UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 1

TÍTULO: Sintetizador de Huellas Dactilares.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autores:

1. *Yusnieira Reyes Pineda.*
2. *Ramón Santana Fernández.*

Tutores:

1. *Ing. Yerandy Arias González.*
2. *Ing. Alexander Leyva Morales.*

La Habana, 13 de junio del 2011

"Año 53 de la Revolución"

‘‘Es justamente la posibilidad de realizar un sueño, lo que hace que la vida sea interesante”.
Paulo Coelho

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio. Para que así conste firmamos el presente:

_____ _____

Autor: Yusnieira Reyes Pineda ***Autor: Ramón Santana Fernández***

_____ _____

Tutor: Ing. Yerandy Arias González. ***Tutor: Ing. Alexander Leyva Morales.***

DATOS DE CONTACTO

Tutor: Ing. Yerandy Arias González(yariasg@uci.cu)

Profesor graduado de Ingeniería en Ciencias Informáticas. Se desempeña como jefe del Departamento de Biometría del Centro CISED (Centro de Identificación y Seguridad Dactilar). Ha participado en eventos científicos como UCIENCIA, RECPAT, Fórum de Ciencia y Técnica, TNT y fue seleccionado como mención al premio CITMA en el curso 2008-2009. Posee dos años de experiencia como profesor recién graduado, el curso 2009-2010 ejerció como tutor de tesis, obteniendo esta última la máxima puntuación.

Tutor: Ing. Alexander Leyva Morales(almorales@uci.cu)

El profesor es Ingeniero en Ciencias Informáticas. Actualmente pertenece al Departamento de Biometría del Centro CISED (Centro de Identificación y Seguridad Dactilar). Ha participado en eventos científicos como UCIENCIA, RECPAT, Fórum de Ciencia y Técnica. Tiene experiencia impartiendo asignaturas del departamento de programación. Durante el curso 2009-2010 se desempeñó como oponente de tesis, obteniendo esta la calificación máxima de 5 puntos.

AGRADECIMIENTOS

En verdad las grandes metas de la vida son difíciles de alcanzar, no por eso dejamos de empeñar alma corazón y vida en busca de sus logros y menos cuando podemos palpar parte de todo ese sacrificio, sientes la recompensa a tanto esfuerzo, dedicación y amor entregado. Gracias a mis esfuerzos y al de muchas personas que siempre se han mantenido a mi lado hoy realizo uno de mis grandes sueños.

Especialmente quiero agradecer a mi mamá por ser la luz de mi vida, mi máxima inspiración, por su infinito amor, confianza y dedicación, por ser mi tesoro más preciado, sin ella no hubiese sido posible llegar hasta aquí. Te quiero mucho.

A mis hermanos Frank, Eduard y Abelito que son mi vida y muy especiales para mí, siempre están presente en cada latido de mi corazón.

A Diosbel que lo quiero como mi propio padre, por su confianza, su cariño, su apoyo, por siempre estar ahí cuando te necesito y por existir en nuestras vidas.

A Edgar por toda su amistad, cariño, confianza, apoyo que me ha brindado.

A mi compañero de Tesis que ha sido como un hermano mayor para mí a lo largo de estos 5 años.

A mi tutor Ing. Yerandy Arias González por su esfuerzo, por los conocimientos y apoyo que me brindó para lograr con éxitos este trabajo de diploma.

A todos mis compañeros y amigos que siempre estuvieron presente en las buenas y en las malas. En especial a mis mejores amigos, que siempre han estado juntos a mí, dándome lo mejor de sí:

A mi familia por su ayuda, comprensión y el apoyo que siempre me han dado.

A la Revolución y a nuestro Comandante Fidel Castro, por habernos dado la oportunidad de estudiar en esta universidad.

Yusnieira Reyes Pineda

Quisiera comenzar agradeciendo a todas aquellas personas que siempre han estado para mí, dándome su apoyo y su confianza:

A mi padre que ha hecho de mi todo lo que soy, me ha guiado y mostrado el camino con sus consejos y su sabiduría.

A mi madre que siempre ha estado presente, brindándome su apoyo y su cariño, a ustedes dos que han vivido cada minuto de mi vida sin importar que tan lejos este, por su confianza infinita y sus interminables lecciones, por estar siempre cuando los necesito.

A mi hermano Ariel por ser justa y exactamente como es, por aparecer con sus ocurrencias y sacarme de quicio, por su apoyo, su comprensión, por sus consejos.

A mi hermano Yaser, por ser esa persona que siempre está presente sin importar nada.

A mis abuelas por haberme apoyado siempre sin importarle otra cosa que no sea mi felicidad.

A mis compañeros de proyecto que más que compañeros los considero amigos.

A mis amistades, por ser leales y comprensivas.

A mi compañera de tesis que con paciencia y dedicación ha ido creando un espacio en mi vida.

En fin todas las personas que de una manera u otra han hecho posible que este hoy aquí, gracias por su cariño, su afecto, por sus interminables horas de paciencia y enseñanza.

Ramón Santana Fernández

DEDICATORIA

Dedico este trabajo que es la realización de un gran sueño y el logro de mucho esfuerzo y sacrificio:

A mi madre por se lo mas importante en mi vida.

A mi abuelita por ser otra madre para mí, abue te quiero mucho.

A mis hermanos, a Diosbel y a mi familia.

Yusnieira Reyes Pineda

Dedicado a Ramón, Lourdes y Ariel, que son no parte de mi vida, sino mi vida entera.

Ramón Santana Fernández

Resumen

En este trabajo se desarrolla un generador de huellas dactilares que realiza el proceso de adquisición de bases de datos de manera automática, capaz de sustituir el proceso tradicional de obtención de las mismas. La huella se construye a partir de una serie parámetros definidos por el usuario, comenzando por la selección del dedo, ubicando las posiciones de las singularidades (cores y deltas), generando minucias e introduciéndole roturas y ruido, logrando construir huellas dactilares lo más reales posibles junto a su vector característico.

Con la elaboración de esta aplicación se podrán generar suficientes muestras para conformar bases de datos y de esta manera disponer de un software competitivo capaz de generar las huellas dactilares necesarias, almacenarlas en bases de datos, dando la posibilidad de contar con un software para el beneficio de las instituciones nacionales que precisen de este tipo de herramientas, lo que ahorraría cuantiosos gastos al país, además este producto por su calidad también se podría comercializar permitiendo la entrada de divisas.

Palabras claves: huellas dactilares, algoritmos biométricos, reconocimiento de personas, base de datos.

Tabla de contenido

Índice de Figuras

Índice de Tablas

Introducción

Con el avance de la sociedad moderna el estudio de la Biometría ha posibilitado el desarrollado de diversos métodos automáticos para la identificación de los individuos basados en uno o más rasgos conductuales o físicos intrínsecos de una persona. En estos últimos años estas técnicas han aumentado desde usar simplemente la huella dactilar, hasta emplear distintos métodos teniendo en cuenta varias medidas físicas y de comportamiento.

Las primeras aplicaciones de las técnicas biométricas tuvieron lugar dentro del ámbito legal, particularmente en el campo forense. Sin embargo, debido a la expansión tecnológica de la sociedad en las últimas dos décadas, surge la necesidad de diseñar sistemas automáticos de alta seguridad capaces de identificar a los diferentes individuos a partir de sus rasgos biométricos.

Estas técnicas biométricas son utilizadas para el control de acceso en muchos países desarrollados, siendo aplicables en los diferentes medios o lugares que requiera autenticación por ejemplo en bancos, pasaportes electrónicos y cajeros automáticos. En cada caso, se mide cierta combinación de las características inherentes y se comparan automáticamente con plantillas almacenadas en un archivo o en una base de datos para realizar la autenticación.

El reconocimiento de personas mediante huellas dactilares es uno de los procedimientos biométricos más utilizados, dadas sus propiedades de unicidad, permanencia y fiabilidad que actualmente se conocen y en pocos años posiblemente sea parte de la vida diaria. En sus inicios, las identificaciones las realizaba una persona especializada, de manera visual, ubicando las minucias en ambas huellas y luego comparándolas para determinar si pertenecían a la misma persona. En la actualidad se han desarrollado sistemas que realizan esta labor de manera automática, logrando comparar varios cientos de huellas en minutos. Estos avances en la identificación de las huellas han abierto un gran campo en el área de la seguridad y la identificación personal.

En los últimos años el fraude de identidad ha creado la necesidad de perfeccionar la tecnología biométrica de reconocimiento de personas. Esto trae como consecuencia que se cuente con aplicaciones que provean datos que puedan ser usados en la realización de pruebas a algoritmos biométricos para tener un alto nivel de precisión de su funcionamiento. Aplicaciones como el SFINGE es usada con este fin, de hecho es la única de su tipo en el mundo que provee una base de datos artificial para ser utilizada en el proceso de prueba.

En Cuba la Biometría es un campo joven que comienza a desarrollarse, sin embargo para los procesos de prueba de algoritmos biométricos de extracción y cotejo de minucias, aún no se cuenta con una herramienta que permita la generación de bases de datos de prueba, por lo que para conformar las mismas es necesario convocar cientos de personas, esto trae varios inconvenientes, la participación de tantas personas en el proceso tiende a ser costosa tanto en términos de tiempo como de dinero, además existe la posibilidad de que ocurran pequeños errores en la recolección, clasificación y posterior almacenamiento de las huellas dactilares.

Por todas estas razones impiden un desarrollo exitoso para la realización de las pruebas a los algoritmos de reconocimiento de huellas dactilares, se conforma la **situación problémica**. Ante este inconveniente se plantea el siguiente **problema científico** ¿Cómo facilitar la generación de bases de datos, para la realización de pruebas a algoritmos de cotejo y extracción de minucias en huellas dactilares?

Para dar solución al problema expuesto se define como **objeto de estudio** la generación artificial de huellas dactilares para generar bases de datos. Se determina como **campo de acción** la generación de minucias, creación de la imagen e introducción de roturas y ruido.

La investigación se rige por la siguiente **idea a defender**: Desarrollar una aplicación que genere huellas dactilares artificiales, para contar con suficientes muestras que permitan conformar base de datos de pruebas.

Para dar solución al problema antes planteado se define como **objetivo general** de la investigación implementar una aplicación para la generación de bases de datos de huellas dactilares y su vector característico.

Para llevar a cabo el objetivo del trabajo se plantean las siguientes **tareas de investigación.**

- Caracterizar las diferentes aplicaciones que se han desarrollado hasta la actualidad para la generación y obtención de huellas dactilares.
- Estudiar el proceso de obtención y generación de huellas dactilares.
- Estudiar el estándar del vector característico.
- Describir el proceso dirigido a la obtención de las huellas dactilares de manera artificial.

- Estudiar la transformada de Gabor.
- Estudiar algoritmos para estimar la frecuencia y orientación de las crestas.
- Implementar algoritmo para estimar frecuencia y orientación de las crestas.
- Estudiar los principales algoritmos a utilizar para la generación de minucias.
- Implementar algoritmos de generación de minucias.
- Estudiar algoritmos para introducir ruido en la huella.
- Implementar algoritmos para simular la presión de la huella en el lector.
- Realizar pruebas a la aplicación.

El presente trabajo de diploma consta de 3 capítulos estructurados de la siguiente manera:

Capítulo 1: **Fundamentación Teórica**, se describen los principales conceptos abordados, se realiza un estudio del estado del arte del tema tratado a nivel internacional, nacional y en la Universidad y se hace un estudio de las principales tecnologías, metodologías y herramientas para darle solución al problema planteado.

Capítulo 2: **Características del Sistema**, se definen las características del sistema realizando un análisis del dominio de la aplicación, se modelan los diagramas de clases del análisis y del diseño dejando todo listo para la etapa de implementación. Se hace referencia a los patrones utilizados para construcción del sistema.

Capítulo 3: **Implementación y Prueba**, se tratan aspectos relacionados con la construcción de la solución que se propone, se modela el diagrama de componente y por último se realizan las pruebas del sistema, donde se valida el funcionamiento de los requisitos funcionales.

Capítulo 1: Fundamentación Teórica

1.1 Introducción

En el presente capítulo se describen los principales conceptos asociados al dominio de la aplicación. Se realiza un estudio del arte sobre las aplicaciones de generación de base de datos de huellas dactilares existentes en el mundo, en Cuba y en la Universidad, así como de las principales tecnologías, metodologías y herramientas, analizando sus características, ventajas y desventajas, buscando las más indicadas para la construcción de la solución a desarrollar.

1.2 Sistemas biométricos

Son sistemas automatizados que se utilizan para labores de biometría, como la verificación e identificación de los individuos, utilizando características físicas y comportamientos precisos, es decir, un sistema que fundamenta sus decisiones de reconocimiento mediante una característica personal que puede ser reconocida o verificada de manera automatizada.

En la actualidad existen sistemas biométricos que basan su acción en el reconocimiento de diversas características, como puede apreciarse en la figura 1. Las técnicas biométricas más conocidas son nueve y están basadas en los siguientes indicadores biométricos: (a) Rostro, (b) Termograma Facial, (c) Huella dactilar, (d) Geometría de la mano, (e) Venas de las manos, (f) Iris, (g) Patrones de la retina, (h) Voz, (i) Firma. (1)

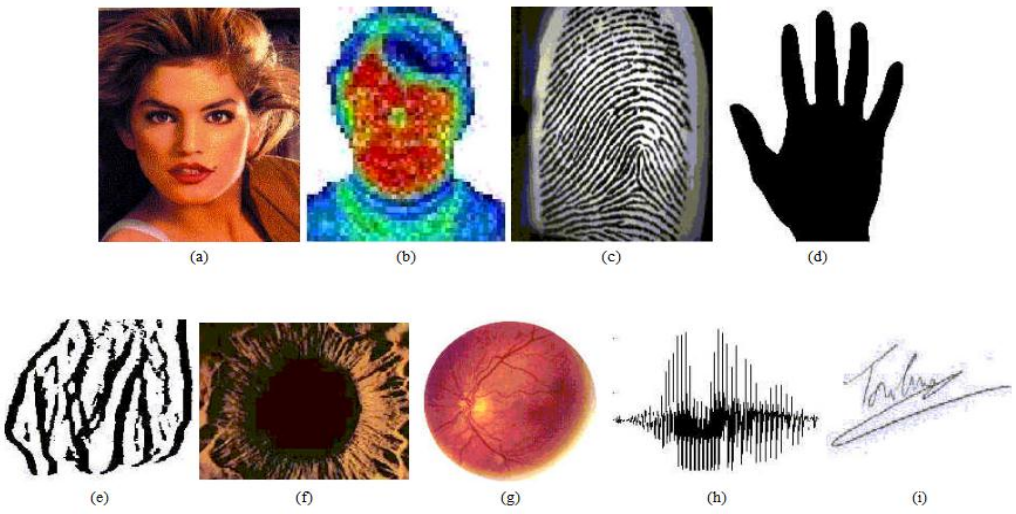

Figura 1. Técnicas biométricas actuales

1.3 ¿Qué son las huellas dactilares?

Una huella dactilar es una característica individual utilizada como medio de identificación de los individuos. Están constituidas por rugosidades que forman salientes y depresiones. Las salientes se denominan crestas papilares y las depresiones surcos inter-papilares. En las crestas se encuentran las glándulas sudoríparas. El sudor que éstas producen contiene un aceite el cual es retenido en los surcos de la huella, de tal manera que cuando el dedo hace contacto con una superficie, queda un residuo, produciendo el facsímil o negativo de la huella. Es posible identificar el tipo de huella que posee cada individuo, ya que las huellas dactilares son una característica única de los mismos y se pueden clasificar en cinco tipos: espiral, lazo derecho, lazo izquierdo, arco y arco tendido, que se pueden observar en la figura 2. Cabe señalar que en un mismo individuo, la huella de cada uno de sus dedos es diferente. (2)

Figura 2. Tipos de huellas dactilares

Cada uno de los cinco tipos de huellas tiene puntos focales que se utilizan para la clasificación.

- **Espiral:** Los patrones en forma de espiral tienen muchos círculos que salen de un punto de la huella como muestra la figura 3, las mismas cuentan con dos o más puntos deltas.

Figura 3. Huella dactilar espiral

- **Lazo derecho**: Se caracterizan porque las crestas que forman su núcleo nacen en el costado izquierdo de la huella y hacen su recorrido hacia la derecha, para luego dar vuelta sobre sí mismas y regresar al mismo punto de partida. Estas tienen un punto delta que se encuentra ubicado al lado derecho de la huella.

Figura 4. Huella dactilar lazo derecho

- **Lazo izquierdo**: Al igual que el lazo derecho, estas tienen un punto delta, pero éste se ubica del lado izquierdo de la huella. Las crestas papilares que forman el núcleo nacen a la derecha y su recorrido es a la izquierda para dar vuelta sobre sí mismas y regresar al mismo punto de partida.

Figura 5. Huella dactilar lazo izquierdo

- **Arco:** Los patrones de arco tienen líneas que empiezan en un lado de la huella, van hacia el centro curvándose hacia arriba y salen del otro lado de la huella, formando lo que se asemeja a una fila de arcos apilados, no cuentan con un punto delta.

