FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO “EURÍPIDES DE MARÍLIA” – UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

CELSO RICARDO SOARES MOREIRA

CRIAÇÃO DE BANCO DE DADOS PARA TESTE DE SISTEMA DE RECONHECIMENTO DE IMPRESSÃO DIGITAL

MARÍLIA
2006

CELSO RICARDO SOARES MOREIRA

CRIAÇÃO DE BANCO DE DADOS PARA TESTE DE SISTEMA DE RECONHECIMENTO DE IMPRESSÃO DIGITAL

Monografia apresentada ao Curso de Ciência da Computação, da Fundação de Ensino "Eurípides Soares da Rocha", Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador:
Profª. Drª. Fátima de Lourdes dos Santos Nunes Marques

MARÍLIA
2006

CELSO RICARDO SOARES MOREIRA

CRIAÇÃO DE BANCO DE DADOS PARA TESTE DE SISTEMA DE RECONHECIMENTO DE IMPRESSÃO DIGITAL

Banca examinadora da monografia apresentada ao Programa de Bacharel do UNIVEM, para obtenção do título de bacharel em Ciência da Computação.

Resultado: _____

ORIENTADOR: _____
Profª. Drª. Fátima de Lourdes dos Santos Nunes Marques

1º EXAMINADOR: _____
Prof. Dr. Márcio Eduardo Delamaro

2º EXAMINADOR: _____
Prof. Dr. Edmundo Sérgio Spoto

Marília, _____de _____de 2006

Dedico este trabalho à memória de minha mãe e ao meu pai, por tudo o que eles fizeram por mim nesta existência... serei eternamente grato.

Obrigado.

AGRADECIMENTOS

Agradeço a todas as pessoas que direta ou indiretamente me auxiliaram na execução deste trabalho.

A todos os professores do Curso de Ciência da Computação.

À minha esposa Veridiana e a minha filha Isabela, pela paciência.

E a Deus, por minha vida.

“Não são as perdas e nem as caídas que podem fazer fracassar a nossa vida, e sim a falta de coragem para levantarmos e seguirmos adiante”.

Samael Aun Weor

“Homem! Conheça-te a ti mesmo e conhecerás o universo e os Deuses”.

Sócrates

“Quem quiser vir após mim, negue a si mesmo, pegue a sua cruz e siga-me”.

Jesus Cristo

MOREIRA, Celso Ricardo Soares. Criação de banco de dados para teste de sistema de reconhecimento de impressão digital. 2006-103f. Monografia (Bacharelado em Ciência da Computação) – Curso de Ciência da Computação, Fundação de Ensino "Eurípides Soares da Rocha", Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM.

RESUMO

Este trabalho tem por objetivo criar um banco de dados de impressões digitais a partir de estudo e programas confeccionados para aquisição e armazenamento de dados. Para isso, foi feito um estudo sobre biometria, aquisição, verificação e classificação de impressões digitais e software de aquisição de impressão digital. Foram feitas as classes para armazenamento das impressões digitais no banco de dados e no final é possível realizar a busca de impressões digitais mais semelhante no banco de dados, gerando resultados satisfatórios.

Palavras-chave: Banco de Dados; Impressões Digitais.

MOREIRA, Celso Ricardo Soares. Criação de banco de dados para teste de sistema de reconhecimento de impressão digital. 2006-103f. Monografia (Bacharelado em Ciência da Computação) – Curso de Ciência da Computação, Fundação de Ensino "Eurípides Soares da Rocha", Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM.

ABSTRACT

This work has for objective to create a database of fingerprints from study and programs implemented for acquisition and data storage. For this, it was made a study on biometria, acquisition, verification and classification of fingerprints and software of acquisition of fingerprint. The class for storage of the fingerprints in the data base had been made and in the end the search of fingerprints more similar in the data base can be carried through, generating resulted satisfactory.

Keywords: Data base; Fingerprints.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS

AFIS - Automated Fingerprint Identification System

DB1 - Banco de Dados 1

DB2 - Banco de Dados 2

DB3 - Banco de Dados 3

DB4 - Banco de Dados 4

FBI - Federal Bureau of Investigation

GIF - Graphics Interchange Format

IBM - International Business Machines

NIST - National Institute of Standards and Technology

TIFF - Tagged Image File Format

SUMÁRIO

INTRODUÇÃO

A biometria está ganhando cada vez mais espaço no dia-a-dia das pessoas. O que antigamente parecia apenas uma ficção científica das telas dos cinemas, hoje é uma realidade na identificação de pessoas.

A biometria é o estudo das características físicas dos seres humanos. Por meio dessas características é possível diferenciar um indivíduo do outro. Essa identificação pode ser feita por diversas partes do corpo humano, como formato do rosto, impressão digital, íris, palma da mão ou com características pessoais, como voz e assinatura.

Com a globalização e o avanço tecnológico a biometria cresceu devido à necessidade cada vez maior de identificar as pessoas.

Essa identificação vai desde controle de acesso a ambientes restritos, controle de freqüência no trabalho, substituição de senhas etc., até a captura de terroristas por identificação facial.

Nos dias de hoje já há carros e casas que se abrem com um simples toque de dedo.

Objetivos

O objetivo desse projeto é criar de um banco de dados e uma forma de recuperação de impressões digitais a partir de estudo e programas confeccionados para aquisição e armazenamento de dados com a finalidade de permitir testar um software para reconhecimento de impressões digitais, previamente desenvolvido.

Justificativa

Os programas de aquisição de impressão digital têm um custo elevado, com a criação desse banco de dados de impressão digital para um software pronto ter-se-á um programa de aquisição de impressão digital com banco de dados a um custo mínimo.

Além desta introdução, este trabalho apresenta os seguintes capítulos:

- Capítulo 1 trata da revisão bibliográfica, onde é descrito o que é biometria, como fazer o reconhecimento de uma impressão digital, métodos de classificação e comparação de uma impressão digital, pontos de identificação, processamento de imagem, apresentação de trabalhos correlatos e banco de dados para teste de sistema de reconhecimento.
- Capítulo 2 aborda a metodologia de desenvolvimento do projeto e os resultados obtidos.
- Ao final, apresenta-se as Conclusões, apêndices e anexos relevantes para o trabalho.

CAPÍTULO 1 - REVISÃO BIBLIOGRÁFICA

A biometria é o estudo das características físicas dos seres humanos. É por meio dessas características físicas que se pode diferenciar um indivíduo do outro. Essa identificação pode ser feita utilizando-se o reconhecimento de diversas partes do corpo humano: formato do rosto, impressão digital, íris, palma da mão, e também por alguns comportamentos como, por exemplo, a voz e assinatura (Figura 1) (LIN *et al.*, 1982).

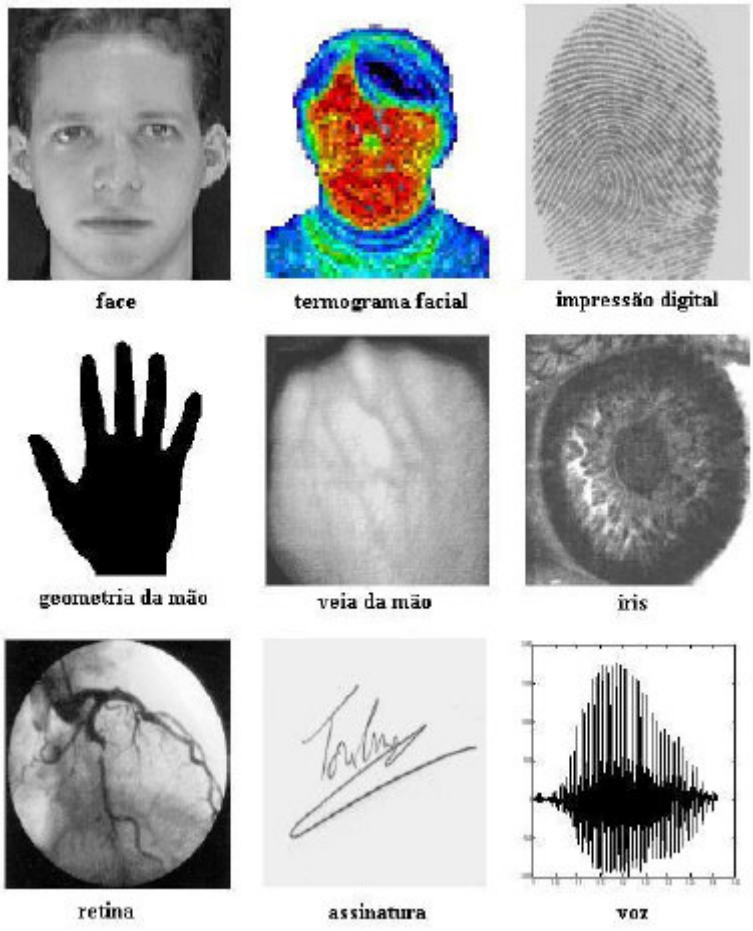

Figura 1 - Diferentes Características Biométricas (Perenha, 2003)

O reconhecimento de impressão digital é um dos meios mais usados para identificar um indivíduo, dentre os citados na Figura 1. Cada pessoa tem uma impressão digital única, que difere das demais. O reconhecimento de impressão digital tem sido amplamente utilizado em diversos ramos, como: segurança, educação, saúde, etc.

O uso de dispositivos de aquisição de impressões digitais estão se tornando cada vez mais comuns nos dias de hoje. As aquisições por esses dispositivos são chamadas de sistemas automáticos e têm por finalidade o reconhecimento de indivíduos utilizando como meio de autenticação sua característica física: a impressão digital.

O seu principal objetivo é com relação à segurança. Existem várias formas de utilização:

1) no local de trabalho e nas escolas, para liberação das catracas, servindo também como folha de presença e cartão de ponto;
2) nas auto-escolas e em concursos, para evitar fraudes;
3) nas consultas médicas, para identificação e também para evitar fraudes;
4) para substituir as senhas de acesso em softwares, na internet e em transações bancárias usando microcomputadores;
5) controle de acesso a locais restritos

O sistema de verificação de impressão digital vem há muito tempo auxiliando na elucidação de crimes (FBI, 1984).

Antigamente as análises de impressão digital eram feitas manualmente por especialistas, conforme pode ser observado na Figura 2. Com os Sistemas Automáticos de Identificação de Impressão Digital (*Automated Fingerprint Identification System* - AFIS) tudo ficou mais fácil, porém, para que isso aconteça a impressão digital deve ser tratada, passando por várias etapas (JAIN *et al.*, 1997).

Figura 2- Análise manual de impressão digital por um perito (Costa, 2001)

1.1 Reconhecimento de impressão digital

1.1.1 Conceitos básicos sobre processamento de imagens

Nesta seção são abordados alguns conceitos básicos sobre processamento de imagens para que se possa compreender melhor como as impressões digitais são adquiridas, classificadas e comparadas.

Ballard *et al*. (1982) afirmam que a formação da imagem ocorre quando a radiação que interagiu com objetos físicos é registrada por um sensor. Então, a imagem é uma representação do objeto físico que pode ser armazenada, manipulada e interpretada de acordo com as necessidades do interessado.

De acordo com Gonzalez *et al*. (2005), o termo imagem refere-se à função bidimensional de intensidade da luz $f(x,y)$ onde x e y denotam as coordenadas espaciais e o valor de f em qualquer ponto (x,y) é proporcional ao brilho (ou níveis de cinza) da imagem naquele ponto. Na Figura 3 é ilustrada a convenção dos eixos.

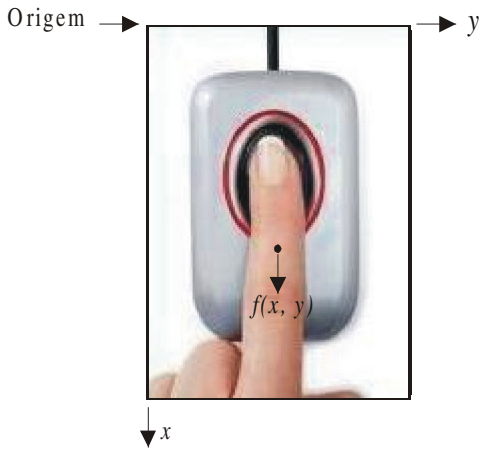

Figura 3 - Convenção dos eixos para representação de imagens digitais (adaptado de Gonzalez *et al*., 2005)

Uma imagem digital é uma imagem $f(x,y)$ discretizada, afirmam Gonzalez *et al*. (2005), tanto em coordenadas espaciais quanto em brilho. Uma imagem digital pode ser considerada como sendo uma matriz cujos índices de linhas e de colunas identificam um

ponto no espaço, e o correspondente valor do elemento da matriz identifica a cor naquele ponto. Os elementos dessa matriz digital são chamados de elementos da imagem, elementos da figura, "*pixels*" ou "*pels*", estes dois últimos, abreviações de "*picture elemenst"* (elementos da figura).

Embora o tamanho de uma imagem digital varie de acordo com a aplicação, é vantajoso selecionar matrizes retangulares com tamanhos e números de níveis de cinza que sejam potências inteiras de 2. Por exemplo, um tamanho típico comparável em qualidade de imagem com aquela de uma TV preto e branco seria uma matriz de 512 x 512 pixels, com 128 níveis de cinza (Gonzalez *et al*., 2005).

Como a luz é uma forma de energia, $f\ (x,\ y)$ deve ser positiva e finita, conforme indica a Equação 1.

$$0 < f\ (x,\ y) < \infty \tag{1}$$

Gonzalez *et al*. (2005) relatam que as imagens que as pessoas percebem em atividades visuais corriqueiras consistem de luz refletida dos objetos. A natureza básica de $f(x,y)$ pode ser caracterizada por dois componentes: (1) a quantidade de luz incidindo na cena sendo observada e (2) a quantidade de luz refletida pelos objetos na cena. Apropriadamente, esses componentes são chamados de iluminação e reflectância, respectivamente, e são representados por $i(x,y)$ e $r(x,y)$. O produto das funções $i(x,y)$ e $r(x,y)$ resulta $f(x,\ y)$, conforme mostra a Equação 2:

$$f(x,\ y) = i(x,y)\ .\ r(x,y) \tag{2}$$

onde $0 < i(x,y) < \infty$

e $0 < r(x,y) < 1)$

A Equação 2 indica que a reflectância é limitada entre 0 (absorção total) e 1 (reflexão total). A natureza de $i(x,y)$ é determinada pela fonte de luz e $r(x,y)$ é determinada pelas características dos objetos na cena.

Para ser adequada para processamento computacional, uma função *f(x, y)* precisa ser digitalizada tanto espacialmente quanto em amplitude. A digitalização das coordenadas espaciais (x,y) é denominada amostragem da imagem e a digitalização da amplitude é chamada quantização.

Considerando que uma imagem contínua *f(x, y)* é aproximada por amostras igualmente espaçadas, arranjadas na forma de uma matriz com *N* linhas e *M* colunas, tem-se que a imagem pode ser representada como indica a Equação 3, em que cada elemento é uma quantidade discreta:

$$f(x,y) \approx \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0, M-1) \\ f(1,0) & f(1,1) & \dots & f(1, M-1) \\ \vdots & & & \\ f(N-1,0) & f(N-1,1) & \dots & f(N-1, M-1) \end{bmatrix} \quad (3)$$

Ballard *et al*. (1982) e Gonzalez *et al*. (1987) relatam que o motivo de definir a imagem matematicamente é a possibilidade de transformar ou retirar dela informações importantes, permitindo manipular o seu conteúdo. Denomina-se *processamento de imagem* ao grande conjunto de operações que pode se aplicar em uma matriz que representa uma imagem.

A seguir são apresentados alguns relacionamentos básicos entre *pixels* (Gonzalez *et al*., 2005).

a) Vizinhança

Um *pixel p* nas coordenadas (*x,y*) possui quatro vizinhos horizontais e verticais, cujas coordenadas são dadas por (x + 1, y), (x - 1, y), (x, y + 1), (x, y -1).

Esse conjunto de *pixels*, chamado vizinhança-de-4 de *p*, é representado por $N_4(p)$. Cada *pixel* está a uma unidade de distância de *(x, y)*, sendo que alguns dos vizinhos de *p* ficarão fora da imagem digital se *(x, y)* estiver na borda da imagem.

Os quatros vizinhos diagonais de *p* possuem como coordenadas (x + 1,y + 1), (x + 1,y - 1), (x - 1,y + 1), (x - 1,y - 1) e são denotados por $N_D(p)$. Esses pontos, junto com a vizinhança-de-4, são chamados vizinhança-de-8 de *p*, representada por $N_8(p)$.

b) Adjacência

Gonzalez *et al*. (1987) afirmam que adjacência é a característica de um par de *pixels* vizinhos que compartilham uma borda ou um vértice, sendo que:

a) "adjacente por borda" ou "4-adjacente" é um par de *pixels* de uma imagem que compartilham uma borda;

b) "adjacente por vértice" ou "8-adjacente" é um par de *pixels* de uma imagem que compartilham um vértice.

c) Conectividade

A conectividade entre *pixels*, dizem Gonzalez *et al*. (2005), é um conceito importante usado no estabelecimento das bordas de objetos e componentes de regiões em uma imagem. Para estabelecer se dois *pixels* estão conectados, é preciso determinar se eles são de alguma forma adjacentes e se seus níveis de cinza satisfazem um certo critério de similaridade.

Seja *V* o conjunto dos valores de níveis de cinza usados para definir conectividade; por exemplo, em uma imagem binária, $V = \{1\}$ para conectividade de *pixels* com Valor 1. Em uma imagem em níveis de cinza, para conectividade de *pixels* com uma escala de valores de intensidade, digamos entre 32 e 64, segue que $V = \{32,33,..., 63,64\}$. Consideremos três tipos de conectividade:

(a) Conectividade-de-4. Dois *pixels* p e q assumindo valores em V são conectados se q está no conjunto $N_4(p)$;

(b) Conectividade-de-8. Dois *pixels* p e q assumindo valores em V são conectados se q está no conjunto $N_8(p)$;

(c) Conectividade-de-m (conectividade mista). Dois *pixels* p e q assumindo valores em V são conectados se:

i) q está em $N_4(p)$ ou

ii) q está em $N_D(p)$ e o conjunto $N_4(p) \cap N_4(q)$ for vazio

A conectividade mista é uma modificação da conectividade-de-8 introduzida para eliminar as conexões por múltiplos caminhos, que freqüentemente aparecem quando a conectividade-de-8 é usada.

d) Distância entre *pixels*

Para os *pixels* p, q e z, com coordenadas (x,y), (s,t) e (u,v) respectivamente, D é uma função distância ou métrica se:

(a) $D(p, q) \geq 0$, ($D(p, q) = 0$ se e somente se $p = q$);

(b) $D(p, q) = D(q, p)$;

(c) $D(p, z) \leq D(p, q) + D(q, z)$.

A distância Euclidiana entre p e q é definida conforme mostra a Equação 4

$$D_e(p, q) = [(x - s)^2 + (y - t)^2]^{1/2} \quad (4)$$

Para essa medida de distância, os *pixels* tendo uma distância de (x, y) menor ou igual a algum valor r são os pontos contidos em um disco de raio r centrado em (x, y).

A distância D_4 (também chamada distância ”city block”, “quarteirão”) entre p e q é definida como pode-se ver na Equação 5

$$D_4(p, q) = |x - s| + |y - t| \quad (5)$$

Nesse caso os *pixels*, tendo uma distância D_4 de (x, y) menor ou igual a algum valor r, formam um losango centrado em (x, y). Por exemplo, se *pixels* com distância $D_4 \leq 2$ de (x, y) (o ponto central) formam os seguintes contornos de distância constante:

$$\begin{array}{ccccc} & & 2 & & \\ & 2 & 1 & 2 & \\ 2 & 1 & 0 & 1 & 2 \\ & 2 & 1 & 2 & \\ & & 2 & & \end{array}$$

Os *pixels* com $D_4 = 1$ são os vizinhos-de-4 de (x, y).

A distância D_8 (também chamada distância xadrez) entre p e q é definida como vê-se na Equação 6

$$D_8(p, q) = \max(|x - s|, |y - t|) \tag{6}$$

Nesse caso, os *pixels* com distância D_8 de (x, y) menor ou igual a algum valor r, formam um quadrado centrado em (x, y). Por exemplo, se *pixels* com distância $D_8 \leq 2$ de (x, y) (o ponto central) formam os seguintes contornos de distância constante:

$$\begin{array}{ccccc} 2 & 2 & 2 & 2 & 2 \\ 2 & 1 & 1 & 1 & 2 \\ 2 & 1 & 0 & 1 & 2 \\ 2 & 1 & 1 & 1 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{array}$$

Os *pixels* com $D_8 = 1$ são os vizinhos-de-8 de (x, y).

Os conceitos anteriores podem ser aplicados no processamento de imagens, que é dividido em três níveis, cada qual com suas funções específicas, (Gonzalez *et al*., 1987):

- Processamento de baixo nível: responsável pela remoção de dados indesejáveis e realce de dados importantes;
- Processamento em nível médio: é a parte do processamento que identifica formas significantes. A esse processo damos o nome de “segmentação”.
- Processamento em alto nível: é responsável pela ligação da imagem com algum banco de conhecimento.

1.1.2 Métodos de Classificação de impressões digitais

A técnica de classificação está baseada no princípio da similaridade de padrões. Um objeto será identificado como "x" se suas características coincidirem, o mais próximo possível das características do modelo de "x", armazenado em um banco de dados.

A dificuldade está em determinar esse conjunto de características que devem ser analisadas para chegar ao resultado esperado, isto é, que partes ou pontos da imagem devem ser estudadas para chegar à conclusão de que as duas impressões digitais em questão são da mesma pessoa (JAIN *et al.*, 1999).

1.1.3 Métodos desenvolvidos

Vários métodos foram desenvolvidos durante décadas, com o propósito de classificação de impressões digitais:

- Método Estrutural: faz uma análise das configurações globais dos padrões das impressões digitais. São analisados os núcleos e os deltas e as impressões digitais são classificadas de acordo com o sistema de Henry (HRECHAK *et al.*, 1990);
- Estatístico: com o emprego de técnicas de estatísticas é comum o uso de vetores de características para desenvolver visões geométricas de padrões. Características estatísticas também são usadas para classificação de impressões digitais, onde são calculadas como atributos das cristas e pontos singulares (núcleo e delta) (RAO, 1976);
- Sintático: é usado um conjunto de padrões que representa e classifica padrões de impressões digitais e baseia-se na descoberta da continuidade e paralelismo das cristas, tipos de linhas, pequenos graus de variação local, núcleos e deltas (CHONG *et al.*,1997);
- Matemático: um modelo matemático é desenvolvido para calcular a orientação da crista local, núcleos e deltas;

- Redes Neurais Artificiais: é um sistema de auto-aprendizado que utilizada as direções das cristas e outros aspectos das impressões digitais como entrada de treinamento para diferenciar as diversas classes (WILSON *et al.*, 1994);
- Híbridos: utilização de mais de um método em conjunto para fazer a classificação das impressões digitais.

1.1.4 Principais pontos de comparação em uma impressão digital

- Pontos singulares (núcleos e deltas)

Núcleo e delta são conhecidos como pontos singulares em impressões digitais, e são usados para classificar os padrões de impressões digitais (Figura 4). O núcleo está localizado no centro da impressão digital. O Delta é um ângulo (triângulo) formado pelas cristas: formando uma bifurcação de uma linha simples ou pela brusca divergência de duas linhas paralelas.

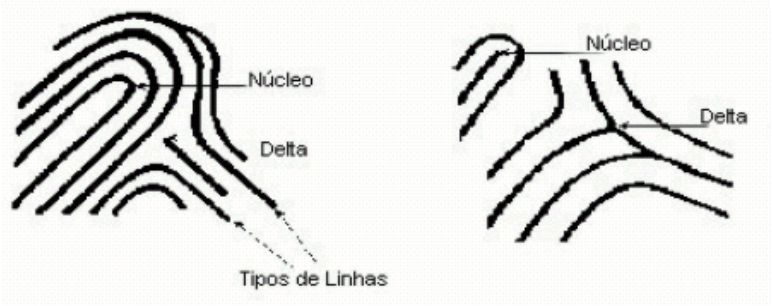

Figura 4 - Ponto delta e ponto núcleo de uma impressão digital (Costa, 2001)

- Sistema de Henry

O sistema de Henry (1905) classifica as impressões digitais em cinco classes, com características próprias, o qual verifica-se na Figura 5: **Arco Plano**: impressão digital sem delta e com linhas que atravessam de um lado ao outro. **Arco Angular**: apresentam elevada acentuação das linhas do centro, com um delta logo abaixo. **Presilha interna** (direita): tem um núcleo e um delta à direita do observador. **Presilha externa** (esquerda): tem

um núcleo e um delta à esquerda do observador. **Verticilo**: a impressão digital apresenta um núcleo e dois deltas, um à direita e um à esquerda.

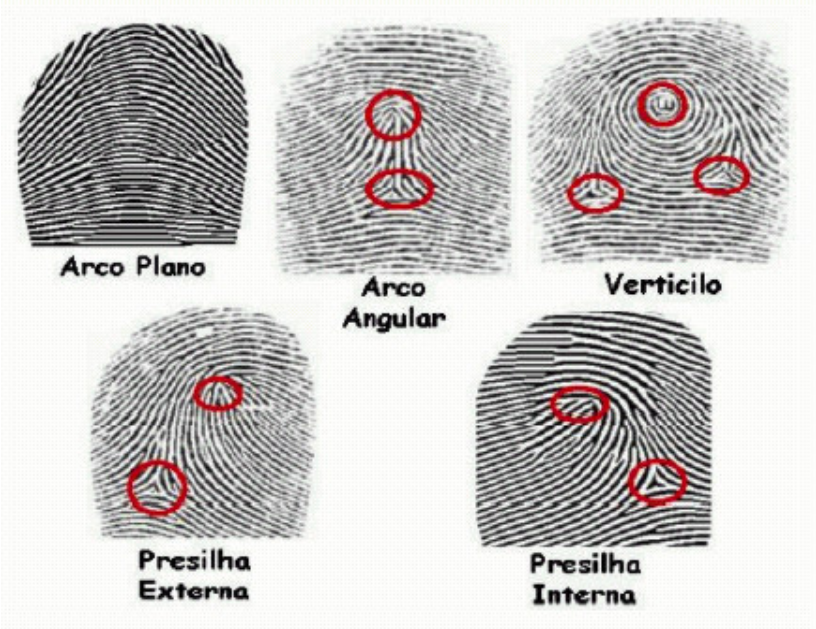

Figura 5 - As cinco classes propostas por Henry: deltas e núcleos destacados em vermelho (Costa, 2001)

Segundo o FBI (*Federal Bureau of Investigation*) foi apurado, na população mundial, que 65% dos padrões de impressões digitais são presilhas, 30% são verticilos e 5% são arcos (MOAYER *et al.*, 1975). Por isso que os banco de dados não são distribuídos uniformemente nas 5 (cinco) classes.

