BIOMETRIC IDENTIFICATION USING FINGERPRINT IN WEB SYSTEMS

Leandro Costa Coelho (Instituto Superior de Tecnologia em Ciências da Computação do Rio de Janeiro, Rio de Janeiro, Brasil) – leandrocosta2@gmail.com

Marcio Belo Rodrigues da Silva (Instituto Superior de Tecnologia em Ciências da Computação do Rio de Janeiro, Rio de Janeiro, Brasil) – mbelo.ist-rio@faetec.rj.gov.br

Fingerprint is one of the most used biometrics' characteristic to identify a unique person in the whole world. Although it has a large use in information systems, its appliance to WEB systems is still rare considering the importance that kind of application has nowadays. This work targets the integration of these technologies in a simple, practical and transparent manner for users, using an example software application – known as ISTBio - to assess this integration.

Keywords: Biometry, WEB, Fingerprint, Identification, Authentication.

IDENTIFICAÇÃO BIOMÉTRICA POR IMPRESSÃO DIGITAL EM SISTEMAS WEB

A impressão digital é uma das características biométricas mais utilizadas atualmente para se identificar um indivíduo em todo o mundo. Embora seja de ampla utilização em sistemas informatizados, o uso desta tecnologia ainda é escasso em ambientes WEB. Este trabalho visa integrar o uso da impressão digital como meio de identificação biométrica em sistemas WEB, de uma maneira simples, prática e transparente para os usuários, utilizando um protótipo funcional – o ISTBio - para avaliar na prática a viabilidade da solução.

Palavras-chave: Biometria, WEB, Impressão digital, Identificação, Autenticação.

Agradecimentos

Aos professores e funcionários do IST-Rio, pelos valiosos ensinamentos e por toda amizade conquistada durante todo nosso caminhar na vida acadêmica.

1. Introdução

As impressões digitais têm um papel fundamental em nossa sociedade. Desde a sua descoberta até o início dos estudos - entre o final do século XIX e início do século XX - as impressões digitais têm sido utilizadas como uma das principais características para se identificar um indivíduo de forma unívoca até hoje.

No início, os processos de aquisição e reconhecimento eram feitos de forma manual, utilizando fichas de papel e tinta especial para o recolhimento das impressões digitais, e precisava de um perito para fazer a identificação das características presentes nas digitais. Devido à aceitação formal do reconhecimento das impressões digitais como um meio de identificação, as bases de dados criadas ficaram tão grandes que inviabilizaram a identificação manual, pois mesmo com um grande número de peritos para analisar as amostras colhidas, eles não seriam capazes de atender toda a demanda a tempo. Com isso, em meados das décadas de 80 e 90, começaram a ser implantados sistemas de reconhecimento automatizados de impressões digitais, que evoluíram rapidamente, chegando a processar até 35 mil impressões digitais por segundo [Griaule Biometrics, 2009].

Com o avanço tecnológico e a queda do custo de produção dos dispositivos de leitura biométrica, o reconhecimento de impressões digitais deixou de ser uma tecnologia voltada apenas para fins forenses e vem sendo utilizada diariamente em tarefas cotidianas como, por exemplo, em caixas eletrônicos, em *check-ins* de aeroportos, no acesso às urnas eletrônicas eleitorais, entre outras. Outrossim, realizou-se uma pesquisa com vistas a analisar o suporte ao uso de dispositivos de leitura biométrica e constatou-se que a maioria das soluções só funciona em ambientes locais (*desktop*) e que o suporte dessas a um sistema *WEB*, quando existem, é precário e limitado, devido à falta de pacotes de desenvolvimento disponibilizados pelos fabricantes.

Por conta disso, iniciou-se uma busca por soluções que permitam o uso desses dispositivos em ambientes *WEB*, tendo como pontos-chave a maior compatibilidade possível com os leitores biométricos existentes no mercado, a independência de *browsers* e sistemas operacionais, e a interoperabilidade entre linguagens de programação. Espera-se, com isso, possibilitar a criação de um sistema simples, prático e transparente para o usuário final, tendo como ambiente de testes para a demonstração de viabilidades técnicas das soluções prospectadas um protótipo totalmente funcional.

2. Situação Atual

O uso de tecnologias biométricas vem aumentando consideravelmente no mundo moderno. Seja na fechadura de uma porta, no acesso a funções restritas do aparelho celular ou até mesmo para dar a ignição em um carro. Esta tecnologia deixou de ser algo que só existia em filmes de ficção científica e vem se tornando cada vez mais presente em nosso dia a dia. De acordo com um relatório apresentado pelo *International Biometric Group* (IBG) [IBG, 2008] sobre o uso de tecnologias biométricas em sistemas em todo o mundo, relativo aos anos de 2004 a 2008, constatou-se a hegemonia do uso das impressões digitais em relação aos demais tipos, com uma fatia significativa de 48% dos sistemas pesquisados, conforme mostra a figura 1.

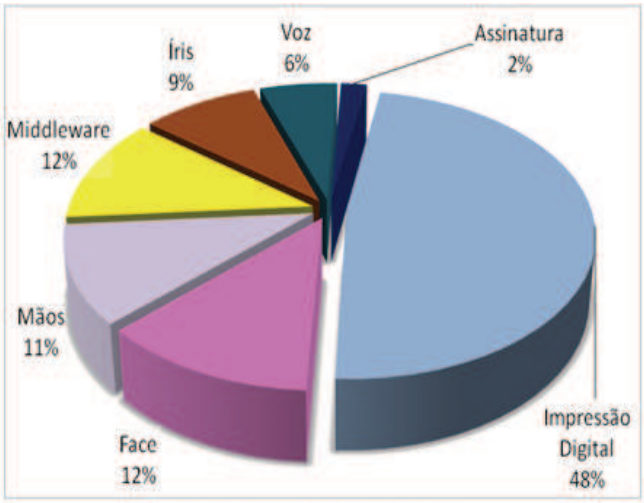

Figura 1: Gráfico sobre o uso de tecnologias biométricas no mercado

Apesar de existirem inúmeros sistemas em funcionamento em todo o Brasil, há dois deles que merecem destaque por serem utilizados em dois grandes órgãos governamentais: no Departamento Estadual de Trânsito (DETRAN) e no Tribunal Superior Eleitoral (TSE).

O DETRAN utiliza a tecnologia biométrica para realizar o cadastro de condutores em seu sistema e para acompanhar todo o processo de formação dos mesmos, controlando o cumprimento da carga horária de aulas obrigatórias de cada aluno e a execução dos testes teóricos e práticos para a aquisição ou renovação da Carteira Nacional de Habilitação (CNH). O cadastro das impressões digitais dos alunos ocorre durante o exame médico. Com isso, os alunos já estarão aptos para o início das aulas, onde a impressão digital é utilizada como um cartão de ponto, ou seja, para marcar os horários de entrada, intervalo (quando aplicável) e saída do aluno, tendo assim um controle exato da carga horária de aula cumprida nos Centros de Formação de Condutores (CFC). Esta solução foi implantada oficialmente no dia 6 de setembro de 2007 e vem funcionando satisfatoriamente até o momento.

O TSE, por sua vez, está em fase de testes da tecnologia biométrica no processo de votação eleitoral. A finalidade principal é eliminar o número de fraudes no processo, tendo na impressão digital uma forma confiável de identificação do indivíduo. Nesse novo sistema, o próprio eleitor libera automaticamente a urna biométrica para a votação, mediante o reconhecimento da sua impressão digital. Caso surja alguma dúvida com relação à identidade do eleitor, ou quando a digital do mesmo não for reconhecida pelo sistema biométrico, o mesário terá a sua disposição uma folha de votação com as fotos de todos os eleitores da seção, à qual poderá recorrer, em caráter excepcional, para liberar a urna através de uma senha mestre para que o eleitor possa efetuar o seu voto [TSE, 2008]. Nas eleições de 2008, foram realizados testes do novo sistema em três cidades brasileiras: Colorado do Oeste, em Rondônia; São João Batista, em Santa Catarina; e Fátima do Sul, em Mato Grosso do Sul. Nesta última, só ocorreram três casos onde não foi possível efetuar a leitura das digitais, pois as

mãos dos eleitores estavam muito machucadas, o que levou à utilização do processo de verificação manual. Com o sucesso desses testes, o TSE pretende implantar a urna biométrica no Brasil inteiro, num prazo estimado de 5 a 10 anos.

No que tange aos tipos de ambientes de sistemas informatizados, observa-se uma continuada ascensão de sistemas baseados em *WEB*, em especial os que fazem uso de tecnologias combinando *HTML*, *JavaScript*, *AJAX*, *Flash* e *Applets Java*; as chamadas aplicações *WEB* ricas. A figura 2, retirada de [Dr.Dobbs, 2010], corrobora com essa visão.