Figura 6. Huella dactilar arco

- **Arco tendido**: Semeja un arco. Es un tipo de línea que rápidamente nace y se pierde en un paso de ángulo, estos tienden a ser asociados con los patrones de huellas tipo arco tendido.

Figura 7. Huella dactilar arco tendido

1.3.1 Vector característico

El vector característico está estandarizado por la norma ISO 19794-4, presenta la forma TLV (Tag-Lenght-Value) donde se especifica que detrás de una etiqueta se encuentra la longitud del valor y posteriormente el valor del mismo, tiene como objetivo guardar la forma en que es etiquetada la base de datos, las posiciones de las singularidades y las minucias que posee la huella dactilar. Esta información sirve para el intercambio entre las organizaciones que se basan en sistemas automatizados de identificación o verificación sobre la base de la información de huellas dactilares.

1.4 Sintetizador de huellas dactilares

Un sintetizador de huellas dactilares es un software encargado de generar bases de datos de huellas dactilares, que contiene la imagen de la huella dactilar y el vector característico, el cual guarda la posición de las minucias y de las singularidades (cores y delta) de las huellas dactilares generadas. Este proceso de generación se lleva a cabo mediante un conjunto de pasos ordenados, comenzando por la selección de la forma del dedo al cual pertenecerá la huella, ubicando las posiciones de los cores y los deltas, generando las minucias, introduciéndole roturas y ruido, con el objetivo de obtener una huella dactilar lo más real posible.

1.4.1 ¿Por qué un sintetizador de huellas dactilares?

La generación sintética de huellas dactilares es una técnica eficaz para superar el problema de la recogida de grandes bases de datos de muestras para con el objetivo de realizar pruebas a algoritmos de verificación dactilar, proceso costoso en tiempo que resulta tedioso para los implicados, sin contar con los pequeños errores que se pueden ir cometiendo durante el proceso de toma de muestras y almacenamiento de las huellas. Un sintetizador de huellas dactilares es un sistema que

genera huellas dactilares junto a su vector característico, no pudiéndose obtener con igual precisión mediante otro proceso.

1.4.2 SFINGE

Actualmente existe una aplicación llamada **SFINGE** que genera huellas dactilares de manera artificial. El **SFINGE** es un producto comercial muy competente y sofisticado, pero muy costoso por lo que muchos países subdesarrollados no pueden adquirirlo como Cuba. La aplicación completa tiene un costo de 3900 euros para instituciones académicas como esta universidad. (3)

SFINGE se utiliza para crear grandes bases de datos de huellas dactilares, lo que permite a los algoritmos de reconocimiento ser simplemente probados y optimizados. Por ejemplo, una base de datos contiene 100.000 huellas dactilares puede ser generado por lotes en aproximadamente un día y en una sola PC, siendo estas huellas extremadamente realistas.

El software **SFINGE** comprende un conjunto de pasos para la generación de una huella dactilar, los primeros tres pasos crean la imagen y los restantes obtienen la huella dactilar artificial. Dichos pasos se explican a continuación.

1. Generación de la zona de la huella dactilar.

En dependencia de la variación de los cinco parámetros ilustrados en la figura se constituye el área de adquisición.

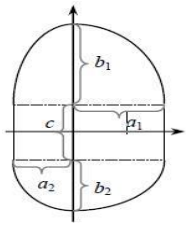

Figura 8. Modelo de la forma de la huella dactilar

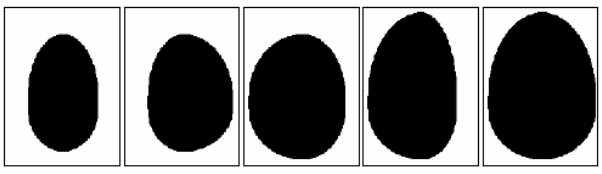

Figura 9. Ejemplos de siluetas de huellas dactilares

2. Orientación para la generación de imágenes.

En 1993 Sherlock y Monro proponen un modelo de orientación que consiste en la orientación de una imagen computacional a partir del conocimiento de sus singularidades (cores y deltas). En dicho modelo la imagen se encuentra en un plano complejo, y la orientación local de cada una de las crestas se tiene por la fórmula descrita a continuación, en la cual las singularidades están descritas como deltas (ds) y loops (ls). Las coordenadas de la orientación hacia el punto z del plano viene descrita por θ en la siguiente ecuación:

$$\theta = \frac{1}{2}\left[\sum_{i=1}^{nd} \arg(z - ds) - \sum_{i=1}^{nc} \arg(z - ls)\right]$$

La función argumento (arco tangente) nos retorna la parte real del número complejo que se calcula en su interior por la resta de los dos puntos, la singularidad y el punto z. El modelo plantea la generación de la imagen de la siguiente manera, primero se selecciona la clase de huella dactilar de manera aleatoria, se generan las singularidades de manera aleatoria pero bajo las restricciones que les impone cada clase, excepto por la clase arco, la cual no tiene singularidades, pero la generación de la misma no presenta un gran problema dentro del modelo, considerándose de manera separada. La generación de la clase arco y la orientación de la misma se realiza mediante la variación de la frecuencia y la amplitud en una función sinusoidal simple.

La línea de ridges de la huella dactilar real no puede ser completamente determinada por las singularidades (cores y delta) en este modelo, aunque se obtienen resultados bastante alentadores con el mismo, sirviendo de punto de comienzo en la generación de la imagen de una huella dactilar. Uno de los problemas que contiene este modelo se encuentra en que la orientación de la imagen generada por el modelo de Sherlock y Monro, con la misma posición del lazo y el delta tiene claras diferencias entre las orientaciones de la línea real de canto y los elementos correspondientes en la orientación de la imagen: en particular, las regiones por encima del loop y entre el loop y el delta no están bien modelados. Por lo que en 1996 Vizcaya y Gerhardt proponen un modelo basado en el antiguo modelo de Sherlock y Monro, en el cual se le introducen mayores grados de libertad a los ángulos, este modelo está determinado por la fórmula:

$$\theta = \frac{1}{2}\left[\sum_{n=1}^{n_d} g_{dsi}\left(\arg(z - d_{si})\right) - \sum_{n=1}^{n_c} g_{lsi}\left(\arg(z - l_{si})\right)\right]$$

Donde $g(\propto), \text{para } k \in (\{l_{s1} \dots l_{sn}, d_{s1} \dots d_{sn}\})$ son piezas de un modelo lineal que tiene como objetivo corregir la alineación de la imagen con respecto a la alineación que propone el modelo de Sherlock y Monroe.

$$g(\alpha) = \bar{g}_k(\alpha_i) + \frac{\alpha - \alpha_i}{\frac{2\pi}{L}}\left(\bar{g}_k(\alpha_{i+1}) - \bar{g}_k(\alpha_i)\right)$$

Para $\alpha_i \leq \alpha \leq \alpha_{i+1}, \alpha_i = -\pi + \frac{2\pi}{L}$

Donde irá tomando valores desde 0 hasta L-1, cada uno de estos valores tienen como función corregir la orientación de la imagen en el ángulo dado. Los ángulos se distribuyen de manera uniforme entre $-\pi$ y π.

En el software **SFINGE** se usa solamente este modelo para la corrección de la imagen. Un detallado estudio de Cappelli, Maio, y Maltoni (2000b) describe que L= 8 es un valor razonable, y deriva rangos apropiados para la orientación de la imagen mientras se genera.

3. Frecuencia de generación de imágenes.

Cappelli, Maio y Maltoni luego de la inspección de un gran número de imágenes llegan a la conclusión de que la generación de la frecuencia de la imagen podría ser hecha totalmente de manera aleatoria, mediante lo observado llegan a la conclusión de que por encima del bucle la región más al norte y por debajo del delta la región más al sur, presenta una menor frecuencia del canto en comparación con el resto de la huella dactilar, por lo que en el software SFINGE se realiza de una manera diferente, se le proporciona una menor frecuencia en la parte superior e inferior de la imagen.

4. Generación de patrones de ridge.

Después de conocida la orientación y la frecuencia de la imagen se realizan una generación determinista del patrón de los ridges, esto incluye la generación de las minucias. Primeramente se deberá definir a priori la cantidad, el tipo y la localización de las minucias, luego de ello se genera una imagen a escalas de grises, comenzando por la vecindad de las minucias y expandiéndolas para conectar con las diferentes regiones hasta completar la imagen. La construcción de tan complejo proceso necesitará de reglas complejas descritas de manera estricta. Pero un enfoque más elegante seria realizar el proceso a partir de un conjunto de símbolos y reglas de producción que genere las huellas. Sin embargo el **SFINGE** usa un método más sencillo, se toma un punto inicial negro entre

tantos blancos y de manera iterativa se va mejorando la imagen a partir de Filtros de Gabor, este proceso se va ajustando a partir del canto local y la frecuencia, las terminaciones, bifurcaciones e islas son colocadas de manera aleatoria, el filtro aplicado a cada pixel tiene la forma:

$$g(x, y: \theta, f) = e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)} * \cos[2\pi * f * (x * \sin\theta + y * \cos\theta)]$$

Donde θ y f son la correspondiente orientación y frecuencia, el parámetro σ determina el ancho de la banda del filtro, es ajustado de acuerdo a la frecuencia local, posibilitando que el filtro no tenga más de tres picos locales. En particular el valor de σ es determinado por la ecuación:

$$e^{-\left((\frac{3}{2f})^2/2\sigma^2\right)} = 10^{-3}$$

Muchos son los factores que contribuyen a la generación de una base de datos de huellas dactilares, al realizar la variación de esta para obtener las demás, entre ellos está el desplazamiento en los ejes de traslación y rotación en los mismos, las diferentes posiciones que puede adoptar en el sensor, la distorsión producida por la presión del dedo contra el sensor, las variaciones en los ridges, y otros son factores importantes, que se evalúan luego de tener la primera huella dactilar y con cuyas variaciones podemos obtener una base de datos de huellas dactilares del tamaño que desee el usuario.

En la figura 10 se muestra la definición de una función de corrección aplicada al modelo de Sherlock y Monro.

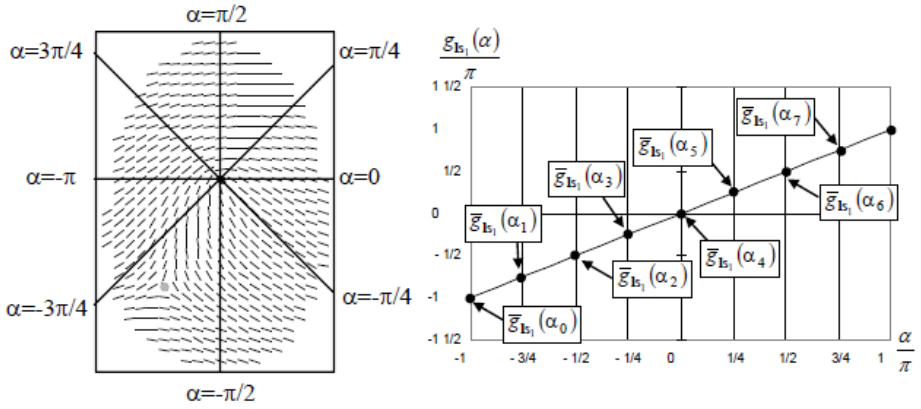

Figura 10. Sherlock y Monro

Los efectos de la modificación de los parámetros que controlan la función de asignación se destacó en la orientación de la imagen correspondiente (Vizcaya y Gerhardt, 1996).

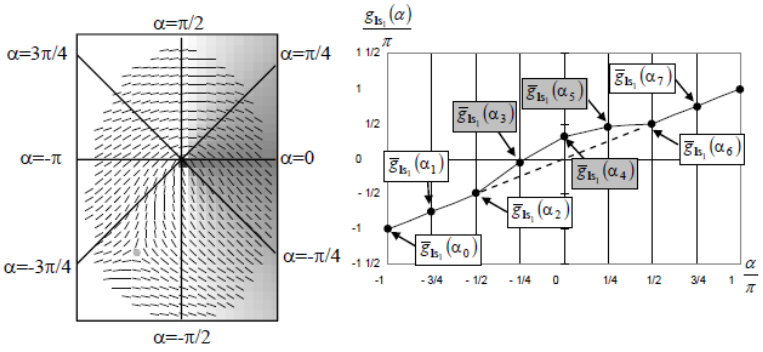

Figura 11. Vizcaya y Gerhardt

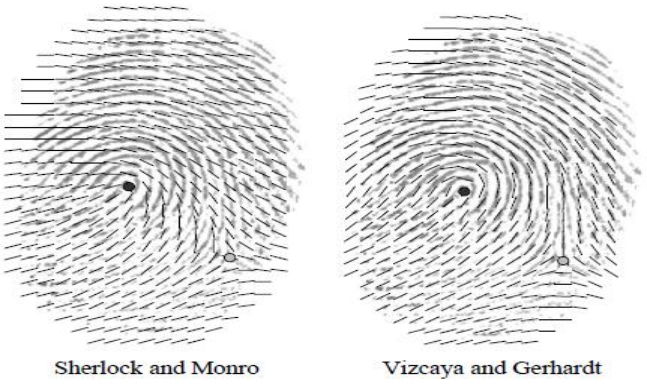

Figura 12. Comparación de las imágenes hechas por ambos métodos

5. Definición de la porción de huella dactilar.

La humedad de la piel y la presión del dedo sobre el sensor tienen efectos similares en la adquisición de imágenes: cuando la piel está seca o la presión es baja, las crestas aparecen más claras, mientras que cuando la piel está húmeda o la presión es alta, las crestas aparecen más grueso como muestra la figura 13. Para simular diferentes grados de humedad / presión, se le aplica a la imagen operadores morfológicos, el operador de la erosión se aplica para simular la presión baja o la piel seca y el operador de la dilatación para la piel de alta presión o húmedo.

Figura 13. Humedad de la piel

6. Distorsión de la huella dactilar.

Una de las principales características que distingue a las diferentes impresiones del mismo dedo es la presencia de distorsiones no lineales, debido principalmente a las deformaciones de la piel, según las diferentes posiciones del dedo sobre el sensor. Debido a la elasticidad de la piel, la aplicación de presión, produce distorsiones no lineales (compresión o estiramiento) de las huellas dactilares adquiridas. Con el fin de mejorar la adecuación de la huella, se aplica la interpolación de Lagrange que se emplea para obtener imágenes deformadas suavizadas de escala de grises.

Figura 14. Distorsión del dedo

7. Perturbación y traslación/rotación.

Durante la adquisición de huellas dactilares, varios factores contribuyen al deterioro de la imagen, lo que produce una imagen con ruido: la irregularidad de los bordes, la presencia de pequeños poros en las crestas, la presencia de pequeñas crestas prominentes, lagunas, el desorden y el ruido debido a la presión no uniforme de los dedos contra la sensor. Además, la huella dactilar por lo general no está centrada en la imagen, la cual se le aplica rotación y traslación.

La fase de perturbación secuencial realiza los siguientes pasos:

- Aislar los píxeles blancos asociados a los valles en una capa separada.
- Añadir ruido en forma de pequeñas manchas blancas de tamaño variable y forma. La cantidad de ruido se incrementa con la inversa de la distancia de la frontera de huellas dactilares.
- Suavizar la imagen resultante con un cuadro de un promedio de 3 × 3 del filtro.
- Superponer la capa de valle a la imagen resultante.
- Rotar y Trasladar la imagen.

8. Generación del fondo de la huella.

La salida de la fase de perturbación es una imagen de la huella dactilar que parece realista, pero el fondo sigue siendo completamente blanco. Con el fin de generar fondos similares a los de la imagen de la huella dactilar adquirida con un sensor determinado, un modelo estadístico basado en la transformación de KL es adoptado. El modelo requiere un conjunto de imágenes de fondo como se muestra en la figura 15, se calcula un sub-espacio lineal que representa las principales variaciones en el fondo de formación de imágenes y luego se usa para generar aleatoriamente nuevos fondos. (4)

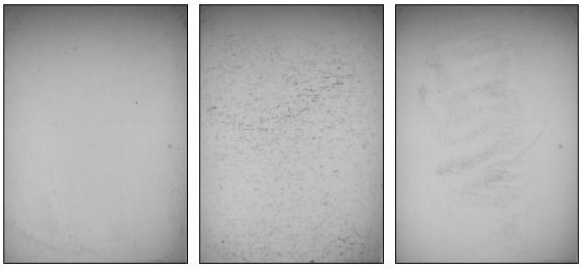

Figura 15. Fondos para la huella

1.5 Tecnología, metodología y herramientas empleadas

1.5.1 Metodología de desarrollo del software

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos de software. Indican, paso a paso, todas las actividades a realizar para lograr el producto informático deseado, además de las personas que deben participar en el desarrollo de las actividades y qué papel deben tener. Detallan la información que se debe producir como resultado de una actividad y la información necesaria para comenzarla.