- Pontos característicos ou minúcias (Detalhes de Galton)

As minúcias (Figura 6) são acidentes que se encontram nas cristas (linhas da impressão digital), elas podem ser linhas que terminam repentinamente ou se bifurcam, dessa forma pode-se estabelecer a unicidade das impressões digitais, conforme vê-se na Figura 7.

A crista final (*ridge ending*) é definida como linhas que terminam abruptamente. A crista bifurcada (*bifurcation*) são linhas que formam uma bifurcação. Ilha (*islands*) ou lago são duas bifurcações conectadas. Crista curta (*short ridge*) são linhas muito pequenas. Esporas (*spur*) é formada por uma crista bifurcada e uma crista final. Cruzamento (*crossover*)

é duas bifurcações conectadas em paralelo, em ângulos opostos, com um caminha conectado (HRECHAK *et al.*, 1990).

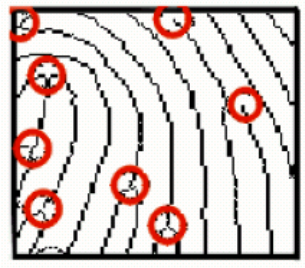

Figura 6 - Pontos característicos ou minúcias – cristas finais e cristas bifurcadas (Costa, 2001)

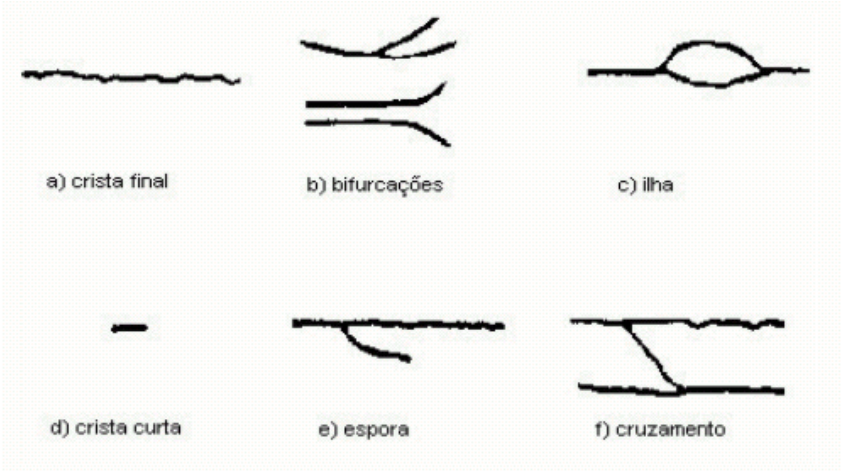

Figura 7 - Aspectos de impressões digitais (detalhes de Galton) (Costa, 2001)

1.1.5 Pré-Processamento de Imagem

Para que as imagens de impressão digital, capturadas por um dispositivo de aquisição, fiquem prontas para serem comparadas, elas devem passar por um pré-processamento, através de algumas técnicas a saber:

1) Filtro de Contraste

   A aplicação do filtro de contraste, como pode ser visto na Figura 8, reduz as distorções. Para que isso ocorra o valor do *pixel* é alterado, baseado em vizinhança. Dada uma imagem calcula-se o valor médio de intensidade do *pixel*, numa vizinhança 5x5, por exemplo, se o valor do *pixel* analisado for menor que a

média do bloco ele receberá o valor zero, caso contrário mantêm-se o valor original (HONG *et al.*, 1996).

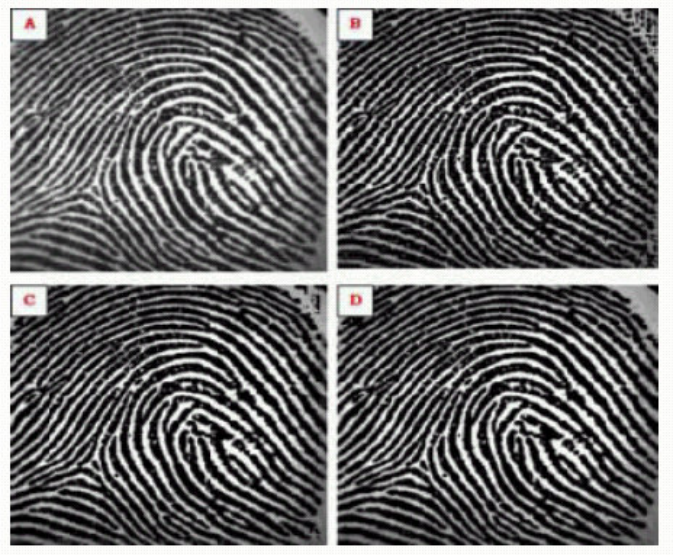

Figura 8 - Filtro de contraste (A) Imagem Original, (B) grade 5x5, (C) grade 10x10, (D) grade 20x20 (Costa, 2001)

2) Binarização ou Limiarização (*Threshold*)

Esta técnica converte imagens em tons de cinza para imagens binárias (preto/branco) (RATHA *et al.*,1995). Verifica-se o valor de intensidade do *pixel* e através da equação de *threshold*, como vê-se na Figura 9, atribui-se o valor 0 (preto) ou 255 (branco) ao *pixel*.

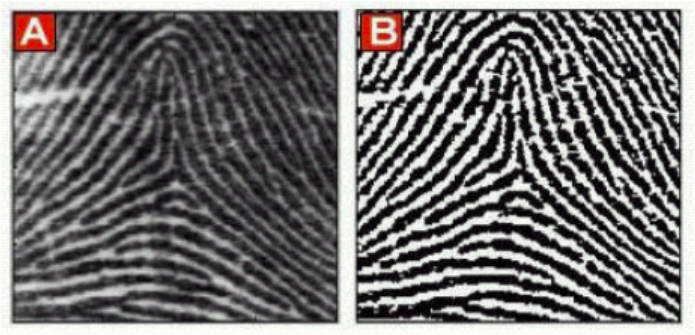

Figura 9 - (a) Imagem Original, (b) Imagem após *threshold* com grade 10 (Costa, 2001)

3) Afinamento (*Thinning*)

É uma técnica usada para remover pontos isolados no fundo da imagem e ângulos retos ao longo de bordas do objeto (HUNG,1993). Utiliza-se um

algoritmo de afinamento (Figura 10) para saber, através da vizinhança, se o *pixel* vai ser apagado ou não, isto é, se recebe o valor 1 ou 0. Este algoritmo é aplicado várias vezes até a imagem ficar bem "enxuta".

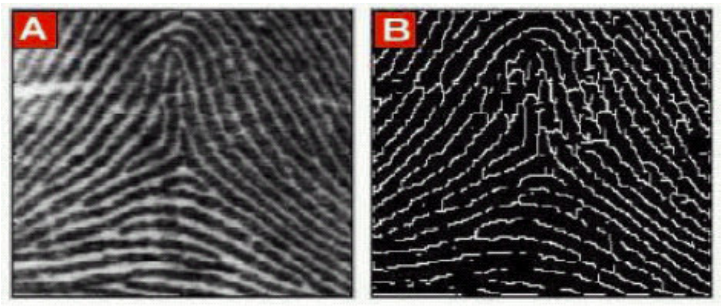

Figura 10 - (a) Imagem Original, (b) Imagem após thinning aplicado 5 vezes (Costa, 2001)

1.1.6 Classificação e Verificação de Impressões Digitais

Existem dois sistemas pelo qual se pode classificar as impressões digitais:

- Bruto: As impressões digitais são distribuídas em 5 classes, como proposto pelo sistema de Henry (HENRY,1905).
- Refinado: neste nível a comparação das impressões digitais é baseada nos aspectos de Galton (Cristas finais e bifurcadas) (HONG *et al.*,1998a). Usam-se os pontos singulares (núcleos e deltas) para evitar problemas de translação e rotação.

As seguintes etapas são executadas para a classificação:

1) Cálculo da Imagem Direcional

As cristas têm normalmente 3 tipos de atributos: a coordenada x, a coordenada y, e a direção(θ). Na Figura 11 vê-se um gráfico dessa representação.

Para calcular a direção de um *pixel* usa-se uma máscara 9x9 (ou 17x17) e com o auxílio de 8 (ou 16) equações pode-se calcular uma das 8 (ou 16) direções possíveis, o resultando disso pode-se ver na Figura 12.

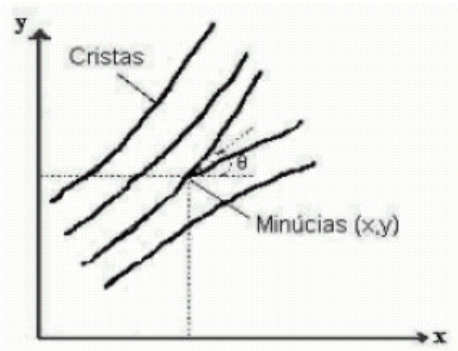

Figura 11 - Componentes das minúcias (Costa, 2001)

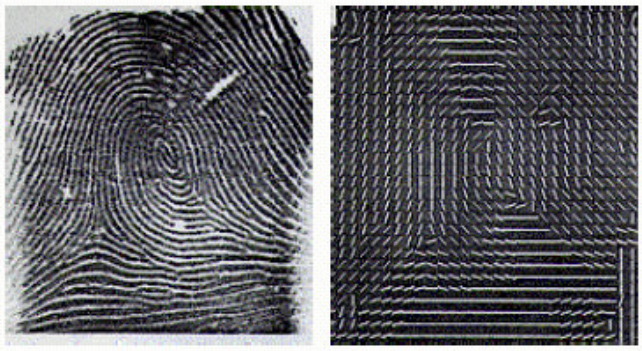

Figura 12 - (a) Imagem Original (b) Imagem Direcional (Costa, 2001)

2) Suavização do Mapa Direcional

Existem vários métodos para calcular a suavização. No caso deste trabalho, dois foram usados:

a) seno-cosseno, o qual se verifica na Figura 13.

b) Moda, representada na Figura 14.

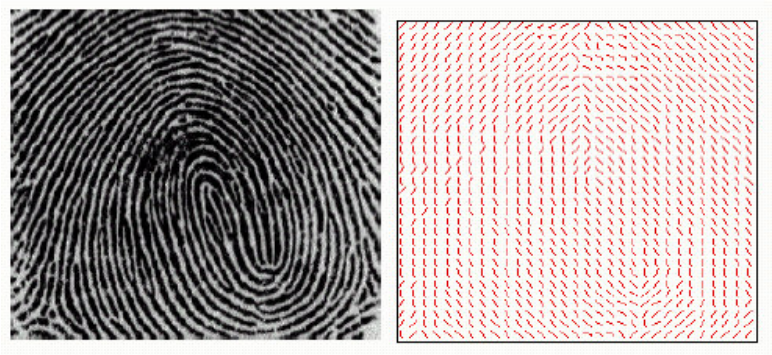

Figura 13 - Suavização do mapa direcional usando seno e cosseno (Costa, 2001)

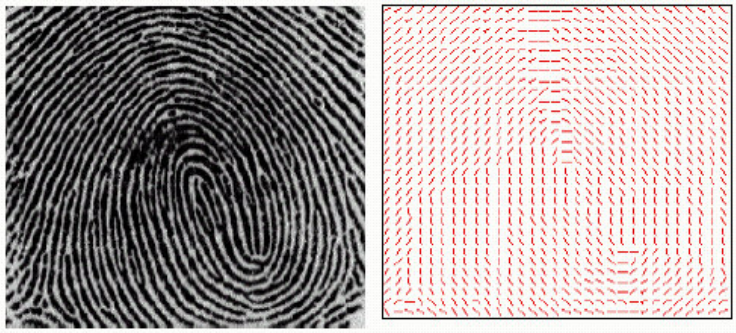

Figura 14 - Suavização do mapa direcional usando a moda (Costa, 2001)

A moda tem sido mais eficiente: divide-se a imagem em blocos e de acordo com os *pixel* vizinhos se estabelece um valor médio de direção. Para suavizar o mapa, basta considerar a direção que aparece com maior freqüência, isto é, a minoria vai ganhar a direção da maioria.

3) Índice de Poincaré

O método de Poincaré, que está representado na Figura 15, é útil para classificar os *pixels* em ordinário, núcleo e delta. Para isso considera-se uma vizinhança 2x2 e calcula-se a somatória dos ângulos. O *pixel* vai ser classificado como ordinário se o ângulo resultante for 0º (grau), núcleo se for fixado em +180º (graus), delta se for fixado em -180º (graus).

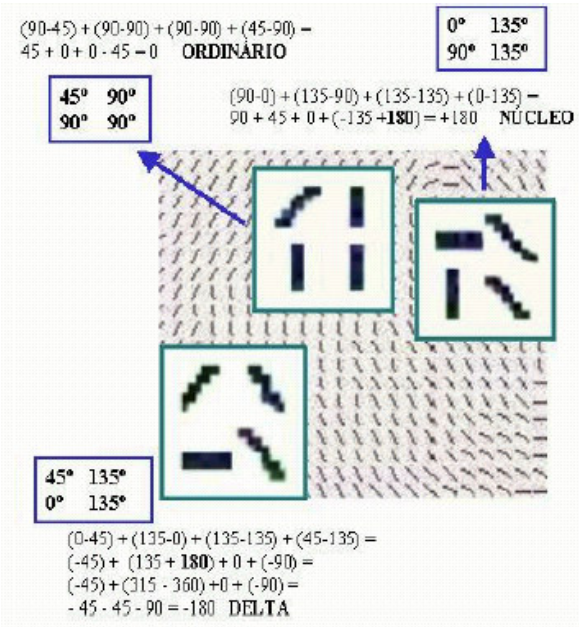

Figura 15 - Cálculo do índice de Poincaré após suavização do mapa direcional (Costa, 2001)

A seguir pode-se verificar o Diagrama de Blocos do Algoritmo de classificação das impressões digitais, que pode ser melhor analisado na Figura 16.

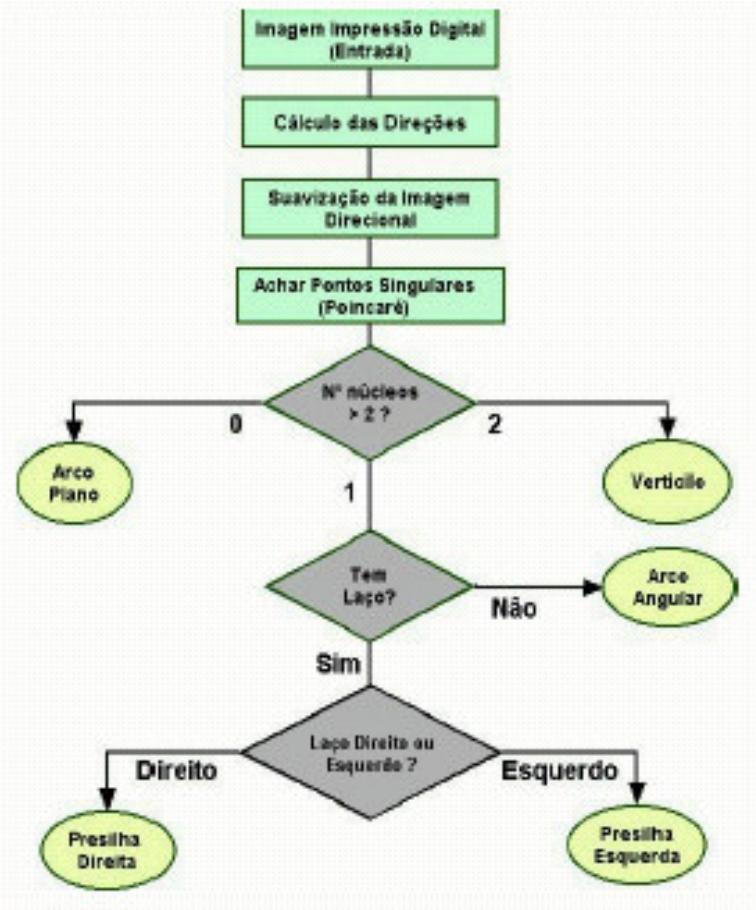

Figura 16 - Diagrama de Blocos do Algoritmo de classificação (Costa, 2001)

4) Etapas de Verificação

Após a extração das minúcias, elas serão comparadas com outras minúcias de impressões digitais armazenadas em um banco de dados. Para a comparação são levados em consideração posição, número, tipo de minúcias e, algumas vezes, a orientação.

A impressão digital de entrada é comparada somente com o subconjunto do banco de dados pelo qual ela foi classificada, para restringir comparações e ganhar tempo em processamento.

5) Aspectos de Extração

O conceito de *crossing number* (RATHA *et al.*,1995; TAMURA, 1978) é muito usado para detecção das minúcias. Ele determina a propriedade de um *pixel* simplesmente

contando o número de transições preto e branco existentes na vizinhança-de-8 (3x3) do *pixel* que está sendo processado.

Pontos finais e bifurcados são detectados usando propriedades de *crossing number*. Se for uma minúcia de ponto isolado, o *pixel* terá propriedade de *crossing number* igual a 0; se for ponto final, igual a 1; se for ponto contínuo, igual a 2; se for ponto bifurcado, igual a 3; se for ponto de cruzamento, igual a 4, como pode-se observar na Quadro 1;

Quadro 1 - Propriedade dos *pixels* (Costa, 2001)

Crossing Number (propriedades)	Minúcias
0	Ponto isolado
1	Ponto final
2	Ponto contínuo
3	Ponto bifurcado
4	Ponto cruzamento

6) Comparação de Minúcias (*Matching*)

Uma impressão digital de entrada é comparada com todas as impressões digitais de sua classe (classificação segundo o sistema de Henry). O algoritmo usado para verificação considera a posição (x,y) e o tipo de minúcia (crista final ou bifurcada). Portanto, duas minúcias são consideradas coincidentes quando localizadas na mesma posição e pertencer ao mesmo tipo (mesmo número de *crossing number*).

1.1.7 Trabalhos Correlatos

1.1.7.1 Verificação de impressão digital on-line (*On-Line Fingerprint Verification*)

Anil (1997) fez um sistema de verificação de impressão digital em 2 estágios, extração e casamento de minúcias, usando o algoritmo de Ratha (Ratha *et al*.,1995). O casamento se dá por uma base de alimentação flexível. Ele compara a impressão digital de

entrada com o modelo armazenado no banco de dados, compensando deformações entre as impressões digitais.

Os testes foram feitos com dois tipos diferentes de scanners de impressão digital e usou dois bancos de dados: o primeiro contém 10 imagens, uma de cada dedo, de 18 indivíduos, totalizando 180 imagens de impressões digitais, no tamanho 380 x 380 *pixels*, capturadas com o scanner da Identix. O segundo contém 10 imagens, uma de cada dedo, de 61 indivíduos, totalizando 610 imagens de impressões digitais, no tamanho 640 x 480 *pixels*, capturadas com o scanner da Digital Biometrics.

Os resultados obtidos foram os seguintes: em um sistema de verificação de impressão digital on-line, um usuário entra com a sua identidade e o sistema fará o casamento das impressões digitais unicamente com os seus modelos armazenados. Portanto, a imagem de impressão digital de entrada foi casada com as outras nove imagens de impressão digital desse mesmo indivíduo, que estava no banco de dados. Se mais da metade das nove impressões digitais casarem com o valor inicial de 25 casamentos ou mais, então esta impressão digital é considerada válida. Numa taxa de verificação de 100% pode-se alcançar uma taxa de rejeição de 16% nos dois conjuntos de teste, e esse valor ainda poderá ser reduzido se for melhorada a qualidade das impressões digitais.

Usando uma estação de trabalho Sparc 20, demorou 8 segundos para a verificação completa. Os experimentos tiveram resultados de alta precisão.

1.1.7.2 Aumento de Imagem de Impressão Digital: Algoritmo e Avaliação de Performance (*Fingerprint Image Enhancement: Algorithm and Performance Evaluation*)

Lin (1998) realizou um trabalho de aquisição de impressão digital de forma automática. A extração de minúcias e o casamento seguro e automático têm se demonstrado crítico, devido à qualidade de impressão digital de entrada e a performance do algoritmo de

extração de minúcia. Para melhorar essa performance deve-se melhorar a qualidade da imagem da impressão digital de entrada. Para isso é essencial a aplicação de um algoritmo de aumento de impressão digital, no módulo de entrada de minúcia, que é o objetivo desse trabalho.

Os testes foram feitos com dois bancos de dados diferentes: o primeiro teste foi feito com 50 imagens de impressões digitais de baixa qualidade, obtidas da IBM. Inicialmente foi analisado o índice de melhoria sem o algoritmo de aumento de impressão digital. Em seguida foi aplicado o algoritmo de aumento de impressão digital de entrada e verificou-se um aumento nos índices de melhoria. Portanto, concluiu-se que o algoritmo de aumento de impressão digital melhora a qualidade das imagens de impressões digitais, melhora a precisão e a confiança da extração da minúcia.

O segundo teste foi analisar a performance do algoritmo de aumento de impressão digital do banco de dados do MSU (700 imagens de 10 indivíduos). Para isso, foi utilizado o sistema de verificação de impressão digital on-line. Inicialmente não foi aplicado o algoritmo de aumento de impressão digital, as imagens de entrada foram diretamente casadas com o banco de dados. Em seguida foi aplicado o algoritmo de aumento de impressão digital para cada imagem de impressão digital de entrada, imagens estas que foram sendo casadas com o banco de dados. O que se notou foi uma melhoria significativa na performance do sistema de verificação de impressão digital on-line.

Para concluir este trabalho foi desenvolvido um algoritmo de aumento de impressão digital que melhora a claridade da crista e das estruturas de vales. A performance do algoritmo foi avaliada usando índice de melhoria da extração da minúcia e a performance do sistema de verificação de impressão digital on-line que inclui o algoritmo de aumento de impressão digital em seu módulo de extração de minúcia. O algoritmo de aumento de impressão digital foi capaz de melhorar ambos os casos, aumentando a qualidade das

impressões digitais de entrada, aumentando a velocidade de verificação e recuperando áreas corrompidas de impressões digitais.

1.1.7.3 Um Sistema de Verificação de Impressão Digital Baseado em Casamento Triangular e Curvatura de Tempo Dinâmico (*A Fingerprint Verification System Based on Triangular Matching and Dynamic Time Warping*)

Kovács (2000) desenvolveu um trabalho com a proposta de fazer a identificação de uma pessoa por meio de impressão digital, adquirida on-line e em tempo real, usando casamento de minúcias. O casamento consiste em duas fases: a primeira é a extração de informações essenciais da imagem de referência, off-line, e a segunda é o casamento propriamente dito, on-line. As informações são obtidas por filtro e procedimentos de extração de minúcias e a identificação é feita por casamento triangular lutando contra a forte deformação de impressão digital, devido à fricção estática e o rolamento de dedo. O casamento é finalmente validado por curvatura de tempo dinâmico.

O banco de dados usado foi o "NIST Special Database 4", que tem 2000 pares de impressões digitais (4000 no total). As imagens são obtidas por scaneamento de impressões digitais tintadas em papel em 500 dpi. As imagens são de 8 bit/*pixel* tamanho 512 x 512. O banco contém imagens de boa qualidade e imagens deformadas, 6% delas não poderiam ser reconhecidas nem por um perito humano. Após o casamento das minúcias, 80% das impressões digitais foram identificadas, 20% como falsa negativa, destes 20%, 30% não foi reconhecido nem por perito e 70% foi identificado. Destes 70% que não tinha sido identificado, 43% tinha taxa de distorção acima do permitido, 13% tinha impressão digital com o dedo rodado e 44% tinha o número de minúcias insuficiente para fazer o casamento.

Essa porção pode ser reduzida se for aceito o valor falsa positiva: 85% das impressões digitais foram identificadas, 15% como falsa negativa, destes 15%, 43% não foi reconhecido nem por perito e 57% foi identificado. Destes 57% que não tinha sido

identificado, 56% tinha taxa de distorção acima do permitido, 42% tinha impressão digital com o dedo rodado e 2% tinha o número de minúcias insuficiente para fazer o casamento.

Os resultados apresentados foram: o casamento de minúcias foi feito usando algoritmo de casamento triangular e verificação final usando curvatura de tempo dinâmico. O casamento triangular é rápido e eficiente enquanto a curvatura de tempo dinâmico supera as deformações, permitindo baixar a taxa de falsos positivos.

1.2 Banco de Dados para teste de sistema de reconhecimento

No trabalho de Anil *et al.* (1997) foram utilizados dois tipos de banco de dados diferentes: no primeiro banco de dados tinha 10 imagens, uma de cada dedo, de 18 indivíduos, totalizando 180 imagens de impressões digitais, no tamanho 380 x 380 *pixels*, capturadas com o scanner da Identix. O segundo banco de dados era composto de 10 imagens, uma de cada dedo, de 61 indivíduos, totalizando 610 imagens de impressões digitais, no tamanho 640 x 480 *pixels*, capturadas com o scanner da Digital Biometrics.

No trabalho feito por Lin *et al.* (1998), também foram utilizados dois tipos de banco de dados diferentes, a saber: no primeiro banco de dados tinha 50 imagens de impressões digitais de baixa qualidade, obtidas da IBM. No segundo banco de dados do MSU tinha 700 imagens de 10 indivíduos.