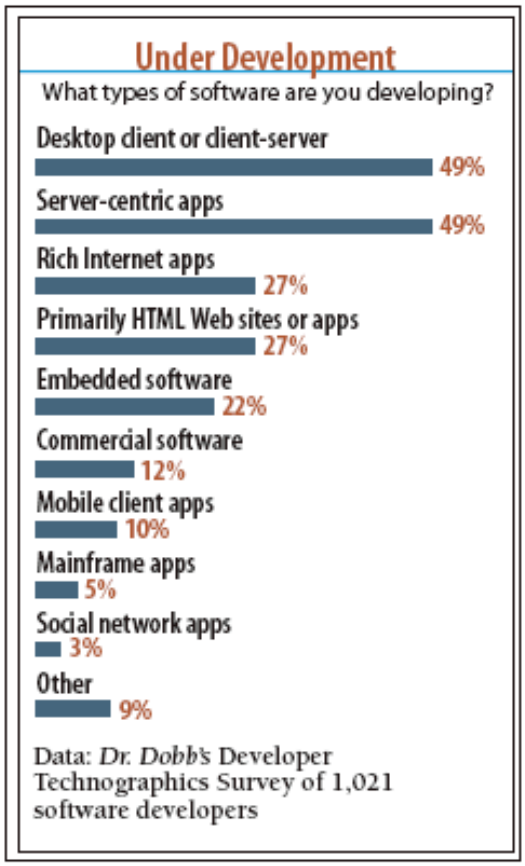

Figura 2: Tipos de software em desenvolvimento

3. Aspectos da Biometria por Impressão Digital

Os sistemas de identificação biométrica têm a impressão digital analisada como uma série de linhas escuras e de espaços em branco, nas quais as linhas escuras representam os sulcos ou cumes (parte elevada da pele) e os espaços em branco representam os vales (parte rasa da pele) entre estes sulcos. Sendo as minúcias (*minutiae*) as principais características de uma impressão digital, os sistemas se baseiam primeiramente nelas e/ou na posição e no sentido dos términos dos sulcos e das bifurcações (separações) ao longo da trajetória de um sulco.

3.1. Classificação das Minúcias

Quanto à classificação, as minúcias dividem-se em seis tipos, conforme exibido na figura 3, retirada de [Júnior, A.A.B., 2007], e descritos em seguida.

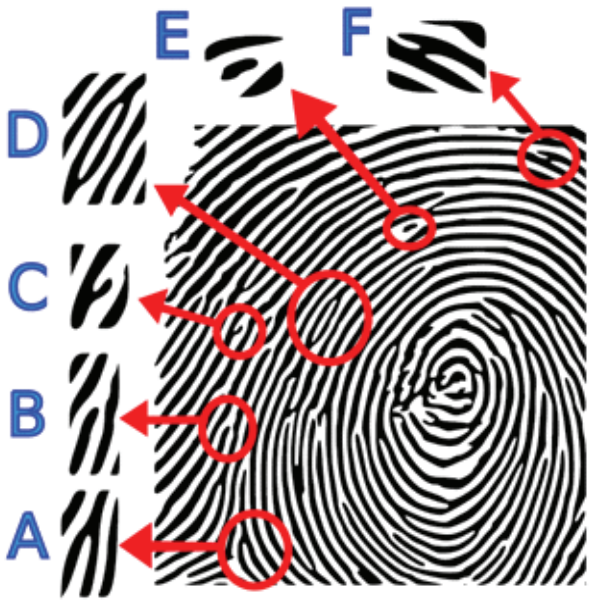

Figura 3: Classificação das minúcias

(A) Bifurcações: um traço que se divide em dois outros traços;

(B) Cristas Finais ou Terminações: um traço que termina abruptamente;

(C) Esporas ou Ramificações: um traço curto que sai de um mais longo;

(D) Ilhas ou Lagos: um traço que se divide em dois e depois se une novamente;

(E) Cristas Curtas ou Traço Curto: um traço que percorre um pequeno caminho e termina;

(F) Cruzamentos ou Pontes: um traço que sai de um traço e termina em outro;

3.2. Hardware envolvido

Basicamente, os dispositivos biométricos de leitura de impressões digitais têm seu hardware composto de *platen* ou *scanner*, geralmente feitos de vidro, plástico, polímero ou silício [Navati S., 2002], e de um módulo que contém os circuitos necessários para a recepção e o envio da informação digitalizada para o sistema, conforme exibido na figura 4. Em alguns casos, o módulo é responsável por todo o processo de extração, cadastramento e reconhecimento de impressões digitais.

Figura 4: Módulo (à esquerda) interligado ao platen ou scanner (à direita)

Quanto às tecnologias de leitura empregadas nesses dispositivos, estas podem ser classificadas como óticas, capacitivas ou por ultrassom, conforme exemplificado na figura 5.

Figura 5: Tecnologias e dispositivos - ótico, capacitivo e por ultrassom, respectivamente

Leitores Óticos

O leitor ótico é a tecnologia mais antiga e amplamente utilizada atualmente. Ele funciona através de uma câmera que registra a imagem da impressão digital que está de encontro ao *platen* de vidro ou de plástico, sobre o qual os sulcos e os vales digitalizados aparecem como linhas pretas, cinzas e brancas, conforme exibido na figura 6, retirada de [Levesque, 2002]. Com isso, a câmera adquire uma série de imagens, permitindo que o software subjacente avalie a qualidade da impressão digital e gere um modelo (*template*) para o registro ou verificação.

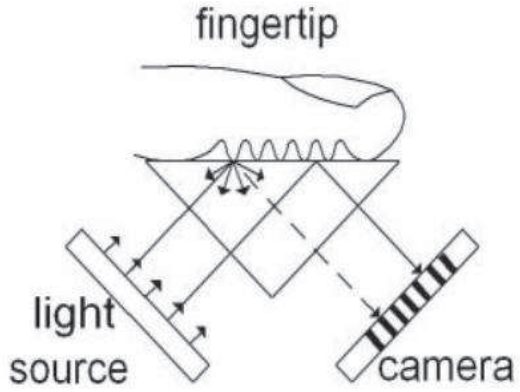

Figura 6: Funcionamento dos leitores óticos

Dentre as vantagens de utilização dos leitores óticos, podemos citar o baixo custo desses dispositivos em comparação com outros modelos, a sua alta resolução de captura de imagem – podendo chegar até 500 DPI (pontos por polegada) – e a sua imunidade a danos eletrostáticos.

Dentre as desvantagens, podemos citar a tendência em mostrar impressões latentes como impressões digitais reais e a suscetibilidade para falsos dedos. Essas vulnerabilidades foram exploradas em experimentos realizados pelos apresentadores do programa *MythBusters* [MythBusters, 2006], seriado produzido e exibido pelo canal *Discovery Channel*, que obtiveram sucesso ao burlar leitores óticos com moldes de dedos feitos em gelatina balística, contendo a impressão digital cadastrada nos sistemas biométricos e até mesmo com ela impressa com qualidade laser em um papel branco comum.

Leitores Capacitivos

Os leitores capacitivos ganharam uma aceitação considerável desde a sua introdução comercial em 1998. Usando um chip de silício como um *platen*, o processo de aquisição das imagens é baseado nas características capacitivas dos sulcos e vales da impressão digital, onde o sensor atua como uma das placas de um capacitor e a impressão digital atua como a outra. Dentre as tecnologias utilizadas na leitura das impressões digitais, os dispositivos podem utilizar os métodos de capacitância ativa, que geram um pequeno campo eletromagnético que se estende além da superfície de contato, para que seja possível a leitura da camada viva da pele e a capacitância passiva, que mede até o ponto de contato dos dedos com o *platen*, conforme exibido na figura 7, retirada de [Mainguet, 2009].

