de resultado: devuelve BioAPIERR_BSP_NOT_LOADED

- **8.8 Assertion 2d - BioSPI_BSPUnload_Confirm**

Descripción: esta aserción prueba si llamando a la función BioSPI_BSPUnLoad realmente se descarga el BSP
Resultado esperado: BioAPIERR_BSP_NOT_LOADED
Resultado obtenido: PASSED
Descripción de resultado: devuelve BioAPIERR_BSP_NOT_LOADED

- **8.9 Assertion 3a - BioSPI_BSPAttach_ValidParam**

Descripción: esta aserción prueba si llamando a la función BioSPI_BSPAttach con parámetros de entrada válidos devuelve BioAPI_OK
Resultado esperado: BioAPI_OK
Resultado obtenido: PASSED
Descripción de resultado: devuelve BioAPI_OK

- **8.10 Assertion 3b - *BioSPI_BSPAttach_InvalidUUID***

Descripción: esta aserción prueba si llamando a la función BioSPI_BSPAttach con un UUID no válido devuelve BioAPIERR_INVALID_UUID
Resultado esperado: invalid UUID
Resultado obtenido: FAILED
Descripción de resultado: devuelve invalid imput pointer.

- **8.11 Assertion 3c - *BioSPI_BSPAttach_InvalidVersion***

Descripción: esta aserción prueba si llamando a la función BioSPI_BSPAttach con una versión incompatible devuelve BioAPIERR_INCOMPATIBLE_VERSION
Resultado esperado: invalid UUID
Resultado obtenido: PASSED
Descripción de resultado: Devuelve BioAPIERR_INCOMPATIBLE_VERSION

- **8.12 Assertion 3d - *BioSPI_BSPAttach_InvalidBSPHandle***

Descripción: esta aserción prueba si llamando a la función BioSPI_BSPAttach con un BSP handle no válido devuelve un error
Resultado esperado: BioAPIERR_INVALID_BSP_HANDLE
Resultado obtenido: FAILED
Descripción de resultado: Devuelve BioAPI_OK

- **8.13 Assertion 4a - BioSPI_BSPDetach_ValidParam**

Descripción: esta aserción prueba si llamando a la función BioSPI_BSPDetach con un modulo handle válido devuelve BioAPI_OK.
Resultado esperado: BioAPI_OK.
Resultado obtenido: PASSED
Descripción de resultado: Devuelve BioAPI_OK.

- **8.14 Assertion 4b - *BioSPI_BSPDetach_InvalidBSPHandle***

Descripción: esta aserción prueba si llamando a la función BioSPI_BSPDetach con un modulo handle no válido devuelve un error
Resultado esperado: InvalidBSPHandle
Resultado obtenido: PASSED
Descripción de resultado: Devuelve InvalidBSPHandle

- **8.15 Assertion 4c - *BioSPI_BSPDetach_Confirm***

Descripción: esta aserción prueba si llamando a la función BioSPI_BSPDetach realmente se termina la sesión añadida (attach session)
Resultado esperado: la llamada a BioSPI_Enroll devuelve BioAPIERR_INVALID_BSP_HANDLE
Resultado obtenido: PASSED
Descripción de resultado: Devuelve BioAPIERR_INVALID_BSP_HANDLE

- **8.16 Assertion 5a - *BioSPI_FreeBIRHandle_ValidParam***

Descripción: esta aserción prueba si llamando a la función BioSPI_FreeBIRHandle con un BIR handle válido libera el BIR handle.
Resultado esperado: BioSPI_GetBIRFromHandle devuelve BioAPIERR_INVALID_BIR_HANDLE
Resultado obtenido: PASSED
Descripción de resultado: Devuelve BioAPIERR_INVALID_BIR_HANDLE

- **8.17 Assertion 5b - *BioSPI_FreeBIRHandle_InvalidBSPHandle***

Descripción: esta aserción prueba si llamando a la función BioSPI_FreeBIRHandle con un BSP handle no válido devuelve un error.
Resultado esperado: BioAPIERR_INVALID_BSP_HANDLE
Resultado obtenido: PASSED
Descripción de resultado: Devuelve BioAPIERR_INVALID_BSP_HANDLE

- **8.18 Assertion 5c - *BioSPI_FreeBIRHandle_InvalidBIRHandle***

Descripción: esta aserción prueba si llamando a la función BioSPI_FreeBIRHandle con un BIR handle no válido devuelve un error
Resultado esperado: BioAPIERR_INVALID_BIR_HANDLE
Resultado obtenido: PASSED
Descripción de resultado: Devuelve BioAPIERR_INVALID_BIR_HANDLE

- **8.19 Assertion 6a - *BioSPI_GetBIRFromHandle_ValidParam***

Descripción: esta aserción prueba si llamando a la función BioSPI_GetBIRFromHandle con parámetros validos devuelve BioAPI_OK.
Resultado esperado: BioAPI_OK.
Resultado obtenido: FAILED
Descripción de resultado: BioSPI_GetBIRFromHandle Devuelve BioAPIERR_INVALID_BIR_HANDLE

- **8.20 Assertion 6b - *BioSPI_GetBIRFromHandle_InvalidBSPHandle***

Descripción: esta aserción prueba si llamando a la función BioSPI_GetBIRFromHandle con BSP handle no válido devuelve un error
Resultado esperado: BioAPI_INVALID_BSP_HANDLE
Resultado obtenido: PASSED
Descripción de resultado: Devuelve BioAPI_INVALID_BSP_HANDLE

- **8.21 Assertion 6c - *BioSPI_GetBIRFromHandle_InvalidBIRHandle***

Descripción: esta aserción prueba si llamando a la función BioSPI_GetBIRFromHandle con BIR handle no válido devuelve un error
Resultado esperado: BioAPI_INVALID_BIR_HANDLE
Resultado obtenido: PASSED
Descripción de resultado: Devuelve BioAPI_INVALID_BIR_HANDLE

- **8.22 Assertion 7a - *BioSPI_GetHeaderFromHandle_ValidParam***

Descripción: esta aserción prueba si llamando a la función BioSPI_GetHeaderFromHandle con parámetros validos devuelve BioAPI_OK.
Resultado esperado: BioAPI_OK.
Resultado obtenido: FAILED
Descripción de resultado: BioSPI_GetHeaderFromHandle Devuelve BioAPI_INVALID_BIR_HANDLE

- **8.23 Assertion 7b *BioSPI_GetHeaderFromHandle_InvalidBSPHandle***

Descripción: esta aserción prueba si llamando a la función BioSPI_GetHeaderFromHandle con un modulo Handle no válido devuelve un error
Resultado esperado: BioAPI_INVALID_BSP_HANDLE
Resultado obtenido: PASSED
Descripción de resultado: Devuelve BioAPI_INVALID_BSP_HANDLE

- **8.24 Assertion 7cBioSPI_GetHeaderFromHandle_InvalidBIRHandle**

Descripción: esta aserción prueba si llamando a la función BioSPI_GetHeaderFromHandle con un BIR Handle no válido devuelve un error
Resultado esperado: BioAPI_INVALID_BIR_HANDLE
Resultado obtenido: PASSED
Descripción de resultado: Devuelve BioAPI_INVALID_BIR_HANDLE