Kovács (2000), utilizou o banco de dados "NIST Special Database 4", que tem 2000 pares de impressões digitais (4000 no total). As imagens são obtidas por digitalização de impressões digitais tintadas em papel em 500 dpi. As imagens são de 8 bits/*pixel* tamanho 512 x 512 *pixels*. O banco contém imagens de boa qualidade e imagens deformadas, 6% delas não poderiam ser reconhecidas nem por um perito humano.

Bancos de dados para teste de impressão digital podem ser encontrados no site http://bias.csr.unibo.it/fvc2002/, são eles: banco de dados 1 (DB1) contém 80 imagens no

formato TIFF, extraídas pelo sensor ótico "TouchView II" da empresa Identix, com tamanho de 388x374 (142 Kpixels) e resolução de 500 dpi, em tons de cinza. O banco de dados 2 (DB2) contém 80 imagens no formato TIFF extraídas pelo sensor ótico "FX2000" da empresa Biometrika, com tamanho de 296x560 (162 *Kpixels*) e resolução de 569 dpi, em tons de cinza. O banco de dados 3 (DB3) contém 80 imagens no formato TIFF extraídas pelo sensor de capacidade "100 SC" da empresa Precise Biometrics, com tamanho de 300x300 (88 Kpixels), e resolução de 500 dpi, em tons de cinza. O banco de dado 4 (DB4) contém 80 imagens no formato TIFF extraídas pelo Gerador de Impressão Digital Sintético “SfinGe v2.51”, com tamanho de 288x384 (108 *Kpixels*), e resolução de 500 dpi, em tons de cinza, vide Figura 17.

Figura 17 - Impressões digitais dos bancos de dados DB1, DB2, DB3 e DB4 (FVC2000, 2006)

Outros bancos de dados para teste de impressão digital podem ser adquiridos no NIST (*National Institute of Standards and Technology*), onde se obtém banco de dados técnicos e científicos. No site: http://www.itl.nist.gov/iad/894.03/databases/ defs/dbases.html (www.nist.gov), há diversos tipos de banco de dados, diferenciando-os em quantidade, resolução, tipos de impressão digital, etc., ressalta-se que Kovács (2000) utilizou um banco de dados “NIST Special Database 4” deste site.

CAPÍTULO 2 – METODOLOGIA

A partir desse capítulo apresenta-se a metodologia do projeto, desde a instalação dos softwares utilizados até a confecção das classes.

Após a instalação do Firebird 1.5.2.4731 deve-se confirmar se ele está iniciado. Para isso deve-se abrir o Painel de Controle no Menu Iniciar, Configurações (Figura 18) e acionar o ícone Firebird 1.5 Server Manager. Será apresentada uma tela com a seguinte mensagem no seu cabeçalho: "*O Firebird service is running*" (Figura 19). O passo seguinte é criar uma porta para o Firebird 1.5. Ainda no Painel de Controle deve ser selecionado o item Firewall do Windows (Figura 18). Na tela que se abrirá deve ser selecionado a aba Exceções (Figura 20), depois deve ser selecionado adicionar porta e digite no campo nome "Firebird" e no campo número da porta digite "3050" (Figura 21), deixe a opção TCP marcada e deve ser selecionado *OK,* e a porta já estará criada (Figura 22).

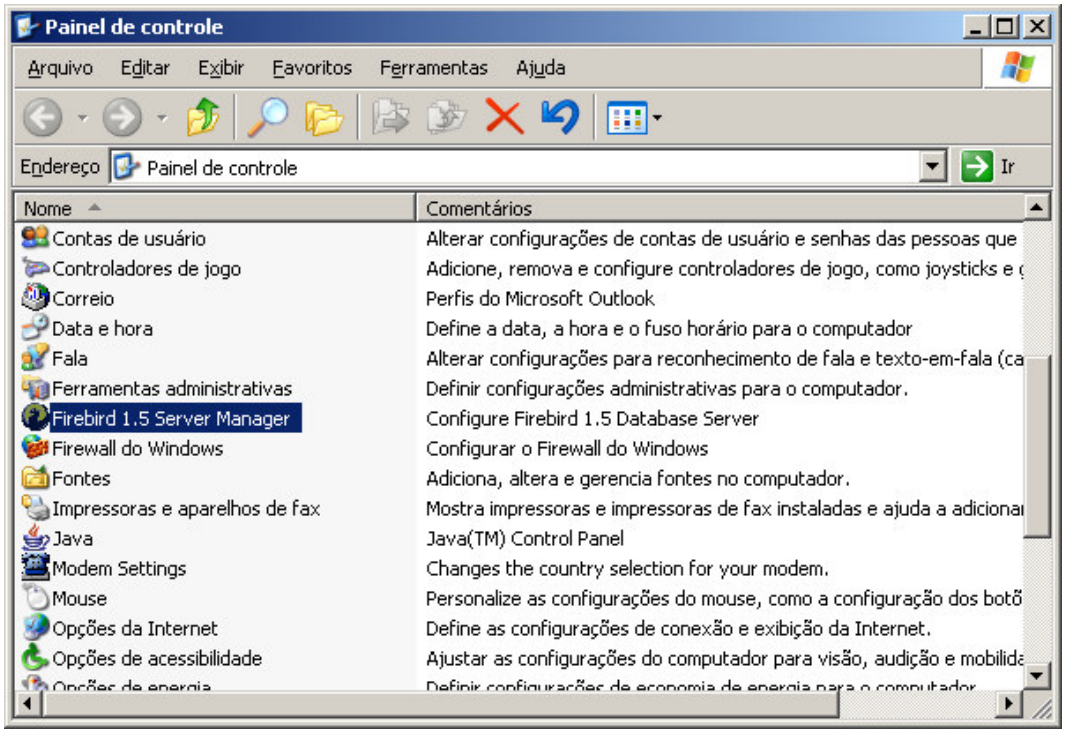

Figura 18 – Painel de Controle – Firebird

Figura 19 – Controle do Servidor Firebird

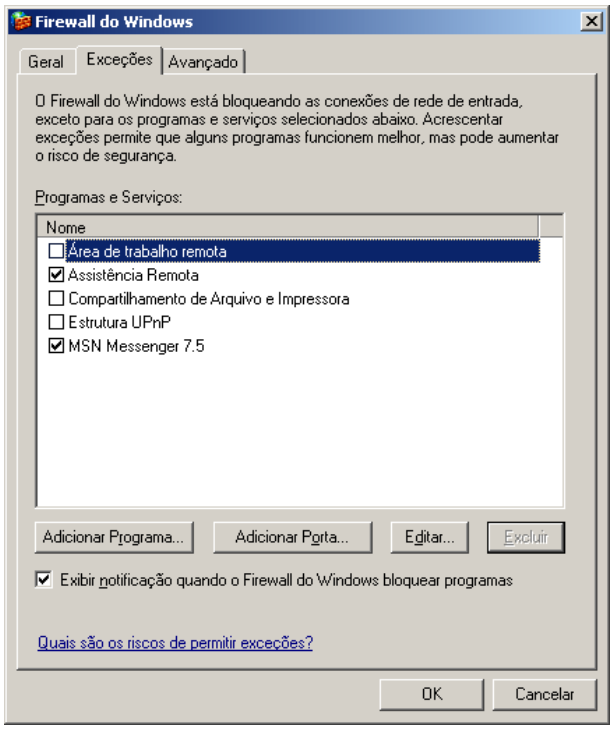

Figura 20 – Firewall do Windows

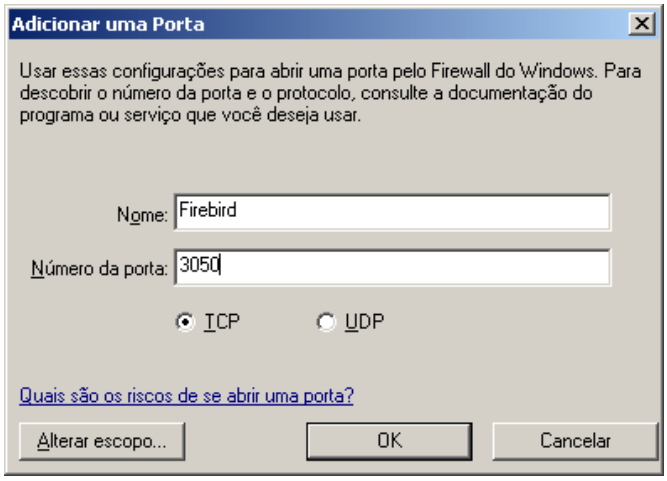

Figura 21 – Adicionar uma Porta

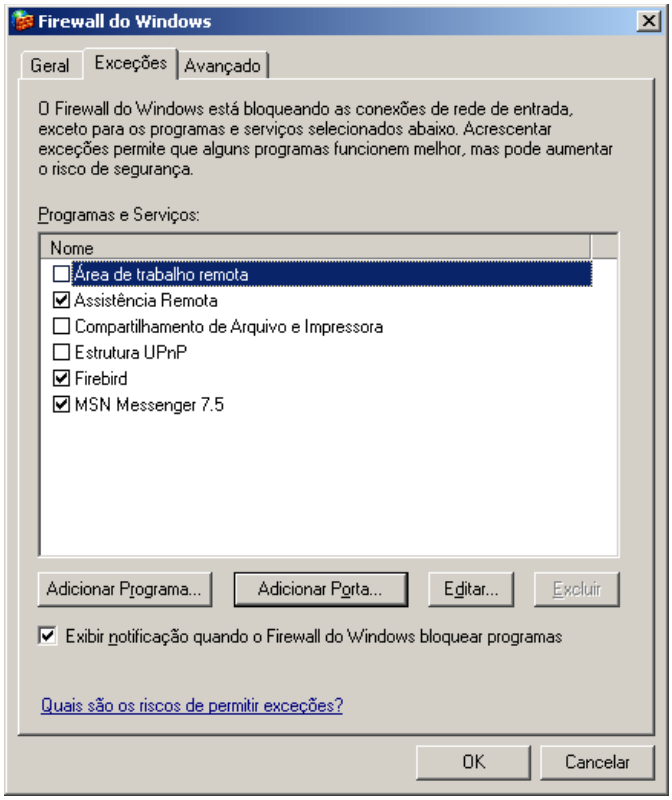

Figura 22 – Firewall do Windows 2

2.1 Criação do Banco de Dados

Para criar um banco de dados no Firebird deve-se executar o programa IBExpert e no menu Database escolher a opção Create Database, em seguida abrirá uma tela, como pode-se ver na Figura 23, a qual deve-se selecionar algumas configurações, as configurações que serão passadas foram as configurações utilizadas nesse trabalho.

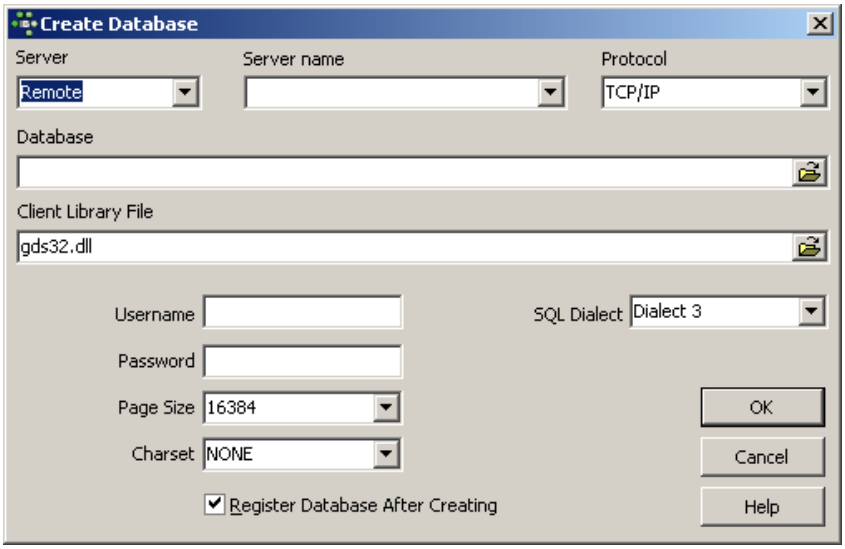

Figura 23 – Criação do Banco de Dados 1 - DB

No campo Server selecionar Local, no campo Database deve ser digitado o nome do banco de dados ou deve ser selecionado a aba da direita para selecionar um diretório e o nome do arquivo. O banco de dados criado terá a extensão “GDB”. No campo Username digitar “SYSDBA” e no campo Password digitar “masterkey” e finalmente no campo Charset selecionar a opção “ASCII”, as demais opções não se alteram, vide Figura 24, deve ser selecionado *OK*.

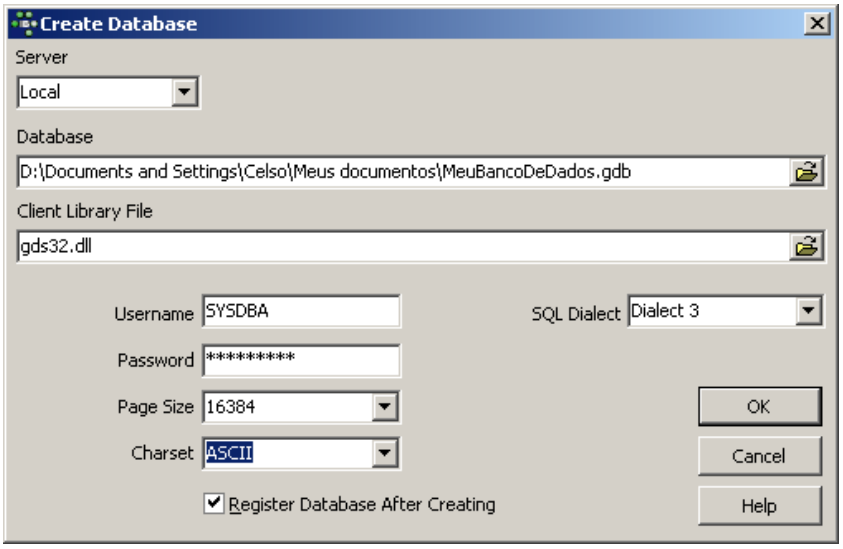

Figura 24 – Criação do Banco de Dados 2 - DB

Feito isso, abrirá uma tela para registrar o banco de dados, nessa tela deve-se selecionar a opção Firebird 1.5 (Figura 25), no campo Server Version e no campo Database Alias escreva o apelido que terá o seu banco de dados criado, por exemplo: TCC, os demais

campos ficam inalterados, deve ser selecionado *Register* para registrar o banco de dados criado.

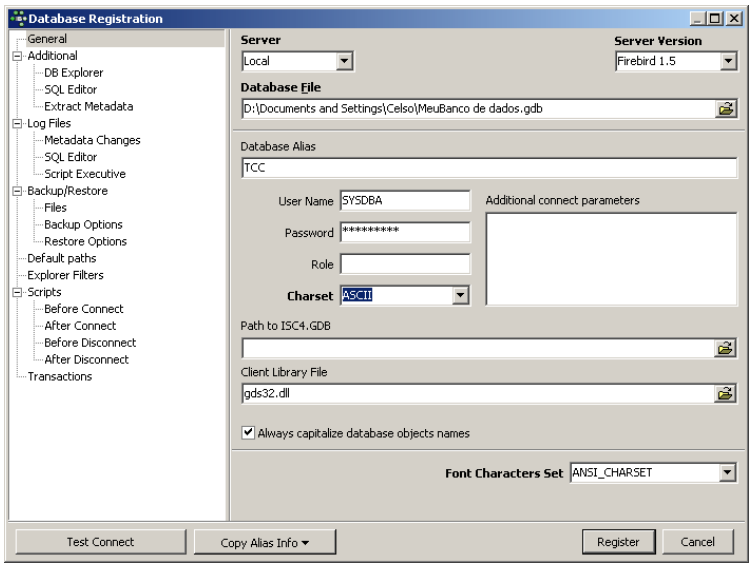

Figura 25 – Registro do Banco de Dados - DB

Para conectar ao banco de dados selecionar duas vezes em cima do apelido (Alias) que foi dado para o seu banco de dados, na palheta *Database Explorer*, que fica do lado esquerdo da tela. Agora já é possível criar tabelas no seu banco de dados para isso basta acionar o botão da direita do mouse em cima da palavra *Tables*, e deve ser selecionado *new Table* (Figura 26).

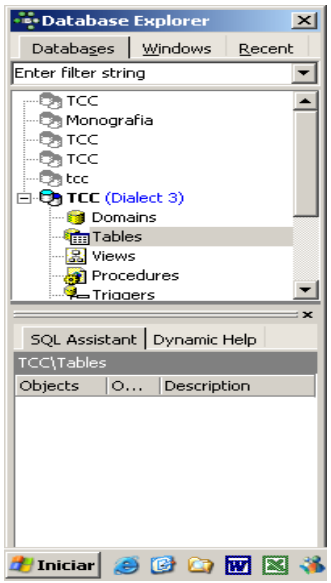

Figura 26 – Explorer do Banco de Dados – DB

Na interface apresentada na Figura 27 deve-se substituir o nome *NEW_TABLE* pelo nome da sua tabela; logo abaixo pode-se criar os campos, substituir o nome *NEW_FIELD* pelo nome do primeiro campo da sua tabela, no campo *Field Type* selecionar o tipo do seu campo, e no campo *Size* colocar o tamanho do seu campo. Para inserir um novo campo é só acionar a seta para baixo.

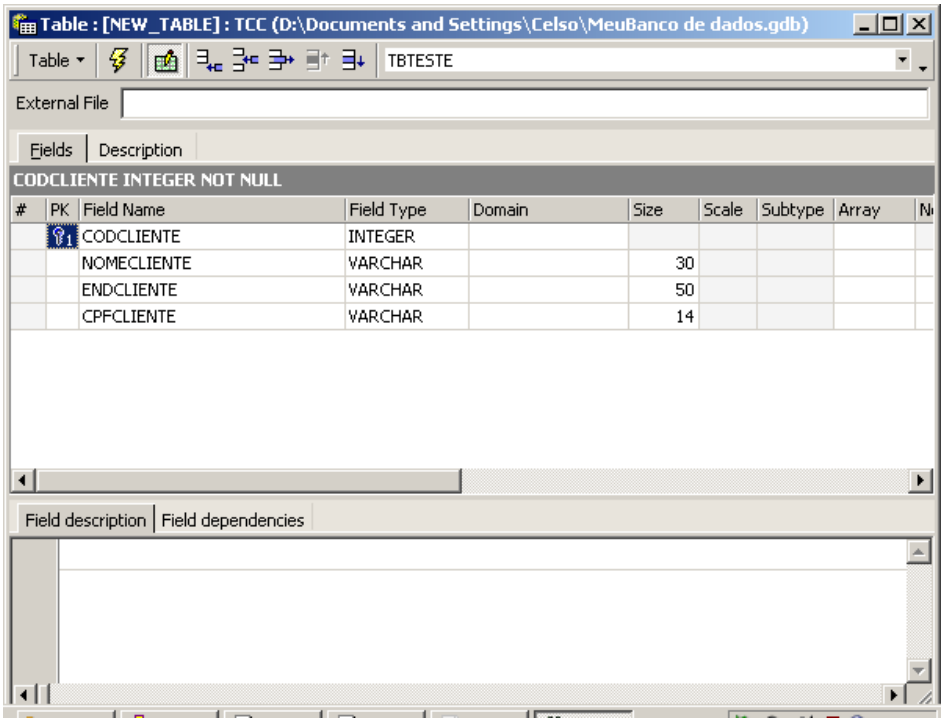

Figura 27 – Tabela do Banco de Dados - DB

Para finalizar deve ser selecionado o ícone com o desenho de um raio e abrirá uma nova tela (Figura 28) mostrando como será criada a tabela *(Creating Table* TBTESTE), deve ser selecionada a opção *commit* para salvar os dados no banco, dessa forma os campos serão numerados e a tabela está criada, como se pode ver à esquerda no *Database Explorer* letra B da Figura 30. Para ver ou para cadastrar dados na tabela é só acionar a aba *Data*, letra C da Figura 30.

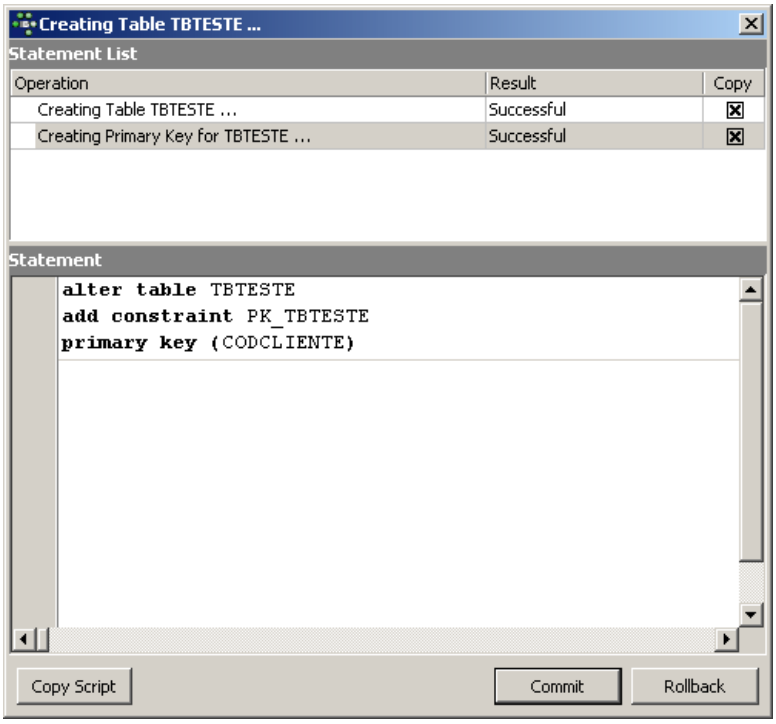

Figura 28 – Criando a Tabela TBTESTE - DB

O código do banco para criar as tabelas pode ser visto na Figura 29. Com comando *create* desse código cria-se a tabela de nome IDIGITAL, contendo os campos COD do tipo inteiro e que é a chave primária, o campo NOME do tipo varchar com cinqüenta posições, o campo CPF do tipo varchar de 15 posições e o campo IMAGEM do tipo varchar de 100 posições.

```
CREATE TABLE IDIGITAL (
   COD     INTEGER NOT NULL,
   NOME    VARCHAR(50),
   CPF     VARCHAR(15),
   IMAGEM  VARCHAR(100)
);
```

Figura 29 – Criação da tabela do Banco de Dados - DB

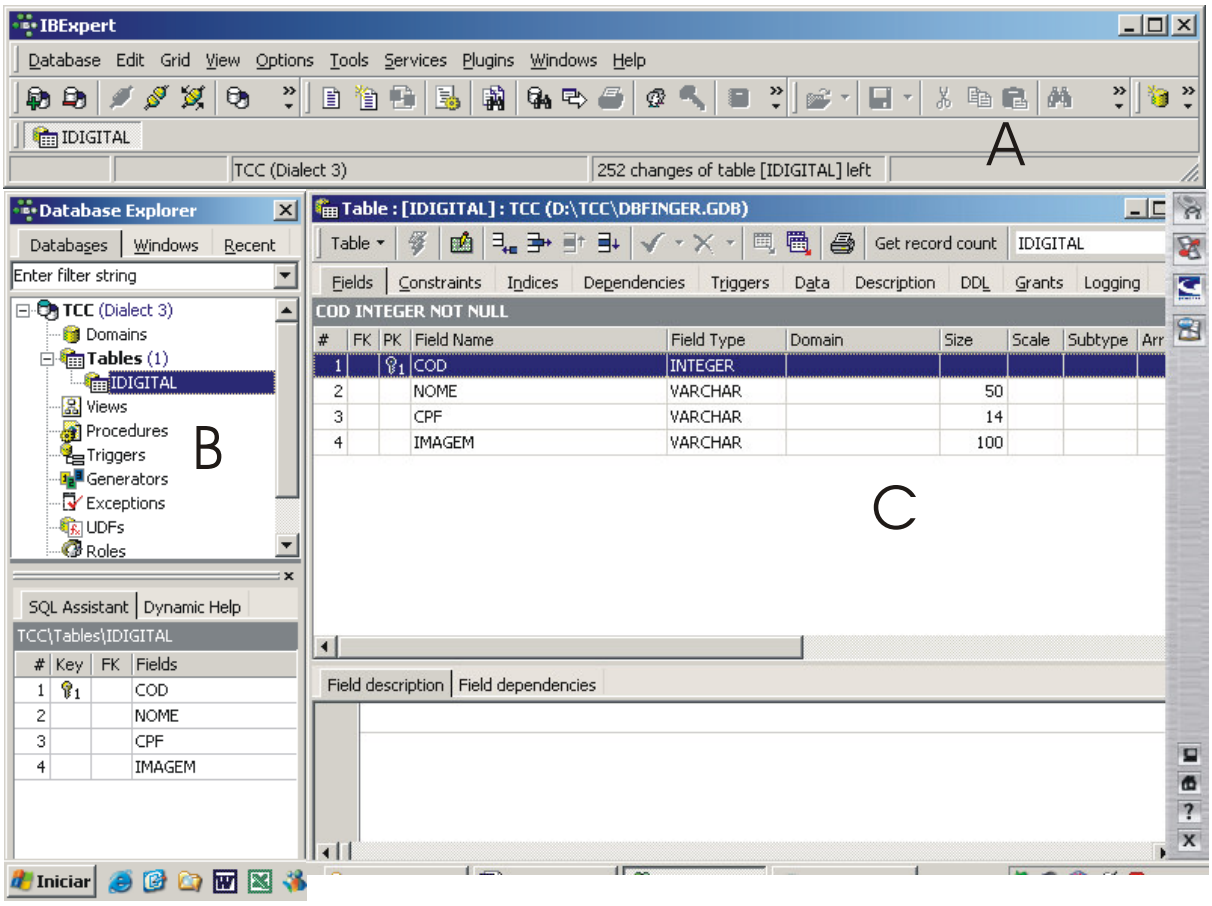

Figura 30 – Tela do IBExpert

2.2 Criação do Software

Nesta seção é apresentada a implementação do *software* que tem por objetivo armazenar imagens de impressão digital utilizando um banco de dados em um software de comparação de impressão digital.