Las metodologías se encargan de elaborar estrategias de desarrollo de software que promuevan prácticas centradas en las personas o los equipos, orientadas hacia la funcionalidad y la entrega, de comunicación intensiva y que requieren implicación directa del cliente. Una metodología de desarrollo de software permite producir organizada y económicamente software de alta calidad, siguiendo una serie de pasos donde se utilizan un conjunto de técnicas, notación y normas de documentación pre establecidas. (5)

1.5.1.1 Metodologías ágiles

La metodologías ágiles se caracterizan principalmente por el uso de técnicas para agilizar el desarrollo del software, así como de una mayor flexibilidad para adaptarse a los cambios en los requisitos del proyecto, los individuos y las interacciones entre ellos son más importantes que las herramientas y los procesos empleados, es más importante crear un producto de software que funcione que escribir documentación exhaustiva, la capacidad de respuesta ante un cambio es más importante que el seguimiento estricto de un plan, la colaboración con el cliente debe prevalecer sobre la negociación de contratos. (6)

Programación Externa (XP): es una metodología ágil centrada en potenciar las relaciones interpersonales como punto importante para el éxito en el desarrollo del software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los miembros del equipo de desarrollo y propiciando un ambiente de trabajo adecuado. XP se basa fundamentalmente en la realimentación continua con el cliente y el equipo de desarrollo, y la simplicidad en la solución implementada. Entre las características esenciales que presenta XP se encuentran las historias de usuarios, técnica utilizada para especificar los requisitos del software. (7)

Scrum: es una metodología de desarrollo de software iterativa e incremental, desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle, es un término utilizado en el juego de rugby que llevado a la ingeniería significa que el equipo se agrupará para llegar con el apoyo de cada uno de los miembros del proyecto a obtener un producto con calidad que es el objetivo fundamental. Esta metodología se conforma por ciclos de desarrollos llamados sprint, los cuales se definen en un período de quince a treinta días, en el cual el equipo crea un incremento de software utilizable. Las acciones que forman parte de cada sprint vienen de un conjunto de requisitos de alto nivel priorizados que definen el trabajo a realizar. Estos elementos que forman parte del sprint se determinan durante una reunión, entonces, el equipo determina la cantidad de trabajo que puede comprometerse a completar durante el siguiente sprint. (8)

1.5.1.2 Metodologías pesadas

Las metodologías pesadas son aquellas que están guiadas por una fuerte planificación durante todo el proceso de desarrollo, en la cual se establecen estrictamente las actividades involucradas, los roles definidos, los artefactos que se deben producir, las herramientas y notaciones que serán utilizadas así como el modelado y documentación detallada. El proceso de desarrollo de software llevado a cabo a partir de este tipo de metodología es mucho más controlado, con cierta resistencia a posibles cambios, además de poseer numerosas políticas o normas. (9)

RUP

El Proceso Unificado Racional, es un proceso de desarrollo de software, orientado a objetos, preparado para desarrollar grandes y complejos proyectos, unifica los mejores elementos de metodologías anteriores y utiliza el Lenguaje Unificado de Modelado UML, como lenguaje de representación visual. En RUP se han agrupado las actividades en grupos lógicos definiéndose 9 flujos de trabajo principales. Los 6 primeros son conocidos como flujos de ingeniería y los tres últimos como de apoyo. Flujos de trabajo:

- Modelamiento del negocio.
- Requerimientos.
- Análisis y diseño.
- Implementación.
- Prueba (Testeo).
- Instalación.
- Administración del proyecto.
- Administración de configuración y cambios.
- Ambiente.

El proceso de ciclo de vida de RUP se divide en cuatro fases bien conocidas llamadas Inicio, Elaboración, Construcción y Transición.

I**nicio**: Se describe el negocio y se delimita el proyecto describiendo sus alcances con la identificación de los casos de uso del sistema, que orientarán la funcionalidad.

Elaboración: Se define la arquitectura del sistema y se obtiene una aplicación ejecutable que responde a los casos de uso que la comprometen. A pesar de que se desarrolla a profundidad una parte del sistema, las decisiones sobre la arquitectura se hacen sobre la base de la comprensión del sistema completo y los requerimientos (funcionales y no funcionales), identificados de acuerdo con el alcance definido.

Construcción: Se obtiene un producto listo para su utilización que está documentado y tiene un manual de usuario. Se obtiene 1 o varios reléase del producto que han pasado las pruebas. Se ponen estos reléase a consideración de un subconjunto de usuarios. Es la fase más prolongada de todas.

Transición: El reléase ya está listo para su instalación en las condiciones reales. Se corrigen los últimos errores. Se llama transición porque se transfiere a las manos del usuario, pasando del entorno de desarrollo al de producción.

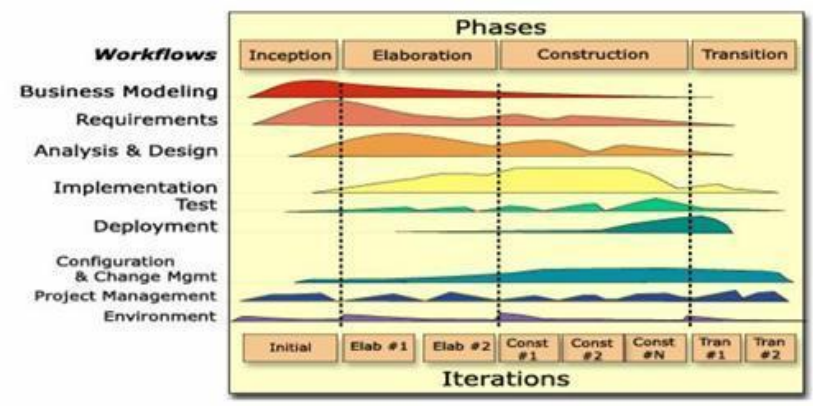

Figura 16. Ciclo de vida de RUP

El ciclo de vida de RUP se caracteriza por:

- Dirigido por Casos de Uso: Los casos de uso reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los casos de uso guían el proceso de desarrollo pues los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso (cómo se llevan a cabo).

- Centrado en la arquitectura: La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente.

- Iterativo e Incremental: RUP propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros. Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software. (10)

1.5.2 Lenguaje de modelación

Un lenguaje para el modelado de objetos es un conjunto estandarizado de símbolos y de modos de disponerlos para modelar un diseño de software orientado a objetos. El uso de un lenguaje de modelado es más sencillo que la auténtica programación, pues existen menos medios para verificar efectivamente el funcionamiento adecuado del modelo. Esto puede suponer también que las interacciones entre partes del programa den lugar a sorpresas cuando el modelo ha sido convertido en un software que funciona. (11)

Lenguaje Unificado de Modelado UML

Por sus siglas en inglés, (Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

Es importante resaltar que UML es un "lenguaje" para especificar y no para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

Se puede aplicar en el desarrollo de software entregando gran variedad de formas para dar soporte a una metodología de desarrollo de software, pero no especifica en sí mismo qué metodología o proceso usar. (12)

1.5.3 Herramienta CASE

CASE es una sigla, que corresponde a las iniciales de: Computer Aided Software Engineering; y en su traducción al Español significa Ingeniería de Software Asistida por Computación. El concepto de CASE es muy amplio; y una buena definición genérica, que pueda abarcar esa amplitud de conceptos, sería la de considerarla como la aplicación de métodos y técnicas a través de las cuales se hacen útiles a las personas comprender las capacidades de las computadoras, por medio de programas, procedimientos y su respectiva documentación. El uso de estas herramientas, permite desarrollar proyectos informáticos que tengan como objetivo la automatización de procedimientos administrativos; o sea las herramientas CASE representan una forma que permite modelar los Procesos de Negocios de las empresas y desarrollar los Sistemas de Información Gerenciales. (13)

Rational Rose Enterprise

El Rational Rose, es una herramienta CASE desarrollada por Rational Corporation, basada en UML, que permite crear los diagramas que se van generando durante el proceso de Ingeniería en el desarrollo del software. Las personas que desarrollaron RUP son miembros de Rational Corporation, por lo que el mismo es completamente compatible con esta metodología, brinda muchas facilidades en la generación de la documentación del software que se está desarrollando, además posee un gran número de estereotipos predefinidos que facilitan el proceso de modelación del software. Dicha herramienta es capaz de generar el código fuente de las clases definidas en el flujo de trabajo de diseño, pero tiene la limitación de que aún hay varios lenguajes de programación que no soporta o que solo lo hace a medias. (14)

Visual Paradigm

Visual Paradigm utiliza el lenguaje de modelado estándar UML, diseñada para realizar el modelado visual de software que utiliza la programación orientada a objetos. Como características más sobresalientes se puede destacar que permite generar a partir del modelado, código PHP 5, realizar ingeniería tanto directa como inversa y generar la documentación del proyecto automáticamente en varios formatos como Web o PDF, es una herramienta multiplataforma que se puede utilizar tanto en Linux como en Windows, por lo que cumple con las políticas de migración a software libre, es un

software multiplataforma lo cual favorece el modelado independientemente del sistema operativo que se utilice. Permite la generación automática de diagramas a partir de las descripciones de los casos de uso, por ejemplo diagramas de secuencia, permitiendo la agilidad en el trabajo del analista. (15)

1.5.4 Lenguaje de programación

Los lenguajes de programación se usan para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana. Está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

Al proceso por el cual se escribe, se prueba, se depura, se compila y se mantiene el código fuente de un programa informático se le llama programación. Los lenguajes de programación facilitan la tarea de programación, ya que disponen de formas adecuadas que permiten ser leídas y escritas por personas, a su vez resultan independientes del modelo de computador a utilizar. (16)

Lenguaje C++

Bjarne Stroustrup crea una versión experimental denominada "C with Classes" (C con clases) hacia 1979, con la intención de proporcionar una herramienta de desarrollo para el kernel Unix en ambientes distribuidos; el objetivo de este nuevo lenguaje era mejorar algunas características del C pero manteniendo su basamento en el mismo. Luego, en 1983 adquiere el nombre de “C++” debido a su esencia de C y con mejoras como su operador de incremento numeral (++).

C++ es un lenguaje imperativo orientado a objetos, es un súper conjunto de C, que nació para añadirle cualidades y características de las que carecía. El resultado es que como su ancestro, sigue muy ligado al hardware subyacente, manteniendo una considerable potencia para programación a bajo nivel, pero se la han añadido elementos que le permiten también un estilo de programación con alto nivel de abstracción.

C++ no es un lenguaje orientado a objetos puro (en el sentido en que puede serlo Java por ejemplo), se trata simplemente del sucesor de un lenguaje de programación hecho por programadores (de alto nivel) para programadores, al cual le han ido añadiendo todos los elementos que la práctica aconsejaba como necesarios, con independencia de su belleza o purismo conceptual.

Es versátil, flexible, conciso y muy eficiente. Por muchos años fue el preferido en el desarrollo de aplicaciones. Se ha utilizado para implementar el núcleo de sistemas como Windows. (17)

Lenguaje C#

C# (leído en inglés “See Sharp” y en español “C Almohadilla”) es el nuevo lenguaje de propósito general diseñado por Microsoft para su plataforma .NET. Sus principales creadores son Scott Wiltamuth y Anders Hejlsberg, éste último también conocido por haber sido el diseñador del lenguaje Turbo Pascal y la herramienta RAD Delphi.

Aunque es posible escribir código para la plataforma .NET en muchos otros lenguajes, C# es el único que ha sido diseñado específicamente para ser utilizado en ella, por lo que desarrollar usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes. Por esta razón, se suele decir que C# es el lenguaje nativo de .NET.

Presenta las siguientes características:

- Sencillez.
- Modernidad.
- Orientación a objetos.
- Orientación a componentes.
- Gestión automática de memoria.
- Seguridad de tipos.
- Instrucciones seguras.
- Sistema de tipos unificado.
- Eficiencia.
- Compatibilidad. (18)

Lenguaje Java

Este es un lenguaje desarrollado por la compañía Sun Microsystem en los años noventa. Está inspirado en C++ y se proyectó con la finalidad de obtener un producto de pequeñas dimensiones, simple y portátil sobre diferentes plataformas y sistemas operativos ya sea a nivel de código fuente como a nivel de código binario. Es un lenguaje orientado a objetos, eso implica que su concepción es muy próxima a la forma de pensar humana, genera ficheros de clases compiladas, pero estas son en realidad interpretadas por la Máquina Virtual de Java, quien mantiene el control sobre las clases que se estén ejecutando. El mismo es multiplataforma y seguro: La máquina virtual, al ejecutar el código java, realiza comprobaciones de seguridad, además el propio lenguaje carece de características inseguras, como por ejemplo los punteros.

Su sintaxis ha sido trabajada mejorando la de C++ logrando mayor sencillez y legibilidad. Presenta mayor robustez al simplificar la gestión de memoria y eliminar las complejidades del manejo explícito de punteros. Se puede compilar y ejecutar en cualquier plataforma de sistema operativo por ejemplo en Windows, Solaris o Linux gracias a su máquina virtual. Su desarrollo ha sido rápido y exitoso debido a la gran cantidad de grandes empresas colaboradoras que han dado su aporte para enriquecerlo. La ejecución de programas escritos en Java suele comportarse más lenta que la de aplicaciones de otro lenguaje haciendo un uso voraz de recursos como memoria y procesador. Esto se hace más notorio si la ejecución se basa en cálculos matemáticos complejos o si la aplicación presenta un diseño cargado de componentes visuales. (19)

1.5.5 Entornos integrados de desarrollo

Un entorno integrado de desarrollo (IDE), es un programa informático compuesto por un conjunto de herramientas de programación, utilizadas por los programadores para desarrollar código. Esta herramienta puede estar pensada para su utilización con un único lenguaje de programación o bien puede dar cabida a varios de estos.

Las herramientas que normalmente componen un entorno de desarrollo integrado son las siguientes: un editor de texto, un compilador, un intérprete, unas herramientas para la automatización, un depurador, un sistema de ayuda para la construcción de interfaces gráficas de usuario y, opcionalmente, un sistema de control de versiones.

Hoy día los entornos de desarrollo proporcionan un marco de trabajo para la mayoría de los lenguajes de programación existentes en el mercado como es el caso de C++, C#, Java, Python y Visual Basic.

Además es posible que un mismo entorno de desarrollo tenga la posibilidad de utilizar varios lenguajes de programación, como es el caso de Eclipse. (20)

Entre los entornos integrados de desarrollo se encuentran los siguientes:

- Eclipse
- Zend Studio
- NetBeans
- Visual Studio.NET

Eclipse

Eclipse fue desarrollado originalmente por IBM. Es ahora desarrollado por la Fundación Eclipse, una organización independiente que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Es un entorno de desarrollo integrado, de Código abierto y Multiplataforma. Mayoritariamente se utiliza para desarrollar lo que se conoce como "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores, es una potente y completa plataforma de Programación, desarrollo y compilación de elementos tan variados como sitios web, programas en C++ o aplicaciones Java. Además contiene una atractiva interfaz que lo hace fácil y agradable de usar.

La base para Eclipse es la Plataforma de cliente enriquecido (del Inglés Rich Client Platform RCP). Los siguientes componentes constituyen la plataforma de cliente enriquecido:

- Plataforma principal: inicio de Eclipse, ejecución de plugins.
- OSGi: una plataforma para bundling estándar.
- El Standard Widget Toolkit (SWT).
- JFace: manejo de archivos, manejo de texto, editores de texto.
- El Workbench de Eclipse: vistas, editores, perspectivas, asistentes. (21)

NetBeans

NetBeans se refiere a una plataforma para el desarrollo de aplicaciones de escritorio usando Java y a un entorno de desarrollo integrado (IDE) desarrollado usando la Plataforma NetBeans.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados de manera independiente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software. NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. Sun MicroSystems fundó el proyecto de código abierto NetBeans en junio 2000 y continúa siendo el patrocinador principal de los proyectos. (22)

Visual Studio .NET

Es un entorno de desarrollo integrado para sistemas operativos Windows. Soporta varios lenguajes de programación tales como C++, C#, J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros. Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET a partir de la versión net 2002. Así se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles. (23)

1.6 Fundamentación de las herramientas, metodologías y tecnologías a utilizar

El rápido desarrollo de las tecnologías en el mundo hace que se experimenten constantes cambios, nuevas técnicas, metodologías y herramientas surgen para facilitar y proporcionar mejoras en la evolución de la información. El sistema a desarrollar pertenece al proyecto Dactilab del Departamento de Biometría del Centro de Identificación y Seguridad Dactilar (CISED), este centro cuenta con un marco de trabajo definido para el desarrollo de los sistemas informáticos que desarrolla. A continuación se describen las tecnologías, metodologías y herramientas definidas para desarrollar el sistema.

Se considera a RUP el proceso de desarrollo más general de los existentes actualmente, posee grandes ventajas para el análisis, implementación y documentación de sistemas orientados a objetos. Esta metodología está preparada para desarrollar todo tipo de proyectos, es muy organizada y genera desde sus inicios una documentación robusta y especificada de todo el proceso de manera general. La arquitectura provee la estructura sobre la cual guiar el trabajo en iteraciones, mientras que los casos de uso definen las metas y dirigen el trabajo en cada iteración.