- **8.25 Assertion 7dBioSPI_GetHeaderFromHandle_BIRHandleNotFreed**

Descripción: esta aserción prueba si después de llamar a la función BioSPI_GetHeaderFromHandle el BIR handle no ha sido liberado
Resultado esperado: BioAPI_OK.
Resultado obtenido: UNDECIDED
Descripción de resultado: GetHeaderFromHandledevuelve BioAPIERR_INVALID_BIR_HANDLE

- **8.26 Assertion 8a - *BioSPI_EnableEvents_ValidParam***

Descripción: esta aserción prueba BioSPI_EnableEvents con parámetros de entrada validos
Resultado esperado: BioAPI_OK.
Resultado obtenido: PASSED
Descripción de resultado: devuelve BioAPI_OK

- **8.27 Assertion 8b - *BioSPI_EnableEvents_InvalidBSPHandle***

Descripción: esta aserción prueba BioSPI_EnableEvents con un modulo Handle inválido
Resultado esperado: InvalidBSPHandle
Resultado obtenido: PASSED
Descripción de resultado: BioSPI_EnableEvents devuelve InvalidBSPHandle

- **8.32 Assertion 10a - *BioSPI_CreateTemplate_PayloadSupported***

Descripción: : esta aserción prueba BioSPI_CreateTemplate con parámetros válidos y un valor de Payload en este caso(0x0000008)
Resultado esperado: BioAPI_OK
Resultado obtenido: PASSED
Descripción de resultado: BioAPI_OK

- **8.33 Assertion 10b - *BioSPI_CreateTemplate_BIRHeaderQuality***

Descripción: esta aserción prueba BioSPI_CreateTemplate con parámetros válidos y un valor de quality esperado de entre 0 y 100
Resultado esperado: BioAPI_OK y quality entre 0 y 100
Resultado obtenido: FAILED
Descripción de resultado: devuelve BioAPI_OK pero el valor de quality no es correcto

- **8.34 Assertion 10c - *BioSPI_CreateTemplate_OutputBIRDataType***

Descripción: esta aserción prueba BioSPI_CreateTemplate con párametros válidos el nuevo Template BIR se espera que tenga el nivel de procesamiento
Resultado esperado: BioAPI_BIR_DATA_TYPE_PROCESSED
Resultado obtenido: FAILED
Descripción de resultado: no coincide con BioAPI_BIR_DATA_TYPE_PROCESSED

- **8.35 Assertion 10d - *BioSPI_CreateTemplate_OutputBIRPurpose***

Descripción: esta aserción prueba que el Purpose de CreateTemplate es el mismo que el del Captured BIR
Resultado esperado: igual Purpose
Resultado obtenido: PASSED
Descripción de resultado: valores de Purpose iguales

- **8.36 Assertion 10e - BioSPI_CreateTemplate_InputBIRDataType**

Descripción: esta aserción prueba CreateTemplate con un nivel de procesamiento no válido en el input BIR
Resultado esperado: un error
Resultado obtenido: UNDECIDED
Descripción de resultado: GetHeaderFromHandle devuelve BioAPIERR_INVALID_BIR_HANDLE

- **8.37 Assertion 10f - *BioSPI_CreateTemplate_Inconsistent_Purpose***

Descripción: : esta aserción prueba BioSPI_CreateTemplate con input BIR no valido en su valor de Purpose
Resultado esperado: BioAPIERR_INCONSISTENT_PURPOSE
Resultado obtenido: FAILED
Descripción de resultado: devuelve BioAPI_OK

- **8.38 Assertion 11a - *BioSPI_Process_ValidParam***

Descripción: esta aserción prueba la función BioSPI_Process con párametros validos y el nivel de procesamiento devuelto
Resultado esperado: BioAPI_OK y BioAPI_BIR_DATA_TYPE_PROCESSED
Resultado obtenido: PASSED
Descripción de resultado: BioAPI_OK y BioAPI_BIR_DATA_TYPE_PROCESSED

- **8.39 Assertion 11b - *BioSPI_Process_BIRHeaderQuality***

Descripción: esta aserción prueba la función BioSPI_Process con parámetros validos y un valor de quality esperado de entre 0 y 100
Resultado esperado: BioAPI_OK y Quality entre 0 y 100
Resultado obtenido: PASSED
Descripción de resultado: devuelve BioAPI_OK y Quality entre 0 y 100

- **8.40 Assertion 11c - *BioSPI_Process_OutputBIRPurpose***

Descripción: esta aserción prueba la función BioSPI_Process y que los valores de Purpose del processed BIR y del captured BIR son coincidentes.
Resultado esperado: BioAPI_OK y mismo Purpose
Resultado obtenido: FAILED
Descripción de resultado: devuelve purpose a y b distintos

- **8.41 Assertion 11d - BioSPI_Process_BuildsProcessedBIR**

Descripción: esta aserción prueba la función BioSPI_Process y el tipo de procesamiento
Resultado esperado: processed BIR.
Resultado obtenido: PASSED
Descripción de resultado: processed BIR.

- **8.42 Assertion 11e - BioSPI_Process_InputBIRDataType**

Descripción: esta aserción prueba la función BioSPI_Process con un nivel de Processed en el input BIR y si la llamada al process falla
Resultado obtenido: UNDECIDED
Descripción de resultado: GetBIRFromHandle: BioAPIERR_INVALID_BIR_HANDLE

- **8.43 Assertion 12a - *BioSPI_VerifyMatch_ValidParam***

Descripción: esta aserción prueba la función BioSPI_VerifyMatch con parámetros validos devuelve BioAPI_OK.
Resultado esperado: BioAPI_OK.
Resultado obtenido: FAILED
Descripción de resultado: BioAPIERR_INVALID_DATA

- **8.44 Assertion 12b - *BioSPI_VerifyMatch_Payload***

Descripción: esta aserción prueba que la función BioSPI_VerifyMatch soporta el uso de payload.
Resultado esperado: BioAPI_OK.
Resultado obtenido: FAILED
Descripción de resultado: BioAPIERR_INVALID_DATA

- **8.45 Assertion 12c - *BioSPI_VerifyMatch_Inconsistent_Purpose***

Descripción: esta aserción prueba que la función BioSPI_VerifyMatch devuelve error cuando tiene un purpose no válido en su input BIR
Resultado esperado: BioAPIERR_BSP_INCONSISTENT_PURPOSE
Resultado obtenido: UNDECIDED
Descripción de resultado: BioAPIERR_INVALID_DATA

Una vez expuestos estos datos correspondientes a la BCS y La CTS vamos a poner uno de los casos de prueba como referencia de código, ya que la estructura es la misma en todas las aserciones. Únicamente se llama diferentes funciones para su comprobación.

```
//8.37 Assertion 10f - BioSPI_CreateTemplate_Inconsistent_Purpose//


1. Solo ejecutamos el caso si está en modo automático (ComboBox1->ItemIndex == 1) O si es elegida como opción (CheckBox31->Checked == true) en el modo manual (CheckBox31->Checked == true)
```

```
if ( ( CheckBox31->Checked == true)&&(ComboBox1->ItemIndex == 0)|| (ComboBox1-
>ItemIndex == 1))

{
```

2. Llamamos al load, inicializamosBioAPI y cerramos el trabajo anterior del Framework.

```
BioAPI_RETURN bar10f = 0;

AnsiString rep35="iniciando: *ASSERTION 10f* -
BioSPI_CreateTemplate_Inconsistent_Purpose";
ListBox1->AddItem(rep35,NULL);

BioAPI_Terminate();
InicializarBioAPI();
Bio_Load(&m_pBSPs->BSPUuid);
```