Após a criação da base de dados, citada anteriormente, iniciou-se a geração do *software* para o armazenamento das impressões digitais, utilizando-se a linguagem de programação Java e o SGBD Firebird 1.5 com o auxílio do Ibexpert que é o editor do Banco de dados.

Na Figura 31 é apresentado o diagrama de classe do software desenvolvido.

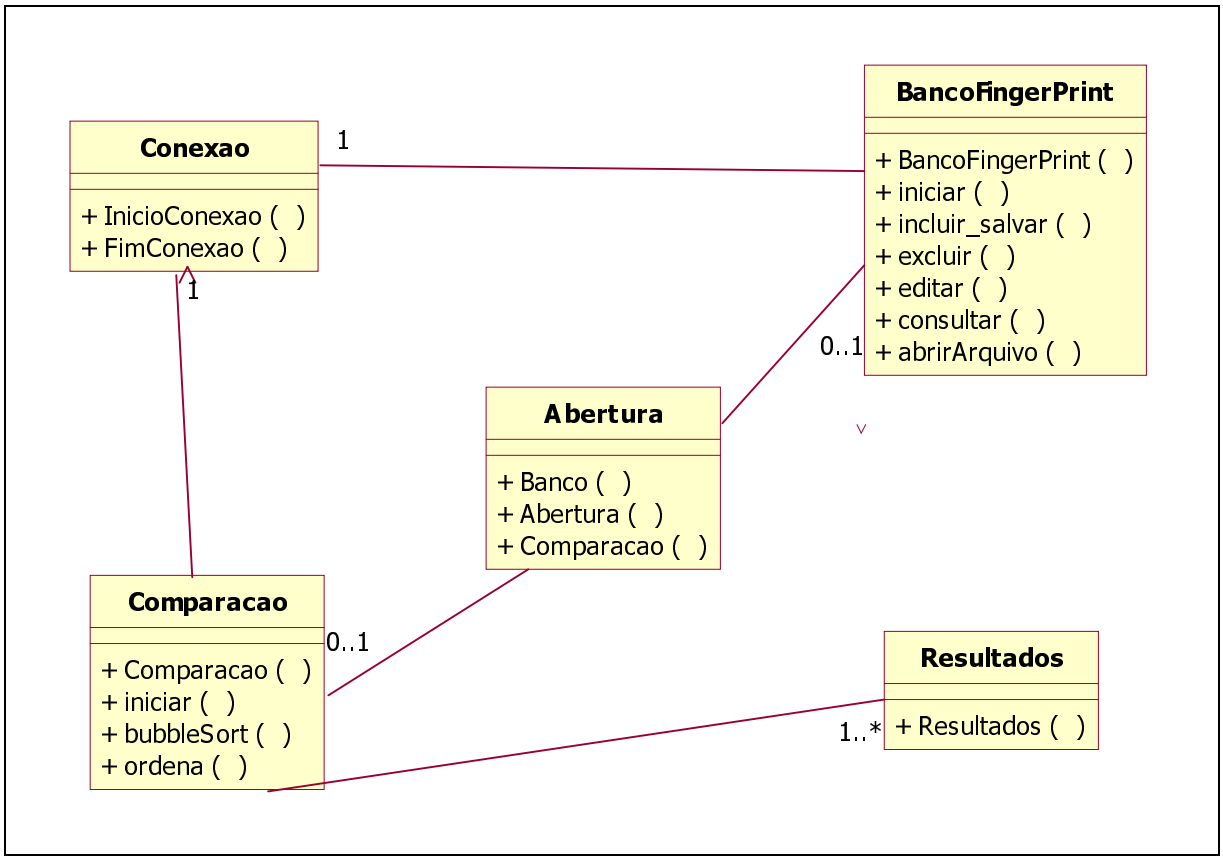

Figura 31 – Diagrama de Classes

2.2.1 Interfaces

A classe Abertura gera a tela inicial (Figura 32), por meio da qual é possível acessar a Manutenção das impressões digitais e acessar a classe que faz a Comparação das impressões digitais.

Figura 32 – Tela inicial - Software

Como pode ser visto no ANEXO A, a classe Abertura tem os métodos Banco() e Comparação(). O método Banco() instancia um objeto da classe BancoFingerPrint e o método Comparação()instancia um objeto da classe Comparação.

2.2.2 Estabelecendo uma conexão com o Banco de Dados

Como verifica-se na Figura 33 existem classes disponíveis em Java para fazer a conexão com o Banco de Dados. Em seguida, foi feita a Classe Conexao, com os métodos InicioConexao(), para conectar ao Banco de Dados e FimConexao(), para desconectar do Banco de Dados. No método InicioConexao() utiliza-se a classe Class.forName para registrar o drive e o DriverManager.getConnection(url, "SYSDBA", "masterkey") estabelece uma conexão.

```
 public static Connection InicioConexao() throws Exception
 {
                try // CONECTANDO BD              IP    PORTA    BD
                {   String url = "jdbc:firebirdsql:127.0.0.1/3050:D:/TCC
231106/DBFINGER.gdb";
                   Class.forName("org.firebirdsql.jdbc.FBDriver");
                        conexao = DriverManager.getConnection(url,"SYSDBA","masterkey");
        System.out.println("CONEXAO COM O FIREBIRD REALIZADA COM SUCESSO
!");
                        conexao.setAutoCommit(false);
                }catch (SQLException e)
                {   System.out.println("ERRO NA CONEXAO COM O FIREBIRD : "+ e);
                }
                return conexao;
 }

 public static void FimConexao()
 {
                try
                {
                        conexao.close(); // FECHA A CONEXÃO
                        System.out.println("PROGRAMA DESCONECTADO...");
                }
                catch(SQLException f)
                {
                        System.out.println("ERRO AO FECHAR O ARQUIVO!!! " + f);
                }
        } // fim da conexão
```

Figura 33 – Trecho do Código Conexao - Software

2.2.3 Manipulando o Banco de Dados

Para manipular o Banco de Dados criou-se a classe BancoFingerPrint que armazena os dados na tabela DBFINGER.GDB do banco de dados. Essa classe gera uma tela (Figura 34) que é composta das opções incluir, excluir, editar e consultar, com os seus respectivos métodos, alterando os dados diretamente no banco de dados.

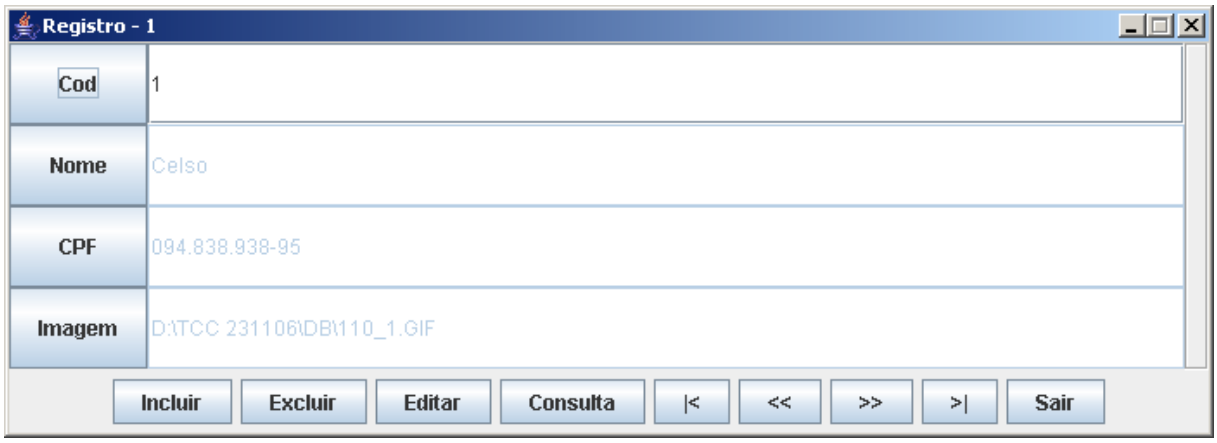

Figura 34 – Tela de Cadastro - Software

A opção incluir permite cadastrar uma nova impressão digital com os seguintes dados: código, nome, CPF e imagem. O código seqüencial é gerado automaticamente e da imagem guarda-se o caminho (*path*). A opção excluir elimina o registro selecionado. A opção editar permite alterar o registro selecionado. A opção consulta exibe um dado registro. A consulta é executada pelo número do código ou pelo nome. Também é possível navegar pelos registros utilizando-se as teclas: início (|<), retroceder (<<), avançar (>>) e fim (>|) e a última opção é o botão sair.

Para poder navegar de registro em registro foi criado um registro que é carregado toda vez que essa classe é chamada, como pode ser visto na Figura 35. Sempre que há uma alteração, esta é feita tanto no banco de dados quanto no array.

```
  public void iniciar()
  {             //carrega todo o banco de dados para o registro e
mostra o primeiro
                sql = "SELECT * FROM IDIGITAL ORDER BY COD";
                indice=0;
                cont = 0;
                try
                {
                      rs = st.executeQuery(sql);
                      while (rs.next())
                      {
                            registro[indice][cod] =
rs.getString("COD");
                            registro[indice][nome] =
rs.getString("NOME");
                            registro[indice][cpf] =
rs.getString("cpf");
                            registro[indice++][imagem] =
rs.getString("IMAGEM");
                            cont++;
                      }
                }
                catch(SQLException e)
                {
                      System.out.println("ERRO NA BUSCA : " + e);
                }

                indice=0;
                  tf.setText(registro[indice][cod]);
                  tf1.setText(registro[indice][nome]);
                  tf2.setText(registro[indice][cpf]);
                  tf3.setText(registro[indice][imagem]);
                set_title("set");
                 tf_set_enable(false);
                 bt_verificar();
  }
```

Figura 35 – Método Iniciar - Software

2.2.4 Selecionando uma impressão digital para ser comparada com as outras do banco de dados.

A classe Comparacao abre uma interface onde pode ser selecionado um nome, utilizando um *combobox*, pelo qual trará na tela uma impressão digital com os dados de seu proprietário: código, nome, CPF e o *path* da imagem (Figura 36).

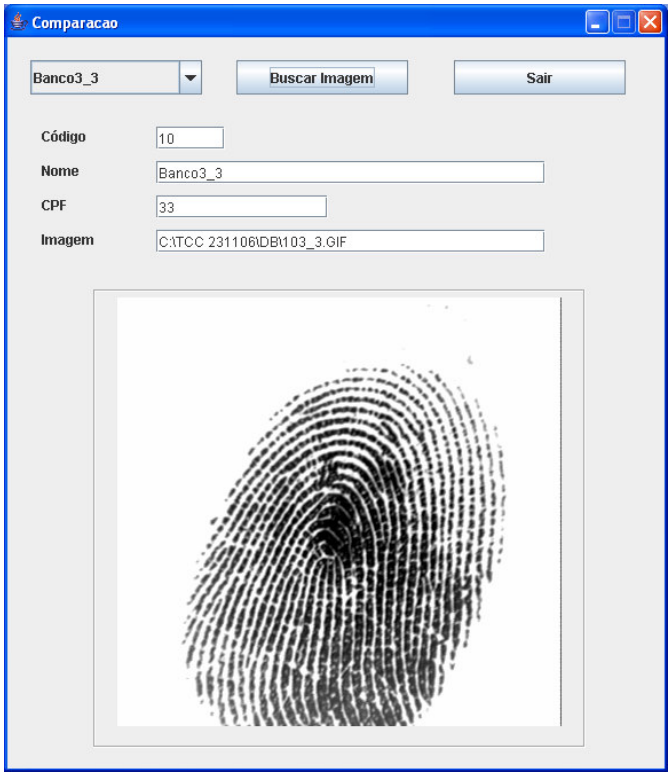

Figura 36 – Tela Comparação - Software

Na Figura 36 pode-se observar dois botões, um para encerrar a execução (sair) e outro para buscar a imagem (Buscar Imagem). Esse último, quando acionado, fará 3 (três) perguntas: 1) "Digite a porcentagem a ser analisada, a partir do centro [30 a 100]." (Figura 37); 2) "Digite o raio a ser analisado, a partir do centro [5 a 25]." (Figura 38) e 3) "Quantas impressões mais semelhantes você quer ver? [1 a 5]" (Figura 39).

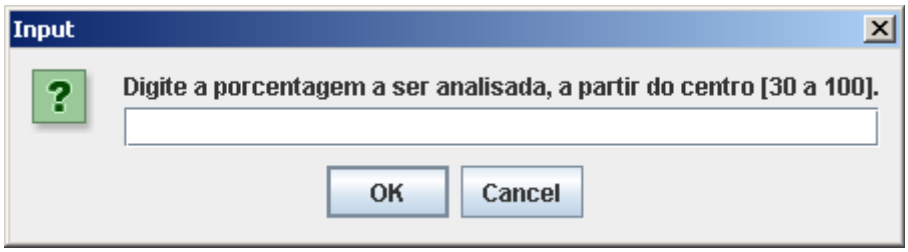

Figura 37 – Pergunta da Porcentagem - Software

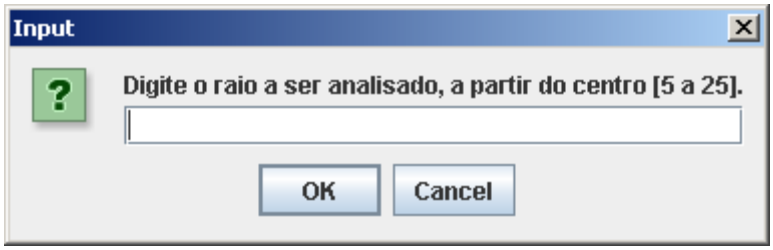

Figura 38 – Pergunta do Raio - Software

Figura 39 – Pergunta Quantidade de Impressões - Software

Com esses parâmetros o programa faz a comparação da impressão digital selecionada com todas as outras impressões digitais do banco de dados, guardando o número das minúcias coincidentes e o código delas. Após abrirá uma nova tela (Figura 40), gerada pela classe Resultados, que trará o código, nome e as impressões digitais que mais coincidem com a impressão digital selecionada, é importante salientar que é fornecida a quantidade de impressões mais coincidem que foram selecionadas na última pergunta: "Quantas impressões mais semelhantes você quer ver? [1 a 5]" (Figura 39). Quanto maior for o número da porcentagem e do raio escolhidos o programa achará mais minúcias e maior será a chance do programa achar a impressão coincidente.

A Figura 40 é o resultado da busca da Figura 36 com os seguintes parâmetros: porcentagem 100, raio 25, impressões semelhantes 1.

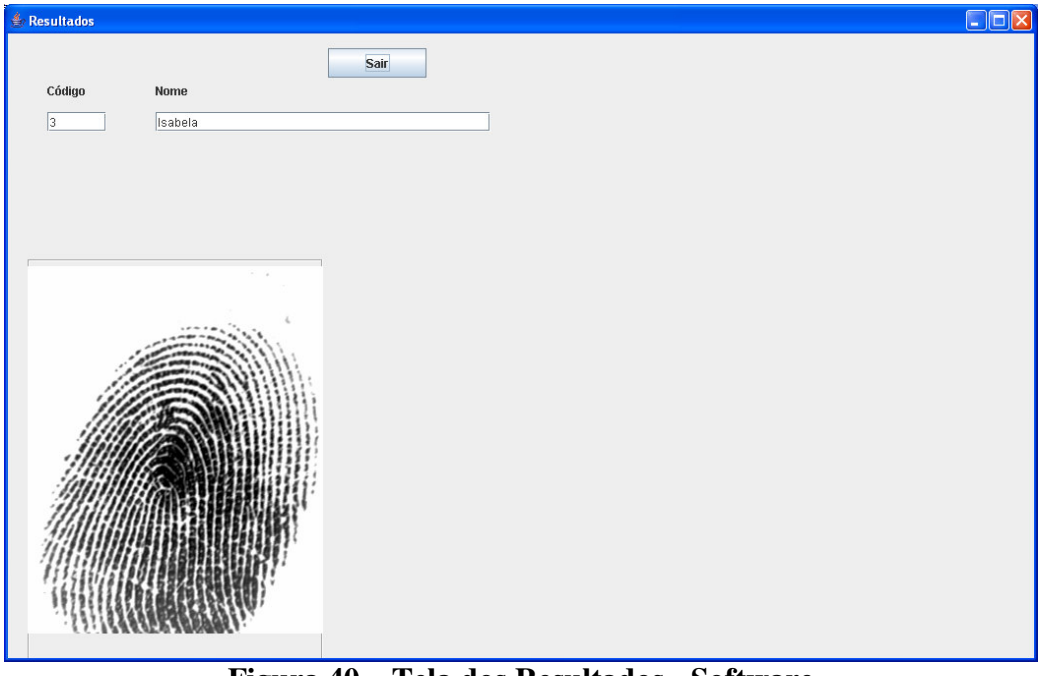

Figura 40 – Tela dos Resultados - Software

Para a captura das minúcias das impressões digitais foi utilizado o software FingerPrint (software pronto) que faz um tratamento em cada impressão digital, faz a extração das minúcias e em seguida faz o “casamento” da minúcias e retorna a quantidade de minúcias coincidentes.

2.3 Resultados

O que pode-se notar é que têm impressões digitais que geram muitas minúcias (Figura 41) e outras impressões digitais geram poucas minúcias. E quando uma impressão digital qualquer é comparada com essa que gera muitas minúcias, a tendência é que um número maior de minúcias sejam coincidentes, enquanto que essa mesma minúcia se comparada com uma impressão digital que gera poucas minúcias, a tendência é que haja menos casamentos. Já quando se compara duas impressões digitais idênticas, o mesmo número de minúcias será o número de casamento, portanto, nesse caso, o programa apresenta resultados satisfatórios, encontrando as impressões idênticas.

A Figura 41 é de baixa qualidade, a impressão que se tem é que foi colocado a impressão digital em um papel e depois foi adquirida por um *scanner*, sem contar que ela está de ponta cabeça.

Figura 41 – Impressão Digital com baixa qualidade

Foram utilizadas impressões digitais do banco de dados (FVC2000, 2006). As impressões digitais foram colocadas em um mesmo diretório e quando do cadastro foram armazenados os *path* das imagens.

Para testar o software foi tomada uma impressão digital (Figura 42) nas quais foram feitas várias modificações: 1) A imagem foi clareada e gravada com o nome 102_2 Clara.gif; 2) A imagem foi deslocada para baixo e gravada com o nome 102_2 Descida.gif; 3) A imagem foi desfocada e gravada com o nome 102_2 Desfoque.gif; 4) A imagem foi escurecida e gravada com o nome 102_2 Escura.gif; 5) A imagem foi escurecida e engrossada as linhas da impressão digital e gravada com o nome 102_2 Escura Borrada.gif; 6) A imagem foi melhorada e gravada com o nome 102_2 Nitida.gif; e 7) A imagem foi enclinada para a direita e gravada com o nome 102_2 Virada.gif;

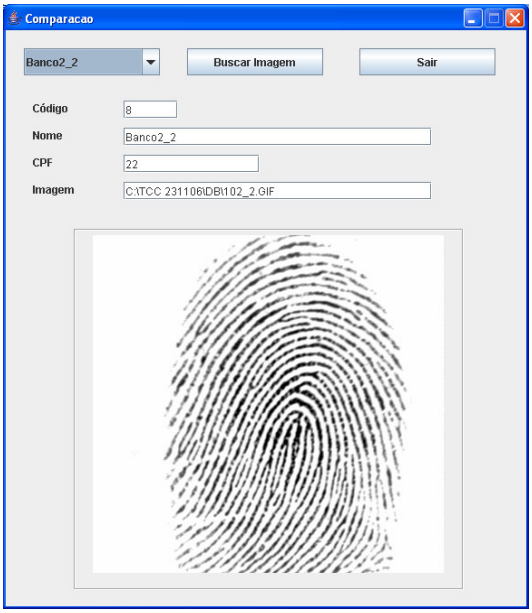

Figura 42 – Impressão Digital 102_2.gif

Após começaram-se os testes e para todos eles o número de imagens a ser retornada é 3, gerando os seguintes resultados:

1.º) Figura selecionada 102_2.gif (Figura 42), porcentagem escolhida 100%, raio 25, tempo gasto 15 s, depois de fazer os casamentos das minúcias retornaram as imagens

respectivamente nessa ordem (Figura 43): a) 102_2 Clara.gif b) 102_2 Escura.gif e c) 102_2 Escura borrada.gif. Nesse caso o programa foi 100% eficiente retornando só variações da imagem principal e em pouco tempo.

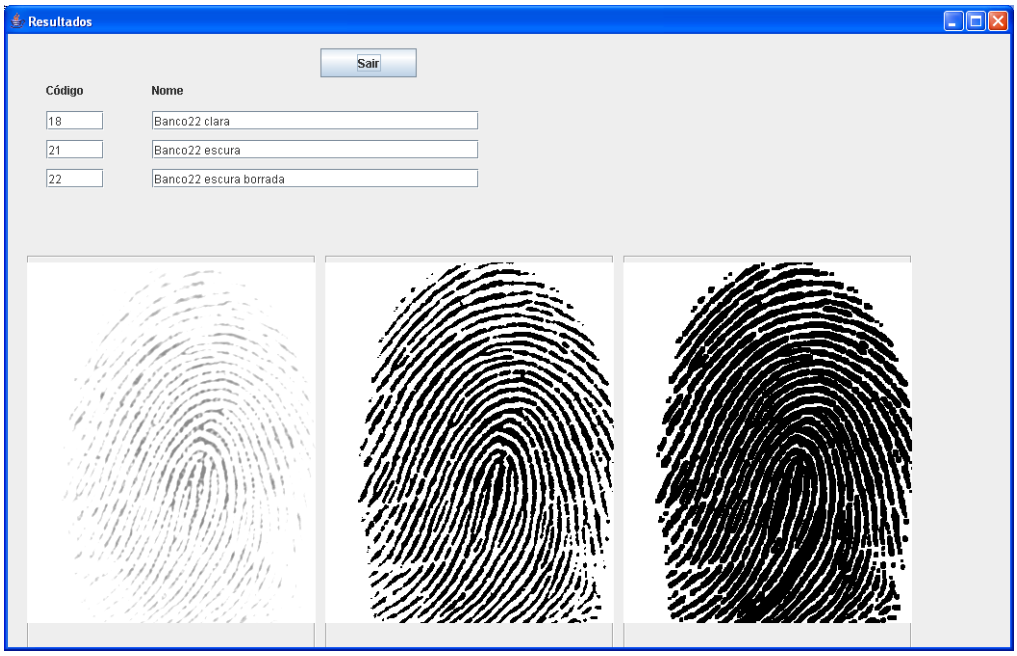

Figura 43 – Resultado do primeiro teste

2.º) Figura selecionada 102_2.gif, porcentagem escolhida 90%, raio 20, tempo gasto 15 s, retornaram as imagens: a) 102_2 Clara.gif b) 102_2 Escura.gif e c) 102_2 Virada.gif. Também com 100% de eficiência retornando só variações da imagem principal e em pouco tempo.

3.º) Figura selecionada 102_2 Clara.gif (Figura 44), porcentagem escolhida 100%, raio 25, tempo gasto 23 s, retornaram as imagens (Figura 45): a) 102_2.gif b) 102_2 Escura.gif e c) 102_2 Escura Borrada.gif. O que pode-se notar é que a primeira imagem que retorna é justamente a imagem original, novamente teve 100% de eficiência e o tempo aumentou um pouco. Na Figura 46 pode-se ver a quantidade de minúcias que foram casadas, respectivamente, com cada imagem do banco de dados.

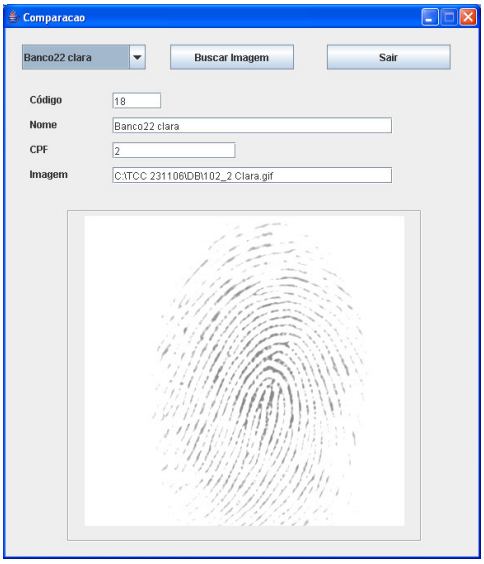

Figura 44 – Impressão Digital 102_2 Clara.gif

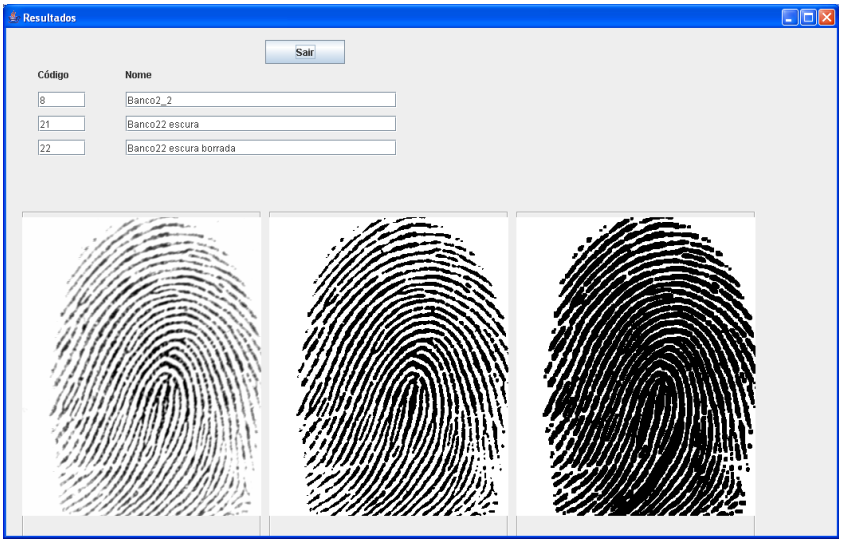

Figura 45 - Resultado do terceiro teste

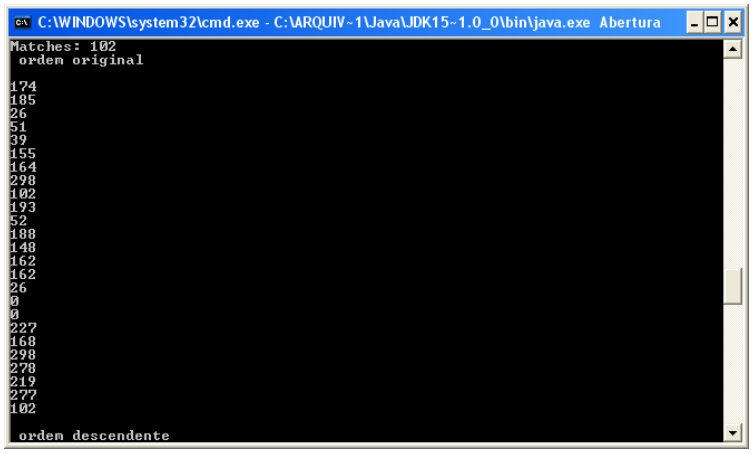

Figura 46 – Quantidade de casamentos realizados no terceiro teste

4.º) Figura selecionada 102_2 Desfoque.gif (Figura 47), porcentagem escolhida 100%, raio 20, tempo gasto 46 s, retornaram as imagens (Figura 48): a) 102_2 Descida.gif b) 102_2 Escura Borrada.gif e c) 102_2 Nitida.gif. Este foi o teste que mais tempo demorou, no entanto continuou trazendo as variações da imagem principal.