La herramienta utilizada para la modelación del sistema es Visual Paradigm, brinda la gran ventaja que es software libre. Utiliza un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación y por sus características cubre todo el ciclo de vida de un proyecto.

Como entorno integrado de desarrollo se usará Visual Studio porque permite a los desarrolladores crear aplicaciones de alta calidad con gran rapidez, más seguras, confiables y administrables en cualquier entorno que soporte la plataforma .NET. Es una plataforma de ejecución intermedia multilenguaje.

Para la implementación se utiliza el Lenguaje de Programación C# desarrollado por la empresa Microsoft Corporation, lenguaje de gran efectividad y hace una recopilación de lo mejor de C++ y Java. Lenguaje de programación de propósito general que ofrece economía sintáctica, control, flujo y estructuras sencillas y un buen conjunto de operadores. En poco tiempo, un programador puede utilizar la totalidad del lenguaje. Es un lenguaje orientado a objetos, tiene una sintaxis muy parecida a Java y posee la potencia de C++.

1.7 Conclusiones

En este capítulo se elaboró un estudio sobre las aplicaciones de generación de base de datos de huellas dactilares a nivel internacional y nacional, llegando a conclusiones que construir una base de datos de huellas dactilares es importante para evaluar el desempeño de los sistemas de reconocimiento de personas. Se realizó un estudio de las características, ventajas y desventajas acerca de las distintas metodologías, tecnologías y herramientas tendientes en el mundo actual, lleno de nuevos avances, a la hora de desarrollar este tipo de aplicaciones.

Capítulo 2: Características del Sistema

2.1 Introducción

En este capítulo se describen las principales características del sistema propuesto, se hace una valoración del negocio, que debido a la poca estructuración de los procesos, se requieren definir conceptos los cuales se puedan agrupar en un modelo de dominio, permitiendo especificar correctamente los requisitos funcionales y representarlos mediante la construcción de un diagrama de casos de uso. Se realiza un análisis y una representación gráfica de diagramas del diseño correspondiente al sistema. Se realizarán los diagramas de clases del análisis, diagramas de clases del diseño y diagramas de interacción, realizándose una breve descripción, se especifican los patrones utilizados, mediante ellos se logra un mejor entendimiento para el posterior desarrollo del sistema.

2.2 Modelo de dominio

Esta investigación se basará en un modelo de dominio, debido a que no se tienen claro los procesos del negocio, el mismo permitirá mostrar los principales conceptos con los que trabaja la aplicación en desarrollo, lo que ayuda a comprender el entorno del sistema. Este modelo va a contribuir más adelante a describir las clases más importantes dentro del contexto la aplicación.

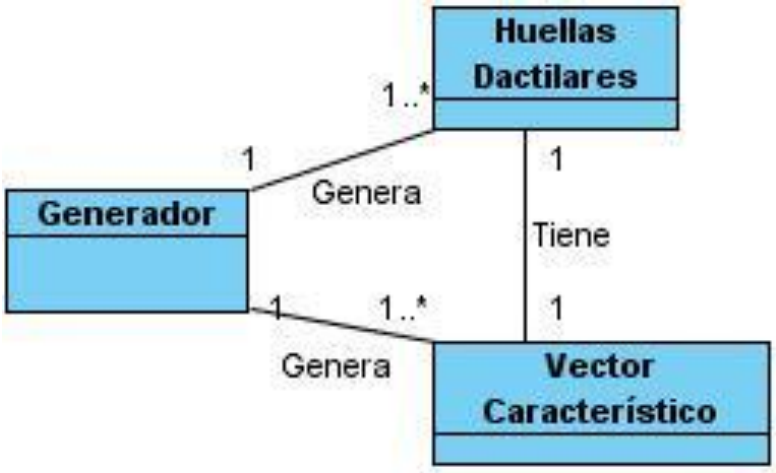

Figura 17. Modelo de dominio.

2.3 Propuesta del sistema

Descripción general de la propuesta de sistema, ¿Cómo debe funcionar?

El sistema consiste en una herramienta para la generación de bases de datos de huellas dactilares artificiales generadas mediante parámetros definidos por el usuario. El objetivo de esta base de datos es la realización de pruebas a algoritmos de extracción y cotejo de minucias. El sistema está conformado por dos módulos: la generación de la huella dactilar y la creación de la base de datos.

Las huellas dactilares son generadas a partir de una serie de parámetros configurados por el usuario, comenzando por la selección de la forma del área de adquisición, ubicando de manera aleatoria las posiciones de las singularidades, generando minucias e introduciéndole roturas, transformaciones elásticas, ruido, aplicándole rotaciones y traslaciones, para construir huellas dactilares lo más reales posibles junto a su vector característico.

La generación de bases de datos de huellas dactilares se realiza mediante la configuración de los parámetros que regulan el proceso, estando presentes en la descripción del requisito funcional generación de bases de datos, pudiendo ser generadas la cantidad de huellas deseadas por el usuario.

2.3.1 Requisitos funcionales

Los requisitos funcionales son características del sistema que expresa una capacidad de acción o una funcionalidad del mismo; generalmente expresada en una declaración en forma verbal. (24)

RF-1 Generar Huellas Dactilares.

1.1 Crear área de trabajo.

1.2 Generar el campo de orientación.

1.3 Crear el mapa direccional de crestas.

1.4 Aplicar transformaciones (desplazamiento, rotación, traslación).

1.5 Agregar ruido.

1.6 Generar el fondo de la imagen.

RF-2 Crear Base de Datos.

RF-3 Obtener el vector característico.

2.3.2 Requisitos no funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Son características que hacen al producto atractivo, usable, rápido o confiable.

RNF-1 Apariencia o interfaz externa

- La interfaz de esta aplicación debe ser sencilla y amigable al usuario.
- Debe tener colores refrescantes a la vista.

RNF-2 Usabilidad

- La aplicación debe ser de fácil manejo para los usuarios que tengan niveles básicos sobre computación.

RNF-3 Rendimiento

- Los tiempos de respuestas del sistema deben ser rápidos, al igual que la velocidad de procesamiento de la información para lograr respuestas rápidas del mismo.

RNF-4 Requisitos de hardware.

- PC Intel Pentium 4 o superior.
- CPU 1.6 GHZ o superior.
- HDD SCCI 80 GB o superior.
- 512 GB de Memoria RAM o superior.

2.4 Modelo de casos de uso del sistema

Una vez definidos los requisitos funcionales del sistema, estos se representarán mediante un diagrama de casos de uso. Para ello primero se debe definir claramente cuáles son los actores que van a interactuar con el sistema, y los casos de uso que representarán todas las funcionalidades del sistema.

Un Caso de Uso es una descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica. Un actor representa un conjunto coherente de roles que juegan los usuarios de los Casos de Uso cuando interactúan con éstos. Los actores pueden ser personas (roles que desempeñan las personas) u otros sistemas de cómputo. (25)

Tabla 1. Actores del sistema

Actores del Sistema	Justificación
Usuario	Representa una persona que es la encargada de hacer todos los procesos que se llevan a cabo para la generación de huellas dactilares.

Tabla 2. Resumen del caso uso: Generar huellas dactilares

CU 1	Generar huellas dactilares
Actor	Usuario
Descripción	El usuario solicita generar huella dactilar que simule una huella real.
Referencia	RF-1

Tabla 3. Resumen del caso uso: Crear base de datos

CU 2	Crear Base de datos
Actor	Usuario
Descripción	El usuario solicita crear una base de datos de huellas dactilares que contiene todas las huellas resultantes de la funcionalidad generar huella dactilar.
Referencia	RF-2

Tabla 4. Resumen del caso uso: Obtener el vector característico

CU 3	Obtener el vector característico
Actor	Usuario
Descripción	El sistema construye el vector característico almacenándolo en la base de datos junto a la huella dactilar correspondiente.
Referencia	RF-3

2.5 Diagrama de caso de uso del sistema.

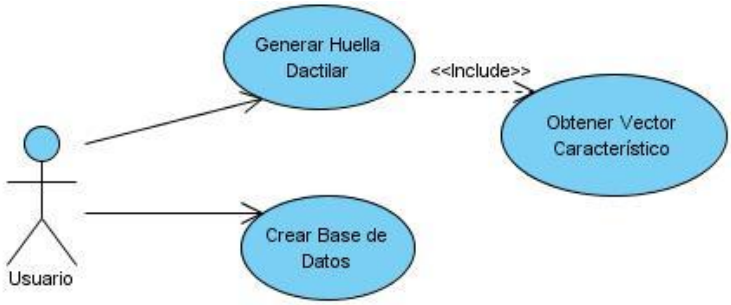

Figura 18. Diagrama de caso de uso del sistema

Descripción de caso de uso del sistema.

Tabla 5. Descripción de caso de uso: Generar huella dactilar

Nombre del caso de uso: Generar huella dactilar	
Actores:	Usuario
Propósito	Generar huellas dactilares que simulen una huella real.
Resumen	El caso de uso inicia cuando el usuario selecciona la opción generar huella dactilar.
Referencia	RF-1
Precondiciones	
Flujo normal de eventos	
Acción del actor	**Respuesta del sistema**
1. El usuario selecciona la opción **FingerPrint Generation.**	2. El sistema muestra una interfaz que posibilita escoger el tamaño de la imagen.
3. El usuario escoge el tamaño de la imagen que se obtendrá como resultado del proceso.	4. El sistema muestra una interfaz para seleccionar el área de adquisición.
5. El usuario escoge las dimensiones deseadas, selecciona el dedo de la huella que se va a tomar y selecciona el botón next.	6. El sistema muestra una interfaz donde se genera el campo de orientación de la imagen.
7. El usuario selecciona la clase de huella, selecciona el botón next.	8. El sistema muestra una interfaz para la generación de los patrones ridges.

9. El usuario selecciona la cantidad de semillas, la densidad de las crestas, inicia la generación y luego selecciona el botón next.	10. El sistema muestra una interfaz para la selección de las roturas que va a tener la huella.
11. El usuario mueve el control deslizante para ajustar la cantidad de roturas y luego selecciona el botón next.	12. El sistema muestra una interfaz para la selección de la zona de contacto del dedo.
13. El usuario mueve los controles deslizantes para seleccionar la región de contacto, aplica y luego selecciona el botón next.	14. El sistema muestra una interfaz para seleccionar la presión del dedo.
15. El usuario mueve el control deslizante seleccionando la presión del dedo, aplica y luego selecciona el botón next.	16. El sistema muestra una interfaz para la distorsión de la huella.
17. El usuario mueve los controles deslizantes de traslación, rotación y elasticidad, aplica y selecciona el botón next.	18. El sistema muestra una interfaz para agregar roturas y ruido a la huella.
19. El usuario mueve los controles deslizantes de roturas y ruido, selecciona la suavidad de la huella, aplica y selecciona el botón next.	20. El sistema muestra una interfaz para darle rotación y traslación a la huella.
21. El usuario mueve los controles deslizantes de rotación y traslación, aplica y finaliza el proceso.	
Flujo alterno	
Pos condiciones	Se generó la huella dactilar.

Tabla 6. Descripción de caso de uso: Crear base de datos

Nombre del caso de uso: Crear base de datos	
Actores:	Usuario
Propósito	Contar con una base de datos para la realización de pruebas a los algoritmos biométricos de reconocimiento de persona mediante su huella dactilar.
Resumen	El caso de uso inicia cuando el usuario solicita la opción crear base de datos para guardar en ella todas las huellas generadas.

Referencia	RF-2
Precondiciones	Se haya generado la huella dactilar.
Flujo normal de eventos	
Acción del actor	**Respuesta del sistema**
1. El usuario selecciona la opción **DataBase Generation.**	2. El sistema muestra una interfaz para escoger los valores iniciales de la huella, la clase de huella, la cantidad a generar y la configuración del proceso.
3. El usuario llena los campos, selecciona la cantidad de huella a generar y presiona el botón start.	4. El sistema muestra una interfaz que muestra el estado del proceso.
Flujo alterno	
Pos condiciones	Se creó la base de datos.

Tabla 7. Descripción de caso de uso: Obtener el vector característico

Nombre del caso de uso: Obtener el vector característico	
Actores:	Usuario
Propósito	Obtener un vector que contenga las posiciones exactas de las singularidades y las minucias de la huella dactilar.
Resumen	El caso de uso inicia cuando se generan las singularidades de la huella dactilar.
Referencia	RF-3
Precondiciones	Se haya inicializado la opción generar huella dactilar.
Flujo normal de eventos	
Acción del actor	**Respuesta del sistema**
	1. Guarda las posiciones de las singularidades.
	2. Guarda las posiciones de las minucias que contiene la huella dactilar.
Flujo alterno	
Pos condiciones	Se obtuvo el vector característico de la huella dactilar.

2.6 Análisis

El análisis es la etapa del flujo de trabajo Análisis y Diseño, que se encarga de refinar y estructurar los requisitos identificados en el capítulo anterior, con el objetivo de lograr una mejor comprensión y descripción de los mismos, que facilite estructurar el sistema en su totalidad.

En la construcción del modelo de análisis se identifican las clases que describen la realización de los casos de uso, los atributos y las relaciones entre ellas. Con esta información se construye el diagrama de clases del análisis. El modelo de análisis es el resultado de la actividad de analizar los casos de uso y su realización suaviza la transición al diseño y se utiliza para tener una visión general de la propuesta de sistema.

2.6.1 Diagrama de clases del análisis

Un diagrama de clases del análisis es un artefacto en el que se representan los conceptos en un dominio del problema. Representa el funcionamiento del mundo real, no de la implementación automatizada del mismo. Las clases del análisis se centran en los requisitos funcionales y entre ellas se establecen relaciones de asociación, agregación / composición, generalización / especialización y tipos asociativos. (26)

RUP propone clasificar a las clases en:

- **Clase Interfaz**: se utilizan para modelar la interacción entre el sistema y los actores.
- **Clase Entidad**: se utilizan para modelar información que posee una vida larga y que a menudo es persistente. Modelan la información y el comportamiento asociado de algún fenómeno o concepto, como una persona, un objeto, o un suceso del mundo real.
- **Clase Controladora**: Coordinan la realización de uno o unos pocos casos de uso coordinando las actividades de los objetos que implementan la funcionalidad del caso de uso. (27)

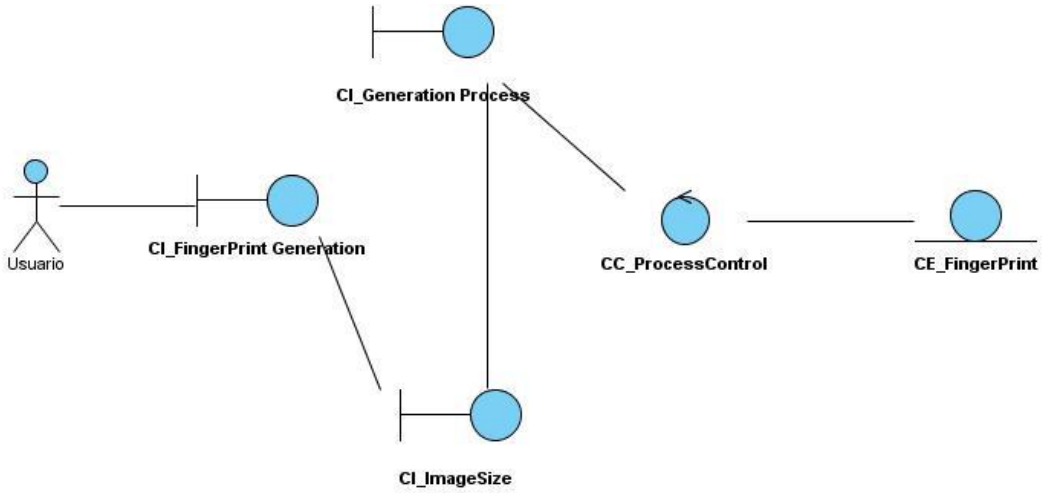

Figura 19. Diagrama de clases del análisis CUS: Generar huella dactilar

Figura20. Diagrama de clases del análisis CUS: Crear base datos

2.6.2 Diagrama de colaboración

Los diagramas de interacción (diagramas de secuencia o diagramas de colaboración) explican gráficamente cómo los objetos interactúan a través de mensajes para realizar las tareas.