3. Llamamos a la primera función en este caso el attach. De ahora en adelante los valores los guardamos en una variable (en este caso bar10f) para ir comprobando más facilmente al depurar que las funciones van devolviendo lo que debe

```
//llamada  attach

AnsiString rep35a="llamamos a la funcion Attach,Capture,y CreateTemplate";
ListBox1->AddItem(rep35a,NULL);

Sleep(200); //parece que con esto no da errores las lamadas a ciertas funciones

bar10f=pAttach(&m_pBSPs->BSPUuid,Version,UnitList,NumUnits, BSPHandle);
```

4. Primera comprobación de resultado intermedio. Si falla el programa, termina el caso de prueba y muestra el mensaje UNDECIDED, esto se hará con cada paso intermedio que marque la norma. Si pasa la prueba se llama a la siguiente funcion y asi sucesivamente

```
//bar10f=0;//para comprobar el else

if (bar10f!= BioAPI_OK) {
AnsiString repaf= "fallo de algun paso intermedio, resultado: UNDECIDED";
ListBox1->AddItem(repaf,NULL);}
else{
```

```
        bar10f=pCapture(BSPHandle,PurposeMalo,Subtype,OutputFormat,CapturedBIR1,Time
out,NULL);

//bar10f=0;//para comprobar el else

if (bar10f!= BioAPI_OK) {
AnsiString repag= "fallo de algun paso intermedio, resultado: UNDECIDED";
ListBox1->AddItem(repag,NULL);}

else{

bar10f=pCreateTemplate(BSPHandle,CapturedBIR,NULL,OutputFormat,NewTemplate,Payload
,NULL);
```

5. Mostramos el resultado esperado por pantalla y comprobamos la condición final que dará como resultado para el Test PASSED o FAILED

```
AnsiString rep35b="valor esperado: BioAPIERR_INCONSISTENT_PURPOSE";
ListBox1->AddItem(rep35b,NULL);

                                                                    if(0x1000115 !=
bar10f){

AnsiString rep35c="no coincide con el resultado esperado";
ListBox1->AddItem(rep35c,NULL);
AnsiString rep35d="FAILED";
ListBox1->AddItem(rep35d,NULL);
AnsiString rep35e="-----------------------------------------------------------------";
ListBox1->AddItem(rep35e,NULL);}

else{

AnsiString rep35f="PASSED";
ListBox1->AddItem(rep35f,NULL);
AnsiString rep35g="-----------------------------------------------------------------";
ListBox1->AddItem(rep35g,NULL);}


                                                                    }
```

6. Por ultimo realizamos las operaciones necesarias para dejar listo el código para una nueva prueba. En este caso es necesario llamar al Detach y Unload

```
 //detach

pDetach(BSPHandle);

//unload

pUnLoad(&m_pBSPs->BSPUuid);

}
```

6.1.6 Funciones implementadas en la aplicación

A continuación se reseña una breve descripción de cada una de las funciones que son utilizadas en los Test de conformidad de las aserciones.

- ***BioSPI_BSPLoad****(IN const BioAPI_UUID *BSPUuid,IN BioSPIRI_EVENTHANDLER BioAPINotifyCallback, IN BioSPI_BFP_ENUMERATION_HANDLER BFPEnumerationHandler, IN BioSPI_MEMORY_FREE_HANDLER MemoryFreeHandler);*

La llamada a esta función provoca la inicialización de un BSP, esta inicialización incluye registrar un manejador de eventos para el BSP.

- ***BioAPI_RETURN BioAPI BioAPI_BSPUnload*** *(const BioAPI_UUID *BSPUuid):*

Esta función provoca que las notificaciones de eventos no sean registrados. De esta forma si un BSP fue cargado con *BioSPI_BSPLoad* puede descargarse con la llamada a esta función

- ***BioSPI_BSPAttach*** *(IN const BioAPI_UUID *BSPUuid, IN BioAPI_VERSION Version, IN const BioAPI_UNIT_LIST_ELEMENT *UnitList, IN uint32_t NumUnits, IN BioAPI_HANDLE BSPHandle);*

Esta función inicializa una **nueva sesión** para el BSP. En primer lugar comprueba la validez de la versión de BSP (comprueba que sean la misma en el sistema y la aplicación). En esta llamada es cuando la aplicación proporciona una **lista de Units** que desea usar para el BSP. Una vez que se ha realizado un *BioSPI_BSPLoad* se puede llamar a esta función múltiples veces para un mismo BSP. De igual forma que en los casos anteriores, la llamada desde una aplicación a esta función implica la ejecución de la función correspondiente .

- ***BioSPI_BSPDetach*** *(BioAPI_HANDLE BSPHandle)*

Esta función libera los recursos establecidos en el *Attach()* del BSP, de esta forma, todos los recursos utilizados en la sesión del BSP se **liberan**. En especial los manejadores de BIR, BSP y de bases de datos.

- ***BioSPI_Enroll*** *(BioAPI_HANDLE BSPHandle, BioAPI_BIR_PURPOSE Purpose, BioAPI_BIR_SUBTYPE Subtype, const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputFormat, const BioAPI_INPUT_BIR *ReferenceTemplate, BioAPI_BIR_HANDLE *NewTemplate, const BioAPI_DATA *Payload, int32_t Timeout, BioAPI_BIR_HANDLE *AuditData, BioAPI_UUID *TemplateUUID):*

El propósito de esta función es crear un nuevo *Template* a partir de una captura realizada con el dispositivo. Este patrón pasará a ser almacenado para su posterior verificación. El BSP puede recibir opcionalmente un Template de Referencia (ReferenceTemplate) como patrón para crear el nuevo Template, que también podrá ser guardado en una base de datos (TemplateUUID). Si, en la etapa de reclutamiento, el BSP tiene que mostrar una interfaz gráfica al usuario, será el propio BSP el encargado de esta misión.

- ***BioSPI_FreeBIRHandle*** *(IN BioAPI_HANDLE BSPHandle, IN BioAPI_BIR_HANDLE Handle);*

El propósito de esta función es liberar el BIR handle conociendo el manejador del BSP.

- ***BioSPI_GetBIRFromHandle*** *(IN BioAPI_HANDLE BSPHandle, IN BioAPI_BIR_HANDLE Handle, OUT BioAPI_BIR *BIR);*

El propósito de esta función es obtener el BIR handle conociendo el manejador del BSP, y generar un BioAPI_BIR de salida.

- ***BioSPI_GetHeaderFromHandle*** *(IN BioAPI_HANDLE BSPHandle, IN BioAPI_BIR_HANDLE Handle, OUT BioAPI_BIR_HEADER *Header);*

Conociendo el manejador del BSP y el manejador del BIR se genera una cabecera de salida del BIR.

- ***BioSPI_EnableEvents*** *(IN BioAPI_HANDLE BSPHandle, IN BioAPI_EVENT_MASK Events);*

Conociendo el manejador del BSP y utilizando la mascara adecuada podemos habilitar o deshabilitar los eventos que nos interesen en una operación.

- ***BioSPI_Capture*** *(BioAPI_HANDLE BSPHandle, BioAPI_BIR_PURPOSE Purpose, BioAPI_BIR_SUBTYPE Subtype, const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputFormat, BioAPI_BIR_HANDLE *CapturedBIR, int32_t Timeout, BioAPI_BIR_HANDLE *AuditData)*

Esta función captura una nueva muestra a partir del dispositivo biométrico elegido y previamente añadido, dicha muestra podrá procesarse posteriormente (solo si es necesario) o ser utilizada como dato final.