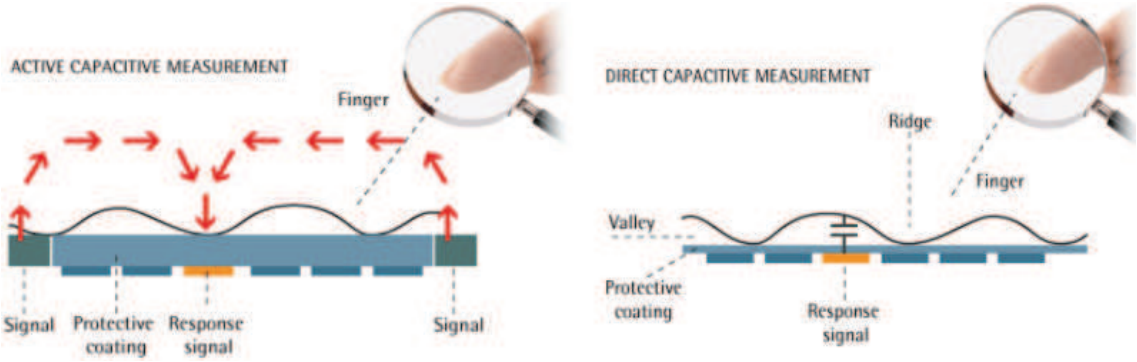

Figura 7: Funcionamento dos leitores capacitivos: capacitância ativa e passiva, respectivamente

Dentre as vantagens de utilização dos leitores capacitivos, podemos citar a alta qualidade de imagem adquirida pelo sensor, o seu tamanho compacto, o baixo custo de produção e seu baixo consumo elétrico.

Dentre as desvantagens, podemos citar como a principal o fato de estes sensores serem facilmente suscetíveis a danos eletrostáticos, o que compromete a sua durabilidade e o seu uso em determinadas situações.

Leitores por Ultrassom

Os leitores por ultrassom são utilizados com menos frequência que os demais tipos de leitores, porém possuem vantagens únicas. Eles trabalham transmitindo ondas acústicas inaudíveis em direção ao dedo do indivíduo, gerando imagens através da medição da impedância entre o dedo, o *platen* e o ar, conforme exibido na figura 8, retirada de [Ultra-Scan, 2002].

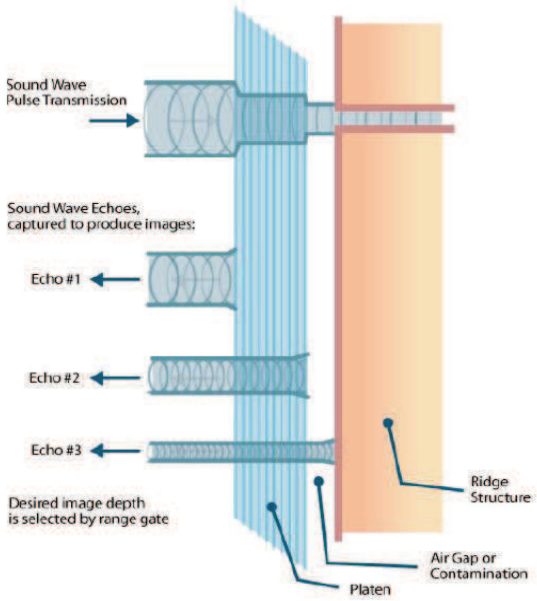

Figura 8: Funcionamento dos leitores por ultrassom

A vantagem na utilização dos leitores por ultrassom, entre outras, é a capacidade de penetrar a sujeira e os resíduos presentes nos dedos do indivíduo e de não serem sujeitos a alguns dos problemas de dissolução de imagens encontrados na maioria dos dispositivos óticos, gerando, assim, imagens de altíssima qualidade.

Há, entretanto, uma grande desvantagem no uso deste tipo de leitor: a quantidade de equipamentos necessários para que a leitura das impressões digitais seja possível, o que torna esta tecnologia cara.

3.3. Software

Desde que os sistemas de identificação biométrica automatizados começaram a ser implantados, nunca foi estabelecido, até hoje, um padrão de comparação e armazenamento das impressões digitais, levando cada fabricante a ter sua própria solução. Apesar disso, todos eles possuem alguns conceitos de comparação em comum, que são os baseados em minúcias e os baseados em correlações.

A comparação por minúcias é a técnica mais popular e amplamente utilizada, baseando-se nas localizações e direções dos pontos de minúcias. Já a comparação por correlações utiliza duas imagens e julga suas semelhanças, realizando a sobreposição de imagens para calcular a correlação entre os *pixels*[1] para diferentes tipos de alinhamentos.

Em qualquer sistema biométrico, o processo engloba as fases de cadastramento (*enrollment*) e de reconhecimento, possuindo este último duas ramificações, que são os processos de verificação (*verify*) e de identificação (*identify*) de um indivíduo. Essas fases são apresentadas em [Pinheiro, 2008].

Na fase de cadastramento, o usuário fornece suas características biométricas, que serão passadas pelo sistema ao módulo de qualidade, que por sua vez irá verificar se a imagem adquirida está em boas condições para a geração de um modelo a ser utilizado em futuras comparações. Tendo este módulo aceitado a imagem, ele a repassa para o módulo extrator de características, que irá rodar um algoritmo para identificar e mapear todas as minúcias encontradas, gerando um resultado que será armazenado no banco de dados em conjunto com os dados pessoais do indivíduo. A figura 9 ilustra o processo de cadastramento.

1 unidade elementar de uma imagem

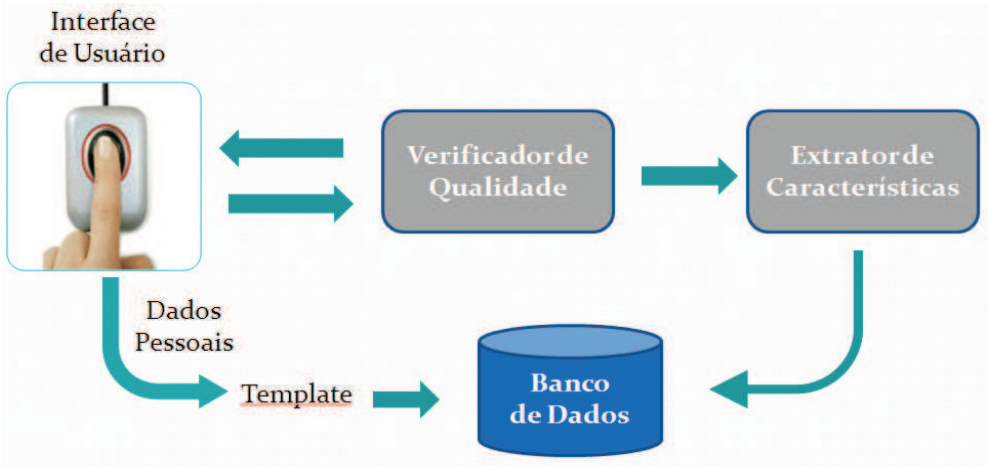

Figura 9: Fase do cadastramento biométrico do usuário

No processo de verificação, o usuário se identifica para o sistema com algo que ele sabe (nome de usuário, matrícula, senha, etc.) e fornece uma de suas impressões digitais para que o sistema possa confirmar ou negar a alegada identidade. De posse da amostra biométrica fornecida, o sistema a repassa para o módulo extrator de características, que irá identificar e mapear as minúcias, enviando o resultado deste processamento para o módulo comparador 1 para 1, que o utilizará na comparação com o modelo biométrico (*template*) desse indivíduo armazenado na fase de cadastramento, determinando se ele é quem realmente alega ser. A figura 10 ilustra o processo de verificação.

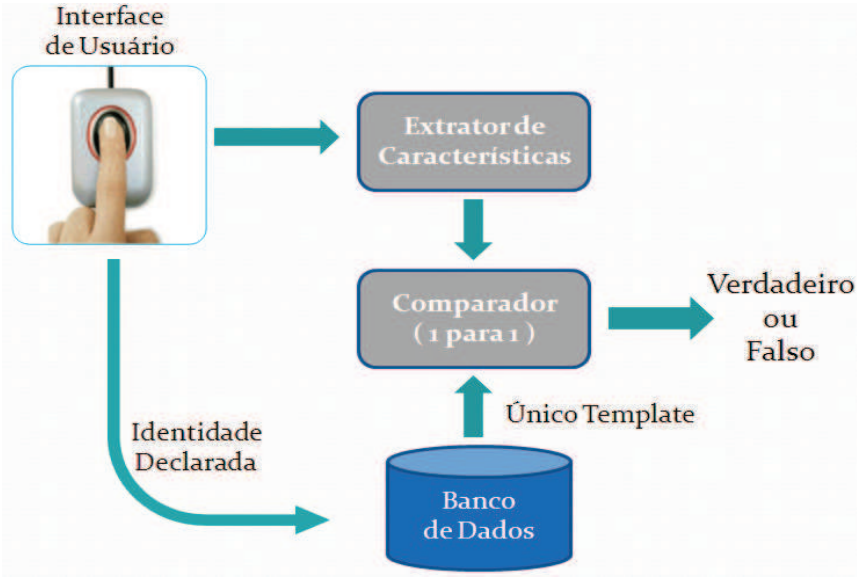

Figura 10: Fase da verificação biométrica do usuário

No processo de identificação, o usuário fornece apenas uma de suas impressões digitais para que o sistema tente identificá-lo, tendo como referência todos os modelos biométricos cadastrados no banco de dados. Em outras palavras, de posse da amostra biométrica fornecida, o sistema a repassa para o módulo extrator de características e este envia o resultado do processamento

para o módulo comparador 1 para N, que irá comparar a informação biométrica com todas informações cadastradas no banco de dados, identificando ou não o usuário. A figura 11 ilustra o processo de identificação.