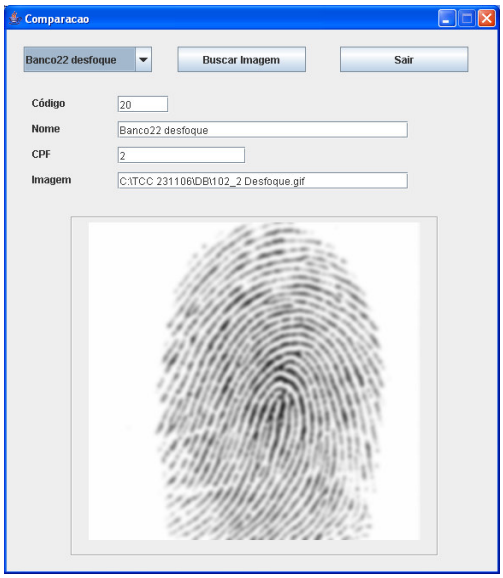

Figura 47 - Impressão Digital 102_2 Desfoque.gif

Figura 48 - Resultado do quarto teste

5.º) Figura selecionada 102_2 Escura Borrada.gif (Figura 49), porcentagem escolhida 80%, raio 15, tempo gasto 24 s, retornaram as imagens (Figura 50): a) 110_1.gif b) 109_1.gif e c) 102_2 Virada.gif. Dos testes realizados com a imagem principal (Figura 42) este foi o único que trouxe imagens diferentes das imagens alteradas da principal. O que nota-se é que a

Imagem selecionada gerou 678 minúnias e a primeira imagem retornada gerou 540 minúcias o qual gerou 202 casamentos entre as minúcias. Estas foram as duas imagens que geraram mais minúcias e conseqüentemente obtiveram mais casamentos. Na Figura 51 pode-se ver a quantidade de minúcias que foram casadas, respectivamente, com cada imagem do banco de dados.

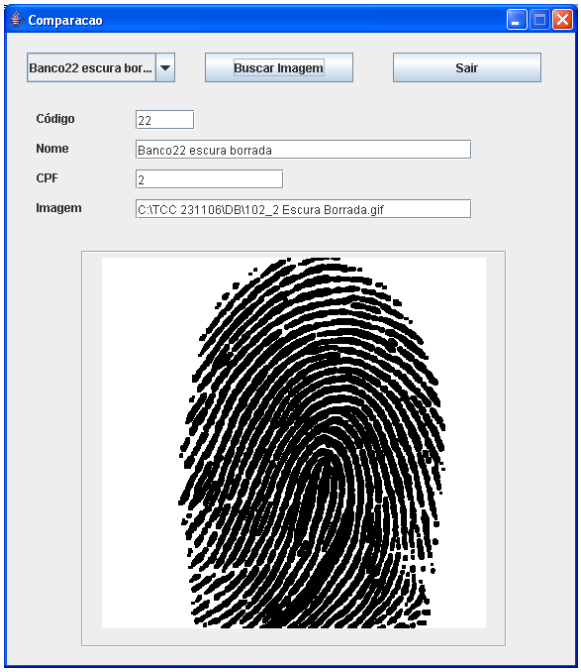

Figura 49 - Impressão Digital 102_2 Escura Borrada.gif

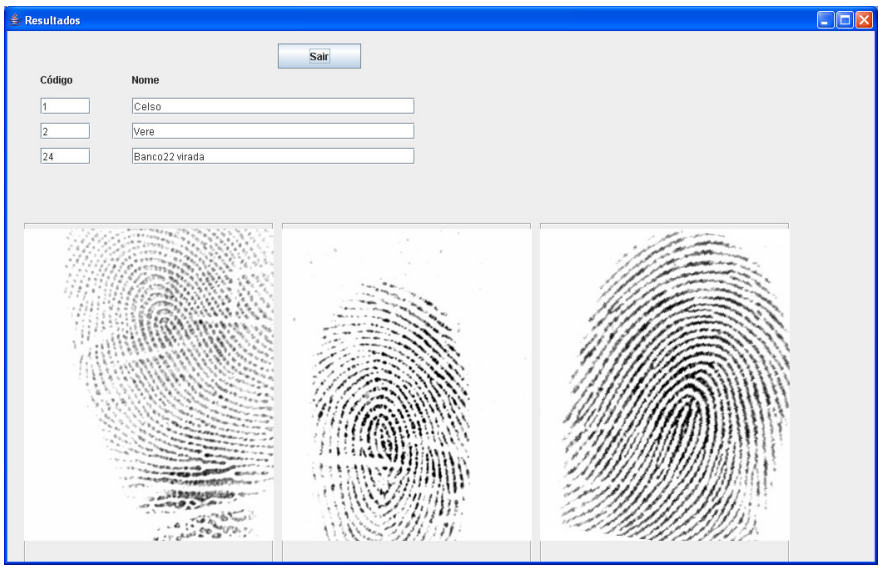

Figura 50 - Resultado do quinto teste

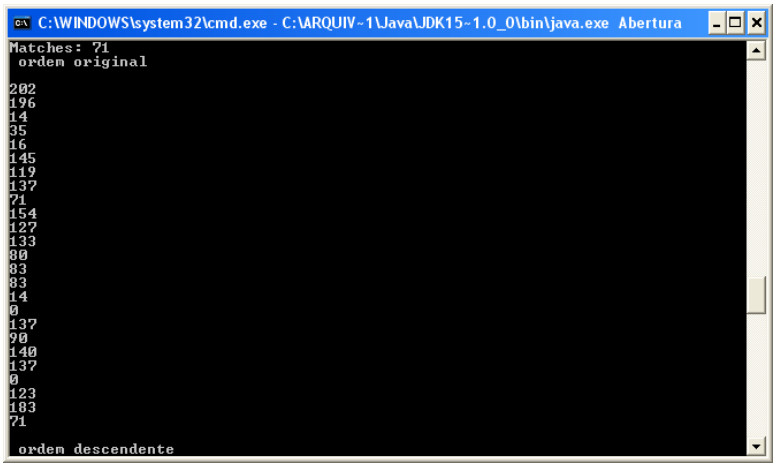

Figura 51 - Quantidade de casamentos realizados no quinto teste

CONCLUSÕES

O *software* confeccionado faz o casamento adequado de duas impressões digitais idênticas, como pode ser observado nos resultados obtidos, porém se a impressão digital for grande e de baixa qualidade, o software irá encontrar muitas minúcias e na realização do casamento com as minúcias de outras impressões digitais gera vários casamentos. Por isso, quase sempre ela é a primeira impressão digital retornada na tela de resultados, gerando resultados errados.

Outro ponto a ser ressaltado é a demora do processamento pois a impressão digital escolhida terá que ser casada com todas as outras impressões digitais e a cada comparação tem-se os seguintes passos: 1) Tratamento da impressão digital selecionada (equalizacao, thresholding, dilatacao2, thinning e limpeza 2) e extração das minúcias 2) Tratamento da impressão digital 1 (equalizacao, thresholding, dilatacao2, thinning e limpeza 2) e extração das minúcias 3) Casamento das minúcias da impressão selecionada com as da impressão digital 1 4) Guarda-se o número de casamento e a posição da impressão digital comparada, que nessa caso foi a impressão digital 1 5) Os 4 (quatro) passos anteriores só que agora comparando com a impressão digital 2 6) idem ao passo 5 ... comparando com a impressão digital 3. E assim sucessivamente até fazer a comparação com a última impressão digital do Banco de dados. Portanto, se tiver 5 (cinco) impressões digitais no banco de dados vai levar pouco tempo para fazer todas as comparações e se tiver 10.000 (dez mil) impressões digitais no banco de dados vai levar muito tempo para fazer todas as comparações. Nos testes realizados o tempo variou de 15 s a 46 s para fazer todas as comparações e em média demorou 30 s.

O índice de aproveitamento nos testes foi de mais de 90% comparando uma imagem principal com as suas variações. Quando selecionada uma imagem para compará-la com a

cópia dela o índice de acerto foi 100%. Portanto, pode-se concluir que o software apresentou resultados satisfatórios.

Para um trabalho futuro deve-se procurar melhorar os dois pontos citados acima. Seria interessante guardar todas as minúcias das impressões digitais na hora do seu cadastro e quando fosse fazer o casamento da impressão digital selecionada com as outras impressões digitais o programa faria a comparação com essas minúcias guardadas, evitando assim todos os passos anteriormente citados.

REFERÊNCIAS BIBLIOGRÁFICAS

BALLARD, D.H; BROWN, C.M. **Computer Vision**. Prentice-Hall Inc, Englewood Cliffs, New Jersey, 1982.

CHONG, Michael M. S.; GAY, Robert K. L.; TAN, Han Ngee; LIU, Jun. **Geometric Framework for Fingerprint Image Classification**. Pattern Recognition, vol. 30, n° 9, pp. 1475-1488, 1997.

COSTA, Silvia Maria Farani. **Classificação e Verificação de Impressões Digitais**. Monografia, 2001.

FBI: Federal Bureau of Investigation. **The Science of Fingerprints: Classification and Uses**. U.S. Government Printing Office, Washington, DC (1984).

FVC2000: Fingerprint Verification Competition. **Databases**. Site: http://bias.csr.unibo.it/ fvc2000/databases.asp, acessado no dia 21/05/2006.

GONZALEZ, R. C.; WINTZ, P. **Digital Image Processing**. Addison-Wesley Publishing Company, United States of America, 1987.

GONZALEZ, Rafael C..; WOODS, Richard E. **Processamento de Imagens Digitais**. Editora Edgard Blücher Ltda, 2005.

HENRY, E. R.. **Classification and Uses of Fingerprints**. Wyman and Sons Ltda., 1905.

HONG, Lin; JAIN, Anil; PANKANTI, Sharath; BOLLE, Ruud. **Fingerprint Enhancement**. Proc. Third IEEE Workshop on Applications of Computer Vision, pp. 202-207, 1996.

HONG, Lin; WAN, Yifei; JAIN, Anil. **Fingerprint Image Enhancement: Algorithm and Performance Evaluation**. IEEE Transactions On Pattern Analysis And Machine Intelligence, Vol. 20, No. 8, August 1998.

HONG, Lin; JAIN, Anil K.. **Integrating Faces and Fingerprints for Personal Identification**. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, n° 12, pp. 1295-1307, 1998.

HRECHAK, Andrew K.; MCHUGH, James A.. **Automated Fingerprint Recognition using Structural Matching**. Pattern Recognition, vol. 23, n° 8, pp. 893-904, 1990.

HUNG, Douglas D. C.. **Enhancement and Feature Purification of Fingerprint Images**. Pattern Recognition, vol. 26, n° 11, pp. 1661-1671, 1993.

JAIN, Anil; HONG, Lin; BOLLE, Ruud. **On-Line Fingerprint Verification**. IEEE Transactions On Pattern Analysis And Machine Intelligence, Vol. 19, No. 4, April 1997.

JAIN, Anil K.; PRABHAKAR, Salil; HONG, Lin. **A Multichannel Approach to Fingerprint Classification**. IEEE Transactions On Pattern Analysis And Machine Intelligence, Vol. 21, No. 4, pp. 348-359, 1999.

KOVÁCS, Z. M. **A Fingerprint Verification System Based on Triangular Matching and Dynamic Time Warping**. IEEE Transactions On Pattern Analysis And Machine Intelligence, Vol. 22, No. 11, November 2000.

LIN, C. H.; LIU, J. H.; OSTERBERG, J. W.; NICOL, J. D. **Fingerprint comparison I – Similarity of Fingerprints**. J. Forensic Sci 27, pp. 290-304, 1982.

MOAYER, Bijan; FU, King Sun. **A Syntactic Approach to Fingerprint Pattern Recognition**. Pattern Recognition, vol. 7, pp. 1-23, 1975.

PERENHA, Rodrigo Afonso. **Sistema de Processamento de Imagens para Reconhecimento de impressão digital**. Monografia, 2003.

RAO, Kameswara. **Feature Extraction for Fingerprint Classification**. Pattern Recognition, vol. 8, pp. 181-192, 1976.

RATHA, Nalini K.; CHEN, Shaoyun; JAIN, Anil K.. **Adaptative Flow Orientation – Based Feature Extraction in Fingerprint Images**. Pattern Recognition, vol. 28, n° 11, pp. 1657-1672, 1995.

TAMURA, Hideyuki; **A Comparison of Line Thinning Algorithms form Digital Geometry Viewpoint**. Proc. Of 4th Int. Jt. Conf. On Pattern Recognition, Kyoto Japan, 1978

WILSON, C. L.; CANDELA, G. T.; WATSON, C. I.. **Neural Network Fingerprint Classification**. Journal of Artificial Neural Networks, 1(2), pp. 203-228, 1994.

APÊNDICES

APÊNDICES A – INSTALAÇÃO DO JAVA JDK1.5.0_09

Para a confecção do software utilizou-se a linguagem Java jdk1.5.0_09 e o SGBD FireBird 1.5, com o auxílio do IBExpert versão 2005.06.07.

Foi instalado o arquivo jdk-1_5_0_09-windows-i586-p.exe de 50.7 MB, que pode ser adquirido no site da Sun: www.java.sun.com, na seção download. Quando iniciar a instalação desse arquivo a primeira tela que se abre é da licença (Figura 52), após a leitura selecionar a opção "*I accept the terms in the license agreement*" e deve ser acionado *next*.

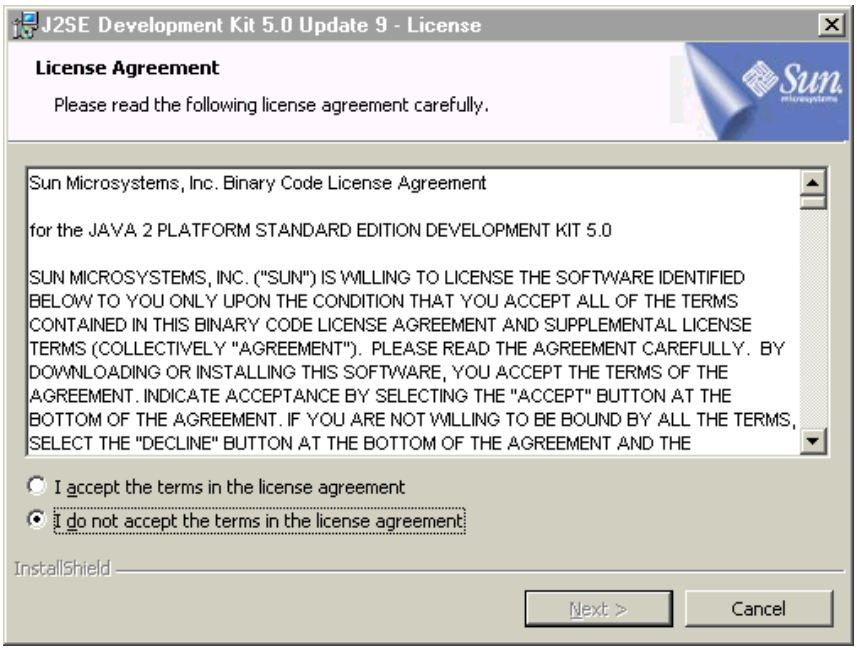

Figura 52 – Licença do Java 1.5.0_09

Na seqüência é perguntado quais os itens do programa JRE se deseja instalar e em qual diretório (Figura 53), após as alterações deve ser selecionado *next*. No caso desse trabalho apenas foi aceito as sugestões da instalação.

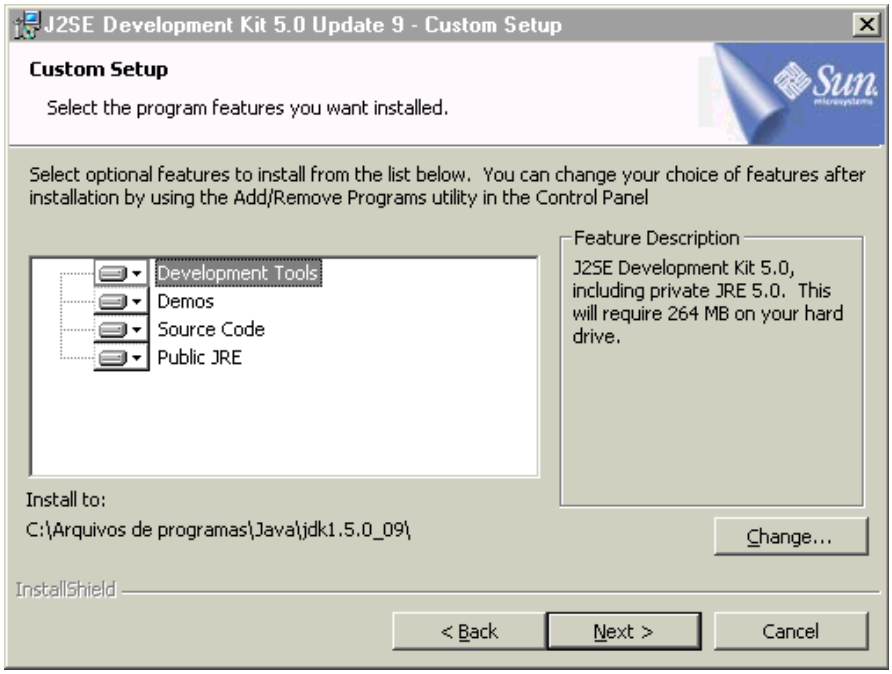

Figura 53 – Custom Setup – JRE Java 1.5.0_09

Imediatamente, então, o programa começa a ser instalado, esse processo é um pouco demorado e dependendo do microcomputador pode levar de 10 a 15 minutos. Posteriormente é perguntado quais os itens do programa J2SE (JDK) se deseja instalar e em qual diretório (Figura 54), após as alterações deve ser selecionado *next*. No caso desse trabalho aqui também foram aceitas as sugestões da instalação.

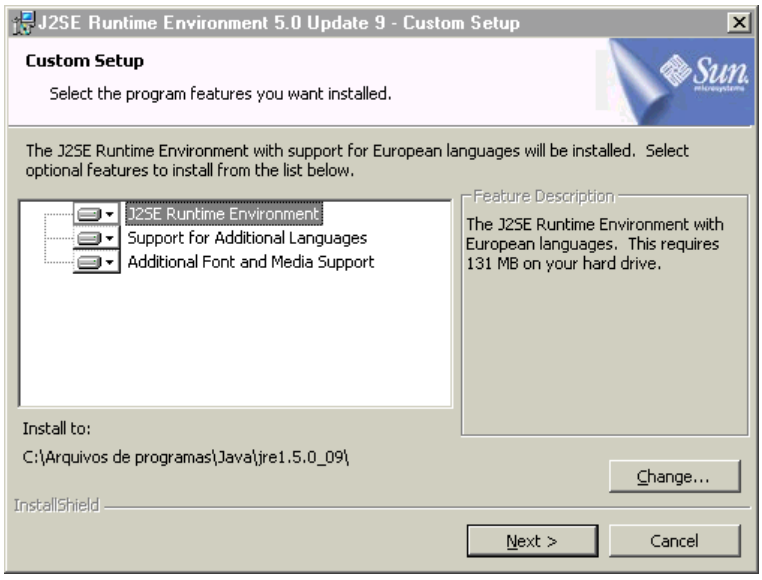

Figura 54 – Custom Setup - JSE Java 1.5.0_09

Em seguida é pedido para selecionar o browser que vai ser registrado com o Java (Figura 55), porém a única opção que ele apresenta é o Microsoft Internet Explorer, caso o seja outro deverá ser alterado pelo Painel de Controle, nesse caso deixou-se a opção Microsoft Internet Explorer, acionar *next* para prosseguir.

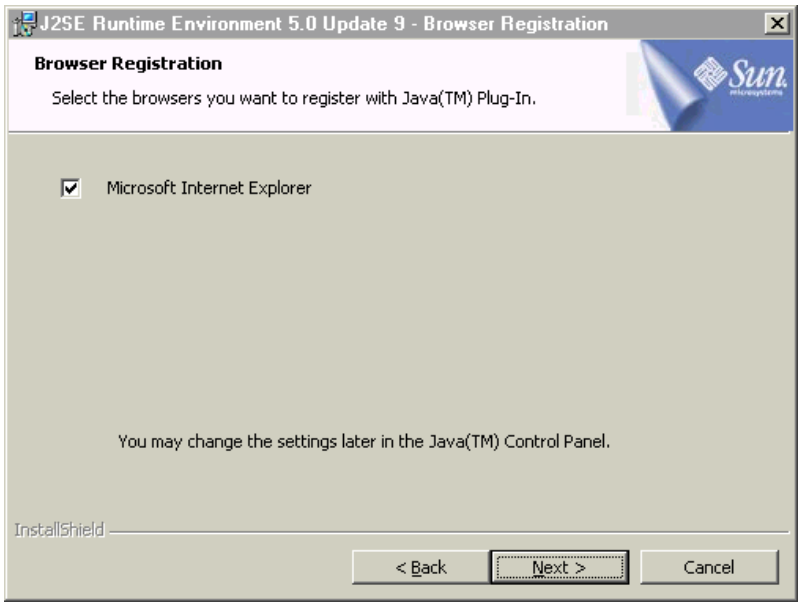

Figura 55 – Browser Registration do Java 1.5.0_09

Após a instalação dos arquivos, que leva de 5 a 10 minutos a instalação está completada (Figura 56) e para finalizar deve ser selecionado *finish*.

Figura 56 – Installation Completed do Java 1.5.0_09

Acessando o diretório c:\Arquivos de Programas\Java pode-se ver os arquivos que foram instalados.

Instalação das Bibliotecas JAI do Java

Após a instalação do Java foi necessário instalar as bibliotecas JAI (Java Advanced Imaging) para poder dar o import javax.media.jai.*, para trabalhar com imagens:

Primeiramente instala-se a biblioteca jai-1_1_2_01-lib-windows-i586-jdk de 5.4 Mb que será colocada no diretório C:\Arquivos de programas\Java\jdk1.5.0_09 do Java.

Quando executa-se este arquivo a primeira tela que se abre (Figura 57) recomenda-se fechar todos os programas para depois continuar o setup clicando no next.

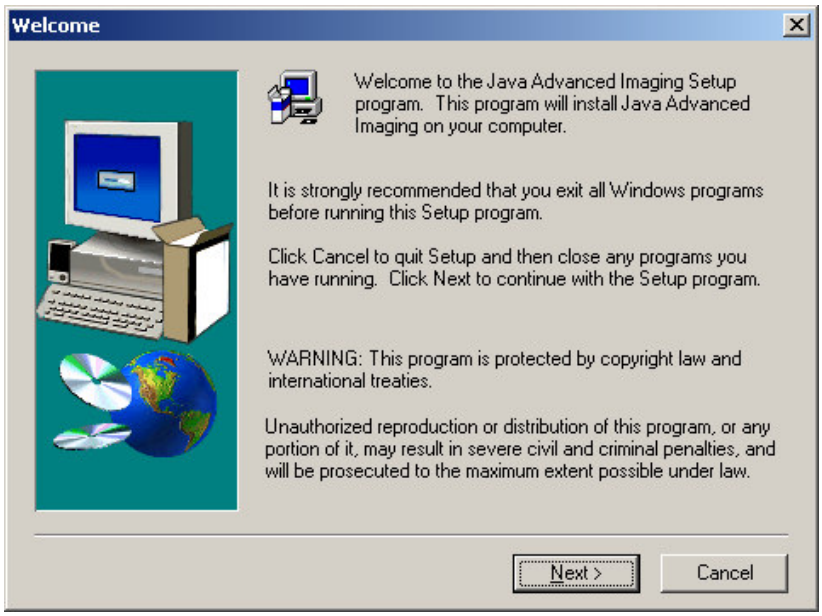

Figura 57 - jai-1_1_2_01-lib-windows-i586-jdk

Na próxima tela (Figura 58) deve-se ler os termos da licença e se de acordo acionar *yes*.