- ***BioSPI_CreateTemplate*** *(BioAPI_HANDLE BSPHandle, const BioAPI_INPUT_BIR *CapturedBIR, const BioAPI_INPUT_BIR *ReferenceTemplate, const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputFormat, BioAPI_BIR_HANDLE *NewTemplate, const BioAPI_DATA *Payload, BioAPI_UUID *TemplateUUID)*

Esta función, básicamente, lo que hace es mantener un *template* que sirva como **patrón** con el cual comparar en futuras verificaciones o identificaciones. Este *template* puede construirse desde cero, mediante el uso de una nueva captura, o usar un *template* de referencia, al cual se le realizan ciertas.

- ***BioSPI_Process*** *(BioAPI_HANDLE BSPHandle, const BioAPI_INPUT_BIR *CapturedBIR, const BioAPI_BIR_BIOMETRIC_DATA_FORMAT *OutputFormat, BioAPI_BIR_HANDLE *ProcessedBIR)*

Esta función es la encargada de procesar los datos intermedios capturados mediante la llamada a *BioSPI_Capture()* para su posterior verificación o identificación. De esta manera logramos unos datos procesados (*ProcessedBIR*) que podrán ser recuperados por la aplicación mediante el uso de la función *BioSPI_GetBIRFromHnadle()*.

- ***BioSPI_VerifyMatch*** *(BioAPI_HANDLE BSPHandle, BioAPI_FMR MaxFMRRequested, const BioAPI_INPUT_BIR *ProcessedBIR, const BioAPI_INPUT_BIR *ReferenceTemplate, BioAPI_BIR_HANDLE *AdaptedBIR, BioAPI_BOOL *Result, BioAPI_FMR *FMRAchieved, BioAPI_DATA *Payload)*

Esta función realiza una comparación directa entre dos BIRs; el BIR procesado y el BIR de referencia. El primero de ellos surge tras la etapa de procesamiento descrita en el apartado anterior, mientras que el segundo es el dato de referencia que se almacena en la etapa de Enroll. La aplicación ha de especificar un valor de FMR como umbral para la verificación.

6.2 Interfaz

Las posibilidades que ofrece la programación orientada a objetos con Borland Builder 2006 son amplias. Nuestra aplicación, al menos en la fase en que esta desarrollada, no dispone de muchos elementos para interactuar con el usuario, debido a que el trabajo que realiza es tomar un BSP, realizar unas pruebas y mostrar resultados.

Se ha pretendido buscar una interfaz sencilla y útil, a la vez que se ha procurado evitar que cualquier información relevante que pueda proporcionar el programa no sea mostrada al usuario. A continuación pasamos a presentar el aspecto de nuestro “BSP Testing Aplication”

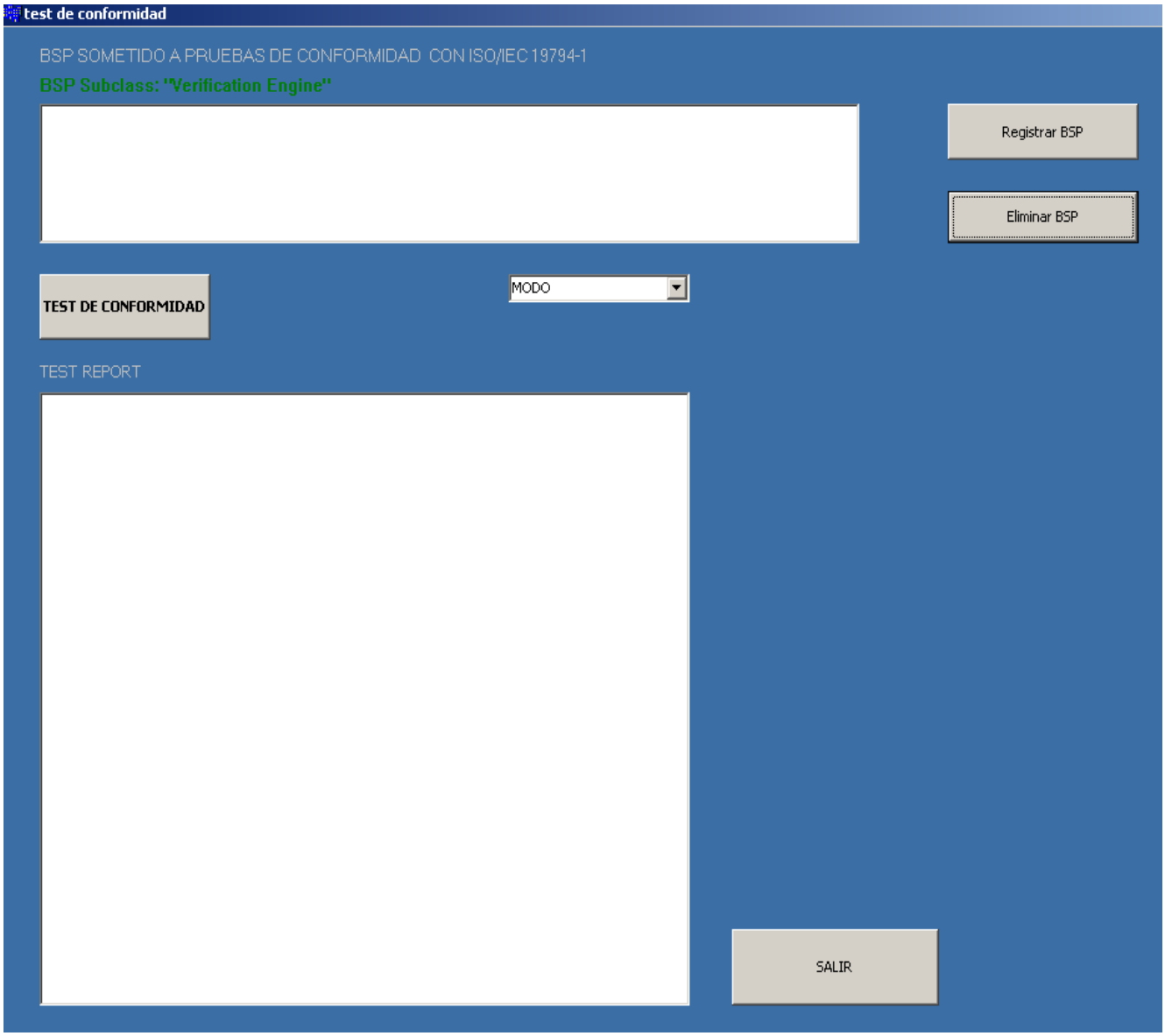


figura 6.4) Interfaz de usuario

Aquí podemos observar la distribución inicial. Tenemos una ventana donde se mostrara el BSP que se carga en el sistema. Tenemos una segunda ventana "TEST REPORT" donde los resultados de las pruebas serán mostrados una vez realizado el Test. Una ventana "combo" donde podemos elegir el modo: automático o manual, y los botones que realizan las operaciones que queramos en cada caso.

figura 6.5) Registrar BSP

Registrar BSP: nos abrirá una ventana donde podremos elegir el lugar de nuestro sistema donde tenemos almacenado el BSP y cargarlo en nuestra aplicación. El archivo que ha de ser cargado es el de extensión ".dll". Debido a posibles conflictos entre dos BSP, se aconseja cargar únicamente el BSP que vaya a ser Testado y nunca tener más de un BSP cargado en la aplicación cuando se quiera hacer un Test.