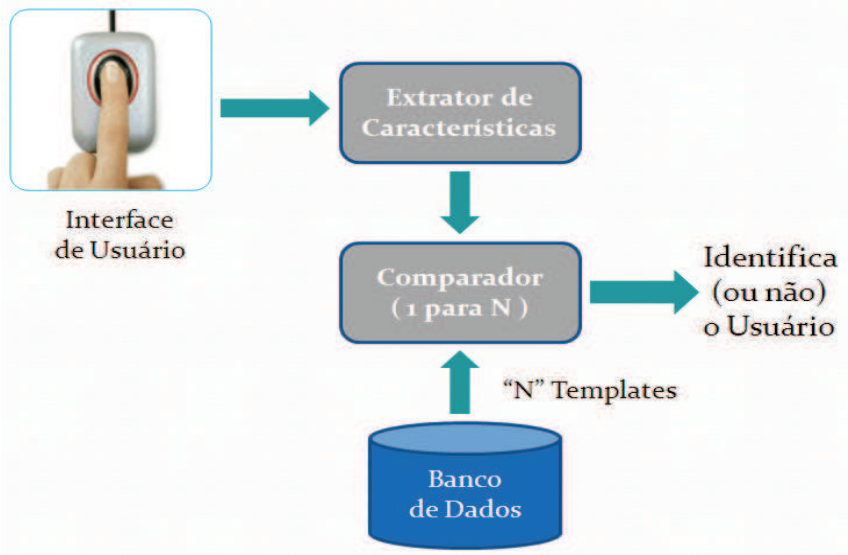

Figura 11: Fase da identificação biométrica do usuário

Uma das utilizações mais interessantes da identificação ocorre em sistemas criminais, onde se busca um indivíduo dentre vários suspeitos. Além disso, em sistemas onde a segurança não é um fator de alta criticidade, a identificação pode ser uma solução prática, como em fechaduras eletrônicas que usam a digital como chave. O problema principal da identificação é o custo computacional de processamento. Quando a base de *templates* sobre a qual se deseja realizar a identificação de um indivíduo é grande, o custo computacional pode ser extremamente alto.

Com a solução criada para este trabalho – o ISTBio – realizou-se a coleta e testes com 28 indivíduos voluntários. Para essa pequena amostra a performance da identificação 1 para N mostrou-se satisfatória com 100% de acertos. Entretanto, dada a arquitetura da solução, espera-se que, ao atingir um determinado limite de *templates* armazenados, tal procedimento torne-se computacionalmente inviável em tempo de resposta. Ressalva-se, entretanto, que na verificação (comparador 1 para 1) tal problema não ocorre, independente da quantidade de *templates* armazenados no banco de dados.

4. Protótipo de Validação da Tecnologia

Com o propósito de provar a viabilidade técnica na criação de uma solução de software baseado em *WEB* que utilize sistemas biométricos, criou-se um protótipo totalmente funcional de software – denominado ISTBio – com as funcionalidades básicas que envolvem o cadastramento, verificação e identificação de indivíduos por meios biométricos.

4.1. Arquitetura Geral

O protótipo implementado consiste no uso de uma A*pplet* Java, responsável pela comunicação com o dispositivo biométrico e pelo tratamento, verificação e identificação das características biométricas presentes na impressão digital, bem

como pela comunicação com o servidor central, onde ficam armazenados todos os dados referentes aos usuários. A linguagem de programação utilizada no servidor é o PHP e o banco de dados é o MySQL. A comunicação entre cliente e servidor é feita através da troca de objetos serializados no formato JSON[1]. A figura 12 destaca os principais componentes da arquitetura do ISTBio e a comunicação entre eles.

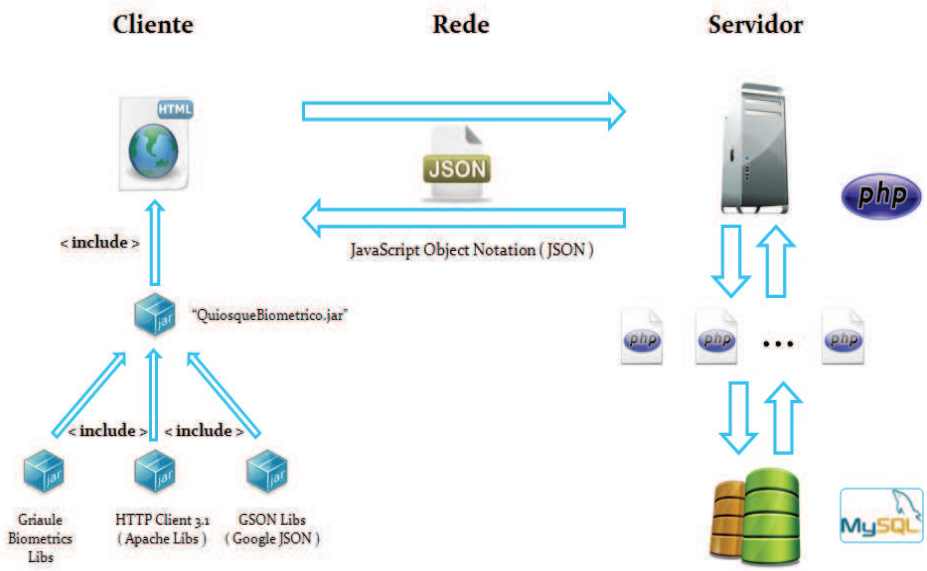

Figura 12: Arquitetura geral da aplicação

4.2. Detalhamento da solução

Estrutura de Arquivos e Suas Funções

O protótipo, tanto a *Applet Java* (cliente) quanto o servidor, foi desenvolvido em camadas e totalmente orientados a objetos. A *Applet Java* foi dividida em cinco arquivos, como exibido na figura 13 e descritos em seguida.

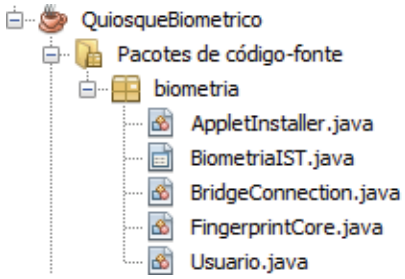

Figura 13: Organização dos arquivos de código-fonte da Applet Java (cliente)

- *AppletInstaller.java*: classe responsável pela instalação dos *drivers* dos dispositivos biométricos que serão utilizados pela *Applet* Java;

1 JSON (JavaScript Object Notation – Notação de Objetos JavaScript) é uma formatação textual leve para a serialização de dados estruturados [RFC4627, 2006].

- *BiometriaIST.java*: classe responsável pela criação da interface do usuário da *Applet* Java e também pelo redirecionamento do navegador para outras páginas do aplicativo *WEB*;

- *BridgeConnection.java*: classe responsável pela conexão/comunicação da *Applet* Java com o servidor;

- *FingerprintCore.java*: classe de controle da aplicação, que implementa as interfaces da API da *Griaule Biometrics* de forma a capturar os eventos disparados em resposta à manipulação do dispositivo biométrico. Ao ocorrer um desses eventos, essa classe também se responsabiliza pela interface com o servidor que hospeda os *templates* biométricos cadastrados;

- *Usuario.java*: classe que representa o *bean* da entidade usuário. O usuário possui dois atributos: a matrícula – chave primária de identificação – e um *template* - onde é armazenado o produto do processamento da impressão digital adquirida.

O servidor, escrito na linguagem PHP, foi elaborado seguindo o padrão de arquitetura de projeto MVC (Model-View-Controller) e segue uma estrutura bem dividida de pastas, conforme exibido na figura 14 e descritos a seguir.