Durante la realización del análisis se utilizan los diagramas de colaboración, pues su principal objetivo es precisamente mostrar las interacciones entre objetos organizadas entorno a objetos y los enlaces entre ellos. (28)

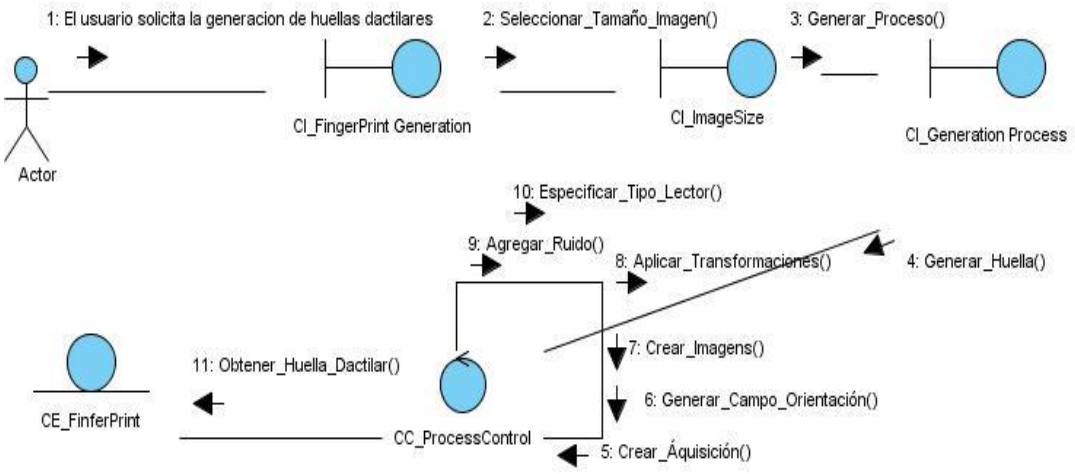

Figura 21. Diagrama de colaboración CUS: Generar huella dactilar

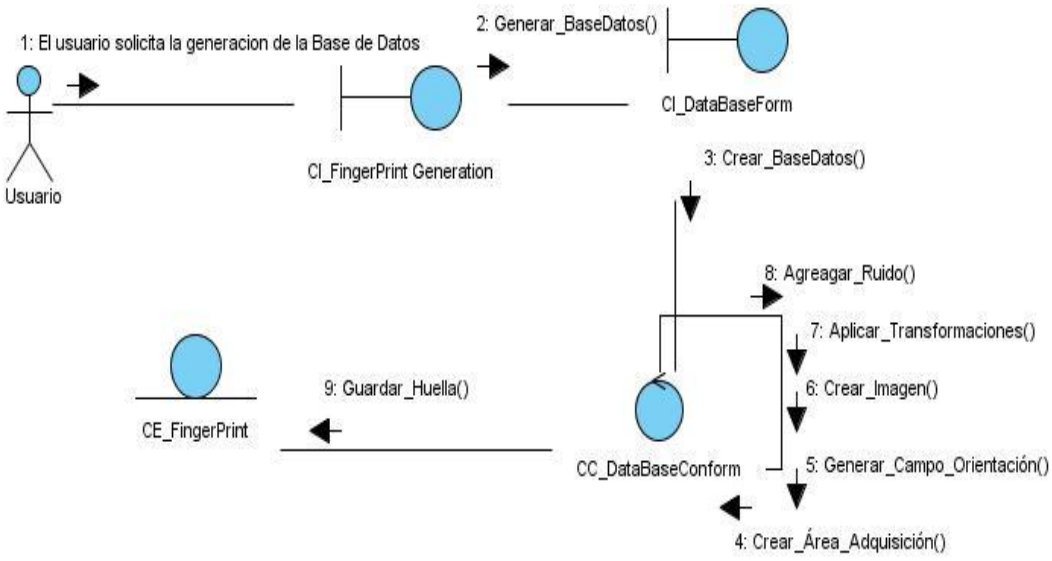

Figura 22. Diagrama de colaboración CUS: Crear base de datos

2.7 Diseño

El propósito fundamental del diseño es indicar como se llevará a cabo el sistema y encontrar su forma (incluida la arquitectura) para que soporte todos los requisitos, incluyendo los no funcionales y las restricciones que se le suponen. Una entrada esencial en el diseño es el resultado del análisis, o sea el modelo de análisis, que proporciona una comprensión detallada de los requisitos. (29)

2.7.1 Patrones de diseño

Los patrones de diseño son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Se consideran como procedimientos para solucionar diversas veces un problema del mismo tipo y son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software. Los desarrolladores lo usan como una forma de reutilizar la experiencia, clasificando las soluciones con términos de común denominación. (30)

En el diseño de la aplicación se usarán los patrones GRASP para la implementación:

- **Patrón Alta Cohesión**: asigna una responsabilidad, de modo que la cohesión permanezca alta. Este patrón incrementa la claridad y facilita la comprensión del diseño e incrementa las capacidades de reutilización.

- **Patrón Experto:** permite darle solución al problema de cuál sería el principio fundamental para asignar responsabilidades en el diseño orientado a objetos. La solución factible es asignar la responsabilidad a la clase contenedora de la información necesaria para cumplir la responsabilidad. El uso de este patrón permitirá a los objetos valerse de su propia información para hacer lo que se les pide, favorece la existencia de mínimas relaciones entre las clases, lo que permite contar con un sistema sólido y fácil de mantener.

- **Patrón Creador:** se considera para la asignación de responsabilidades a las clases relacionadas con la creación de objetos, de forma tal que una instancia de un objeto solo pueda ser creada por el objeto que contiene la información necesaria para ello. El uso de este patrón admite crear las dependencias mínimas necesarias entre las clases, beneficiando el mantenimiento del sistema y brindando mejores oportunidades de reutilización.

- **Patrón Controlador**: se considera para efectuar las asignaciones en cuanto al manejo de los eventos del sistema y definir sus operaciones. Crear una nueva instancia por la clase que tiene la información necesaria para realizar la creación del objeto, usa directamente las instancias creadas del objeto, almacena o maneja varias instancias de la clase y contiene o agrega la clase.

- **Patrón Bajo Acoplamiento:** garantiza que exista una alta reutilización entre las funcionalidades de las clases y una escasa dependencia, contribuyendo al mantenimiento de las mismas. El uso de los patrones Experto y Creador favorecen al bajo acoplamiento entre las clases del sistema. Este patrón se tuvo en cuenta por la importancia que tiene realizar un diseño de clases independientes que puedan soportar los cambios de una manera fácil y permitan la reutilización.

2.7.2 Patrones de arquitectura

La arquitectura del software es el diseño de más alto nivel de la estructura de un sistema. También es denominada Arquitectura Lógica, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software para un sistema informático. Establece los fundamentos para que analistas, diseñadores, programadores y otros roles, trabajen en una línea común que permita alcanzar los objetivos del sistema de información, cubriendo todas las necesidades.

Generalmente, no es necesario inventar una nueva arquitectura software para cada sistema de información. Lo habitual es adoptar una arquitectura conocida en función de sus ventajas e inconvenientes para cada caso en concreto. Así, las arquitecturas más universales son:

- **Cliente-servidor.** Esta arquitectura se divide en dos partes claramente diferenciadas, la primera es la parte del servidor y la segunda la de un conjunto de clientes. Normalmente el servidor es una máquina bastante potente que actúa de depósito de datos y funciona como un sistema gestor de base de datos (SGBD). Por otro lado los clientes suelen ser estaciones de trabajo que solicitan varios servicios al servidor. Ambas partes deben estar conectadas entre sí mediante una red. (31)

- **Tuberías y Filtros.** Se descompone el sistema en módulos funcionales. Interacción sucesiva transformación de flujos de datos. Los datos llegan a un filtro, se transforman y son pasados a través de tubos al siguiente filtro.

- **Arquitectura de tres niveles.** Generalización de la arquitectura cliente-servidor donde la carga se divide en tres partes con un reparto claro de funciones: una capa para la presentación, otra para el cálculo y otra para el almacenamiento. Una capa solamente tiene relación con la siguiente.

- **Arquitectura Orientada a Servicios (SOA**). Establece un marco de diseño para la integración de aplicaciones independientes de manera que desde la red pueda accederse a sus funcionalidades, las cuales se ofrecen como servicios. La forma más habitual de implementarla es mediante Servicios Web, una tecnología basada en estándares e independiente de la plataforma, con la que SOA puede descomponer aplicaciones monolíticas en un conjunto de servicios e implementar esta funcionalidad en forma modular. (32)

- **Modelo Vista Controlador (MVC)** Es un patrón de diseño de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones Web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. (33)

El patrón de arquitectura propuesta para la implementación de la aplicación Sintetizador de Huellas Dactilares es Tuberías y Filtros, la misma provee una estructura para procesar flujos de datos. Cada paso de procesamiento se encapsula en un filtro. Los datos se pasan usando los tubos entre filtros adyacentes. Recombinado los filtros se pueden construir distintas familias de sistemas relacionados.

Estructura

- Filtros:

  - unidades de procesamiento en las arquitecturas de tubos y filtros.
  - los filtros no saben de la existencia o identidad de otros filtros.
  - enriquece, refina y/o transforma los datos.
  - el procesamiento se inicia cuando:
    - ✓ el elemento siguiente solicita datos.
    - ✓ el elemento anterior envía datos.
    - ✓ el filtro tiene un ciclo interno que solicita datos del filtro anterior y envía datos a siguiente.

- Tubos:
    - conectan origen-filtro, filtro-filtro, filtro-destino.
    - si dos filtros activos se comunican con un tubo, éste los sincroniza.
- Origen de datos:
    - input del sistema.
    - brinda al sistema una serie de datos con la misma estructura o tipo:
    - pueden también ser activos o pasivos.
- Destino de datos:
    - a él llegan los resultados del procesamiento del sistema.
    - existen destinos activos y pasivos.

Implementación

- Dividir las tareas en una secuencia de pasos de procesamiento: los procesos deben ser independientes y ordenados.
- Definir el formato de los datos transmitidos a través de cada pipe: flexibilidad vs. performance.
- Decidir implementación de cada tubo: Determinar la implementación de los filtros (activos o pasivos).
- Diseñar e implementar los filtros.
- Diseñar política de errores.
- Instalar el sistema.

Características

- Los tubos no tienen estado interno y simplemente comunican datos entre los filtros.

- Tubos y filtros ejecutan en forma no determinado sobre los datos hasta que éstos se acaban.
- Restricciones de conexión: existe una fuente de datos conectada al puerto input de un filtro; exist un destino de datos conectado al puerto output de un filtro.

Ventajas

- Simplicidad:
    - forma limitada de interacción con el ambiente.
    - la funcionalidad es sólo la composición de funcionalidades más sencillas.
    - no existen interacciones complejas.
    - promueve la reutilización y simplifica el mantenimiento.
    - composición jerárquica: una combinación de tubos y filtros puede mostrarse externamente como un único filtro.
    - por la independencia de procesamiento de los filtros, puede hacerse ejecución paralel o distribuida aumentando la disponibilidad y la performance.

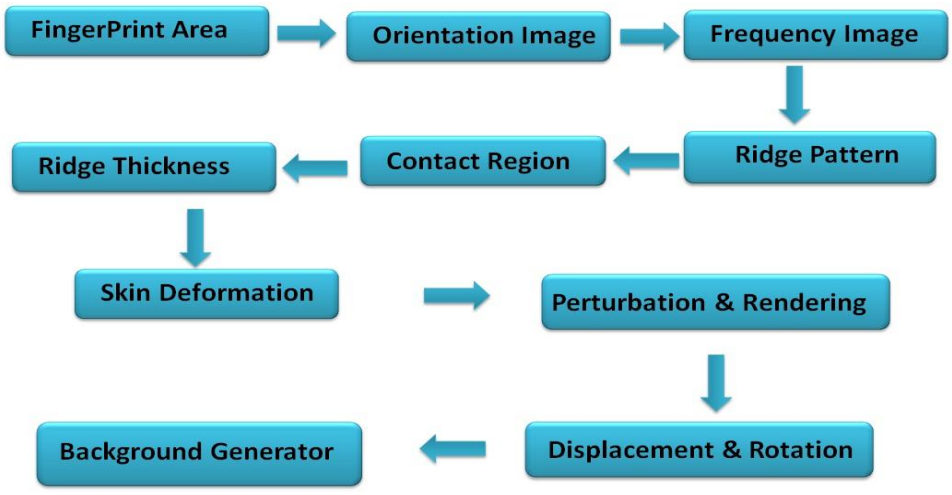

Figura 23. Arquitectura Tuberías y Filtros

2.7.3 Diagrama de clases del diseño

Un diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación. Los diagramas de clases del diseño contienen la siguiente información:

- Clases, asociaciones y atributos.
- Interfaces, con sus operaciones y constantes.
- Métodos.
- Información sobre los tipos de los atributos.
- Navegabilidad.
- Dependencias. (34)

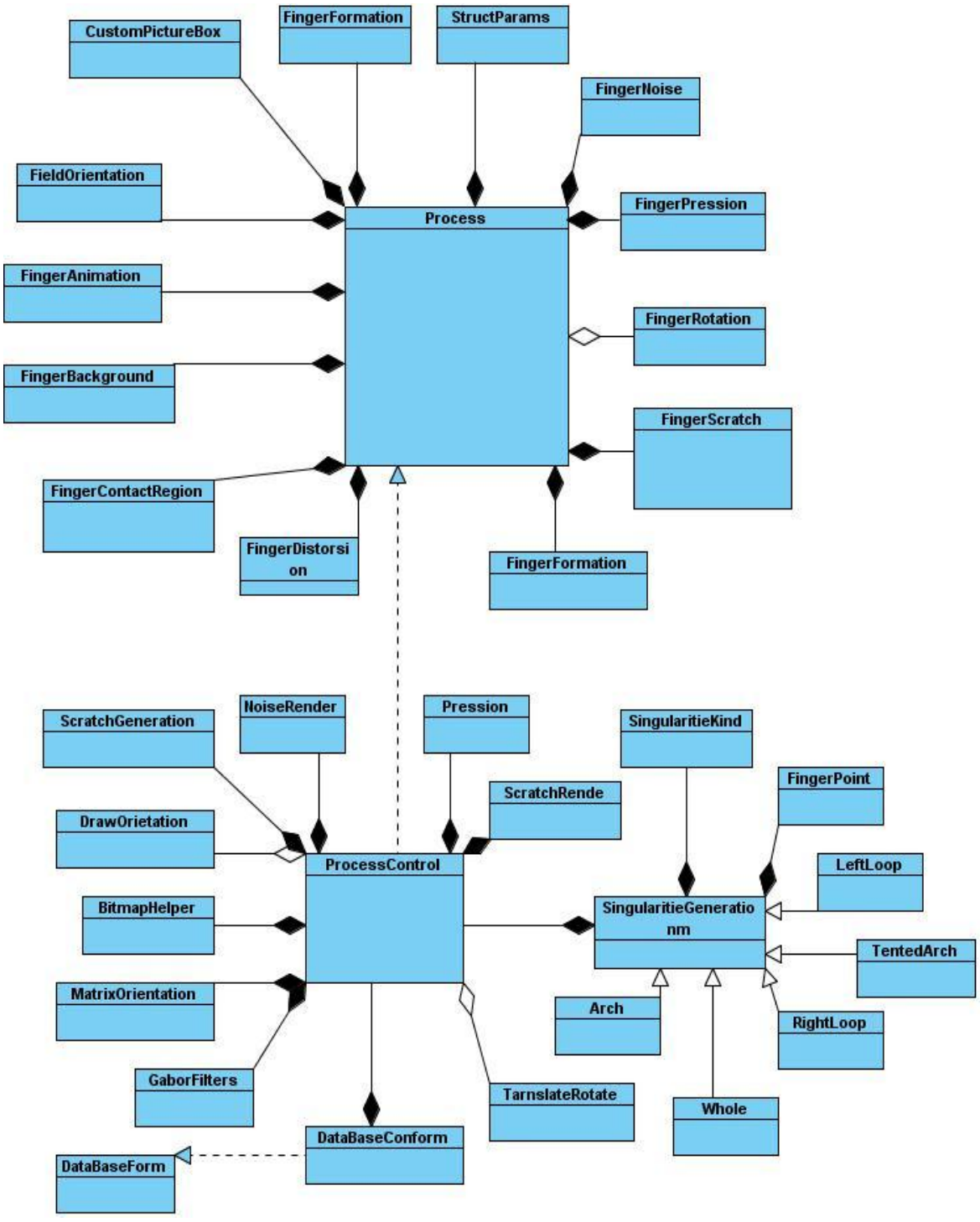

Figura 24. Diagrama de clases del diseño

En el anexo 1 se encuentra la descripción las clases Process y ProcessControl.

2.8 Conclusiones

En este capítulo se expusieron las características del sistema, se construyó el modelo de dominio, lo que dió lugar a la identificación de un conjunto de funcionalidades que debe cumplir el sistema Sintetizador de Huellas Dactilares, expresadas en requisitos funcionales y agrupados en casos de uso del sistema. Se realizaron los diagramas de clases del análisis, diagramas de clases del diseño y diagramas de interacción (diagrama de colaboración) para cada caso de uso identificado, dejando todo listo para dar paso a la construcción del sistema. Como parte de este capítulo se describe la arquitectura y los patrones de diseño utilizados.