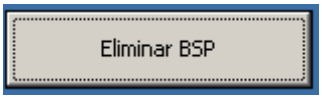


figura 6.6) Eliminar BSP

Eliminar BSP: tras seleccionar el BSP de la ventana con este botón lo descargamos de nuestra aplicación, pero no lo borramos del equipo.

figura 6.7) TEST DE CONFORMIDAD

TEST DE CONFORMIDAD: una vez cargado el BSP y elegido el modo que queremos ejecutar con este botón comienza el proceso y tras unos instantes dependiendo del número de pruebas ejecutadas los resultados serán mostrados en el Test report.

figura 6.8) SALIR

SALIR: cuando queramos abandonar el programa podemos recurrir a este botón que terminará los procesos activos en caso de que todavía se estén ejecutando. Se recomienda no usarlo a mitad de un Test.

Una de las opciones más importantes que el usuario puede elegir es la de Test automático o manual. En la ventana de modo, pinchando en el combo, podemos seleccionar uno de los dos.

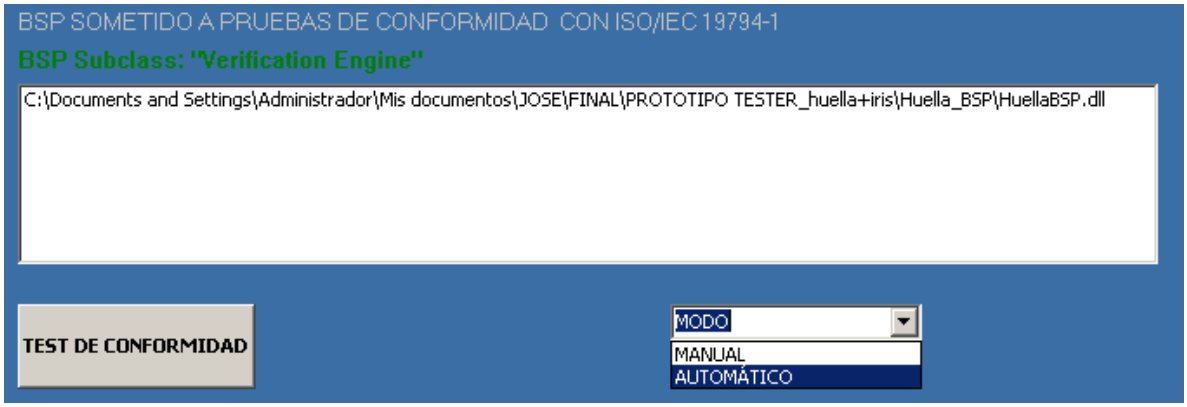


figura 6.9) elección del modo

En caso de que se quiera un Test automático, tras presionar el botón de Test el BSP será sometido a todas las pruebas estipuladas mostrado al finalizar sus resultados. Si lo que queremos es analizar los resultados de pruebas concretas deberemos elegir el modo manual y marcar haciendo clic la ventana correspondiente como se muestra en la figura 6.10

figura 6.10) elección de la prueba

TEST REPORT:

Aquí se muestran los resultados del siguiente modo: se anuncia el inicio del Test y el modo en que se realiza y posteriormente cada prueba incluyendo:

- nombre y número de la aserción
- principales funciones a las que se llama
- valor esperado de los resultados
- no coincidencia con lo esperado en caso de que sea así
- resultado obtenido: PASSED/FAILED

Podemos observarlo en la figura 6.11

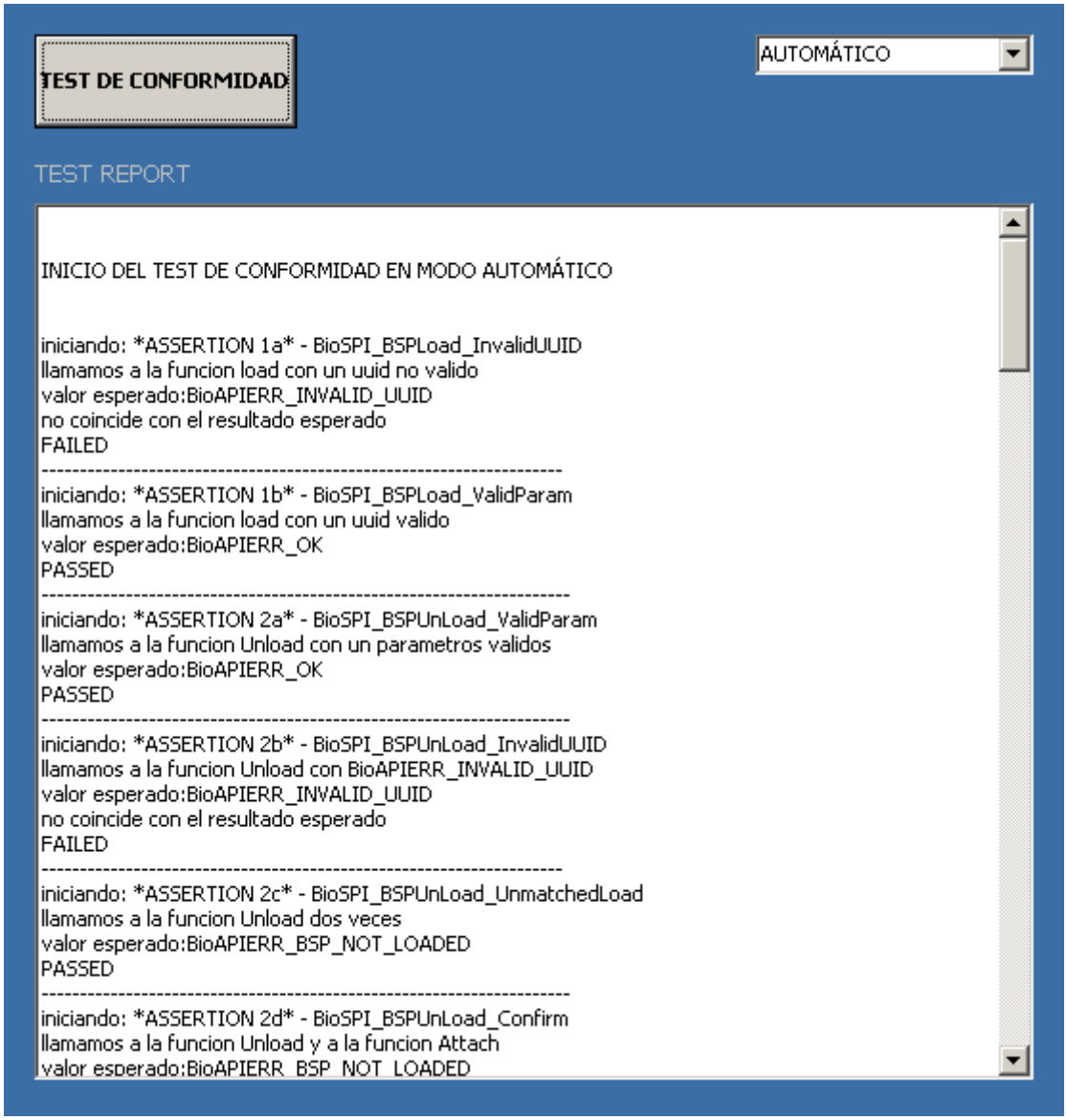


figura 6.11) resultados en el TEST REPORT

En caso de que la prueba no haya podido concluirse por que en el proceso falla algo se comunica y se da el mensaje de UNDECIDED como se observa en la figura 6.12