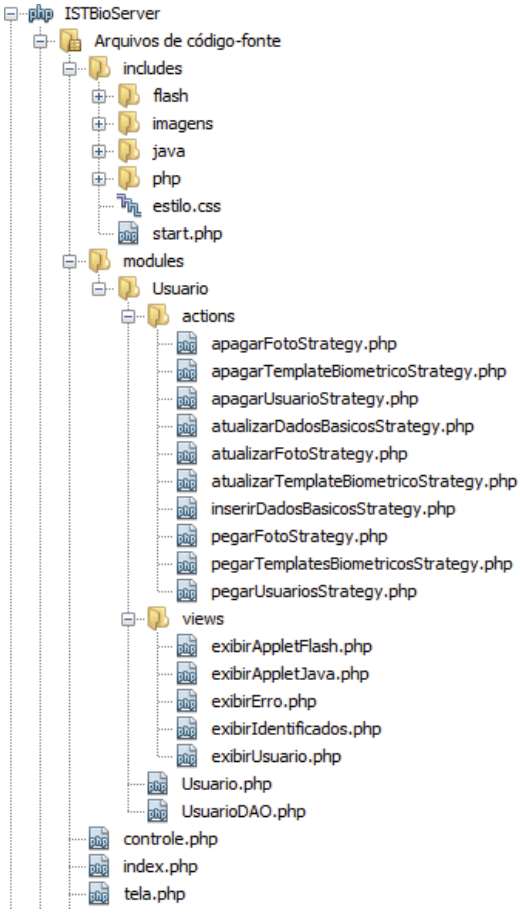

Figura 14: Organização dos arquivos de código-fonte do ISTBioServer

- Pasta *includes*: nesta pasta estão armazenados todos os itens que serão

importados dentro de cada página, como, por exemplo, imagens, arquivos de folha de estilos, bibliotecas de terceiros, etc. Esta é a pasta onde fica armazenada a aplicação cliente (*Applet* Java);

- Pasta *modules*: nesta pasta estão armazenadas subpastas referentes a cada uma das entidades presentes no banco de dados. Em cada subpasta, estão armazenados arquivos referentes às classes de acesso ao banco de dados e o *bean* daquela entidade. Estão armazenadas também outras duas subpastas: a *actions* - que contém os arquivos que executam as ações no modelo e onde são implementadas as regras de negócio – e a pasta *views* - que contém todas as telas de páginas *WEB* referentes àquela entidade.

- Na pasta raiz estão armazenadas a página inicial do site e o controlador principal de toda a aplicação.

Configuração do Ambiente de Execução

Nas configurações do servidor, deve-se verificar no arquivo de configuração do PHP – arquivo php.ini – se a opção *magic_quotes_gpc* está com o valor *Off*. Caso contrário, ocorrerão erros na comunicação da *Applet* Java com o servidor, através da troca de objetos serializados no formato JSON.

Já na camada cliente, os seguintes itens de configuração devem ser verificados para que a solução seja executada normalmente:

- A máquina virtual Java deve estar instalada na máquina do usuário e o *plugin* para a execução de *applets* no navegador devidamente configurado;

- O *driver* do dispositivo biométrico deve estar instalado. Em ambiente Windows, deve ser instalado um dos *drivers* disponíveis na parte de suporte do site da *Griaule Biometrics*. Em ambiente Linux, deve ser instalado um pacote chamado *libfrint0*, que pode ser encontrado nos repositórios das principais distribuições Linux existentes ou através do site do projeto [Fprint, 2010].

Fluxo e Funcionamento da Aplicação

Quando um usuário entra na página inicial do protótipo ISTBio, é exibida uma página *WEB* contendo botões de acesso para cada ação presente em um sistema biométrico – cadastramento, verificação e identificação – conforme exibido na figura 15.

Figura 15: Página inicial do protótipo ISTBio

Cada um destes botões efetuará o redirecionamento para uma mesma página *WEB*, que carregará a aplicação cliente no navegador. A diferença sutil presente nos botões são as ações que os mesmos passam como argumento, que irá dizer à *Applet* Java em que modo ela deverá operar. O trecho de código responsável por obter a ação enviada e carregar a *Applet* Java é exibido no código 1. Nesse mesmo código-fonte observa-se a referência às bibliotecas necessárias para o carregamento da *Applet* Java, com todas as suas dependências, a destacar:

- *Griaule Biometrics*: API biométrica proprietária da empresa *Griaule Biometrics*, responsável pela extração das minúcias e pelo processo de comparação de digitais. A biblioteca está no arquivo *SignedFingerprintSDKJava.jar*:

- *Apache Commons*: conjunto de bibliotecas de código aberto da Fundação Apache (Apache Foundation) que implementam as funcionalidades de requisições *WEB* (protocolo HTTP) na linguagem Java. As bibliotecas individuais são: *commons-codec-1.4.jar*, *commons-httpclient-3.1.jar*, *commons-logging-1.1.1.jar*;

-

- *GSON*: API de código aberto da *Google* que permite que aplicações Java

trabalhem com (de)serialização de objetos no padrão JSON. A biblioteca está empacotada no arquivo: *gson-1.4.jar*;

```
<?php

     // Verificando se o modo de operação da Applet e se a matricula foram passados como argumento
     if(!isset($_POST["modoOperacao"])) {
          header("Location: index.php");
     }
     else if(($_POST["modoOperacao"] == "cadastrar" || $_POST["modoOperacao"] == "verificar") &&
(empty($_POST["matricula"]))) {
          header("Location: index.php");
     }
     else {
          $modoOperacao = ucfirst($_POST["modoOperacao"]);
          $matricula = $_POST["matricula"];
     }
?>

<div class ="mConteudo">
    <h2>Impress&atilde;o Digital</h2>
    <p>Coloque seu polegar direito sobre a superf&iacute;cie de leitura do sensor biom&eacute;trico
e siga as instru&ccedil;&otilde;es exibidas no "Log de Eventos"</p>
    <center>
        <applet code="biometria.BiometriaIST" codebase="includes/java"
                archive="./QuiosqueBiometrico.jar,./lib/SignedFingerprintSDKJava.jar,./lib/commons-
codec-1.4.jar,./lib/commons-httpclient-3.1.jar,./lib/commons-logging-1.1.1.jar,./lib/gson-1.4.jar"
                height="426" width="383" alt="A JVM (Java Virtual Machine) n&atilde;o est&aacute;
instalada no seu sistema!">

            <param name="modoOperacao" value="<?php echo $modoOperacao; ?>">
            <param name="matricula" value="<?php echo $matricula; ?>">
        </applet>
    </center>
    <br />
</div>
```

Código 1: Trecho do código HTML/PHP cliente

Por questões de segurança, uma *Applet* Java é executada dentro de um ambiente seguro – também conhecido como *sandbox* – com direitos de acesso restrito. Como a *Applet* em questão precisa ter acesso ao dispositivo biométrico, gravar os *drivers* dos dispositivos em um diretório temporário na máquina do usuário e usar bibliotecas de terceiros, houve a necessidade de efetuar a assinatura da *Applet* e de todas as bibliotecas utilizadas, com exceção da biblioteca da *Griaule Biometrics*, que já vem assinada com o certificado do próprio fabricante. Com isso, é garantido o perfeito funcionamento da solução, pois fornecerá à *Applet* Java direitos de acesso iguais ao de uma aplicação Java nativa, mediante autorização do usuário, através das janelas de aviso que a máquina virtual Java (JVM) exibirá no navegador.

Uma vez recebidos os argumentos da página HTML, o código de carregamento da *Applet* Java dispara o método *init* da *Applet* (primeiro método executado assim que uma *Applet* é criada). O método *getParameter* permite capturar os argumentos enviados pela página HTML e iniciar a aplicação no modo desejado, conforme exibido no código 2.

```
public class BiometriaIST extends javax.swing.JApplet {

    // Códigos omitidos
    @Override
    public void init() {

        // Pegando os parâmetros passados para Applet pelo HTML
        operacao = this.getParameter("modoOperacao");
        matricula = this.getParameter("matricula");


        try {
            java.awt.EventQueue.invokeAndWait(new Runnable() {
                public void run() {
                    initComponents();
                }
            });
        } catch (Exception ex) {
            ex.printStackTrace();
        }


        // Atribuindo um "text" e uma "tooltip" ao botão, de acordo com a operação desejada
        actionButton.setText(operacao);
        if(operacao.equalsIgnoreCase("cadastrar")) {
            actionButton.setToolTipText("Cadastrar a impressão digital no banco de dados");
            this.escreverLog("Processamento para matrícula \"" + matricula + "\"");
        }
        else if(operacao.equalsIgnoreCase("verificar")) {
            actionButton.setToolTipText("Verificar se a impressão digital corresponde a identificação
declarada");
            this.escreverLog("Processamento para matrícula \"" + matricula + "\"");
        }
        else {
            actionButton.setToolTipText("Tentar identificar sua impressão digital dentro do banco de
dados");
        }
        // Instanciando, instalando e inicializando o SDK
        this.fingerprintSDK = new FingerprintCore(this);
        // Setando o endereço principal do servidor, passando o Host onde a aplicação está instalada
        BridgeConnection.setServidor(this.getCodeBase().getHost());
    }
    // Códigos omitidos
}
```

Código 2: Trecho do código de inicialização da Applet Java

Figura 16: Aplicação cliente carregada no modo de cadastramento

Após a execução, é exibida uma página *WEB* contendo a aplicação cliente no corpo da mesma, como exibido na figura 16.