Capítulo 3: Implementación y Prueba

3.1 Introducción

En este capítulo se realizará la implementación del sistema dándole cumplimiento a los requerimientos planteados al inicio de la investigación, se expondrá además el diagrama de componente sumamente importante dentro del flujo de trabajo de implementación. Se visualizan las interfaces de usuario y se da una explicación de cada una de ellas para un mayor entendimiento acerca del funcionamiento de la aplicación. Se realizan las pruebas donde se valida el funcionamiento de los requisitos funcionales.

3.2 Modelo de implementación

El modelo de implementación describe como los elementos del modelo de diseño y las clases, se implementan en términos de componentes, como ficheros de código fuente, ejecutables, entre otros. También describe como se organizan los componentes de acuerdo con los mecanismos de estructuración disponible en el entorno de implementación y en el lenguaje de programación utilizado y como dependen los componentes uno del otro. (35)

3.2.1 Diagrama de componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones**,** representan todos los tipos de elementos software que entran en la fabricación de aplicaciones informáticas. Las relaciones de dependencia se utilizan en los diagramas de componentes para indicar que un componente utiliza los servicios ofrecidos por otro componente. Un diagrama de componentes representa las dependencias entre componentes software, incluyendo componentes de código fuente, componentes del código binario, y componentes ejecutables. (36)

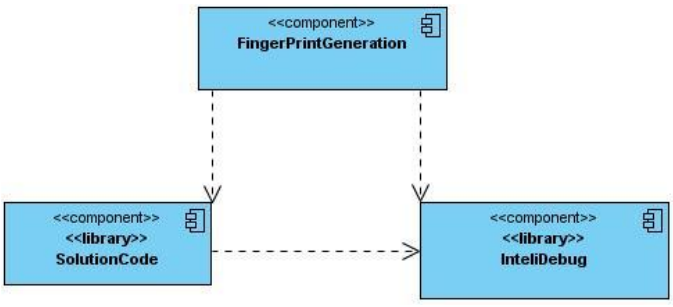

Figura 25. Diagrama de componentes

3.3 Interfaz de usuario

La interfaz gráfica del usuario es el medio por el cual este interactúa con el sistema, por lo que su diseño debe ser amigable, consistente y que no requiera usuarios con un alto nivel informático. Una aplicación con una interfaz bien diseñada, además de un buen diseño gráfico, debe tener una buena navegabilidad, usabilidad y distribución de los contenidos. Este trabajo utilizará en el diseño los principios antes mencionados.

Después de una breve reseña de la aplicación, se muestran las interfaces para que se tenga un mayor entendimiento de lo que hace el sistema.

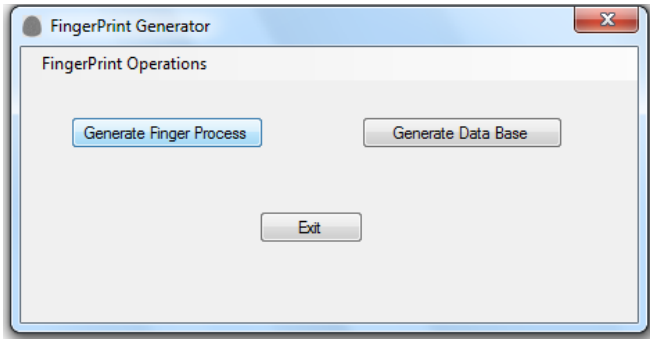

Figura 26. Interfaz FingerPrint Generation

Esta es la interfaz principal de la aplicación, donde el usuario escoge si seleccionar la generación de la huella dactilar o la generación de la base de datos.

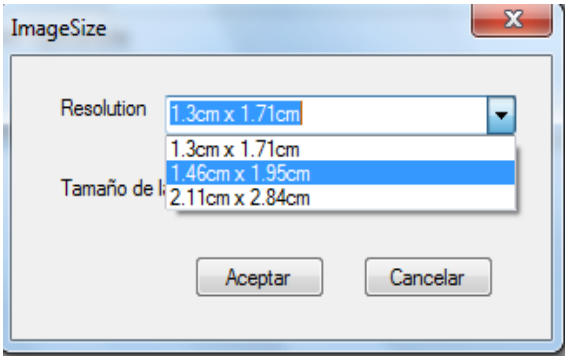

Figura 27. Interfaz ImageSize

El sistema muestra esta interfaz para seleccionar el tamaño de la imagen final de la huella dactilar generada.

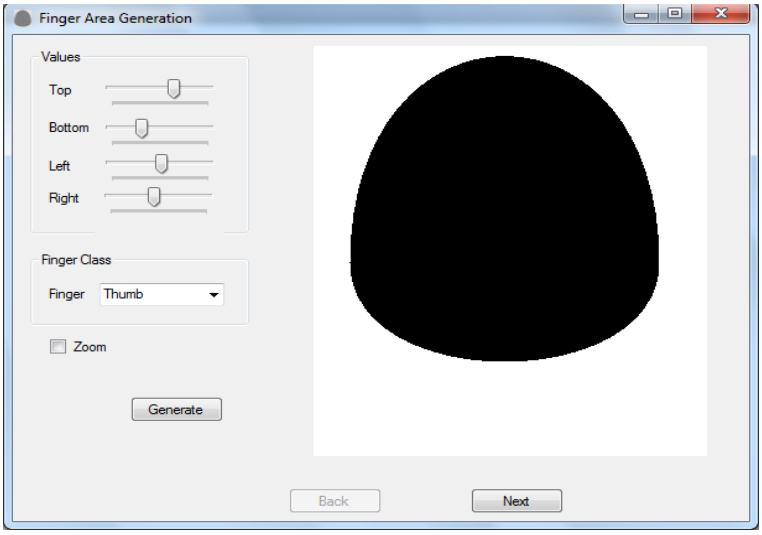

Figura 28. Interfaz Finger Area Generation

Una vez seleccionada el tamaño de la imagen empieza la generación de la huella, comenzando por seleccionar el área de adquisición, escogiendo el dedo al que le quiere tomar la huella, dándole las dimensiones deseadas o generarlas de manera aleatoria.

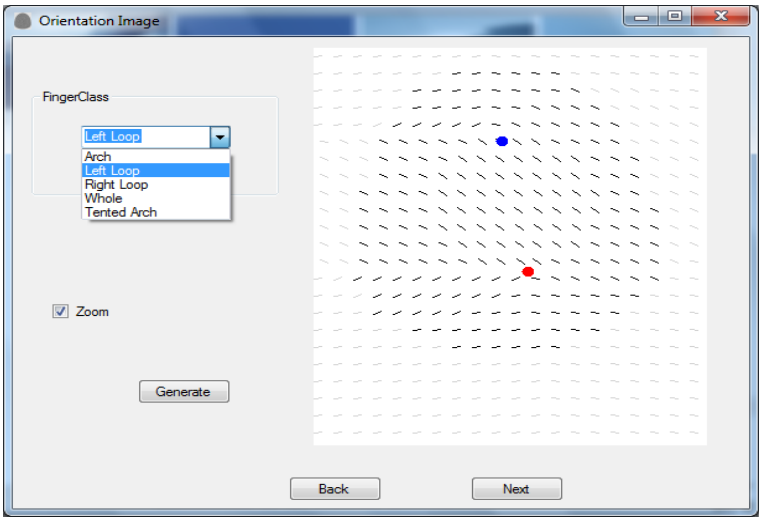

Figura 29. Interfaz Orientation Image

En esta interfaz se escoge la clase de huella que desea generar y a partir de esta se generan las singularidades de manera aleatoria.

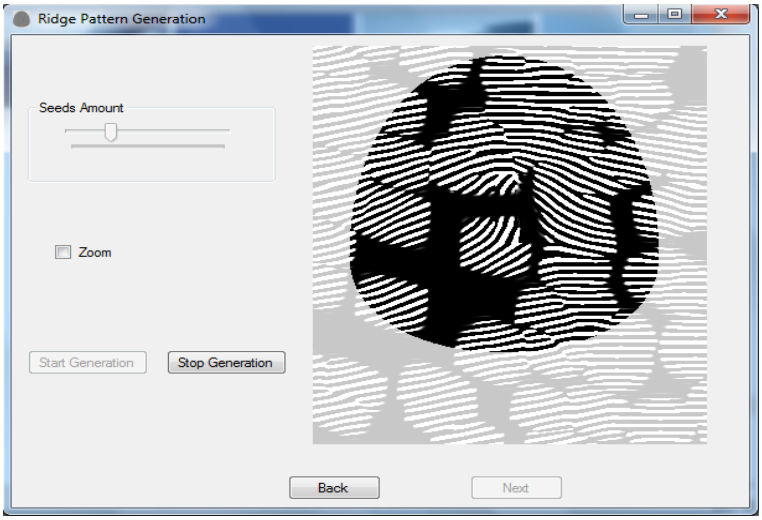

Figura 30. Ridge Pattern Generation

En este caso se genera un conjunto de semillas y luego a partir de esta comienza la generación de la imagen, pudiéndose interrumpir si el usuario desea.

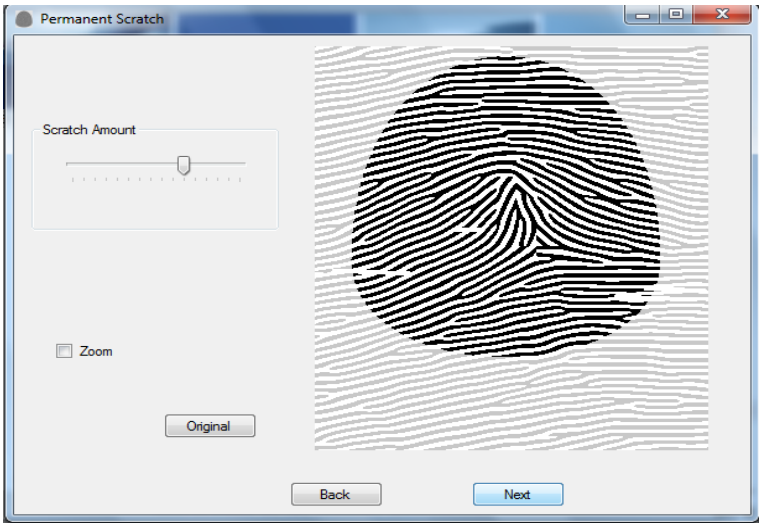

Figura 31. Interfaz Permanent Scratch

En esta interfaz se le aplican roturas de la huella, teniendo en cuenta las pequeñas cicatrices que puede tener una huella real.

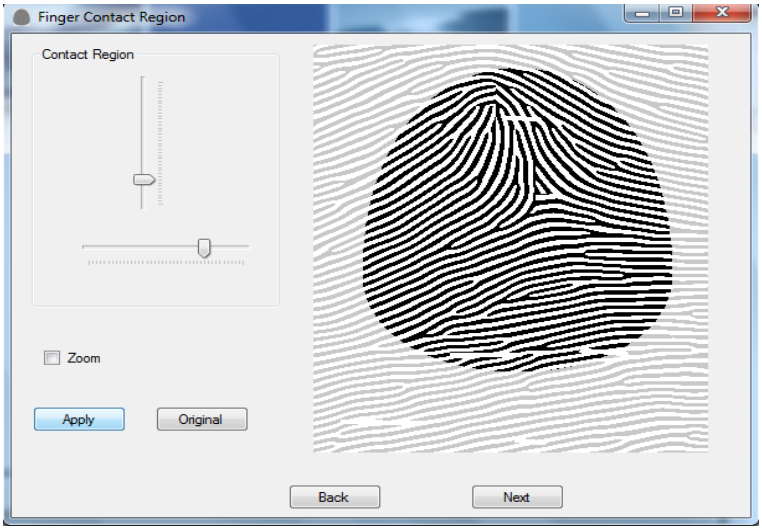

Figura 32. Interfaz Finger Contact Region

Aquí se mueve la región de contacto, debido a que la huella no siempre se encuentra en el centro del lector.

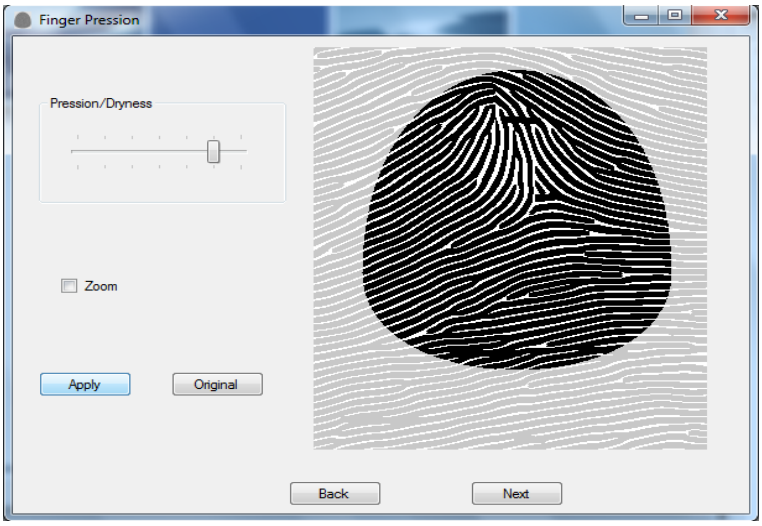

Figura 33. Interfaz Finger Pression

Esta interfaz simula la aplicación de un patrón morfológico de erosión y presión. La erosión es cuando la piel se encuentra seca y aparecen los ridges más finos y la presión cuando la piel está húmeda y los ridges son más gruesos.

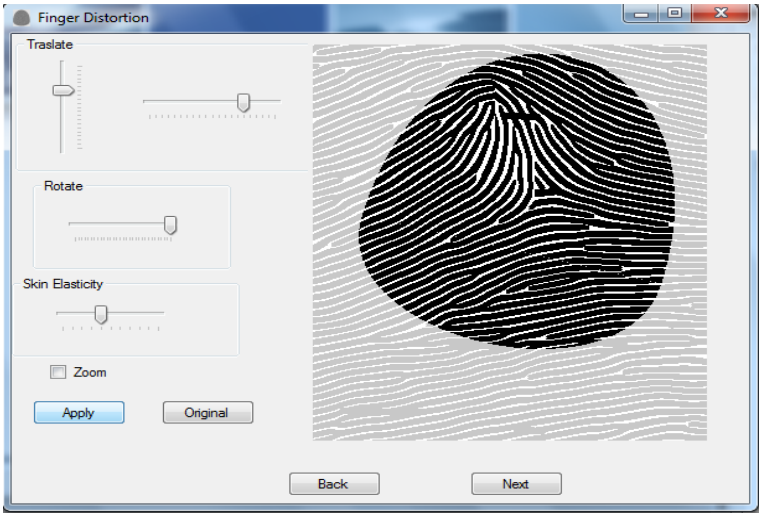

Figura 34. Interfaz Finger Distortion

En este caso se rota y traslada la región de la huella dactilar y se le aplica una distorsión para simular la deformación producida por la propiedad elástica de la piel.

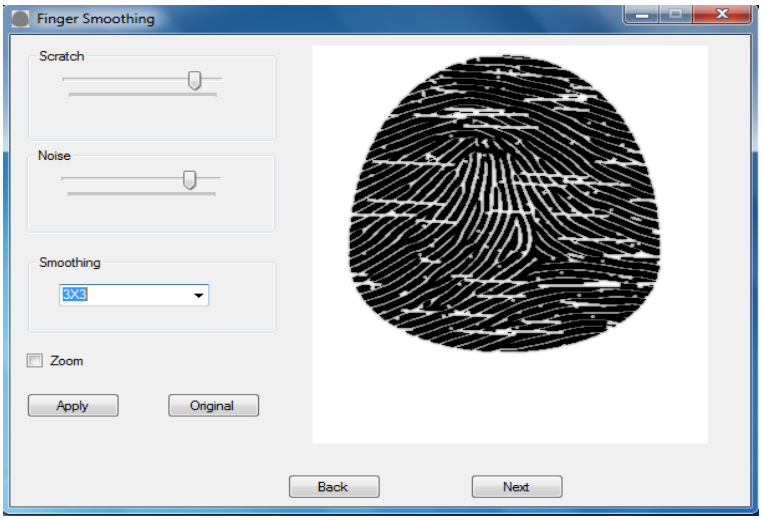

Figura 35. Interfaz Finger Smoothing

En esta interfaz se le introducen roturas permanentes, ruido y desenfocado o suavizado a la huella para obtener una huella lo más real posible.