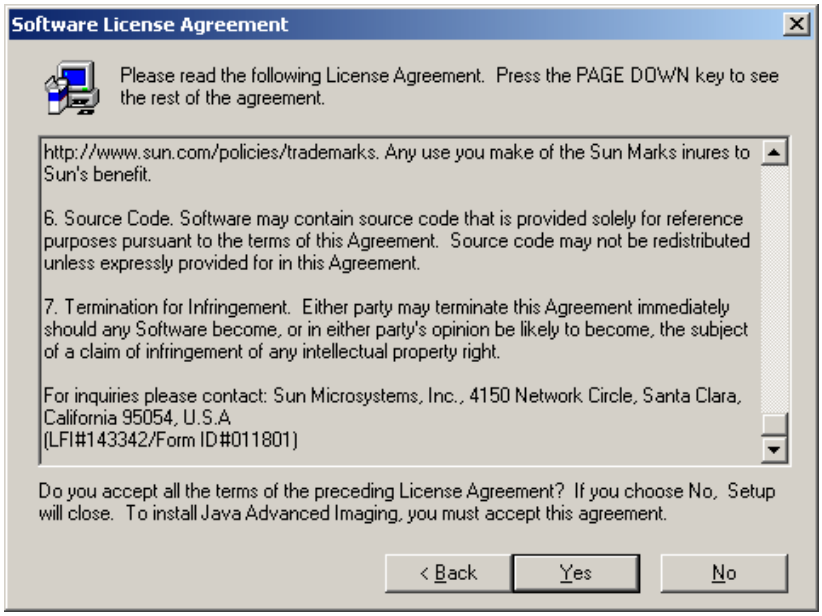

Figura 58 - Software License Agreement

Em seguida mostra-se o diretório onde o JAI – JDK vai ser instalado (Figura 59).

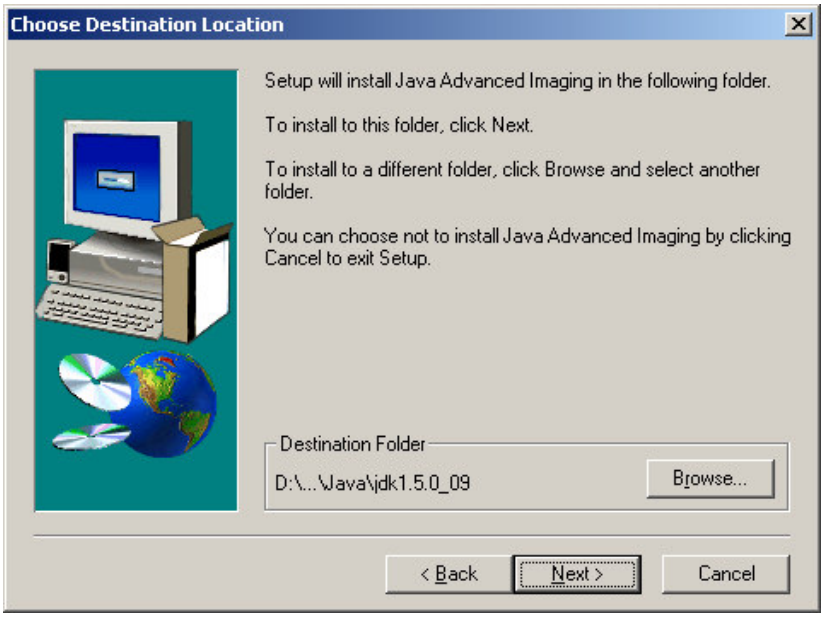

Figura 59 – Choose Destination Location

A próxima biblioteca a ser instalada é jai-1_1_2_01-lib-windows-i586 de 5.4 Mb que será instalada na raiz, no diretório C:\jai-1_1_2_01, obedecendo os 3 (três) passos anteriores.

Por último instala-se a biblioteca jai-1_1_2_01-lib-windows-i586-jre, também de 5.4 Mb que será instalada no diretório C:\Arquivos de programas\Java\jre1.5.0_09, também obedecendo os 3 (três) passos anteriores.

Apêndice B – Instalação e configuração do Banco de Dados Firebird 1.5.2.4731

Para instalar o Banco de dados Firebird 1.5.2.4731 deve-se executar o arquivo Firebird-1.5.2.4731-Win32 de 2.7 MB. Inicialmente pergunta-se qual o idioma vai ser usado durante a instalação, recomenda-se aceitar a sugestão dele que o idioma Português (Standard), conforme a Figura 60, após deve ser selecionado *OK*:

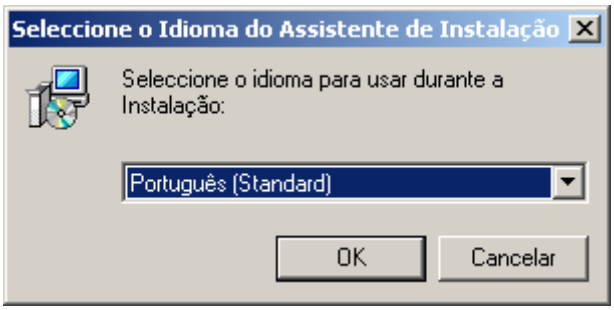

Figura 60 – Seleção de Idioma

Em seguida abre-se a tela inicial dizendo que vai ser feita a instalação do Firebird 1.5.2.4731 e recomenda fechar todas as outras aplicações antes de continuar (Figura 61), deve ser selecionado seguinte.

Figura 61 – Assistente de Instalação

Seguidamente abre-se a tela do Contrato de Licença que deve ser lido e se de acordo selecionar a opção “Aceito o Contrato” (Figura 62) e deve ser selecionado seguinte.

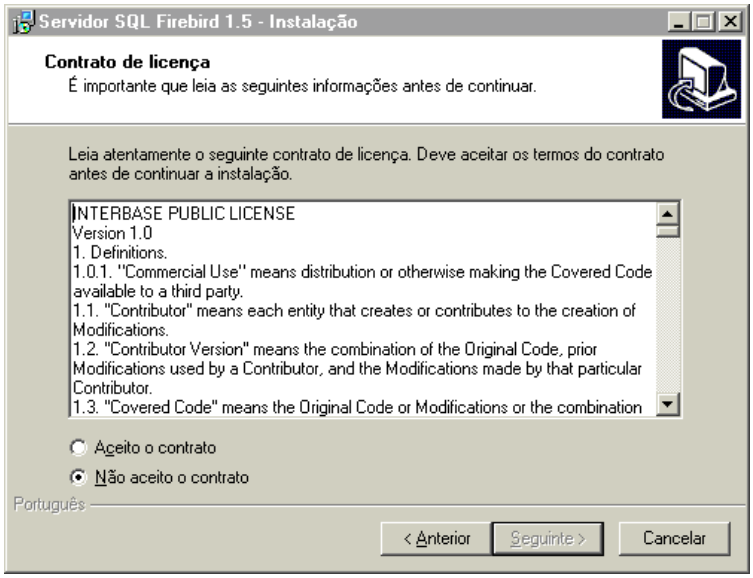

Figura 62 – Contrato de Licença

Ato contínuo abre uma nova tela com informações importantes que também deve ser lida (Figura 63) e deve ser selecionado seguinte para continuar.

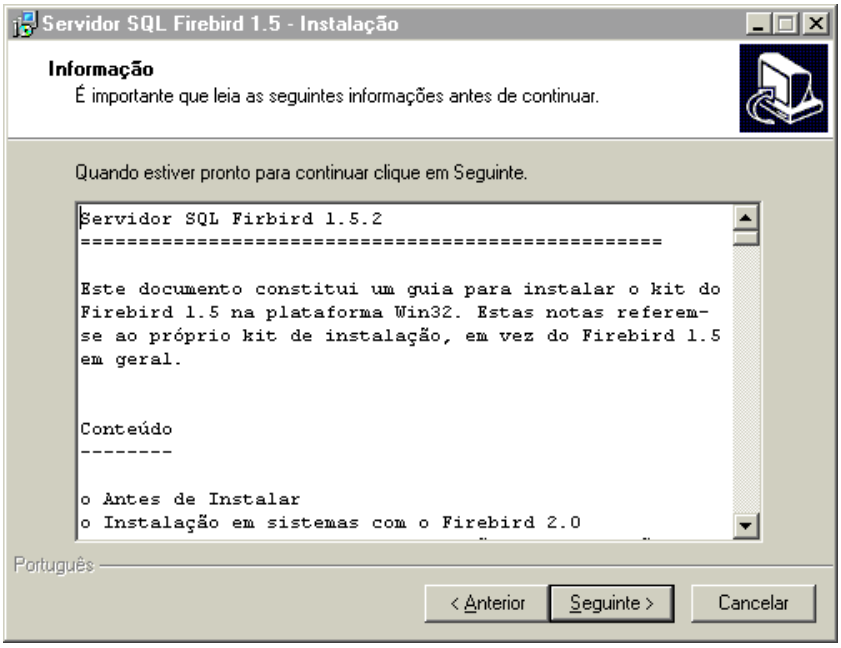

Figura 63 – Informações importantes

A nova tela quer saber em que diretório o Firebird deverá ser instalado e que ele ocupará pelo menos 1.4 Mb de espaço livre em disco (Figura 64). Recomenda-se aceitar essa opção e acionar seguinte, embora nada impede que o diretório de instalação seja alterado.

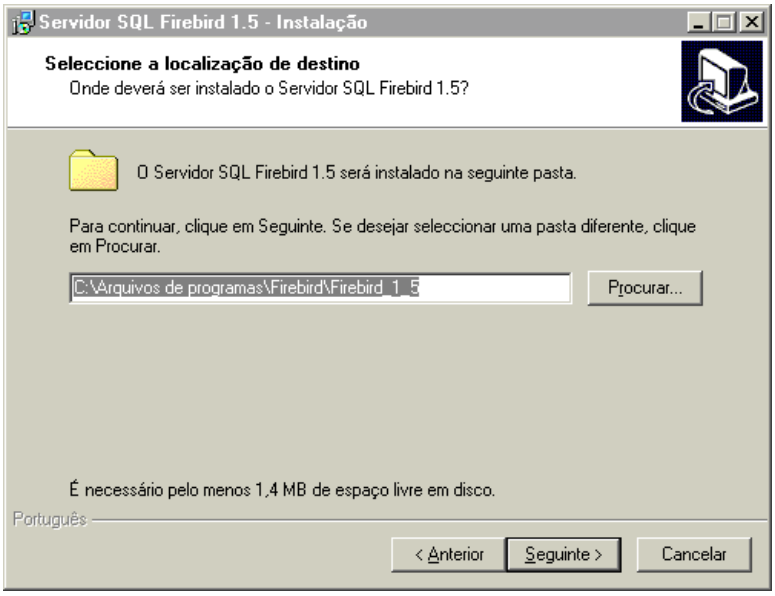

Figura 64 – Localização do Destino

Na próxima tela devem ser selecionados os componentes a serem instalados e que a seleção atual ocupará pelo menos 9.1 Mb de espaço em disco (Figura 65). Nesse caso foram aceitas as sugestões clicando em seguinte.

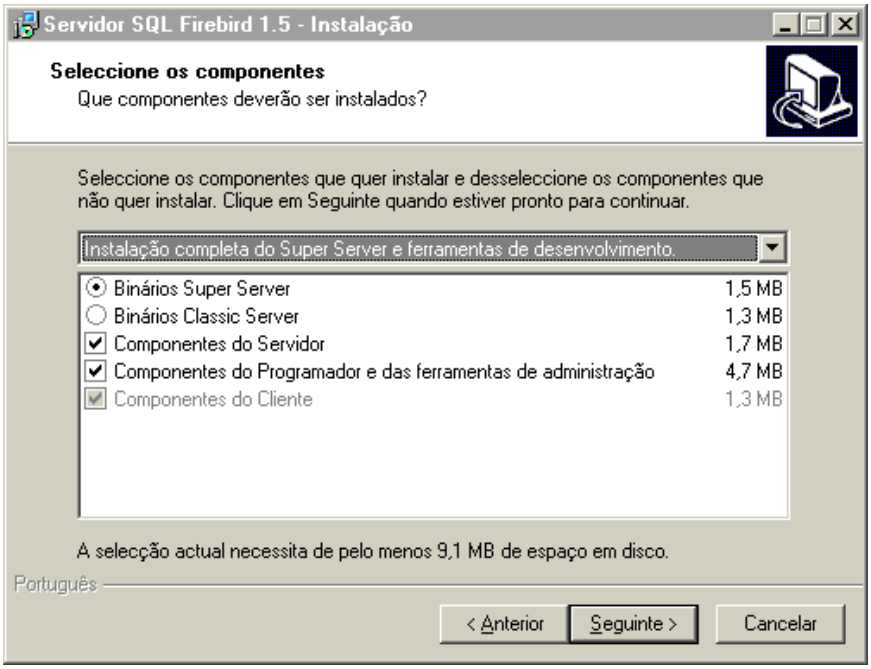

Figura 65 – Seleção dos Componentes

Em seqüência ele mostra a pasta do Menu Iniciar onde serão colocados os ícones de atalho do programa (Figura 66), essa pasta poderá ser alterada ou selecionar a opção não criar ícones e depois deve ser selecionado seguinte.

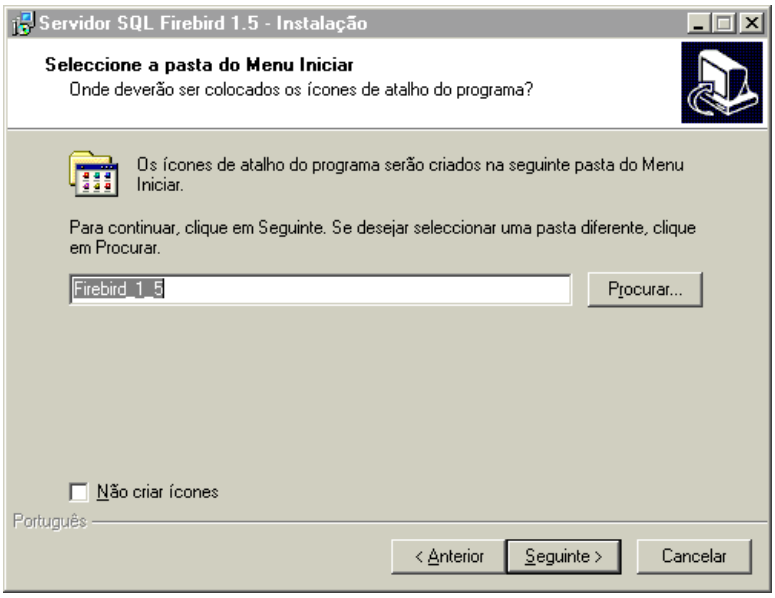

Figura 66 – Seleção da Pasta do Menu Iniciar

Agora é pedido quais as tarefas adicionais serão executadas (Figura 67), selecione as de sua opção e deve ser selecionado seguinte, nesse trabalho foram selecionadas todas as opções. É importante salientar que a opção 4: “Copiar a biblioteca do cliente Firebird para a pasta de <sistema>?” deve ser selecionada, caso contrário deverá ser feito manualmente, o mesmo ocorre com as outras opções. A biblioteca cliente é o arquivo fbclient.dll, que será abordado logo a frente.

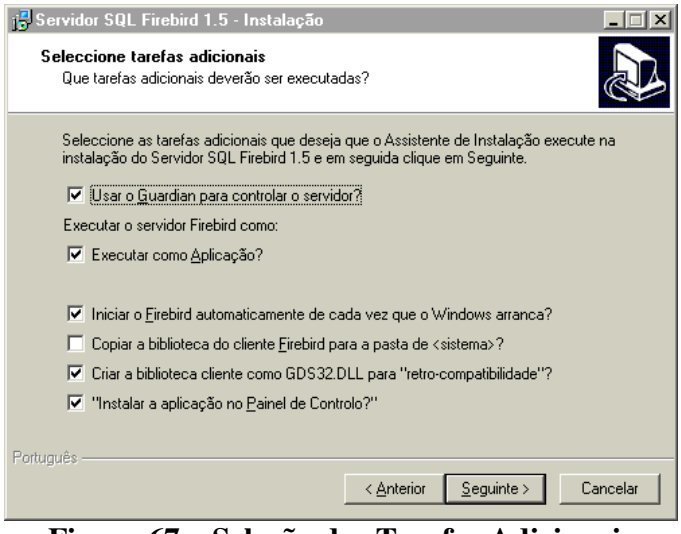

Figura 67 – Seleção das Tarefas Adicionais

A próxima tela confirma que todos os itens selecionados serão instalados e em qual diretório será, deve ser selecionado instalar (Figura 68).

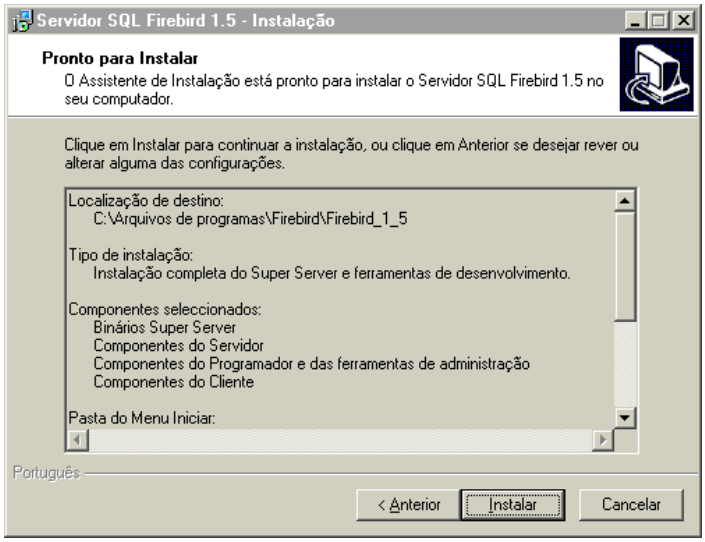

Figura 68 – Pronto para Instalar

Em poucos segundos a instalação é completada e apresenta uma tela com informações (Figura 69), leia e deve ser selecionado seguinte.

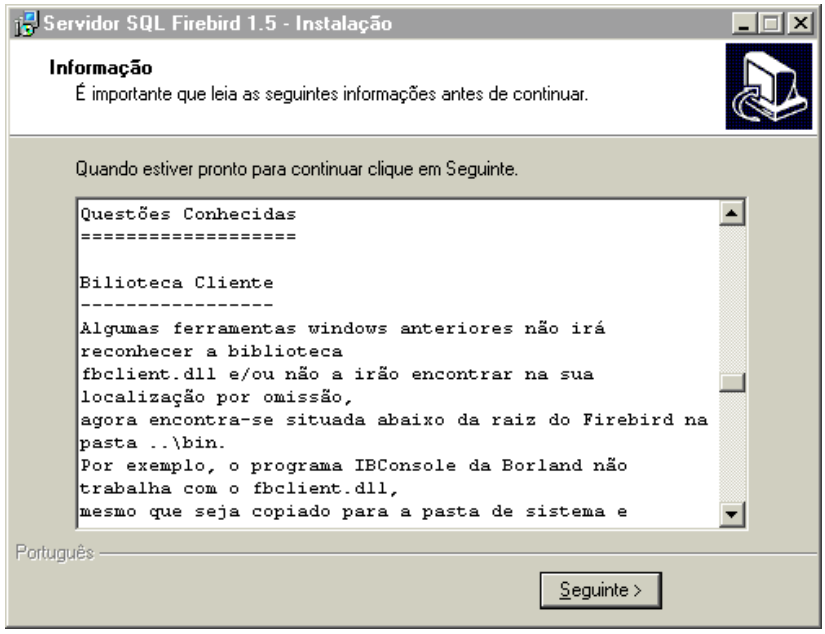

Figura 69 – Outras Informações

Para finalizar é apresentada uma tela dizendo que a instalação está concluída (Figura 70), deve ser selecionado Concluir.

Figura 70 – Instalação Concluída

Agora deve-se instalar os Drives do banco de dados Firebird 1.5, para isso deve-se pegar o arquivo FirebirdSQL-1.5.0RC2JDK_1.4 de 2.1 MB, que está compactado, copiá-lo para um diretório qualquer e descompactá-lo usando um programa apropriado como o Winzip ou o Winrar. Após o arquivo ser descompactado pegar todos os arquivos que estão fora dos diretórios (Figura 71) copiá-los e colá-los no diretório Bin do Java, por exemplo: C:\Arquivos de programas\Java\jdk1.5.0_09\bin, e os arquivos que estão dentro do diretório lib devem ser colados no diretório Lib do Java, por exemplo: C:\Arquivos de programas\ Java\jdk1.5.0_09\lib (Figura 72).

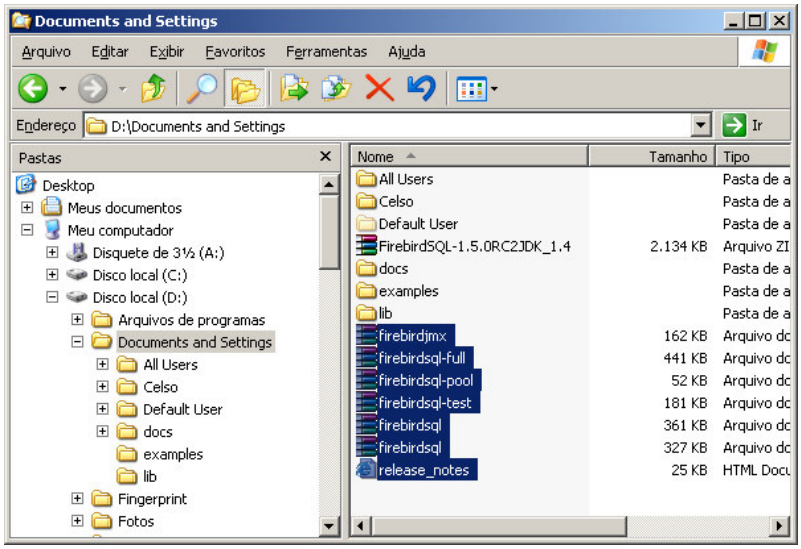

Figura 71 – Explorer – arquivos do diretório bin

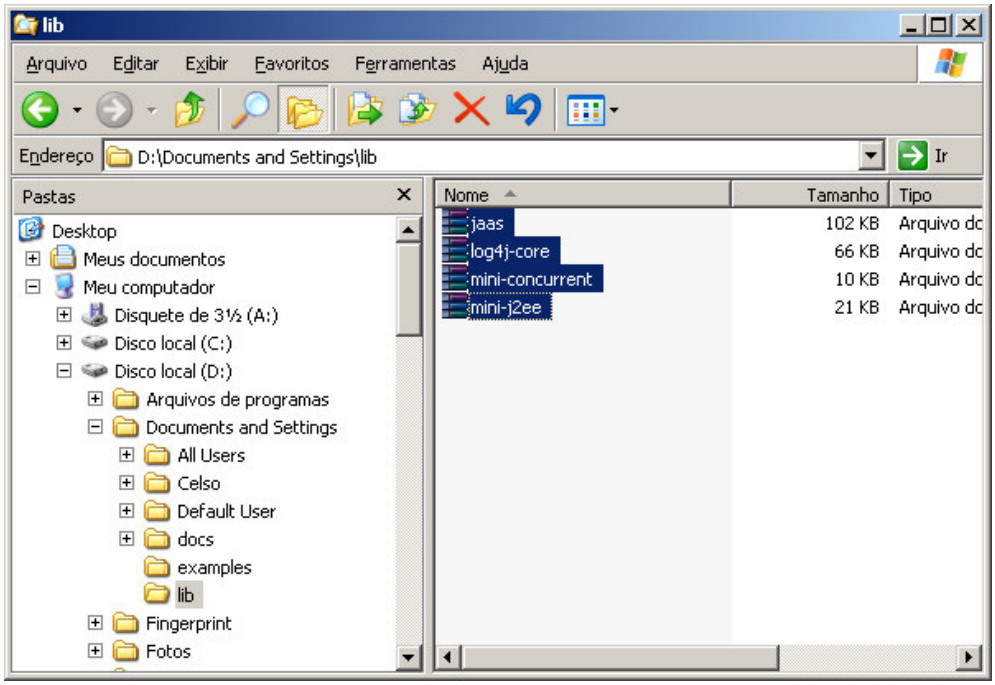

Figura 72 – Explorer – arquivos do diretório lib

Cabe ressaltar que cada versão tem o seu drive, o drive FirebirdSQL-1.5.0RC2JDK_1.4 só serve para o Firebird 1.5 com o Java 1.4 ou 1.5 se o Firebird ou o Java forem de versões diferentes dessas deve-se instalar o drive apropriado.

A próxima providência a ser tomada é copiar o arquivo fbclient.dll que está no diretório C:\Arquivos de programas\Firebird\Firebird_1_5\bin para o diretório BIN do Java, por exemplo: C:\Arquivos de programas\Java\jdk1.5.0_09\bin e também para o diretório onde está o seu banco de dados, por exemplo: C:\TCC. Feito isso, deve-se setar as Variáveis de ambiente do Windows abrindo o menu Iniciar – Configurações – Painel de Controle – ícone Sistema – orelha Avançado (Figura 73) – botão Variáveis de ambiente (Figura 74). Nas Variáveis do Sistema, no item Classpath adiciona-se os seguintes caminhos: C:\TCC\fbclient.dll, C:\Arquivos de programas\Java\jdk1.5.0_09\bin\fbclient.dll, C:\Arquivos de programas\Java\jdk1.5.0_09\lib\ fbclient.dll.