No processo de carregamento da aplicação cliente o método *inicializarSDK* – da classe *FingerprintCore* – é executado. Ele instala os *drivers* necessários para comunicação do dispositivo biométrico com a aplicação – através do método *install* da classe *AppletInstaller* – e inicializa o SDK (*Software Development Kit*) da *Griaule Biometrics*, ativando o dispositivo biométrico, conforme exibido no código 3.

```
// Inicializa o SDK e ativa a captura de digitais
private void inicializarSDK() {


    try {
        // Instala os arquivos do SDK num diretório temporário
        AppletInstaller.install(getClass().getResource("/FingerprintSDKLibs.zip"));
        interfaceUsuario.escreverLog("Carregamento das bibliotecas: OK!");


        // Instanciando um objeto-contexto do SDK
        fingerprintSDK = new MatchingContext();


        // Inicializa a captura das digitais
        GrFingerJava.initializeCapture(this);
        interfaceUsuario.escreverLog("Inicialização da captura: OK!");
    }
    catch(Exception e) {
        interfaceUsuario.escreverLog("Falha na instalação ou inicialização do SDK!", true);
    }
}
```

Código 3: Inicialização do SDK da Griaule Biometrics

Após o carregamento total da *Applet* Java, o SDK da *Griaule Biometrics* passa a monitorar alguns eventos que são essenciais para a interação com o dispositivo biométrico. Seguindo recomendações da própria documentação da empresa, foi criada uma classe que implementa os métodos contidos em três interfaces, conforme mostra o código 4 e descritos em seguida.

```
public class FingerprintCore implements IStatusEventListener, IImageEventListener, IFingerEventListener
{


    // Códigos omitidos


    // Esta função é chamada toda vez que um leitor de digitais for conectado
    public void onSensorPlug(String idSensor) { … }


    // Esta função é chamada toda vez que um leitor de digitais for desconectado
    public void onSensorUnplug(String idSensor) { … }


    // Esta função é chamada toda vez que uma digital é capturada
    public void onImageAcquired(String idSensor, FingerprintImage impressaoDigital) { … }


    // Esta função é chamada toda vez que uma digital é colocada no sensor
    public void onFingerDown(String idSensor) { … }


    // Esta função é chamada toda vez que uma digital é removida do sensor
    public void onFingerUp(String idSensor) { … }


    // Códigos omitidos
}
```

Código 4: Interfaces do SDK da Griaule Biometrics e seus respectivos métodos a serem implementados

- *IStatusEventListener*: interface responsável por receber eventos sobre o estado dos sensores. Nesta interface, devem ser implementados os métodos *onSensorPlug* e *onSensorUnplug*, que são chamados sempre que um sensor biométrico for, respectivamente, conectado ou desconectado do computador;

- *IImageEventListener*: interface responsável por receber as digitais adquiridas por um sensor. Nesta interface, deve ser implementado o método *onImageAcquired*, que é chamado toda vez que uma impressão digital for capturada pelo sensor biométrico;

- *IFingerEventListener*: interface responsável por receber eventos de digitais vindas de um sensor já conectado. Nesta interface, devem ser implementados os métodos *onFingerDown* e *onFingerUp*, que são chamados sempre que o usuário retirar ou colocar sua impressão digital no *platen* do sensor biométrico, respectivamente;

Processamento no Cliente

Depois que uma impressão digital é colhida pelo leitor biométrico, o SDK da *Griaule Biometrics* identifica as minúcias presentes na impressão digital e extrai o *template* da mesma, indicando a sua qualidade, como exibido no código 5.

```
// Extrai um template da imagem / impressão digital atual
public void extrairTemplate() {


    try {
        // Extrai um template da impressão digital atual
        template = fingerprintSDK.extract(impressaoDigital);


        // Notifica o usuário que o template foi extraído e exibe a qualidade do mesmo
        String msg = "Template extraido com sucesso. ";
        switch(template.getQuality()) {
            case Template.HIGH_QUALITY: msg += "Qualidade: Alta."; break;
            case Template.MEDIUM_QUALITY: msg += "Qualidade: Media."; break;
            case Template.LOW_QUALITY: msg += "Qualidade: Baixa."; break;
        }
        interfaceUsuario.escreverLog(msg);


        // Mostra as minúcias, seguimentos e direções na imagem
        interfaceUsuario.mostrarImagem(GrFingerJava.getBiometricImage(template, impressaoDigital));
    }
    catch(GrFingerJavaException e) {
        interfaceUsuario.escreverLog("Falha na extração do template!", true);
    }
}
```

Código 5: Extração do template biométrico

Após o acionamento do botão de ação pelo usuário na interface do aplicativo, é chamado um método presente na classe *FingerprintCore*, cujo comportamento varia de acordo com o modo de operação da *Applet* Java escolhido anteriormente, como exibido no código 6.

```
private void actionButtonActionPerformed(java.awt.event.ActionEvent evt) {

    // Verificando a ação desejada e executando a mesma
    if(operacao.equalsIgnoreCase("cadastrar")) {
        fingerprintSDK.cadastrarImpressaoDigital(matricula);
    }
    else if(operacao.equalsIgnoreCase("verificar")) {
        fingerprintSDK.verificarUsuario(matricula);
    }
    else {
        fingerprintSDK.identificarUsuario();
    }
}
```

Código 6: Trecho da classe FingerprintCore que trata o evento disparado pelo botão de ação

No processo de cadastramento, cria-se um objeto da classe *Usuario* (*bean*) contendo seus dados, sendo passado em seguida para o método sobrecarregado *enviarRequisicao* da classe *BridgeConnection*, que por sua vez serializa o objeto numa *string* codificada na notação JSON – usando a biblioteca GSON - e o envia através de uma requisição POST para o servidor. Após o processamento no servidor, recebe uma mensagem que indica se o usuário foi inserido com sucesso ou não. A implementação da classe *Usuario* e de ambos os métodos podem ser vistos a seguir, nos códigos 7, 8 e 9, respectivamente.

```
public class Usuario {

    // Atributos privados da classe Usuario
    private String matricula;
    private byte[] template;

    // Construtores da classe Usuario

    // Construtor default obrigatorio para a deserialização do objeto
    public Usuario() {
        this(null, null);
    }

    public Usuario(String matricula, byte[] template) {
        this.matricula = matricula;
        this.template = template;
    }

    // Métodos públicos de acesso Get e Set
    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }

    public String getMatricula() {
        return this.matricula;
    }

    public void setTemplate(byte[] template) {
        this.template = template;
    }
    public byte[] getTemplate() {
        return this.template;
    }
}
```

Código 7: Implementação da classe Usuario na Applet Java (bean)

```
// Processo de cadastramento (enrollment)
public void cadastrarImpressaoDigital(String matricula) {


    // Instanciando um novo objeto da classe Usuario
    Usuario novoUsuario = new Usuario(matricula, template.getData());


    // Instanciando um objeto da classe Gson (Objeto Serializador / Deserializador)
    Gson sr = new Gson();


    try {
        // Pegando o objeto responsável pela conexão com o servidor
        BridgeConnection conexao = BridgeConnection.pegarInstancia();
        boolean sucesso = conexao.enviarRequisicao(novoUsuario);
        if(sucesso) {
            interfaceUsuario.escreverLog("Impressão digital cadastrada com sucesso na matrícula \"" +
matricula + "\"!");


            // Preparando a URL para o redirecionamento de página no navegador
            String url = BridgeConnection.getServidor() +
"?modulo=Usuario&view=exibirAppletFlash&matricula=" + matricula;
            interfaceUsuario.redirecionarNavegador(new URL(url));
        }
        else {
            interfaceUsuario.escreverLog("Erro na gravação da impressão digital no banco de dados!",
true);
        }
    }
    catch(IOException e) {
        interfaceUsuario.escreverLog("Falha no envio dos dados ao servidor!", true);
    }
    catch(Exception e) {
        interfaceUsuario.escreverLog(e.getMessage(), true);
    }
}
```