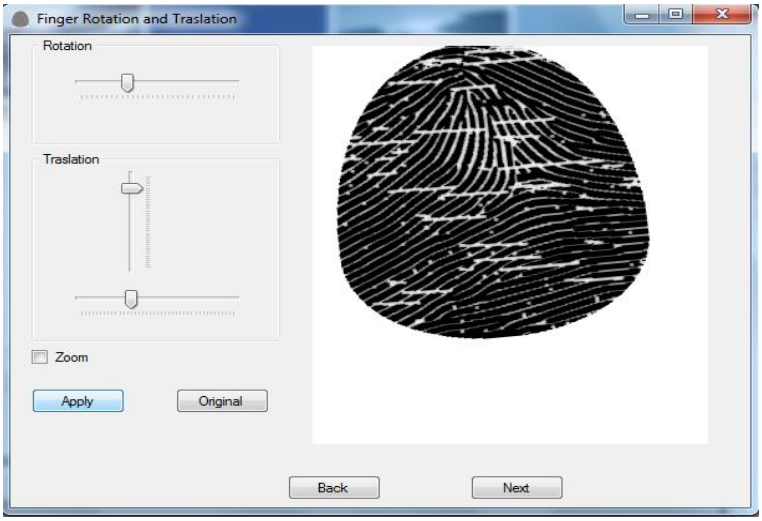

Figura 36. Interfaz Finger Rotation and Traslation

En este caso se rota y se traslada la huella final.

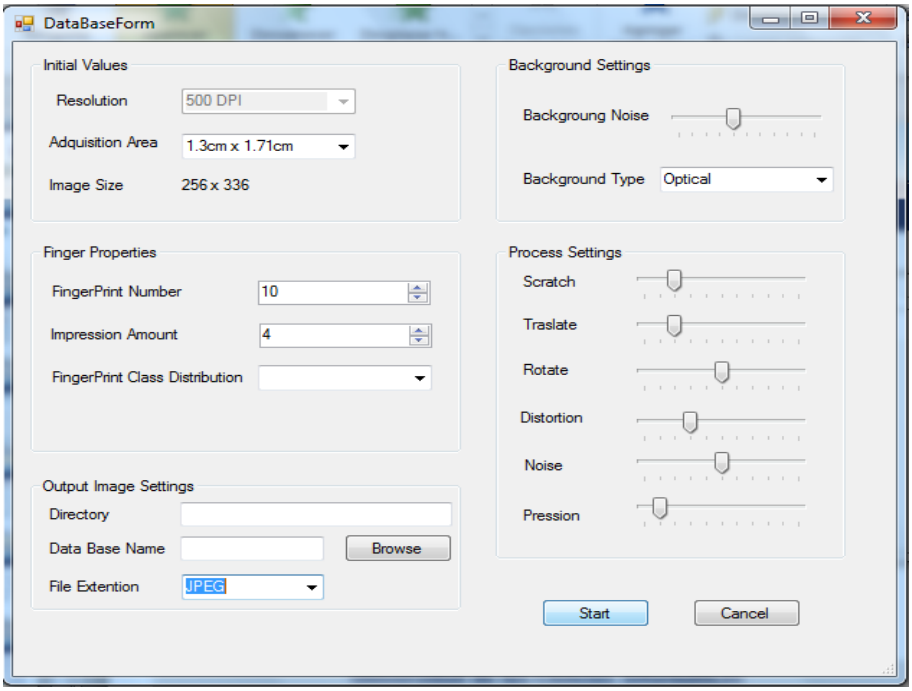

Figura 37. Interfaz DataBaseForm

En esta interfaz se genera la base de datos, se escoge el área de adquisición, el número de huellas y la cantidad de muestras por dedo, se le aplican diferentes transformaciones a las huellas y seguidamente se guarda donde desee el usuario.

3.4 Sintetizador de huellas dactilares vs SFINGE

Para el desarrollo de este sistema se tomó como base la aplicación SFINGE, después de un estudio profundo sobre el funcionamiento de la misma y tomándola como referencia, se decide pasar a la implementación del nuevo Sintetizador de Huellas Dactilares, realizando diversas transformaciones, las cuales se explican a continuación:

- Para seleccionar la forma del área de adquisición, se modela mediante la unión de cuatro beziers y el movimiento de 4 parámetros.

- En la orientación de la generación de imágenes, para formar las clases de huellas dactilar se crea un rectángulo dentro de la huella dactilar, en el cual se conforma la generación de los puntos según la clase en específico, pero siempre dentro de la huella y con una separación de al menos 30 pixeles el uno del otro.

- Teniendo en cuanta las cicatrices que tiene una huella dactilar real, se le aplican roturas a la huella generadas de manera aleatoria.

- Cuando se trata de rotar el área de contacto de la huella, debido que esta no siempre se encuentra en el centro del lector, la rotación se le realiza a la parte de la huella dactilar.

- A la imagen se le aplican operadores morfológicos para simular diferentes grado de humedad, teniendo en cuenta cuando la piel esta seca o húmeda, estas no se fijan con la misma apariencia sobre el lector, estos operadores como la presión y erosión se simula con la aplicación de una matriz de 4x4, que aplica a los pixeles un color u otro de acuerdo al patrón morfológico que se escoja.

- Debido a las deformaciones de la piel o la elasticidad que pueda presentar, se le aplican una mayor cantidad roturas a la huella generadas aleatoriamente.

- La presión no uniforme de los dedos contra el sensor producen un deterioro durante la adquisición de las huellas dactilares, denominándose ruido, debido a este el sistema genera pequeños puntos blancos en toda la huella, aplicándose un suavizado que no es más que la utilización de una matriz de 3x3 o 5x5 a toda la imagen.

- Al culminar el proceso de generación de la huella, se le aplica nuevamente rotación y traslación pero esta vez solo a la huella dactilar.

3.5 Pruebas

La prueba de software es un elemento crítico para la garantía de la calidad del software, constituyen aquellas actividades mediante la cual un sistema es ejecutado bajo determinadas condiciones específicas, enfocada principalmente a la evaluación y determinación de la calidad del producto.

Glem Myers establece varias normas que pueden servir adecuadamente como objetivos de la prueba:

- La prueba es un proceso de ejecución de un programa con la intención de descubrir un error.
- Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
- Una prueba tiene éxito si descubre un error no detectado hasta entonces. (37)

3.5.1 Pruebas de caja blanca

Las pruebas de caja blanca requieren del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones internas de diseño o el código. Se examinan los caminos lógicos del software exponiendo que los casos de prueba se realizan juntos, definidos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado o mencionado.

Mediante los métodos de prueba de la caja blanca, el ingeniero de software puede obtener casos de prueba donde garanticen que:

- ✓ Se ejerciten por lo menos una vez todos los caminos independientes para cada módulo.
- ✓ Se ejerciten todas las decisiones lógicas en sus vertientes verdaderas y falsa.
- ✓ Ejecuten todos los bucles en sus límites y con sus límites operacionales.
- ✓ Se ejerciten las estructuras internas de datos para asegurar su validez.

La prueba del camino básico: Esta prueba permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución.

Los pasos que se siguen para aplicar esta técnica son:

1. A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
2. Se calcula la complejidad ciclomática del grafo.
3. Se determina un conjunto básico de caminos independientes.
4. Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

Fórmulas para calcular complejidad ciclomática.

1. V (G) = (A (Aristas) – N (Nodos)) + 2.
2. V (G) = P (Nodos Predicados) + 1.
3. V (G) = R.

- **Funcionalidad PressionFinger**

```
public Bitmap PressionFinger(int value, GraphicsPath fingerArea)
   {
     Bitmap temp = BitmapHelper.GetBitmap(ImageArray);
     double[,] result = ImageArray; //1
     if (value < 3) //2
       ImageArray = Pression.FingerPrintDyness(temp); //3
     if (value > 3) //4
       ImageArray = Pression.FingerPrintPression(temp); //5
     image = BitmapHelper.GetBitmap(ImageArray); //6

     TraslateRotate.FingerRegion(image, reg); //7
     return image; //8
   }
```

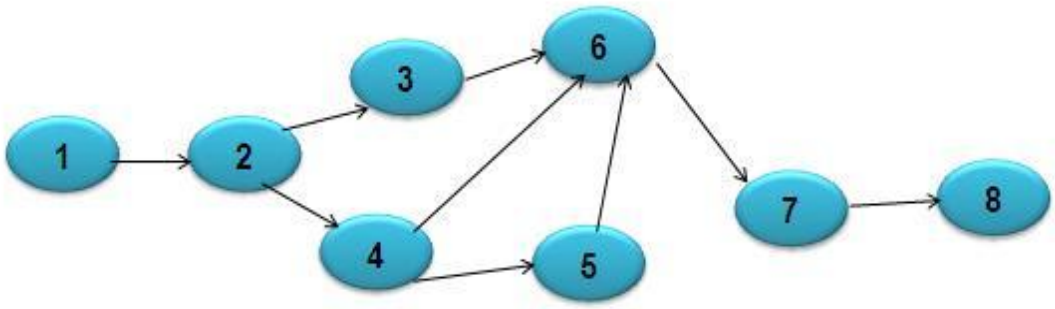

Figura 38. Grafo de flujo: funcionalidad PressionFinger

Complejidad ciclomática para la funcionalidad PressionFinger

- Fórmula1

  V (G) = (A (Aristas) – N (Nodos)) + 2

  V (G) = (9 – 8) + 2 = 3

- Fórmula 2

  V (G) = P (Nodos Predicados) + 1

  V (G) = 2 + 1 = 3

- Fórmula 3

  V (G) = R = 3

Caminos independientes obtenidos y casos de pruebas que obliga a la ejecución de cada camino.

- 1-2-3-6-7-8
  - ✓ Que el valor de la variable value sea menor que 3.
- 1-2-4-6-7-8
  - ✓ Que el valor de la variable value sea igual a 3.
- 1-2-4-5-6-7-8
  - ✓ Que el valor de la variable value sea mayor que 3.

- **Funcionalidad Orientada**

```
public double[,] Orientada(List<FingerPoint> ptos)
   {
     int filas = orientacion.GetLength(0) / bloquesize;
     int columnas = orientacion.GetLength(1) / bloquesize; //1
     for (int i = 0; i < filas; i++) //2
     {
       for (int j = 0; j < columnas; j++) //3
       {
         double angulo = MathHelper.Angulo(ptos, new Point(j * bloquesize, i * bloquesize)); //4
         for (int a = 0; a < bloquesize; a++) //5
         {
           for (int b = 0; b < bloquesize; b++) //6
           {
             orientacion[i * bloquesize + b, j * bloquesize + a] = angulo; //7
           }
         }
       }
     }
     return orientacion; //8
   }
```

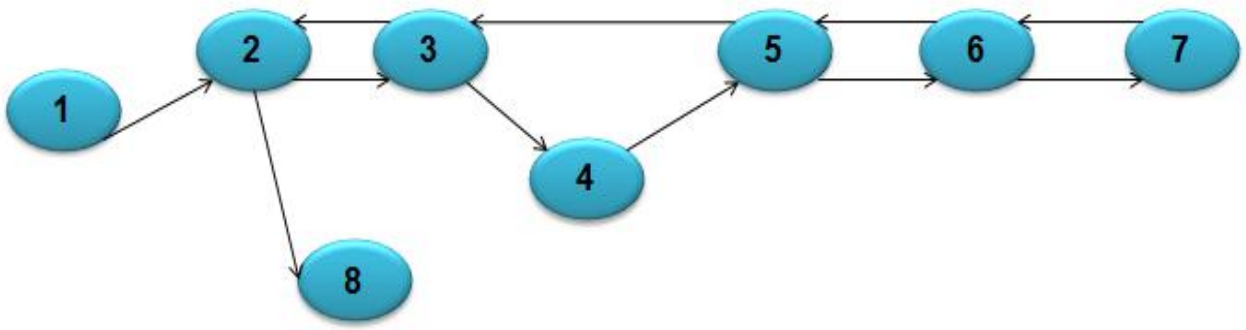

Figura 39. Grafo de flujo: funcionalidad Orientada

Complejidad ciclomática para la funcionalidad Orientada

- Fórmula1

  V (G) = (A (Aristas) – N (Nodos)) + 2

  V (G) = (11 – 8) + 2 = 5

- Fórmula 2

  V (G) = P (Nodos Predicados) + 1

V (G) = 4 + 1 = 5

- Fórmula 3

V (G) = R = 5

Caminos independientes obtenidos y casos de pruebas que obliga a la ejecución de cada camino.

- 1-2-3-4-5-6-7-6-5-3-2-8
  - ✓ Que el valor de la variable i sea menor que la cantidad de filas.
  - ✓ Que el valor de la variable j sea menor que la cantidad de columnas.
  - ✓ Que el valor de la variable a sea menor que el de la variable bloquesize.
  - ✓ Que el valor de la variable b sea menor que el de la variable bloquesize.
- 1-2-3-4-5-6-5-3-2-8
  - ✓ Que el valor de la variable i sea menor que la cantidad de filas.
  - ✓ Que el valor de la variable j sea menor que la cantidad de columnas.
  - ✓ Que el valor de la variable a sea menor que el de la variable bloquesize.
  - ✓ Que el valor de la variable b sea mayor que el de la variable bloquesize.
- 1-2-3-4-5-3-2-8
  - ✓ Que el valor de la variable i sea menor que la cantidad de filas.
  - ✓ Que el valor de la variable j sea menor que la cantidad de columnas.
  - ✓ Que el valor de la variable a sea mayor que el de la variable bloquesize.
- 1-2-3-2-8
  - ✓ Que el valor de la variable i sea menor que la cantidad de filas.
  - ✓ Que el valor de la variable j sea mayor que la cantidad de columnas.

- 1-2-8

  - ✓ Que el valor de la variable i sea mayor que la cantidad de filas.

Funcionalidad Smooth

```
public Bitmap Smooth(int cant)
   {
     Graphics g;
     Bitmap bmp = BitmapHelper.GetBitmap(imageArray);
     Bitmap final = (Bitmap)bmp.Clone();
     Bitmap resul=(Bitmap)bmp.Clone();
     final.MakeTransparent(Color.White); //1
     if (cant == 0) //2
     {
       bmp = Smooting3(bmp);
       resul = Mezclar(bmp, final); //3
     }
     else
       if (cant == 1) //4
       {
         bmp = Smooting5(bmp); resul = Mezclar(bmp, final); //5
       }
     imageArray = BitmapHelper.BitmapArray(resul); //6
     return resul; //7
   }
```

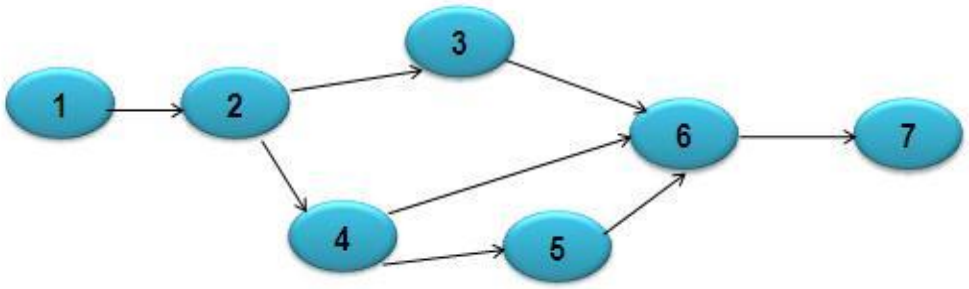

Figura 40. Grafo de flujo: funcionalidad Smooth

Complejidad ciclomática para la funcionalidad Smooth

- Fórmula1

  V (G) = (A (Aristas) – N (Nodos)) + 2

  V (G) = (8 – 7) + 2 = 3

- Fórmula 2

  V (G) = P (Nodos Predicados) + 1

  V (G) = 2 + 1 = 3

- Fórmula 3

  V (G) = R = 3

Caminos independientes obtenidos y casos de pruebas que obliga a la ejecución de cada camino.

- 1-2-3-6-7
  - ✓ Que el valor de la variable cant sea igual a 0.
- 1-2-4-5-6-7
  - ✓ Que el valor de la variable cant sea igual a 1.
- 1-2-4-6-7
  - ✓ Que el valor de la variable cant sea diferente de 1 y de 0.

Para cada funcionalidad en particular se realizó el cálculo de la complejidad ciclomática mediante las fórmulas descritas, por lo que ha dado el mismo valor, lo que significa que existen esos posibles caminos por donde el flujo puede circular, como se demuestra anteriormente. Éste valor obtenido representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado.

3.6 Conclusiones

En el presente capítulo se elaboró finalmente la implementación del sistema propuesto, se describió la estructura y organización del sistema mediante la representación un diagrama de componentes, se visualizaron las interfaces de usuario, dando una explicación de cada una de ella para un mayor entendimiento, se expusieron las diferencias ante la aplicación SFINGE. Finalmente se realizaron las pruebas de caja blanca, validando el funcionamiento y la calidad del sistema.

Conclusiones generales

A lo largo del desarrollo de la presente investigación, se realizó un estudio sobre los sistemas automatizados que generan huellas dactilares existentes en el mundo y en nuestro país, obteniendo un conocimiento profundo de las tendencias actuales de dichos sistemas, demostrando la necesidad de desarrollar una aplicación para la generación de bases de datos de huellas dactilares artificiales junto a su vector característico, de esta manera contar con suficientes muestras que permitan conformar Base de Datos de pruebas necesarias para el desarrollo de algoritmos biométricos de reconocimiento de personas.