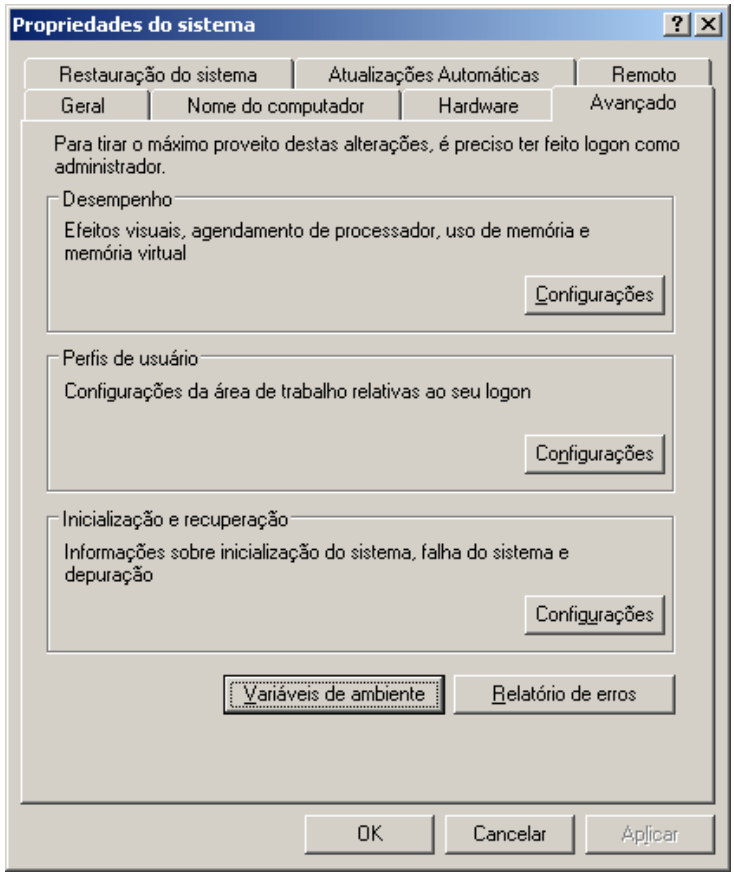

Figura 73 – Propriedades do Sistema

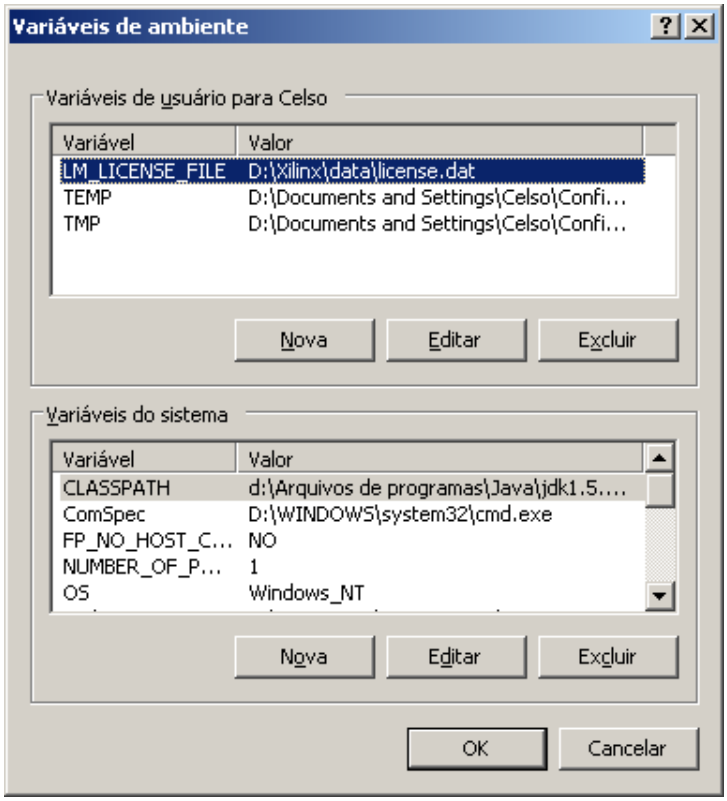

Figura 74 – Variáveis de Ambiente

Apêndice C – Instalação do Ibexpert 2005.06.07

Para instalar o Ibexpert 2005.06.07 da HK-Software que é o editor do Banco de dados Firebird 1.5.2.4731, deve-se executar o arquivo ibep_2005.6.7.1_full de 3.8 Mb que pode ser acessado pelo site www.ipexpert.com. Quando executa-se este arquivo na primeira tela que se abre (Figura 75) recomenda-se fechar todos os programas para depois continuar o setup acionando em *next*.

Figura 75 – Tela inicial - IBExpert

Em seguida é exibida a tela da licença (Figura 76), que após a leitura deve-se selecionar o item "*I accept the agreement*" e acionar *next*. Continuando mostra-se o diretório onde o programa será instalado (Figura 77): C:\Arquivos de programas\HK-Software\IBExpert, é possível alterar o diretório, mas se recomenda instalar no diretório *default* (padrão), e ainda alerta que ele requer 14,9 Mb de espaço no disco, deve ser selecionado *next* para continuar.

Figura 76 – License Agreement - IBExpert

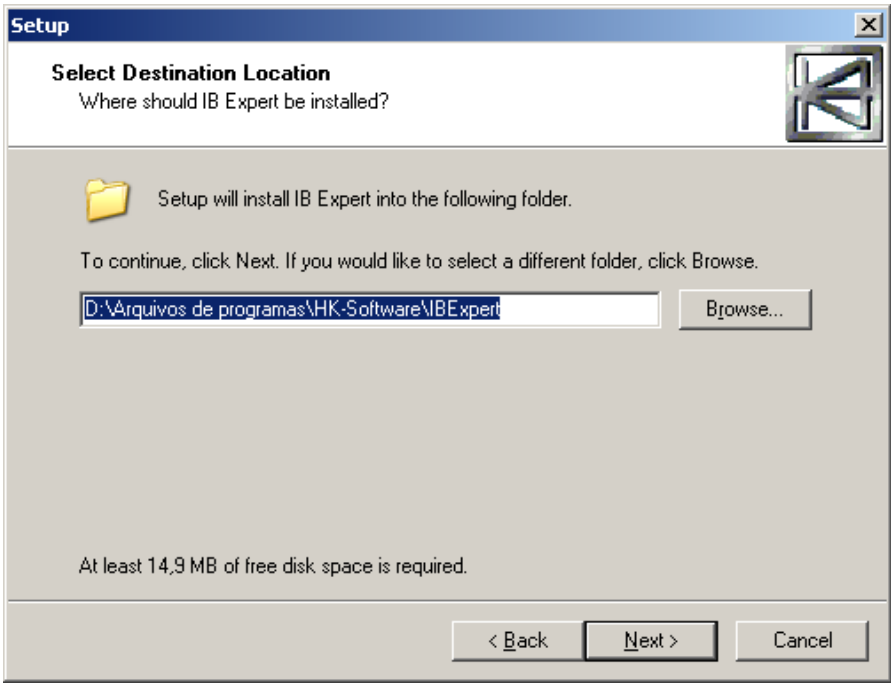

Figura 77 – Seleção do Diretório - IBExpert

Em ato contínuo ele pede a forma que será colocada no Menu Iniciar Programas (Figura 78), se pode alterar ou deixar o que ele sugere “HK-Software\IBExpert”, deve ser selecionado *next* para continuar.

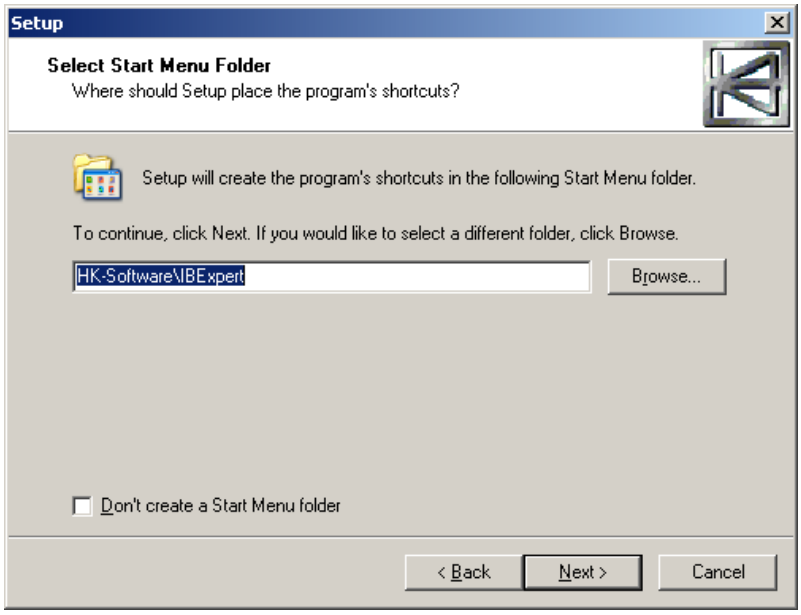

Figura 78 – Seleção do Menu Iniciar - IBExpert

Após são feitas duas perguntas, como vê-se na Figura 79: 1) se quer adicionar um ícone no desktop 2) se quer adicionar um ícone no *Quick Launch* (ícones que ficam ao lado do menu inicia), deve ser selecionado *next* para continuar.

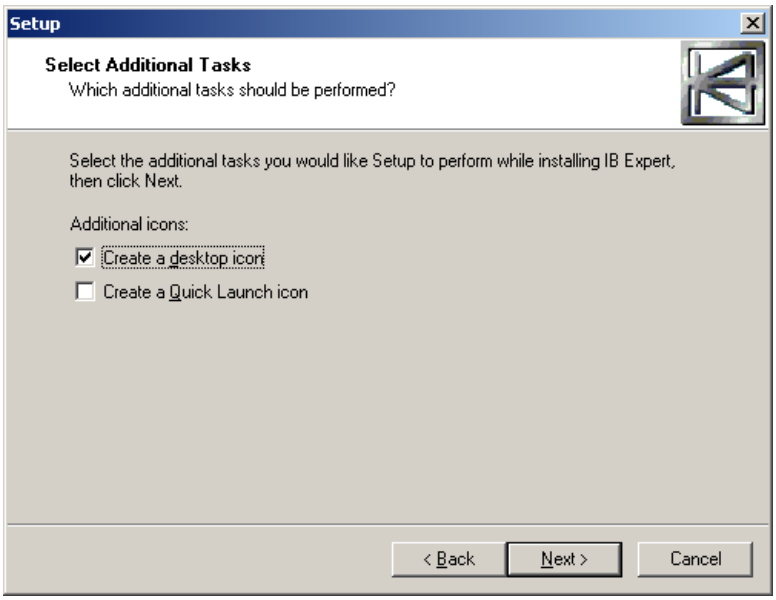

Figura 79 – Seleção Adicional - IBExpert

Antes de começar a instalar o programa é mostrado o diretório onde será instalado, o caminho do menu iniciar programas e os ícones que serão instalados (Figura 80), se estiver tudo *ok* deve ser selecionado *Install* para começar a instalação.

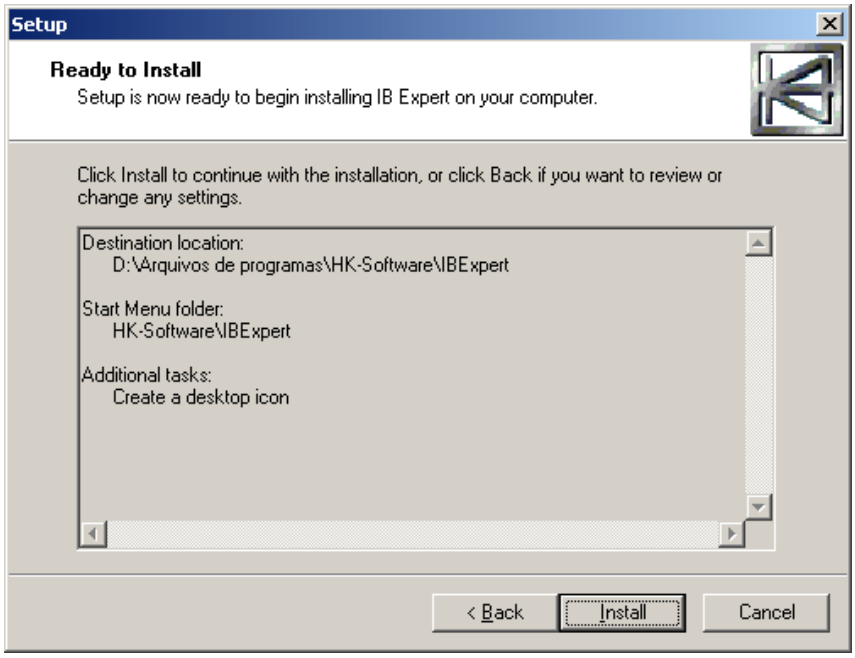

Figura 80 – Confirmação das Opções Selecionadas - IBExpert

Na seqüência, ele mostra uma tela (Figura 81) dizendo que a instalação foi completada e se quiser iniciar o programa deve ser selecionado *Launch IB Expert*, deve ser selecionado *finish* para finalizar a instalação.

Figura 81 – Tela Final - IBExpert

Anexo A – Código Fonte da Classe Abertura

```
import java.lang.Object;
import javax.swing.JComponent;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.event.*;
import java.awt.*;
import java.io.*;
import java.io.File;
import javax.media.jai.*;
import com.sun.media.jai.widget.DisplayJAI;
import java.util.*;
import java.io.*;
import java.awt.*;
import java.lang.*;
import java.sql.*;
import java.applet.Applet;

public class Abertura extends JFrame implements ActionListener
{
  private JButton bt1,bt2,bt3,bt4,bt5;
  private JPanel pn,pn1,pn2;
  private FlowLayout fl;
  private Image img;

  Abertura()
  {  //criacao do layout
     setLayout(null);
     setBackground(new Color(238, 238, 238));
     setSize(700, 250);
     setResizable(false);
     setTitle("Banco de Dados de Impressão Digital");
     bt1 = new JButton("Manutenção de Impressão Digital");
     bt2 = new JButton("Comparação de Impressão Digital");
     bt3 = new JButton("Sair");
      bt4 = new JButton("              @                 ");
      bt5 = new JButton("              @                 ");
     pn = new JPanel();
     pn1 = new JPanel();
      pn2 = new JPanel();

     fl = new FlowLayout();
     BorderLayout lay = new BorderLayout();
     getContentPane().setLayout(lay);

     getContentPane().add(pn1,BorderLayout.WEST);
     getContentPane().add(pn,BorderLayout.CENTER);
     getContentPane().add(pn2,BorderLayout.EAST);
     pn.setLayout(new GridLayout(3,1));
     pn1.setLayout(new GridLayout(1,1));
     pn2.setLayout(new GridLayout(1,1));

     pn.add(bt1);
     pn.add(bt2);
       pn.add(bt3);
       pn1.add(bt4);
       pn2.add(bt5);
```

```
     bt1.addActionListener(this);
     bt2.addActionListener(this);
       bt3.addActionListener(this);

  } //fim da abertura

  public void Banco()
  {
      try
      {
            BancoFingerPrint janela1 = new BancoFingerPrint();
            janela1.setVisible(true);
            janela1.setLocation(50,200);
      }
      catch(Exception e)
      {   System.out.println("   Nao   foi   possivel   abrir   a   classe:
BancoFingerPrint()" + e);
      }
  }

  public void Comparacao()
  {
            Comparacao janela2 = new Comparacao();
            janela2.setVisible(true);
            janela2.setLocation(5,5);
  }

      //tratando os eventos dos botoes
  public void actionPerformed(java.awt.event.ActionEvent e)
  {
        if(e.getSource() == bt1)
        {
              Banco();
        }
        else
        if(e.getSource() == bt2)
        {
            Comparacao();
        }
        if(e.getSource() == bt3)
        {
                 System.out.println("A  TELA  DE  ABERTURA  FOI  FECHADA  COM
SUCESSO!\n");
              System.out.println("FIM DO PROGRAMA.");
              System.exit(0);
        }
  }

  public static void main(String[] arg) throws Exception
  {
            Abertura janela = new Abertura();
            janela.setLocation(50,200);
            janela.setVisible(true);
            System.out.println("A TELA DE ABERTURA FOI INICIADA...\n");
            System.out.println();
  }// fim do main

} // fim da classe
```

Anexo B – Código Fonte da Classe BancoFingerPrint

```
import java.lang.Object;
import javax.swing.JComponent;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.event.*;
import java.awt.*;
import java.io.*;
import java.io.File;
import javax.media.jai.*;
import com.sun.media.jai.widget.DisplayJAI;
import java.util.*;
import java.io.*;
import java.awt.*;
import java.lang.*;
import java.sql.*;

public class BancoFingerPrint extends JFrame implements ActionListener
{
  private  JButton  bt1,bt2,bt3,bt4,bt5,bt6,bt7,bt8,bt9,bt10,bt11,bt12,bt13,
bteste;
  private JPanel pn,pn1,pn2,janelinha,pn3;
  private JTextField tf,tf1,tf2,tf3;
  private FlowLayout fl;
  private String[][] registro = new String[100][4];
  private int indice=0,indic=1,w;
  private int cod=0,nome=1,cpf=2,imagem=3;
  private PlanarImage imagens = null;
  private String sql, auxcodString;
  public int auxcod = 0,cont = 0;
  public static Connection conexao1 =  null;
  public Statement st;
  public ResultSet rs;
  public ImageIcon previa = null;

  BancoFingerPrint()
  {
      try
      {
            conexao1 = Conexao.InicioConexao();
            st = conexao1.createStatement();
            rs = null;
      }
      catch(Exception e)
      { System.out.println(" Erro na conexao..." + e);
      }
        //criacao do layout
     setLayout(null);
     setBackground(new Color(238, 238, 238));
     setSize(700, 250);
     setResizable(false);
     setTitle("Registro - "+ indic);
     bt1 = new JButton("Cod");
     bt2 = new JButton("Nome");
     bt3 = new JButton("CPF");
     bt4 = new JButton("Imagem");
     bt5 = new JButton("Incluir");
```

```
      bt6 = new JButton("Excluir");
      bt7 = new JButton("Editar");
        bt11 = new JButton("Consulta");
        bt12 = new JButton("|<");
      bt8 = new JButton("<<");
      bt9 = new JButton(">>");
        bt13 = new JButton(">|");
      bt10 = new JButton("Sair");
      tf = new JTextField("");
      tf1 = new JTextField("");
      tf2 = new JTextField("");
      tf3 = new JTextField("");
      pn = new JPanel();
      pn1 = new JPanel();
      pn2 = new JPanel();
      pn3 = new JPanel();
      janelinha = new JPanel();
      janelinha.setBorder(BorderFactory.createEtchedBorder());

      fl = new FlowLayout();
      BorderLayout lay = new BorderLayout();
      getContentPane().setLayout(lay);

      getContentPane().add(pn,BorderLayout.WEST);
      getContentPane().add(pn1,BorderLayout.CENTER);
      getContentPane().add(pn3,BorderLayout.EAST);
      getContentPane().add(pn2,BorderLayout.SOUTH);
      pn.setLayout(new GridLayout(4,1));
      pn1.setLayout(new GridLayout(4,1));
      pn3.setLayout(new GridLayout(1,1));
      pn2.setLayout(fl);
      //trata o botao para a localizacao da imagem
      bt4.addActionListener (new ActionListener()
        {
       public void actionPerformed(ActionEvent e)
             {
                   // Chamada ao Procedimento
                   abrirArquivo();
                   System.out.println("1\n");
       }
        }    );

      pn.add(bt1);
      pn.add(bt2);
      pn.add(bt3);
      pn.add(bt4);
      pn1.add(tf);
      pn1.add(tf1);
      pn1.add(tf2);
      pn1.add(tf3);
      pn3.add(janelinha);
      pn2.add(bt5);
      pn2.add(bt6);
      pn2.add(bt7);
      pn2.add(bt11);
      pn2.add(bt12);
      pn2.add(bt8);
      pn2.add(bt9);
      pn2.add(bt13);
      pn2.add(bt10);
      bt5.addActionListener(this);
```

```
     bt6.addActionListener(this);
     bt7.addActionListener(this);
       bt11.addActionListener(this);
       bt12.addActionListener(this);
     bt8.addActionListener(this);
     bt9.addActionListener(this);
       bt13.addActionListener(this);
     bt10.addActionListener(this);
      //zera todo o registrador
       for(int i=0;i<100;i++)
           for(int j=0;j<4;j++)
               registro[i][j] = "";
     iniciar();
  } //fim do metodo

  public void iniciar()
  {               //carrega todo o banco de dados para o registro e mostra
o primeiro
                  sql = "SELECT * FROM IDIGITAL ORDER BY COD";
                  indice=0;
                  cont = 0;
                  try
                  {
                        rs = st.executeQuery(sql);
                        while (rs.next())
                        {
                              registro[indice][cod] = rs.getString("COD");
                              registro[indice][nome] = rs.getString("NOME");
                              registro[indice][cpf] = rs.getString("cpf");
                              registro[indice++][imagem]                  =
rs.getString("IMAGEM");
                              cont++;
                        }
                  }
                  catch(SQLException e)
                  {
                        System.out.println("ERRO NA BUSCA : " + e);
                  }

                  indice=0;
             tf.setText(registro[indice][cod]);
             tf1.setText(registro[indice][nome]);
             tf2.setText(registro[indice][cpf]);
             tf3.setText(registro[indice][imagem]);
                  set_title("set");
             tf_set_enable(false);
             bt_verificar();
  }
  //habilita ou desabilita os botoes
  public void bt_verificar()
  {
      if(registro[indice][cod] == "")
      {
          bt6.setEnabled(false);
          bt7.setEnabled(false);
          bt9.setEnabled(false);
      }
      else
      {
          bt6.setEnabled(true);
          bt7.setEnabled(true);
```

```
          bt8.setEnabled(true);
          bt9.setEnabled(true);
      }
  }
  //habilita ou desabilita os campos
  public void tf_set_enable(boolean x)
  {
     if(x)
     {
        tf1.setEnabled(true);
        tf2.setEnabled(true);
        tf3.setEnabled(true);
     }
     else
     {
        tf1.setEnabled(false);
        tf2.setEnabled(false);
        tf3.setEnabled(false);
     }
  }
  //trata o cabelhaco da tela incrementa ou decrementa o contador
  public void set_title(String x)
  {
      if(x == "set")
      {
           indic = indice + 1;
           setTitle("Registro - "+ indic);
      }
      else
      if(x == "++")
      {
           indic++;
           setTitle("Registro - "+ indic);
      }
      else
      if(x == "--")
      {
           indic--;
           setTitle("Registro - "+ indic);
      }
  }
  // metodo incluir, para incluir os dados no banco e no registro
  public void incluir_salvar()
  {
       if(bt5.getLabel() == "Salvar")
       {
            for(int i=0;i<100;i++)
              if(registro[i][cod] == "")
                    {      w = cont -1;
                           auxcod = Integer.parseInt(registro[w][cod]);
                           auxcod = auxcod + 1;
                           auxcodString = (new Integer(auxcod)).toString();
                           registro[i][cod] = auxcodString;
                  registro[i][nome] = tf1.getText();
                  registro[i][cpf] = tf2.getText();
                  registro[i][imagem] = tf3.getText();
                           sql = null;
                           sql = "INSERT INTO IDIGITAL VALUES (";
                                     sql     =     sql     +     auxcod     +
",'"+registro[i][nome]+"','"+registro[i][cpf]+"','"+registro[i][imagem]+"')
";
```

```
                            try
                            {
                               st.executeUpdate(sql);
                               conexao1.commit();
                                 cont++;
                                 iniciar();
                            }catch(SQLException e)
                            {
                               iniciar();
                                 registro[cont][cod] = "";
                               registro[cont][nome] = "";
                                 registro[cont][cpf] = "";
                               registro[cont][imagem] = "";
                                 System.out.println("ERRO AO GRAVAR DADOS : "
+ e); }
                     break;
                  }
                     bt5.setLabel("Incluir");
              tf_set_enable(false);
                     janelinha.removeAll();
                     janelinha.updateUI();
        }
        else
        {
                limpar();
                bt5.setLabel("Salvar");
                tf_set_enable(true);
        }
  }
  // metodo excluir, para excluir os dados no banco e no registro
  public void excluir()
  {
        auxcod = Integer.parseInt(registro[indice][cod]);
        sql = "DELETE FROM IDIGITAL WHERE cod = " + auxcod;
        try
        {
            st.executeUpdate(sql);
          conexao1.commit();
            cont--;
        }
        catch(SQLException e)
        {
            System.out.println();
          System.out.println("ERRO AO EXCLUIR : " + e);
        }
      registro[indice][cod] = "";
      registro[indice][nome] = "";
        registro[indice][cpf] = "";
      registro[indice][imagem] = "";
      registro[cont][cod] = "";
      registro[cont][nome] = "";
        registro[cont][cpf] = "";
      registro[cont][imagem] = "";
        bt8.setEnabled(false);
        iniciar();

  }
  // metodo editar, para alterar os dados no banco e no registro
  public void editar()
  {
        bt5.setEnabled(false);
```

```
      bt6.setEnabled(false);
        bt8.setEnabled(false);
      bt9.setEnabled(false);
        bt10.setEnabled(false);

      if(bt7.getLabel() == "Salvar")
       {
           registro[indice][nome] = tf1.getText();
           registro[indice][cpf] = tf2.getText();
           registro[indice][imagem] = tf3.getText();
               auxcod = Integer.parseInt(registro[indice][cod]);
                        sql    =    "UPDATE    IDIGITAL    SET    NOME    =
'"+registro[indice][nome]+"', cpf = '"+registro[indice][cpf]+
                                                          "',     IMAGEM     =
'"+registro[indice][imagem]+"' WHERE COD = " + auxcod;
               try
               {   st.executeUpdate(sql);
                     conexao1.commit();
               }catch(SQLException e)
               {   System.out.println();
                     System.out.println("ERRO NO COMANDO UPDATE : " + e);
                }

            registro[indice][nome] = tf1.getText();
            registro[indice][cpf] = tf2.getText();
            registro[indice][imagem] = tf3.getText();
            bt7.setLabel("Editar");
            janelinha.removeAll();
                  janelinha.updateUI();
            tf_set_enable(false);
                  bt5.setEnabled(true);
                  bt6.setEnabled(true);
                  bt8.setEnabled(true);
                  bt9.setEnabled(true);
                  bt10.setEnabled(true);
       }
       else
       {
            bt7.setLabel("Salvar");
                  tf1.setEnabled(true);
            tf2.setEnabled(true);
            tf3.setEnabled(true);
       }
  }
  // metodo consultar, para consultar os dados de uma pessoa, pode ser pelo
codigo ou pelo nome
  public void consultar()
  {
        int op;
        String resultado;
        bt5.setEnabled(false);
      bt6.setEnabled(false);
        bt7.setEnabled(false);
        bt8.setEnabled(false);
      bt9.setEnabled(false);
        bt10.setEnabled(false);
        bt12.setEnabled(false);
        bt13.setEnabled(false);

           resultado  =  JOptionPane.showInputDialog("Digite:  \n"  +  "1  -
Para consulta por código.\n" +
```

```
                                "2 - Para consulta por nome.\n" );
            op = Integer.parseInt(resultado);

            if (op == 1)
            {
                  resultado    =    JOptionPane.showInputDialog("Digite    o
código:");
                  auxcod = Integer.parseInt(resultado);
                  sql = "SELECT * FROM IDIGITAL WHERE cod = "+ auxcod;
            }
            else
            {
                  resultado    =    JOptionPane.showInputDialog("Digite    o
nome:");
                  sql   =   "SELECT   *   FROM   IDIGITAL   WHERE   NOME   =   '"+
resultado+"'";
            }
            try
            {
                  rs = st.executeQuery(sql);
                  rs.next();

                  String aux = "CÓDIGO : " + rs.getString("COD") + "\nNOME
: " + rs.getString("NOME") +
                                          "\ncpf   :   "   +   rs.getString("cpf")   +
"\nIMAGEM : " + rs.getString("IMAGEM");

                  JOptionPane.showMessageDialog(null,aux,"Resulta         da
Pesquisa",JOptionPane.INFORMATION_MESSAGE);
            }
            catch(SQLException e)
            {
                  System.out.println("ERRO NA BUSCA : " + e);
            }
          tf_set_enable(false);
            bt5.setEnabled(true);
            bt6.setEnabled(true);
            bt7.setEnabled(true);
            bt8.setEnabled(true);
            bt9.setEnabled(true);
            bt10.setEnabled(true);
            bt12.setEnabled(true);
            bt13.setEnabled(true);
  }

  // metodo voltar, para retroceder os dados no registro
  public void voltar()
  {
        if(indice == 0)
        {
            indice = 0;
            set_title("set");
            JOptionPane.showMessageDialog(null, "Inicio do registro");
        }
        else
        if(indice > 0)
        {
            indice--;
            tf.setText(registro[indice][cod]);
            tf1.setText(registro[indice][nome]);
            tf2.setText(registro[indice][cpf]);
```

```
            tf3.setText(registro[indice][imagem]);
            set_title("--");
        }
  }
  // metodo avancar, para avancar os dados no registro
  public void avancar()
  {
     if(indice == 99)
     {
      indice = 99;
        set_title("set");
        JOptionPane.showMessageDialog(null, "Fim do registro");
     }
     else
     if(indice < 99)
     {
      indice++;
        tf.setText(registro[indice][cod]);
        tf1.setText(registro[indice][nome]);
        tf2.setText(registro[indice][cpf]);
        tf3.setText(registro[indice][imagem]);
        set_title("++");
     }
  }
  // metodo inicio, para carregar os dados do primeiro registro, quando
acionado o botao inicio
  public void inicio()
  {

      indice = 0;
        tf.setText(registro[indice][cod]);
        tf1.setText(registro[indice][nome]);
        tf2.setText(registro[indice][cpf]);
        tf3.setText(registro[indice][imagem]);
            set_title("set");

  }
  // metodo fim, para carregar os dados do ultimo registro, quando acionado
o botao fim
  public void fim()
  {
      indice = cont - 1;
        tf.setText(registro[indice][cod]);
        tf1.setText(registro[indice][nome]);
        tf2.setText(registro[indice][cpf]);
        tf3.setText(registro[indice][imagem]);
            set_title("set");
  }
  // metodo limpar, para limpar a tela
  public void limpar()
  {
      tf.setText("");
      tf1.setText("");
      tf2.setText("");
      tf3.setText("");
      janelinha.removeAll();
      janelinha.updateUI();
  }
  // metodo actionPerformed, trata os acionamentos dos botoes
  public void actionPerformed(java.awt.event.ActionEvent e)
  {
```

```
   if(e.getSource() == bt5)
   {
         incluir_salvar();
       bt_verificar();
   }
   else
   if(e.getSource() == bt6)
   {
       excluir();
       limpar();

   }
   else
   if(e.getSource() == bt7)
   {
       editar();
   }
   else
   if(e.getSource() == bt8)
   {
       voltar();
       bt_verificar();
   }
   else
   if(e.getSource() == bt9)
   {
       avancar();
       bt_verificar();
   }
   else
   if(e.getSource() == bt11)
   {
       fim();
         avancar();
         //indice++;
       consultar();
       bt_verificar();
   }
   else
   if(e.getSource() == bt12)
   {
       inicio();
       bt_verificar();
   }
   else
   if(e.getSource() == bt13)
   {
       fim();
       bt_verificar();
   }
   else
   if(e.getSource() == bt10)
   {
                Conexao.FimConexao();
                setVisible(false);
   }
}
 //metodo abrirArquivo, para localizar o caminho da impressão digital
 public void abrirArquivo()
 {
        //exibe o dialogo para o usuario escolher o arquivo a ser aberto
```

```
        JFileChooser arquivoEscolhido = new JFileChooser();
        arquivoEscolhido.setFileSelectionMode(JFileChooser.FILES_ONLY);
        arquivoEscolhido.setAcceptAllFileFilterUsed(true);
        arquivoEscolhido.setFileFilter(new FiltroDeImagem());
                                        arquivoEscolhido.setAccessory(new
PreVisualizacao(arquivoEscolhido));
        int resultado = arquivoEscolhido.showOpenDialog(null);
        // usuario clicou no botao cancel do dialogo
        if ( resultado == JFileChooser.CANCEL_OPTION )
            return;
        // obtem o arquivo selecionado
        File nomeArquivo = arquivoEscolhido.getSelectedFile();
        tf3.setText(nomeArquivo.getPath());
        String arq = nomeArquivo.getPath();
        // mostra erro se nome do arq for invalido
        if (nomeArquivo == null || nomeArquivo.getName().equals(""))
            JOptionPane.showMessageDialog(null,"Nome       de       Arquivo
Inválido","Nome de Arquivo Inválido",JOptionPane.ERROR_MESSAGE);
       else
       {
            //carregaArquivo(arq);
                  //Mostrafigura(arquivoEscolhido);

         }
  }
  //metodo carregaArquivo
  public void carregaArquivo(String arq)
  {

      FlowLayout layout = new FlowLayout(FlowLayout.LEFT);
      setPreferredSize(new Dimension(100, 50));
      imagens = JAI.create("fileload",arq);
      DisplayJAI img = new DisplayJAI(imagens);
      //JFileChooser arq1 = new JFileChooser();
      layout.setHgap(-1);
      layout.setVgap(-1);
      janelinha.setLayout(layout);
      janelinha.add(img);
      janelinha.updateUI();
  }

} // fim da classe
```