Código 8: Método cadastrarImpressaoDigital da classe FingerprintCore

```
// Método responsável pelas requisições de cadastramento (enrollment)
// Envio de dados para o servidor via POST
public boolean enviarRequisicao(Usuario usuario)
    throws IOException, Exception {


    // Instanciando um objeto da classe Gson (Objeto Serializador / Deserializador)
    Gson sr = new Gson();


    // Obtendo o equivalente JSON do objeto passado como parâmetro
    String jUsuario = sr.toJson(usuario);


    // Inicializando o cliente HTTP
    HttpClient cliente = new HttpClient();


    String url = urlServidor + "?modulo=Usuario&acao=atualizarTemplateBiometrico";


    // Estabelecendo o método de envio de dados (POST)
    PostMethod metodo = new PostMethod(url);


    // Adicionando os parâmetros que serão enviados
    metodo.addParameter("jsonUsuario", jUsuario);


    // Executando a requisição / método
    if(cliente.executeMethod(metodo) == -1)
        throw new Exception("Excecao capturada no \"-1\" - cadastramento.");


    // Recuperando o retorno vindo do servidor
    String resultado = metodo.getResponseBodyAsString();


    // Liberando / Fechando a conexão
    metodo.releaseConnection();
    return (resultado.equalsIgnoreCase("true")) ? true : false;
}
```

Código 9: Método sobrecarregado enviarRequisicao da classe BridgeConnection – cadastramento

No processo de verificação, passa-se para o método sobrecarregado *enviarRequisicao* da classe *BridgeConnection* o identificador do indivíduo – a matrícula – indicado pelo usuário na página inicial do protótipo. Este, por sua vez, envia o identificador para o servidor através do método POST e captura o retorno que, em caso de sucesso, será um objeto da classe *Usuario* contendo a matrícula e seu respectivo *template* biométrico, codificados no padrão de serialização JSON. O método efetua a deserialização do objeto e o retorna para o método chamador, que por sua vez confronta o *template* vindo do banco de dados com o fornecido pelo o usuário através de métodos fornecidos pelo SDK da *Griaule Biometrics*, obtendo assim, a negação ou a confirmação da identidade que o usuário alega ter. A implementação de ambos os métodos podem ser vistos a seguir, nos códigos 10 e 11, respectivamente.

```
// Processo de verificação (verify)
public void verificarUsuario(String matricula) {


    try {
        // Pegando o objeto responsável pela conexão com o servidor
        BridgeConnection conexao = BridgeConnection.pegarInstancia();


        Usuario usuario = conexao.enviarRequisicao(matricula);


        if(usuario == null) {
            interfaceUsuario.escreverLog("Matricula não encontrada no banco de dados!", true);
            return;
        }


        // Pegando o template de referência
        Template referencia = new Template(usuario.getTemplate());


        // Compara os templates
        boolean combina = fingerprintSDK.verify(template, referencia);


        // Se os templates combinam, mostrar a combinação das minúcias, segmentos e direções
        if(combina) {
            // Mostra as minúcias, seguimentos e direções na imagem assim como as combinações entre
ambas
            interfaceUsuario.mostrarImagem(GrFingerJava.getBiometricImage(template, impressaoDigital,
fingerprintSDK));


            // Notifica que os templates combinaram
            interfaceUsuario.escreverLog("Os templates combinam com score = " +
fingerprintSDK.getScore() + "!");


            // Preparando a URL para o redirecionamento de página no navegador
            String url = BridgeConnection.getServidor() +
"?modulo=Usuario&acao=pegarUsuarios&matricula=" + matricula;
            interfaceUsuario.redirecionarNavegador(new URL(url));
        }
        else {
            // Notifica que os templates não combinaram
            interfaceUsuario.escreverLog("Os templates não combinam com score = " +
fingerprintSDK.getScore() + "!");
        }
    }
    catch(IOException e) {
        interfaceUsuario.escreverLog("Falha no envio / recebimento dos dados do servidor!", true);
    }
    catch(JsonParseException e) {
        interfaceUsuario.escreverLog("Falha na deserialização dos dados vindos do servidor!", true);
    }
    catch(GrFingerJavaException e) {
        interfaceUsuario.escreverLog("Falha na comparação / verificação dos templates!", true);
    }
    catch(Exception e) {
        interfaceUsuario.escreverLog(e.getMessage(), true);
    }
}
```

Código 10: Método verificarUsuario da classe FingerprintCore

```
// Método responsável pelas requisições de verificação (verify)
// Envio de dados para o servidor via POST
public Usuario enviarRequisicao(String matricula)
    throws IOException, JsonParseException, Exception {


    // Inicializando o cliente HTTP
    HttpClient cliente = new HttpClient();


    String url = urlServidor + "?modulo=Usuario&acao=pegarTemplatesBiometricos";


    // Estabelecendo o método de envio de dados (POST)
    PostMethod metodo = new PostMethod(url);
    // Adicionando os parâmetros que serão enviados
    metodo.addParameter("matricula", matricula);


    // Executando a requisição / método
    if(cliente.executeMethod(metodo) == -1)
        throw new Exception("Excecao capturada no \"-1\" - verificacao.");


    // Recuperando o retorno vindo do servidor
    String jUsuario = metodo.getResponseBodyAsString();


    // Liberando / Fechando a conexão
    metodo.releaseConnection();


    if(jUsuario.equalsIgnoreCase("null"))
        return null;


    // Instanciando um objeto da classe Gson (Objeto Serializador / Deserializador)
    Gson sr = new Gson();


    // Restaurando o objeto através da String JSON passada como argumento
    Usuario usuario = sr.fromJson(jUsuario, Usuario.class);


    return usuario;
}
```

Código 11: Método sobrecarregado enviarRequisicao da classe BridgeConnection – verificação

No processo de identificação, de forma semelhante ao que ocorre no de verificação, é submetida uma requisição ao servidor em busca de *template*. A diferença, nesse caso, é que a requisição pede todos os *templates* existentes no banco de dados, recebidos como instâncias da classe *Usuario*. Após a coleta do *template* do usuário que tenta se identificar, essa amostra é comparada com todos os demais recebidos do servidor. Uma lista de potenciais positivos é gerada.

Ao final de qualquer um dos três modos de operação, a *Applet* Java, em caso de sucesso na operação, efetua a chamada de uma nova página WEB, através do método *redirecionarNavegador* presente na classe *BiometriaIST*, como exibido no código 12.

```
// Método que efetua uma nova chamada de página no navegador
public void redirecionarNavegador(URL url) {

    this.escreverLog("Aguarde o redirecionamento do navegador...");

    AppletContext contexto = this.getAppletContext();

    contexto.showDocument(url);
}
```

Código 12: Método redirecionarNavegador da aplicação cliente

Na operação de cadastramento, a *Applet* Java redireciona o navegador para a página *WEB* contendo uma aplicação em *Flash* que acessa a *WebCam* instalada no computador do usuário, de forma a capturar a identidade visual do indivíduo, conforme descrito em [José David e Silva, M.B.R., 2009].

Processamento no Servidor

Toda requisição feita ao servidor passa antes por um código PHP que é o controlador principal da aplicação. Através dos parâmetros, este código sabe qual *Action* deve ser chamada e, logo após, sua respectiva *View*. No servidor, o acesso aos dados é feito através da variável global *$_REQUEST*, que obtém os argumentos enviados pelo cliente. A implementação do controlador pode ser vista no código 13.

```php
<?php

    // Pegando o módulo e a ação desejada pelo método GET
    if(isset($_GET["modulo"]) && (isset($_GET["acao"]))) {
        // Indica qual entidade do sistema / BD será trabalhada
        $modulo = $_GET["modulo"];


        // Indica a ação solicitada a ser feita pelo sistema
        $acao = $_GET["acao"];
        $acao .= "Strategy";


        // Montando a URL da Action a ser chamada
        $url = "modules/" . $modulo . "/actions/" . $acao . ".php";
        // Importando a Action a ser executada
        require_once($url);


        // Instanciando um objeto da classe da Action e chamando o método "execute"
        $acao = new $acao;
        $retorno = $acao->execute();


        if($retorno == "APPLET_JSON") {
            // Imprimindo a string JSON a ser lida pela Applet Java
            echo $_REQUEST["jsonOutput"];
        }
        else if($retorno == "IMAGEM_BLOB") {
            // Indicando que o conteudo que será exibido é uma imagem
            header("Content-type: image/jpeg");


            // Imprimindo a foto vinda de um campo BLOB do BD
            echo $_REQUEST["foto"];
        }
        else {
            // Montando a URL da View a ser chamada
            $view = "modules/" . $modulo . "/views/" . $retorno . ".php";
            // Importando o controlador de telas (Views)
            require_once 'tela.php';
        }
    }
    else if(isset($_GET["modulo"]) && (isset($_GET["view"]))) {
        // Indica qual a entidade do sistema / BD será acessada
        $modulo = $_GET["modulo"];


        // Indica a view a ser acessada pelo sistema
        $tela = $_GET["view"];


        // Montando a URL da View a ser chamada
        $view = "modules/" . $modulo . "/views/" . $tela . ".php";
        // Importando o controlador de telas (Views)
        require_once 'tela.php';
    }
    else {
        header("Location: index.php");
    }
?>
```