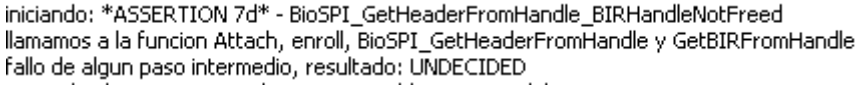


figura 6.12) resultado: UNDECIDED

En caso de elegir el modo manual solo se muestran los resultados correspondientes a las pruebas seleccionadas como podemos ver en la figura 6.13

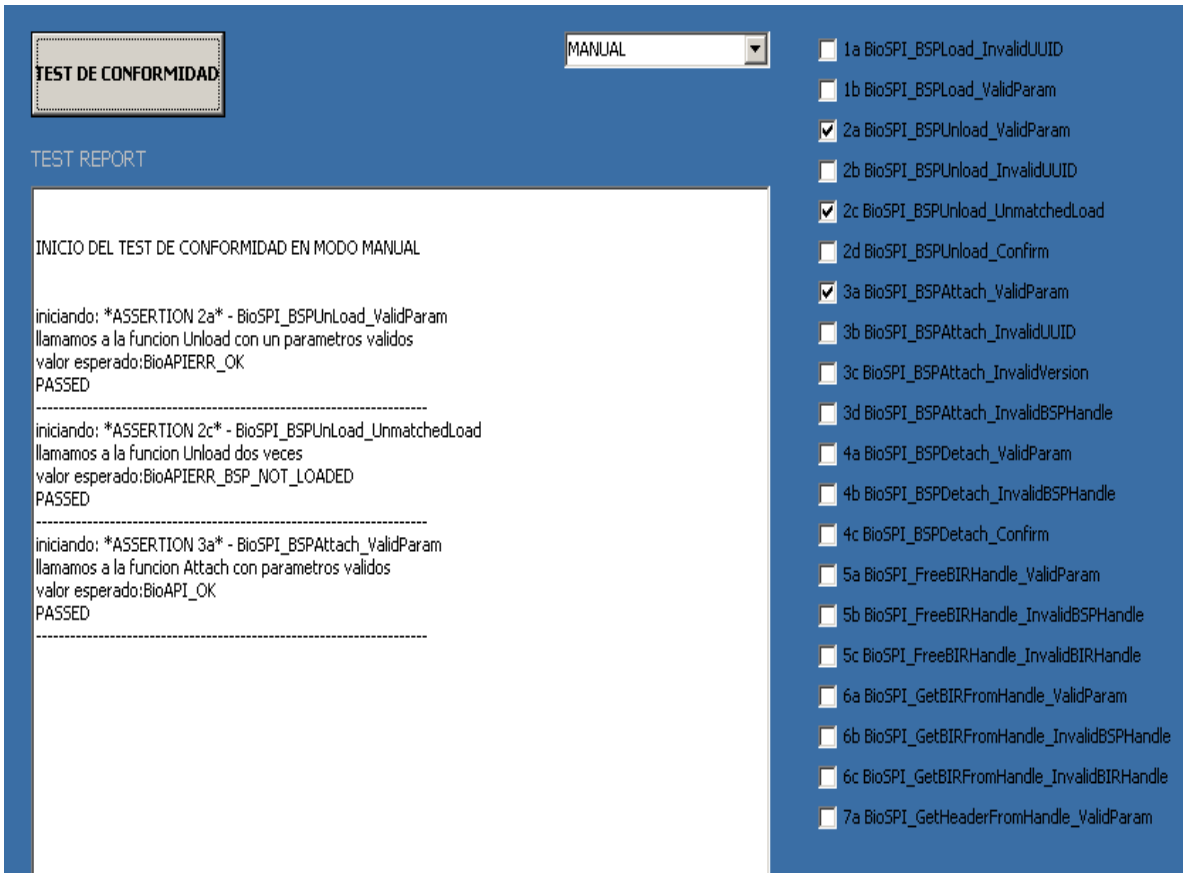
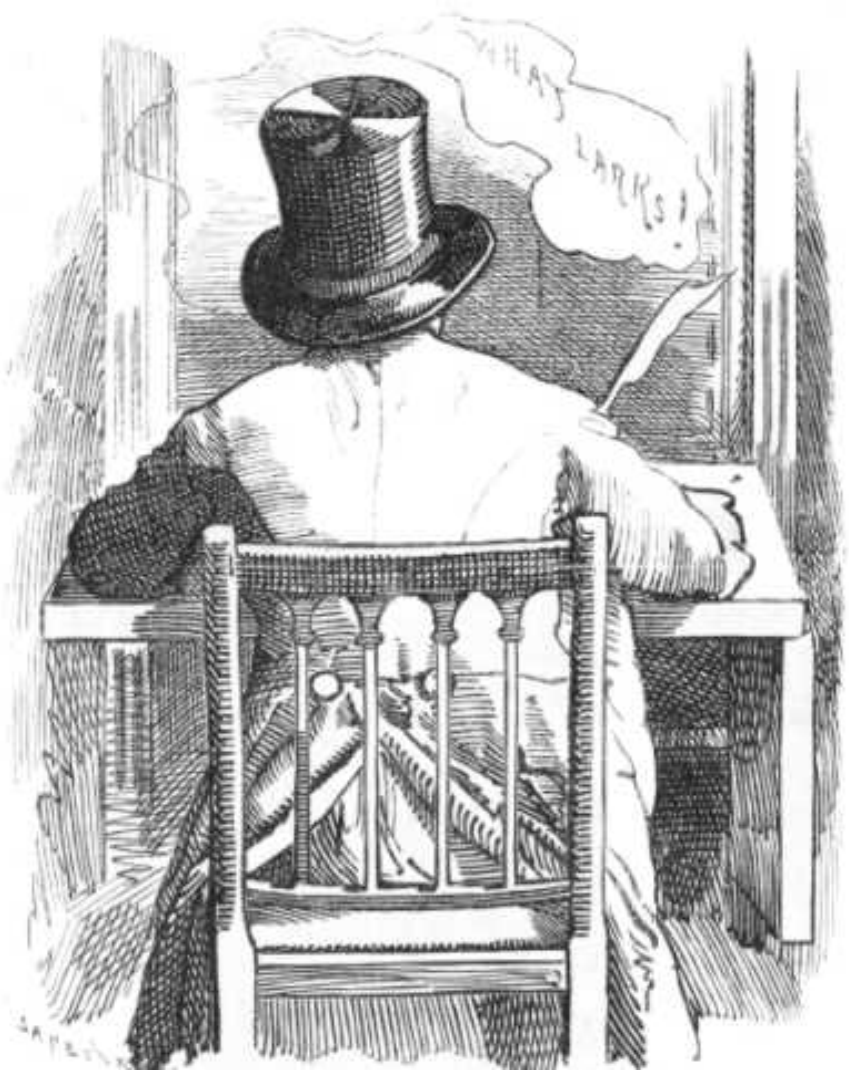


figura 6.13) TEST REPORT en modo manual

CAPÍTULO 7



Conclusiones

7.1 Conclusiones generales

El objetivo de este proyecto ha sido el desarrollo de una aplicación que sea capaz de realizar pruebas de conformidad a uno o varios BSP con respecto a la norma ISO/IEC 19794-1. Para llevarlo a cabo se han utilizado dos BSP basados en reconocimiento por huella e iris, y ha sido necesario realizar alguna modificación en los mismos para poder desarrollar el modelo de pruebas de manera útil para el cometido que nos ocupaba.