Anexo C – Código Fonte da Classe Comparacao

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.lang.Exception;
import javax.swing.JOptionPane;
import java.sql.*;
import java.awt.FlowLayout;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JComboBox;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.AbstractButton;

public class Comparacao extends JFrame
{
      private    JLabel    codigoImpressao,    nomeUsuario,    cpfUsuario,
imagemImpressao;
      private JTextField codCampo, nomeCampo, cpfCampo, imagemCampo;
      public static Connection conexao1 =  null;
      public Statement st = null;
      public ResultSet rs = null;
      private String[][] registro = new String[100][4];
      private int indice=0;
      private int cod=0,nome=1,cpf=2,imagem=3;
      private String sql;
      private FlowLayout fl;
      int auxcod = 0, cont = 0;
      private JComboBox imagesComboBox;
    private JLabel label;
      private String names[] = new String[100];
      private JPanel pn,pn1,pn2,pn3;
      private Icon icons[] = new Icon[100];
      public int nCasamento[] = new int[100];
      public int auxCasamento[] = new int[100];
      public int tamanho1;
      public int vetorAux[][] = new int[100][2];
      public int selecaoAtual;
      public int auxCod;
      public JButton buscarImagem;
      public JButton sairJanela;

      public Comparacao()
      {     //setLayout(null);
            //criacao do layout
          setTitle("Comparacao");
            setSize( 400, 500 );
            setBackground(new Color(238, 238, 238));
/*        pn = new JPanel();
          pn1 = new JPanel();
            pn2 = new JPanel();
```

```
            pn3 = new JPanel();

      getContentPane().add(pn,BorderLayout.NORTH);
      getContentPane().add(pn1,BorderLayout.WEST);
      getContentPane().add(pn2,BorderLayout.EAST);
            getContentPane().add(pn3,BorderLayout.SOUTH);
      pn.setLayout(new GridLayout(1,1));
      pn1.setLayout(new GridLayout(4,1));
      pn2.setLayout(new GridLayout(4,1));
            pn3.setLayout(new GridLayout(1,1));


             pn.add(buscarImagem);
           pn.add(sairJanela);
             pn1.add(codigoImpressao);
             pn1.add(nomeUsuario);
             pn1.add(cpfUsuario);
             pn1.add(imagemImpressao);
             pn2.add(codCampo);
             pn2.add(nomeCampo);
             pn2.add(cpfCampo);
             pn2.add(imagemCampo);
             pn3.add(label);*/

           //buscarImagem.addActionListener(this);
           //sairJanela.addActionListener(this);

            Container container = getContentPane();
        container.setLayout( new FlowLayout() );

            try
            {     //faz a conexao
                  conexao1 = Conexao.InicioConexao();
                  st = conexao1.createStatement();
                  rs = null;
            }
            catch(Exception e)
            { System.out.println(" Erro na conexao..." + e);
            }

            //Container container = getContentPane();
            //container.setLayout(null);

            /* ComboBox */
            /*JComboBox application = new JComboBox(names);
            application.setMaximumRowCount( 5 );
            JPanel p = new JPanel( );
            p.add(application);
            getContentPane( ).add(p, "South");
            p.setBounds(new Rectangle(10, 20, 200, 46));*/


           /* fl = new FlowLayout();
            BorderLayout lay = new BorderLayout();
            getContentPane().setLayout(lay);*/


            /* Botões */
            JButton buscarImagem = new JButton("Buscar Imagem");
            container.add(buscarImagem);
            buscarImagem.setBounds(new Rectangle(2, 515, 187, 46));
```

```
            JButton sairJanela = new JButton("Sair");
            container.add(sairJanela);
            sairJanela.setBounds(new Rectangle(600, 515, 187, 46));

            /* Labels */
            codigoImpressao = new JLabel("Código");
            container.add(codigoImpressao);
            codigoImpressao.setBounds(new Rectangle(30, 78, 94, 16));

            nomeUsuario = new JLabel("Nome");
            container.add(nomeUsuario);
            nomeUsuario.setBounds(new Rectangle(30, 108, 94, 16));

            cpfUsuario = new JLabel("cpf");
            container.add(cpfUsuario);
            cpfUsuario.setBounds(new Rectangle(30, 138, 94, 16));

            imagemImpressao = new JLabel("Imagem");
            container.add(imagemImpressao);
            imagemImpressao.setBounds(new Rectangle(30, 168, 94 ,16));

            /* TextFields */
            codCampo = new JTextField();
            container.add(codCampo);
            codCampo.setBounds(new Rectangle(140, 78, 60, 20));

            nomeCampo = new JTextField();
            container.add(nomeCampo);
            nomeCampo.setBounds(new Rectangle(140, 108, 500, 20));

            cpfCampo = new JTextField();
          container.add(cpfCampo);
            cpfCampo.setBounds(new Rectangle(140, 138, 150, 20));

            imagemCampo = new JTextField();
            container.add(imagemCampo);
            imagemCampo.setBounds(new Rectangle(140, 168, 600, 20));


            /* ACTIONS */
            /* BUTTON HANDLER */
            ButtonHandler handler = new ButtonHandler(); // para tratamento
de evento dos JButtons
            buscarImagem.addActionListener( handler );
            sairJanela.addActionListener( handler );

            iniciar();

          for(int i = 0;i < cont;i++)
            {
           names[i]= registro[i][nome];
               icons[i] = new ImageIcon( registro[i][imagem]);

            }

        imagesComboBox = new JComboBox( names );

      imagesComboBox.addItemListener (

           new ItemListener()
```

```
             {
            public void itemStateChanged( ItemEvent event )
            {
                  if ( event.getStateChange() == ItemEvent.SELECTED )
                        label.setIcon(
icons[imagesComboBox.getSelectedIndex()]);

                        selecaoAtual = imagesComboBox.getSelectedIndex();


                        indice = imagesComboBox.getSelectedIndex();
                        codCampo.setText(registro[indice][cod]);
                        nomeCampo.setText(registro[indice][nome]);
                        cpfCampo.setText(registro[indice][cpf]);
                        imagemCampo.setText(registro[indice][imagem]);
            }
         }
      );

        container.add(imagesComboBox);

      label = new JLabel( icons[ 0 ] );
      container.add( label );

      } // fim comparacao()

        private class ButtonHandler implements ActionListener
            {
                  public void actionPerformed ( ActionEvent event )
            {
                        boolean avancar;
                        if (event.getActionCommand()== "Buscar Imagem")
                        {
                          int porcentagem,raio, semelhantes;
                          String resultado;
                          do {
                                                      resultado           =
JOptionPane.showInputDialog("Digite a porcentagem a ser analisada, a partir
do centro [30 a 100].\n");
                                porcentagem = Integer.parseInt(resultado);
                          }while (porcentagem < 30 || porcentagem > 100);
                          do {
                                                      resultado           =
JOptionPane.showInputDialog("Digite o raio a ser analisado, a partir do
centro [5 a 25].\n");
                                raio = Integer.parseInt(resultado);
                          }while (raio < 5 || raio > 25);

                          do {
                                                      resultado           =
JOptionPane.showInputDialog("Quantas impressões mais semelhantes você quer
ver? [1 a 5]\n");
                                semelhantes = Integer.parseInt(resultado);
                          }while (semelhantes < 1 || semelhantes > 5);

                          for(int i = 0;i < cont; i++)
                          {   System.out.println(" 1 = " + selecaoAtual);
                              System.out.println("\n 2 = " + i);

                              if(selecaoAtual != i)
                              {
```

```
                                    try
                                   {
                                          StoredFinger    tamanho    =    new
StoredFinger();
                                          tamanho1                          =
tamanho.numCasa(registro[selecaoAtual][imagem],registro[i][imagem],
porcentagem, raio);
                                   }
                                   catch(Exception e)
                                     {  System.out.println(" Erro  ao  achar  o
tamanho..." + e);
                                  }
                                     nCasamento[i]    =    auxCasamento[i]    =
tamanho1;
                               }
                               else
                               {
                                     nCasamento[i] = auxCasamento[i] = 0;
                               }
                            } // fim do for

                          System.out.println(" ordem original\n");
                          for ( int k = 0; k < cont; k++ )
                             System.out.println( auxCasamento[ k ]);

                          bubbleSort( auxCasamento );

                          System.out.println("\n ordem descendente\n");
                          for ( int s = 0; s < cont; s++ )
                             System.out.println( auxCasamento[ s ]);

                            System.out.println("\n ordem na matriz\n");
                            for ( int i = 0; i < cont; i++ )
                            {         avancar = false;
                                      for ( int j = 0; j < cont; j++ )
                                      {

                                            if       (auxCasamento[i]       ==
nCasamento[j])
                                            {
                                                  if     (((i     >     0)     &&
(auxCasamento[i] != auxCasamento[i-1])) || (avancar == true))
                                                  {
                                                        vetorAux[i][0]       =
auxCasamento[i];
                                                        vetorAux[i][1] = j;
                                                        break;
                                                  }
                                                  else
                                                  {   if (i == 0)
                                                        {
                                                              vetorAux[i][0]  =
auxCasamento[i];
                                                              vetorAux[i][1]  =
j;
                                                              break;
                                                        }
                                                        else
                                                        {     avancar = true;
                                                              continue;
                                                        }
```

```
                                                    }

                                                }
                                            }
                                    }
                                    //imprime todo a matriz vetorAux[s][0]
                                    for ( int s = 0; s < cont; s++ )
                                    {
                                       System.out.println( vetorAux[s][0]);
                                           System.out.println( vetorAux[s][1]);
                                    }

                                     Resultados janela3 = new Resultados(vetorAux,
registro, cont, semelhantes, selecaoAtual);
                                    janela3.setVisible(true);
                                    janela3.setLocation(5,5);
                                }
                                else
                                {
                                      if (event.getActionCommand()== "Sair")
                                      {
                                            Conexao.FimConexao();
                                            setVisible(false);
                                      }
                                }
                        }
                 }

                public void iniciar()
                {
                          sql = "SELECT * FROM IDIGITAL ORDER BY COD";
                          indice=0;
                          cont = 0;

                          try
                          {
                                rs = st.executeQuery(sql);

                                while (rs.next())
                                {
                                      registro[indice][cod] = rs.getString("COD");
                                      registro[indice][nome] = rs.getString("NOME");
                                      registro[indice][cpf] = rs.getString("cpf");
                                      registro[indice++][imagem]                    =
rs.getString("IMAGEM");
                                      cont++;
                                }
                          }
                          catch(SQLException e)
                          {
                                System.out.println("ERRO NA BUSCA : " + e);
                          }
             }

           public void bubbleSort( int array2[] )
           {
              for ( int pass = 1; pass < cont; pass++ )
                {
                 for ( int element = 0;element < cont - 1;element++ )
                     {
                    if ( array2[ element ] < array2[ element + 1 ] )
```

```
                ordena( array2, element, element + 1 );
            }
         }
      }

      public void ordena( int array3[], int first, int second )
      {
         int hold;

         hold = array3[ first ];
         array3[ first ] = array3[ second ];
         array3[ second ] = hold;
      }
}
```

Anexo D – Código Fonte da Classe Conexao

```
import java.sql.*;

public class Conexao
{
  public static Connection conexao =  null;
  public static Statement st;
  public static ResultSet rs;

  public static Connection InicioConexao() throws Exception
  {
            try // CONECTANDO BD
            {       String  url  =  "jdbc:firebirdsql:127.0.0.1/3050:D:/TCC
231106/DBFINGER.gdb";
                Class.forName("org.firebirdsql.jdbc.FBDriver");
                  conexao                                                 =
DriverManager.getConnection(url,"SYSDBA","masterkey");
            System.out.println("CONEXAO  COM  O  FIREBIRD  REALIZADA  COM
SUCESSO !");
                  conexao.setAutoCommit(false);
            }
            catch (SQLException e)
            {   System.out.println("ERRO NA CONEXAO COM O FIREBIRD : "+ e);
            }
            return conexao;
  }

  public static void FimConexao()
  {
            try
            {
                  conexao.close(); // FECHA A CONEXÃO
                  System.out.println("PROGRAMA DESCONECTADO...");
            }
            catch(SQLException f)
            {
                  System.out.println("ERRO AO FECHAR O ARQUIVO!!! " + f);
            }
      } // fim da conexao
}
```

Anexo E – Código Fonte da Classe Resultados

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.lang.Exception;
import javax.swing.JOptionPane;
import java.sql.*;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import java.awt.image.BufferedImage;
import java.io.*;
import javax.imageio.ImageIO;
import javax.swing.AbstractAction;

public class Resultados extends JFrame
{
      private    JLabel    codigoImpressao,    nomeUsuario,    cpfUsuario,
imagemImpressao;
      private JTextField codCampo[] = new JTextField[100];
      private JTextField nomeCampo[] = new JTextField[100];
      private JTextField imagemCampo[] = new JTextField[100];
      private String[][] reg = new String[100][4];
      private int indice=0;
      private int cod=0,nome=1,cpf=2,imagem=3;
      private String sql;
      int auxcod = 0, cont = 0, vezes = 0;
    private JLabel label[] = new JLabel[100];
      private Icon icons[] = new Icon[100];
      private int vetAux[][] = new int[100][2];
      public int auxCod,sel;

      public  Resultados(int  vet[][],String[][]  regist,  int  contador,  int
qtde, int selecao)
      {
            setTitle("Resultados");
            setSize( 750, 550 );
            setBackground(new Color(238, 238, 238));

            vetAux = vet;
            reg = regist;
            cont = contador;
            vezes = qtde;
            sel = selecao;

            //Container                        container                        =
getContentPane();//container.setLayout(null);

            Container container = getContentPane();
        container.setLayout( new FlowLayout() );

            /* Botões */
            JButton sairJanela = new JButton("Sair");
            container.add(sairJanela);
            sairJanela.setBounds(new Rectangle(100, 15, 187, 46));

            int t = 50;
```

```
            for(int j = 0;j < vezes;j++)
            {     /* Labels */
                  t += 30;
                  codigoImpressao = new JLabel("Código");
                  container.add(codigoImpressao);
                  codigoImpressao.setBounds(new Rectangle(30, t, 94, 16));
                  t += 30;
                  nomeUsuario = new JLabel("Nome");
                  container.add(nomeUsuario);
                  nomeUsuario.setBounds(new Rectangle(30, t, 94, 16));
            }

            t = 50;
            for(int j = 0;j < vezes;j++)
            {     t += 30;
                  /* TextFields */
                  codCampo[j] = new JTextField();
                  container.add(codCampo[j]);
                  codCampo[j].setBounds(new Rectangle(140, t, 60, 20));
                  t += 30;
                  nomeCampo[j] = new JTextField();
                  container.add(nomeCampo[j]);
                  nomeCampo[j].setBounds(new Rectangle(140, t, 500, 20));
          }

          for(int i = 0;i < vezes;i++)
            {
                  auxCod = vetAux[i][1];
                  if(sel != auxCod)
                  {
                      codCampo[i].setText(reg[auxCod][cod]);
                        nomeCampo[i].setText(reg[auxCod][nome]);
                  icons[i] = new ImageIcon( reg[auxCod][imagem]);
                  }
            }

            for(int i = 0;i < vezes;i++)
            {
            label[i] = new JLabel( icons[ i ] );
            container.add( label[i] );
            }

            // ACTIONS BUTTON HANDLER
            ButtonHandler handler = new ButtonHandler();
            sairJanela.addActionListener( handler );

      } // fim Result

      private class ButtonHandler implements ActionListener
      {
            public void actionPerformed ( ActionEvent event )
        {

                  if (event.getActionCommand()== "Sair")
                  {
                        setVisible(false);
                  }
            }
      }
}
```

Anexo F – Código Fonte da Classe StoredFinger (Software pronto)

```
import java.awt.image.*;

public class StoredFinger
{
	protected int nLins, nCols;
	protected int[] minucias;

	public StoredFinger(int l, int c, int[] m)
	{
		nLins = l;
		nCols = c;
		minucias = m;
	}

	public StoredFinger()
	{
	}

	public int[] match(StoredFinger x, int raio)
	{
		int[] vet2 = (int[]) x.minucias.clone();
		int[] vet = (int[]) minucias.clone();
		int cont = 0;

		for (int i = 0; i < vet.length; i++)
		{
			int l = Math.abs(vet[i]);
			int l2 = 0;

			int c = l % x.nCols;
			l = l / x.nCols;
			if ( l >= nLins || c >= nCols )
				continue;
			boolean bateu = false;
			for (int j = 0; j < vet2.length; j++)
			{
				l2 = vet2[j];
				if ( (vet[i] < 0 && l2 >= 0) || (vet[i] >= 0 && l2
< 0) )
				{
					continue;
				}
				l2 = Math.abs(l2);
				int c2 = l2 % nCols;
				l2 = l2 / nCols;
				java.awt.Rectangle circle = new
						java.awt.Rectangle(l2-raio,	c2-raio,
raio + raio, raio + raio);
				if ( circle.contains(l,c) )
				{
					vet[i] = vet2[j] = 0;
					cont++;
					break;
				}
			}

		}
		int[] ret = new int[cont];
```

```
            cont = 0;
            for (int i = 0; i < vet.length; i++)
            {
                  if ( vet[i] != minucias[i] )
                        ret[cont++] = minucias[i];
            }
            return ret;
      }

      public int numCasa(String image1, String image2, int percentagem, int
raio) throws Exception
      {
            WritableRaster wr = null;
            int tamanho;
            GreyImage g1 = GreyImage.createImage(image1);
            GreyImage g2 = GreyImage.createImage(image2);
            wr = g1.getRaster();
            wr.setPixels(0,      0,      wr.getWidth(),      wr.getHeight(),
g1.equalizacao());
            wr.setPixels(0,      0,      wr.getWidth(),      wr.getHeight(),
g1.thresholding(11));
            wr.setPixels(0,      0,      wr.getWidth(),      wr.getHeight(),
g1.dilatacao2());
            wr.setPixels(0,      0,      wr.getWidth(),      wr.getHeight(),
g1.thinning(255));
            wr.setPixels(0,      0,      wr.getWidth(),      wr.getHeight(),
g1.limpeza2(0));
            int[] crVet1 = g1.crossNumber(0);

            wr = g2.getRaster();
            wr.setPixels(0,      0,      wr.getWidth(),      wr.getHeight(),
g2.equalizacao());
            wr.setPixels(0,      0,      wr.getWidth(),      wr.getHeight(),
g2.thresholding(11));
            wr.setPixels(0,      0,      wr.getWidth(),      wr.getHeight(),
g2.dilatacao2());
            wr.setPixels(0,      0,      wr.getWidth(),      wr.getHeight(),
g2.thinning(255));
            wr.setPixels(0,      0,      wr.getWidth(),      wr.getHeight(),
g2.limpeza2(0));
            int[] crVet2 = g2.crossNumber(0);

            int[] minuc1 = g1.getMinucias(crVet1, percentagem);
            int[] minuc2 = g2.getMinucias(crVet2, percentagem);

            StoredFinger    sf1    =    new    StoredFinger(wr.getHeight(),
wr.getWidth(), minuc1);
            StoredFinger    sf2    =    new    StoredFinger(wr.getHeight(),
wr.getWidth(), minuc2);
            int[] m = sf1.match(sf2, raio);

            tamanho = m.length;
            System.out.println("Minucias de 1: " + minuc1.length);
            System.out.println("Minucias de 2: " + minuc2.length);
            System.out.println("Matches: " + tamanho);

            return tamanho;
     }
}
```