Se realizó el modelo de dominio y la identificación de los requerimientos del sistema a implementar quedando plasmadas en un modelo de casos de uso del sistema, siendo este, el punto de partida para comenzar el flujo de trabajo Análisis y Diseño. Como resultado, se obtuvieron los artefactos necesarios para dar inicio al siguiente flujo de trabajo propuesto por la metodología de desarrollo de software RUP, Implementación y Prueba, logrando cumplir el objetivo del trabajo propuesto con la implementación de una aplicación para la generación de bases de datos de huellas dactilares y su vector característico.

Con la obtención de este producto se logra automatizar el proceso de obtención de base de datos, disponiendo así de una aplicación que sea competitiva a nivel internacional, dando la posibilidad de contar con un software para el beneficio de las instituciones nacionales que precisen de este tipo de herramientas, lo que ahorraría cuantiosos gastos al país, además este producto por su calidad también se podría comercializar permitiendo la entrada de divisas.

Recomendaciones

- Continuar la investigación para lograr la optimización de la generación de la imagen.
- Implementar la distorsión no lineal y el fondo de la imagen.
- Optimizar los algoritmos para la aplicación de la presión y erosión a la huella.

Bibliografía citada

1. Ruiz-del-Solar, Javier y Morales L., Domingo. SISTEMAS BIOMÉTRICOS: MATCHING DE HUELLAS DACTILARES. [En línea] [Citado el: 10 de Diciembre de 2010.] http://www2.ing.puc.cl/~iing/ed429/sistemas_biometricos.htm.

2. Machado, Antonio y Díaz, J. Martín. Identificación de huellas dactilares. [En línea] 22 de Junio de 2003. [Citado el: 3 de Diciembre de 20010.] http://centros5.pntic.mec.es/ies.victoria.kent/Rincon-C/Curiosid/Rc-57/Rc-57a.htm.

3. SDC. *ASDA.* ASD : s.n.

4. Davide Maltoni, Dario Maio, Anil K. Jain, Salil Prabhakar. *Handbook of Fingerprint Recognition.*

5. Capítulo 2. IAGP 2005/06. Metodologías de desarrollo de software. [En línea] [Citado el: 10 de Enero de 2011.] http://www.um.es/docencia/barzana/IAGP/Iagp2.html.

6. Sánchez, Jesús Pérez. [En línea] [Citado el: 17 de 2 de 2011.] http://www.willydev.net/InsiteCreation/v1.0/descargas/prev/metodologiasagiles.pdf. 11.

7. Amaro Calderón, Sarah Dámaris Valverde Rebaza, Jorge Carlos. Sociedad de Estudiantes de Ciencia de la Computación. [En línea] 2007. [Citado el: 9 de 2 de 2011.] http://www.seccperu.org/files/Metodologias%20Agiles.pdf. 10.

8. Entorno Virtual de Aprendizaje. [En línea] [Citado el: 5 de Febrero de 2011.] http://eva.uci.cu/file.php/102/Curso_2010-2011/Clases/Semana_02/Seminario_1/Materiales_complementarios/04.Metodologias_de_desarrollo_de_software.pdf.

9. [En línea] [Citado el: 3 de 2 de 2011.] http://eva.uci.cu/file.php/102/Curso_2010-2011/Clases/Semana_02/Seminario_1/Materiales_complementarios/04.Metodologias_de_desarrollo_de_software.pdf. 12.

10. Ivar Jacobson, Grady Booch, James Rumbaugh. *El Proceso Unificado de Desarrollo de Software.* págs. 1-12.

11. Lenguaje de modelado de objetos. [En línea] [Citado el: 12 de Diciembre de 2010.] http://es.wikipedia.org/wiki/Lenguaje_de_modelado_de_objetos.

12. Lenguaje Unificado de Modelado. [En línea] [Citado el: 10 de Diciembre de 2010.] http://es.wikipedia.org/wiki/Lenguaje_Unificado_de_Modelado.

13. Herramienta CASE. [En línea] [Citado el: 5 de Diciembre de 2010.] http://www.scribd.com/doc/3062020/Capitulo-I-HERRAMIENTAS-CASE.

14. Rational Rose Enterprise Edition. [En línea] [Citado el: 5 de Diciembre de 2010.] http://www.ecured.cu/index.php/Rational_Rose_Enterprise_Edition.

15. Baeza, Pablo Nicolás Baeza. *Visual Paradigm DB Visual ARCHITECT SQL.*

16. Lenguajes de programación. *Lenguajes de programación.* [En línea] [Citado el: 2 de Diciembre de 2010.] http://www.lenguajes-de-programacion.com/lenguajes-de-programacion.shtml.

17. Programacion en C++. [En línea] [Citado el: 25 de Noviembre de 2010.] http://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C%2B%2B/Introducci%C3%B3n.

18. C Sharp. [En línea] [Citado el: 2 de Diciembre de 2010.] http://es.wikipedia.org/wiki/C_Sharp.

19. Java (lenguaje de programación). [En línea] [Citado el: 2 de Diciembre de 2010.] http://es.wikipedia.org/wiki/Java_%28lenguaje_de_programaci%C3%B3n%29.

20. Entornos integrados de desarrollo. [En línea] [Citado el: 5 de Enero de 2011.] http://www.atenas.cult.cu/rl/informatica/manuales/sl/introduccion_al_SL/sec-ide.html.

21. Sitio web de la E.U. de Ingeniería Técnica Informática de Oviedo. [En línea] [Citado el: 10 de Enero de 2011.] http://petra.euitio.uniovi.es/~i1667065/HD/documentos/Entornos%20de%20Desarrollo%20Integrado.pdf.

22. NetBeans. [En línea] [Citado el: 27 de Enero de 2011.] http://es.wikipedia.org/wiki/NetBeans.

23. Microsoft Visual Studio. [En línea] [Citado el: 10 de Enero de 2011.] http://es.wikipedia.org/wiki/Microsoft_Visual_Studio.

24. Tecnología y Synergix: Visión de Synergix de los Sistemas de Información y la Ingeniería del Software. [En línea] [Citado el: 15 de Marzo de 2011.] http://synergix.wordpress.com/2008/07/07/requisito-funcional-y-no-funcional/.

25. Ivar Jacobson, Grady Booch, James Rumbaugh. *El Proceso Unificado de Desarrollo de Software.* pág. 125.

26. —. *El Proceso Unificado de Desarrollo de Software.* pág. 165.

27. Entorno Virtual de Aprendizaje. [En línea] [Citado el: 3 de Abril de 2011.] http://eva.uci.cu/file.php/102/Curso_2010-2011/Clases/Semana_10/Conferencia_10/Materiales_complementarios/Introduccion_a_la_Disciplina_Analisis_y_Diseno.pdf.

28. SCRIBD. Diagrama de colaboración. [En línea] [Citado el: 3 de Marzo de 2011.] http://www.scribd.com/doc/11802367/diagramas-de-colaboracion.

29. Ivar Jacobson, Grady Booch, James Rumbaugh. *El Proceso Unificado de Desarrollo de Software.* pág. 205.

30. Gracia, Joaquin. Análisis y Diseño. [En línea] 27 de Mayo de 2005. [Citado el: 2011 de Marzo de 15.] http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php.

31. Alvarez, Sara. Desarrolloweb.com. [En línea] 30 de agosto de 2007. [Citado el: 25 de Marzo de 2011.] http://www.desarrolloweb.com/articulos/arquitectura-cliente-servidor.html.

32. *La Arquitectura Orientada a Servicios (SOA) de Microsoft aplicada al mundo real.* 2006.

33. Arquitectura de software. [En línea] [Citado el: 12 de Marzo de 2011.] http://es.wikipedia.org/wiki/Arquitectura_de_software.

34. Valencia, Maria Eugenia. Diagramas de clases del diseño. [En línea] [Citado el: 10 de Marzo de 2011.] http://eisc.univalle.edu.co/materias/Material_Desarrollo_Software/DISCLASES_A12.pdf.

35. Ivar Jacobson, Grady Booch, James Rumbaugh. *El Proceso Unificado de Desarrollo de Software.*

36. Departamento de Sistemas Informáticos. *Modelo de Implementación: Diagramas de Componentes y Despliegue.* [En línea] [Citado el: 5 de Abril de 2011.] http://www.dsi.uclm.es/asignaturas/42530/pdf/M2tema12.pdf.

37. Entorno Virtual de Aprendizaje. Sobre la Disciplina de Prueba. [En línea] [Citado el: 10 de Mayo de 2011.] http://eva.uci.cu/file.php/259/Curso_2010-2011/Semana_9/Conferencia_7/Materiales_Basicos/Sobre_la_disciplina_de_Prueba.pdf.

Bibliografía consultada

1. Cárdenas, Germaín. Sistema de Reconocimiento de Huellas Dactilares.
2. Clasificación de los Sistemas Biométricos.UNAM - Facultad de Ingeniería Biometría Informática. Disponible en URL: http://redyseguridad.fi-p.unam.mx/proyectos/biometria/clasificacionsistemas/recohuella.html.
3. RaffaeleCappelli**.** Biometric System Laboratory. DEIS-University of Bologna. Disponible en URL:http://biolab.csr.unibo.it/research.asp?organize=Activities&select=&selObj=12&pathSubj=111||12&.
4. DavideMaltoni, DarioMaio, Anil K. Jain, SalilPrabhakar. Handbook of Fingerprint Recognition
5. YoandroHechevarríaToranzo, A. F. D. (2006). Sistema Automatizado para la Planificación Material y Financiera del MINFAR. Facultad de Ingeniería Industrial Centro de Estudios de Ingeniería de Sistemas (CEIS). Ciudad de la Habana., Instituto Superior Politécnico "José Antonio Echeverría". 99.
6. Marcelo Claudio Perissé. HERRAMIENTAS CASE. Disponible en URL: http://cyta.com.ar/biblioteca/bddoc/bdlibros/proyectoinformatico/libro.
7. Lenguajes de programación. Disponible en URL: http://www.lenguajes-de-programacion.com/lenguajes-de-programacion.shtml.
8. Mitchell, W. D. (2001). Java sin errores. Madrid: McGraw-Hill: ISBN: 84-481-3107-1.
9. Jose Carlos Carvajal Riola. METODOLOGÍAS ÁGILES: HERRAMIENTAS Y MODELO DE DESARROLLO PARA APLICACIONESJAVA EE COMO METODOLOGÍA EMPRESARIAL. Disponible es URL http://upcommons.upc.edu/pfc/bitstream/2099.1/5608/1/50015.pdf.
10. Metodologías de desarrollo de software. http://eva.uci.cu/file.php/102/Curso_2010-2011/Clases/Semana_02/Seminario_1/Materiales_complementarios/04.Metodologias_de_desarrollo_de_software.pdf. 12.
11. Rumbaugh, J.; Jacobson, I. y Booch, G.;"El Lenguaje Unificado de Modelado. Manual de referencia .2000. Addison-Wesley.

12. Introducción a la Disciplina de Análisis y Diseño. http://eva.uci.cu/file.php/102/Curso_2010-2011/Clases/Semana_10/Conferencia_10/Materiales_complementarios/Introduccion_a_la_Disciplina_Analisis_y_Diseno.pdf.

13. Diseño. http://eva.uci.cu/file.php/259/Curso_2010-2011/Semana_1/Conferencia_1/Materiales_basicos/Diseno.pdf.

14. PATRONES GRASP. http://sophia.javeriana.edu.co/~lcdiaz/ADOO2006-3/grasp_cpaternostro-lvargas-jviafara.pdf.

15. Patrones de arquitectura. http://es.wikipedia.org/wiki/Patrones_de_arquitectura.

16. PATRONES GRASP. Disponible en URL: http://sophia.javeriana.edu.co/~lcdiaz/ADOO2006-3/grasp_cpaternostro-lvargas-jviafara.pdf.

17. Buschmann, F. Meunier, R. Rohnert, H. Sommerlad, P. Stal. "M. Pattern-Oriented Software Architecture. A System of Patterns. ". 1996.

18. Pipe-And-Filter. http://www.dossier-andreas.net/software_architecture/pipe_and_filter.html.

19. Fase de Construcción.http://eva.uci.cu/file.php/259/Curso_2010-2011/Semana_8/Conferencia_6/Materiales_Basicos/Implementacion.pdf.

20. Diagramas de componente. http://www.info-ab.uclm.es/asignaturas/42530/pdf/M2tema12.pdf.

Glosario de términos

SFINGE: Generación Sintética de Huellas Dactilares.

RUP: El Proceso Unificado de Rational (RUP, en inglés Rational Unified Process) es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

TLV: (Tag-Lenght-Value) donde se especifica que detrás de una etiqueta se encuentra la longitud del valor y posteriormente el valor del mismo.

UML: (Unified Modeling Language) Lenguaje Unificado de Modelado. Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad, es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software.

SGBD: Sistema de Gestor de Base de Datos.

GRASP: acrónimo de General Responsibility Assignment Software Patterns. Patrones de asignación de responsabilidades.

Anexos

Anexo 1: Descripción de las clases

Tabla 8. Descripción de la clase: ProcessControl

Nombre: ProcessControl	
Tipo de Clase: Controladora	
Atributo	**Tipo**
-image	Bitmap
-copy	Bitmap
-imagesize	Size
-imagereal	Size
-pentwith	int
-ran	Random
-path	DrawPath
-imageArray	double[*][*]
-tamanno	int
-reg	Region
-bloque	int
Responsabilidad	
Nombre	**Descripción**
+ProcessControl(Size imagereal, Size imagesize)	Constructor con parámetros.
+ImageArray():double[*][*]	Retorna el valor de imageArray.
+Scratch(Bitmap img, int count): Bitmap	Aplica roturas a la huella.
+TraslateImage(Bitmap bmp, GraphicsPath fingerArea, int horizontal, int vertical): Bitmap	Rotada el área de adquisición de la imagen.
+TraslateImage(GraphicsPath fingerArea, int horizontal, int vertical, int angle): Bitmap	Traslada la imagen de la huella.
+RotateTraslateImageFinal(GraphicsPathfingerarea, intoffsetx, intoffsety, int angle): Bitmap	Traslada y rota toda la huella dactilar.
+PressionFinger(int value, GraphicsPathfingerArea): Bitmap	Aplica de presión o erosión a la huella dactilar.
+FirstNoise(int count, int counts,	Introduce roturas y ruido a la imagen.

GraphicsPathfingerArea): Bitmap	
+Mezclar(Bitmap original, Bitmap mezcla): Bitmap	Aplica una imagen transparente encima de la imagen desenfocada.
+Smooth(intcant): Bitmap	Aplica el desenfocado a la imagen y la mezcla con la imagen transparente.
+Smooting3(Bitmap bmp): Bitmap	Aplica el desenfocado de 3x3.
+Smooting5(Bitmap bmp): Bitmap	Aplica el desenfocado de 5x5.
+BackGroundBitmap(Size imagereal): Bitmap	Genera el fondo de la imagen.

Tabla 9. Descripción de la clase: Process

Nombre: Process	
Tipo de Clase: Interfaz	
Atributos	**Tipo**
-real	Size
-tempimage	double[*][*]
-dr -imagePath	GraphicsPath DrawPath
-usFingerForm	FingerFormation
-usDraw	CustomPictureBox
-usField	FieldOrientation
-row	int
-col	int
-bitmapsize	Size
-pointsize	int
-bloquesize	int
-singularities	SingularitieGeneration
-ptos	List<FingerPoint>
-usScratch	FingerScratch
-usAnimation	FingerAnimation
-usContactRegion	FingerContactRegion
-usPression	FingerPression
-usdistortion	FingerDistorsion

-usNoise	FingerNoise
-usRotation	FingerRotation
-usBackground	FingerBackground
Responsabilidades	
Métodos	**Descripción**
Process(Size real)	Constructor con parámetros.
InitializeUi():void	Inicializa el área de adquisición y se suscribe al evento que dibuja la misma.
DrawFieldOrientation(): void	Prepara las condiciones para la orientación de la huella.
Finger(): void	Generación de la Imagen.
Scratch():void	Mostrar la imagen con las roturas aplicadas.
Traslate():void	Mostrar la imagen con las traslaciones aplicadas.
PressionDyness(int cant): void	Mostrar la imagen con la presión/erosión aplicada.
TrasRotate(): void	Mostrar la imagen con la traslación y la rotación aplicada.
DistortionCenter():void	Prepara la imagen para aplicarle la distorsión.
FingerNoise():void	Muestra la imagen con el ruido y las roturas aplicadas.
FingerTraslateImage():void	Muestra la imagen de la huella dactilar trasladada y rotada.
Background():void	Muestra la imagen recortada y con el fondo generado aplicado.
KeepOriginal(Bitmap image): void	Mantener una copia de la transformación cada vez que termine un paso.
ChooseControl(int control): void	Controla el flujo hacia adelante de la aplicación.
UsFingerFormFingerFormationData(object sender, EventArgs e): void	Crea la forma del dedo perteneciente a la huella dactilar.
Seeds():void	Genera las semillas para los Gabor.
StopGeneration():void	Detener el hilo que se utiliza para la generación de la imagen.
StartGeneration();void	Para Iniciar el Hilo.