La división del trabajo ha sido básicamente en cuatro etapas: 1. Documentación y familiarización con las herramientas de trabajo 2. Búsqueda de un método de comunicación entre aplicación y BSP para la realización de las pruebas de conformidad 3. Programación conforme a las especificaciones de la norma de los casos de prueba. 4. análisis de datos y redacción del presente documento.

En la primera etapa fue fundamental una lectura extensa de las normas de conformidad y del modelo BioAPI que en un principio puede resultar equivoca, difusa o incluso inútil a ratos ya que hasta que no se empieza a trabajar directamente con la estructura de comunicación básica (aplicación-Framework-BSP) no se llega a entender bien de que modo funciona realmente. Con esto quiero decir que hasta el momento de comenzar a buscar el modo de comunicación directa entre la aplicación que se intentaba desarrollar y los BSP, saltándonos el Framework, no empecé a ver exactamente de que manera los distintos procesos informáticos consiguen establecer comunicación entre ellos, y lo que es mas importante: conseguir que lo hiciesen del modo que nosotros queríamos.

La norma de conformidad está dividida en cuatro partes de las cuales han sido necesarias las dos primeras más el texto de la especificación BioAPI. Estos tres documentos han sido un referente y un recurso continuamente consultado para llevar a cabo el desarrollo de la aplicación.

La segunda etapa, ha sido sin duda alguna la mas costosa y larga de todas. La especificación BioAPI 2.0 está prácticamente sin desarrollar en lo que conformidad se refiere y esto implica que los modos de comunicación particulares que las pruebas de conformidad implican no están fijados y/o probados. Hasta el momento solo existe una implementación de referencia y además es muy reciente, lo que provoca que la documentación existente sobre BioAPI 2.0, fuera la proporcionada por OSS.Nokalva y prácticamente nula. En la estructura normal de BioAPI, como hemos visto a lo largo del texto, la aplicación se comunica con los BSP a través del Framework, pero las pruebas de

conformidad exigen que nuestra aplicación, el "BSP Testing Aplication", haga directamente las llamadas al interfaz SPI saltándose al intermediario usual. Esto llevó numerosos intentos y pruebas hasta que se dio con la manera de llamar a las funciones del SPI y utilizar la información que proporcionaban sin tener que recurrir al Framework. A pesar de encontrar un modo de comunicación no se ha conseguido una comunicación totalmente satisfactoria sin el Framework en cuanto a lo que la función BioSPI_Load se refiere. Sin la mediación del citado componente en esta función, no se conseguía establecer un flujo de llamadas correcto y ha sido necesario llamar a BioAPI_Load y utilizar las funciones BioAPI_Terminate e InicializarBioAPI para que el Framework la gestionase y pudiesen ser realizadas todas las demás pruebas del modo en que necesitábamos.

Solucionado el asunto de la comunicación, el trabajo se "simplificó" y "rutinizó". Comenzó entonces una etapa en la que se iban siguiendo los pasos citados en la norma para comprobar cada una de las funciones necesarias para reclamar la conformidad. El mayor escollo de esta parte estuvo en declarar y definir todas aquellas variables y datos necesarios para hacer las pruebas. Esto provocó problemas con los datos utilizados en los BIR y deja planteada una duda sobre lo que la norma pide.

La subclase correspondiente a nuestro BSP coincide con el modelo: "BioAPI Conformant Verification Engines" que contiene el procesado biométrico y los algoritmos de comparación 1:1 pero que no realiza la captura biométrica. Esto implica que los datos de la captura que van dentro del BIR estaban vacíos a la hora de hacer de las pruebas y no se podían llevar a cabo, lo que hizo necesario modificar ligeramente los BSP para que llevasen una mínima información necesaria. Es en este punto donde surge la duda de si la norma puede pedir realizar pruebas sobre ciertas funciones que en esta clase de BSP no se pueden ejecutar al disponer de los datos necesarios.

El proceso descrito se llevó a cabo primero con el BSP de la huella dactilar y posteriormente de una manera casi idéntica con el de Iris. Una vez hecho esto la principal tarea fue adaptar gráficamente la aplicación a las necesidades del proceso facilitando su uso y proponiendo un modo automático que realiza todas las pruebas o un modo manual en el que el usuario puede elegir cuales son las que quiere llevar a cabo. Tras un control de errores y una organización básica del código el "trabajo de campo" finalizó. El uso de BioAPI según lo explicado en este proyecto puede parecer que plantea ciertos escollos o dificultades, pero se puede afirmar que tras un cierto "duro trabajo" proporciona múltiples ventajas, como: la reusabilidad, la seguridad, la separación de las capas de presentación y lógica, etc...

En lo que concierne a la redacción, lo principal ha sido intentar relatar de un modo claro cual ha sido el propósito y de que manera se ha conseguido, dando una información concreta de las herramientas que han sido necesarias para ello. La parte más trabajosa se la ha llevado el intento de elaborar un texto organizado y exento de cualquier intención de abrumar con datos o demasiadas explicaciones farragosas. Espero sinceramente haberlo conseguido.

7.2 Decisiones de Diseño

La primera decisión de diseño fue utilizar como referencia para la carga y descarga de los BSP en el "BSP Testing Aplication"un pequeño programa desarrollado por Raúl Alonso Moreno llamado *registrar BSP*. A partir de que el BSP estuviese o no cargado el sistema trataríamos de buscar la comunicación directamente con el.

Dado que el paso de los BSP por las pruebas requiere que sean añadidos a la aplicación y que habíamos utilizado como base el registrar BSP se desestimó una idea inicial de desarrollar una aplicación para cada BSP por separado. Con la funcionalidad de carga y descarga podemos utilizar el mismo interfaz para cada BSP que queramos someter al Test.

Según la norma con cada prueba ha de generarse un Test log, pero el principal objetivo de establecimiento de comunicación y análisis de las pruebas nos pareció más importante, lo que motivó que se hayan incluido los paquetes XML con los que se puede generar el Test log, pero no se gestiona por la aplicación.

La intención de hacer útil y sencillo el programa nos ha llevado a introducir los dos modos ya comentados anteriormente (manual y automático) que permiten realizar los Test de todas las pruebas o seleccionar una o varias pruebas concretas y ver los resultados.

7.3 Líneas Futuras

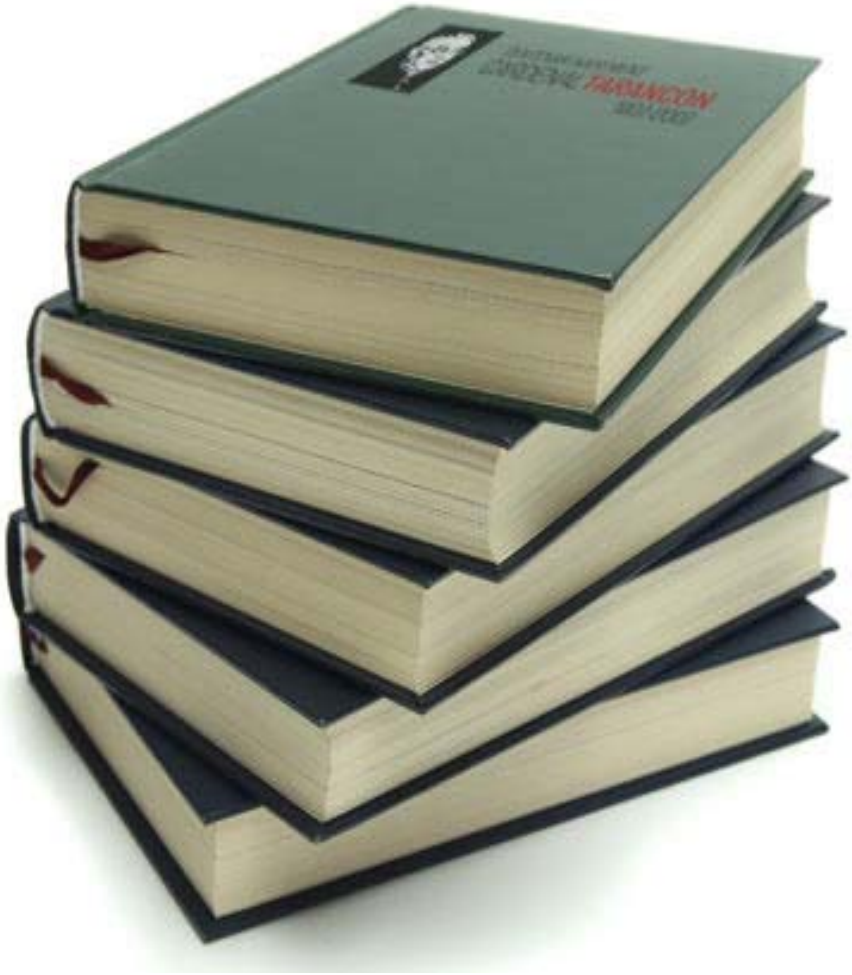
Existen varias mejoras posibles para continuar desarrollando el sistema elaborado. Como primera futura ampliación podemos citar el Test log mencionado en el apartado anterior y que la aplicación generaría con cada prueba (automática o manual). Los archivos XML deberían ser asociados a la información suministrada por el usuario de la aplicación y el proveedor de la IUT y generados con cada Test realizado.

Para llevar a cabo el tipo de comunicación que la norma requiere las DLL correspondientes a los BSP son añadidas al mismo proyecto en que la aplicación se desarrolla en C++ Builder. Este proceso hace necesarios ciertos pasos antes de cargar el BSP y Testar. La búsqueda de una mayor automatización y rapidez de este proceso podría suponer una gran mejora en este tipo de sistemas.

Así mismo conseguir que las llamadas que necesariamente en esta aplicación se hacen a través del Framework (como es lo referente al BioSPI_Load) se hagan directamente al interfaz SPI sería una importante aportación y adecuación a los dictados de la norma.

Por ultimo mencionar que este tipo de aplicaciones se deberían poder concebir para cualquier tipo de BSP por lo que ampliar el rango de acción a otras subclases y a datos de cualquier tipo supondría un importante avance.

BIBLIOGRAFÍA



A continuación se van a listar una serie de referencias a páginas Web o documentos extraídos de libros o trabajos que han sido de gran ayuda como consulta y como guía para la elaboración del proyecto.

REFERENCIAS

- **[NOK08]Página web de la empresa que ha desarrollado el Framework**

http://www.nokalva.com

- **[API08] Página Web de BioAPI:**

http://www.bioapi.org/

- **[ALO07] Raúl Alonso Moreno "Proyecto Fin de Carrera: adaptación de un sistema de identificación biométrico para cumplir la norma ISO/IEC 19784-1",2007**

- **[Bio08] Página Web de Biofoundry**

http://www.biofoundry.com/products/bioapi20_RI.html

- **[POR03] Javier Portillo. U. Carlos III "Sistemas de identificación Biométrica"elaborado para el *Centro de Difusión de Tecnologías ETSIT-UPM Diciembre, 2003***

- **[HOM08] Página web de la empresa desarrolladora de software biometrico homini**

http://www.homini.com/historia.htm

- **[ISO08] Página oficial de ISO.**

http://www.iso.org/iso/en/aboutiso/isomembers/index.html

- **[Fer06] Paloma Ferruelo Soler, "Proyecto Fin de Carrera: Identificación biométrica en dispositivos integrados", 2006**

- **[BAR08] Página web dedicada a la biometria del gobierno argentino**

http://www.biometria.gov.ar/index.php/documentos/60-reconocimiento-del-iris-

- **[KIM08] Página web del fabricante y distribuidor de sistemas biometricos kimaldi:**

http://www.kimaldi.com/area_de_conocimiento/biometria/que_es_la_biometria

- **[DIS08] Página web de la empresa española Disbyo que desarolla soluciones en el campo biométrico**

http://www.disbio.com/index.php?/Contenido/Informacion-biometria.html

- **[WIK08] Página web de la enciclopedia Online wikipedia:**

http://www.wikipedia.org

PRESUPUESTO

En esta parte final, quedan recogidos los costes globales de personal y material que la realización de este proyecto fin de carrera conlleva. Para calcular los costes asociados al tiempo empleado en el desarrollo del proyecto, se ha estimado la duración del mismo en 6 meses. La inversión necesaria de horas al día que se ha tomado han sido 6.

1. FASES DE TRABAJO

La siguiente Tabla muestra las fases del proyecto y el tiempo, aproximado, invertido en cada una de ellas. El tiempo total por tanto dedicado a su elaboración ha sido de 860 horas, repartidas en las fases que se especifican a continuación:

1º FASE	Documentación	150 horas
2º FASE	Exploración de las posibilidades de Borland	150 horas
3º FASE	pruebas	60 horas
4º FASE	Implementación de la aplicación	300 horas
5º FASE	Redacción de la memoria	200 horas

2. COSTES DE PERSONAL

Los costes que se desprenden de los recursos humanos requeridos para el proyecto en concepto de mano de obra, son catalogados como costes de personal. Para su cálculo se ha tenido en cuenta la tabla de honorarios del Colegio Oficial de Ingenieros Técnicos Industriales, donde se establecen unas tarifas de 60,88 €/hora (Baremos 2008 COIT).

Personal	**Horas de trabajo**	**€/hora**	**Total (€)**
Ingeniero Técnico	960 horas	60,88	58444,8
Administrativo	120 horas	19	2280
Coste Total del personal			60724,8

3. COSTES DE MATERIAL

En este hay que incluir el uso de ordenadores, licencias, instalaciones usadas… y hay que tener en cuenta el tiempo de uso. Por lo tanto hay que aplicar un coeficiente reductor al coste total. Para ello tomamos la vida media del material en, aproximadamente, 3 años y, como ya se ha indicado, el uso de ellos ha sido de 6 meses, por lo tanto el coeficiente reductor queda en un 83,33 % o, lo que es lo mismo, los gastos son solo el 16,66 % del total.

Equipo	**Total (€)**
Bsp de prueba	300
Material de oficina y gastos varios	250
Licencia de Windows XP	100
Licencia de Office 2003	100
Licencia de Borland 2006	3950
Coste Total del material	4700
Coste Total ponderado (16,66%)	783,02

4. COSTES TOTALES

En la siguiente tabla quedan reflejada la suma de todos los costes:

Coste	**Total (€)**
Coste de personal	60724,8
Coste de material	783,02
Base imponible	75676,46
I.V.A (16%)	12108,23
Presupuesto total	149292,51