Código 13: Controlador principal da aplicação no servidor

Quando uma solicitação de cadastramento de *template* biométrico é feita, o método *atualizarTemplateBiometricoStrategy* é chamado no servidor, que recebe um objeto da classe *Usuario* via POST, serializado no padrão JSON, e o deserializa, criando um novo objeto da classe *Usuario* com o mesmo estado que ele tinha quando enviado pelo cliente. Na sequência, esse objeto é enviado ao método *atualizarTemplateBiometrico* da classe *UsuarioDAO*, que efetivará sua gravação no banco de dados, retornando um indicador de sucesso. Este resultado será capturado posteriormente pela aplicação cliente. A implementação da classe *Usuario* e de ambos os métodos podem ser vistos a seguir, nos códigos 14, 15 e 16, respectivamente.

```php
<?php

    class Usuario {

        // Atributos privados da classe Usuario
        private $matricula;
        private $nome;
        private $template;
        private $foto;

        // Construtor da classe Usuario
        function __construct() {
            // Por convenção, um Bean só pode ter o construtor default
        }

        // Métodos públicos de acesso Set e Get
        public function setMatricula($matricula) {
            $this->matricula = $matricula;
        }

        public function getMatricula() {
            return $this->matricula;
        }

        public function setNome($nome) {
            $this->nome = $nome;
        }

        public function getNome() {
            return $this->nome;
        }

        public function setTemplate($template) {
            $this->template = $template;
        }

        public function getTemplate() {
            return $this->template;
        }

        public function setFoto($foto) {
            $this->foto = addslashes($foto);
        }

        public function getFoto() {
            return $this->foto;
        }

        // Método que permite que a funcao "json->encode" acesse atributos protegidos e privados
        public function toArray() {

            $aux = null;
            foreach($this as $key => $value) {
                $aux[$key] = $value;
            }
            return $aux;
        }

        // Sobrescrita do método "toString"
        public function __toString() {
            return "Matricula: " . $this->matricula . " - Nome: " . $this->nome;
        }
    }
?>
```

Código 14: Implementação da classe Usuario no servidor PHP (bean)

```php
<?php
    // códigos omitidos

    class atualizarTemplateBiometricoStrategy implements iStrategy {

        public function execute() {
            // Pegando o parâmetro passado via POST pela Applet Java
            $jsonUsuario = $_POST["jsonUsuario"];

            // Instanciando o objeto (de)serializador JSON
            $json = new Services_JSON();

            // Restaurando o objeto através da String JSON passada como argumento
            // >>> Por padrão, esse método retorna um objeto genérico (stdClass)
            $obj = $json->decode($jsonUsuario);

            // Terminando a restauração do objeto, criando uma instância de sua verdadeira classe
            $usuario = new Usuario();
            $usuario->setMatricula($obj->matricula);
            $usuario->setTemplate($obj->template);

            // Abrindo conexão com o BD
            Conexao::conectar();

            // Acessando a camada de persistência através do DAO correspondente
            $resultado = UsuarioDAO::atualizarTemplateBiometrico($usuario);

            // Fechando conexão com o BD
            Conexao::desconectar();

            // Retornando o resultado da "inserção" para a Applet Java
            $_REQUEST["jsonOutput"] = ($resultado) ? "true" : "false";

            // Retornando a string que indica ao controlador que o resultado da Action deve ser
            // tratado de forma diferente, pois a saída será enviada para a Applet Java
            return "APPLET_JSON";
        }
    }
?>
```

Código 15: Código referente a Action atualizarTemplateBiometricoStrategy

```php
public static function atualizarTemplateBiometrico(Usuario $usuario) {

    $sql = "UPDATE usuario SET template = '" . serialize($usuario->getTemplate()) . "' WHERE matricula =
'" . $usuario->getMatricula() . "'";
    return mysql_query($sql);
}
```

Código 16: Código referente ao método atualizarTemplateBiometrico da classe UsuarioDAO

Quando uma solicitação de acesso aos *templates* biométricos é feita, o método *pegarTemplatesBiometricosStrategy* é chamado no servidor. Se for recebido como argumento do cliente a matrícula do usuário, o sistema entra no modo de verificação, e por isso o método *pegarTemplatesBiometricos* da classe *UsuarioDAO* será invocado, o qual acessará o banco de dados e retornará um objeto da classe *Usuario*, caso o mesmo seja encontrado. Caso não seja enviado o argumento matrícula, o sistema entra no modo de identificação, e então é disparado o método para capturar todos os perfis biométricos cadastrados no banco de dados, retornado através de uma coleção de objetos da classe *Usuario*. Em ambos os casos, o resultado será serializado no padrão JSON para que, posteriormente, possa ser capturado pela aplicação cliente. As implementações de ambos os métodos podem ser vistas a seguir, nos códigos 17 e 18, respectivamente.

```php
<?php

    // códigos omitidos
    class pegarTemplatesBiometricosStrategy implements iStrategy {

        public function execute() {

            // Abrindo conexão com o BD
            Conexao::conectar();

            // Instanciando o objeto (de)serializador JSON
            $json = new Services_JSON();

            // Verificando se o dado foi passado via POST
            if(isset($_POST["matricula"])) {

                // Verificação
                // Pegando o parâmetro passado via POST
                $matricula = $_POST["matricula"];

                // Acessando a camada de persistência através do DAO correspondente
                // Pegando um único objeto da classe Usuario com o template biométrico
                $template = UsuarioDAO::pegarTemplatesBiometricos($matricula);

                // Fechando conexão com o BD
                Conexao::desconectar();

                // Verificando se foi retornado um objeto da classe Usuario ou não
                if($template == null) {
                    $_REQUEST["jsonOutput"] = "null";
                }
                else {
                    // Passando o objeto para o seu equivavente JSON pelo método toArray() do objeto
                    $_REQUEST["jsonOutput"] = $json->encode($template->toArray());
                }
            }
            else {

                // Identificação
                $templates = UsuarioDAO::pegarTemplatesBiometricos();

                // Fechando conexão com o BD
                Conexao::desconectar();

                // Verificando se foi retornado um array de objetos da classe Usuario ou não
                if($templates == null) {
                    $_REQUEST["jsonOutput"] = "null";
                }
                else {

                    // Array utilizado na serialização dos objetos da classe Usuario
                    $jsonTemplates = array();

                    foreach($templates as $template) {
                        array_push($jsonTemplates, $json->encode($template->toArray()));
                    }

                    // Retornando o array serializado JSON para a Applet
                    $_REQUEST["jsonOutput"] = $json->encode($jsonTemplates);
                }
            }

            // Retornando a string que indica ao controlador que o resultado da Action deve ser
            // tratado de forma diferente, pois a saída será enviada para a Applet Java
            return "APPLET_JSON";
        }
    }
?>
```

Código 17: Action pegarTemplatesBiometricosStrategy

```php
public static function pegarTemplatesBiometricos($matricula = null) {

    $sql = "SELECT matricula, template FROM usuario ";

    // Pega apenas um template do BD (verificação)
    if($matricula != null) {
        $sql .= "WHERE matricula = '" . $matricula . "'";

        $resultado = mysql_query($sql);
        if(!$resultado) {
            return null;
        }

        $usuario = mysql_fetch_object($resultado, "Usuario");
        $usuario->setTemplate(unserialize($usuario->getTemplate()));
        return $usuario;
    }

    // Pega todos os templates do BD (identificação)
    $sql .= "ORDER BY matricula";

    $resultado = mysql_query($sql);
    if(!$resultado) {
        return null;
    }

    $templates = array();
    while($registro = mysql_fetch_object($resultado, "Usuario")) {
        $registro->setTemplate(unserialize($registro->getTemplate()));
        array_push($templates, $registro);
    }
    return $templates;
}
```

Código 18: Método pegarTemplatesBiometricos da classe UsuarioDAO

5. Conclusão

O protótipo desenvolvido demonstrou ser viável a utilização de elementos biométricos para a identificação de indivíduos em ambientes baseados na *WEB*. Vários membros desta instituição colaboraram com a utilização do protótipo, realizando o cadastramento de suas informações biométricas para a realização de testes de identificação e verificação. O resultado mostrou-se bastante satisfatório.

Como proposta para trabalho futuro, sugerimos a prospecção e implantação de alternativa em software livre para substituir a comercial que foi utilizada neste protótipo.

Referências

Dr.Dobbs, 2010. “What Developers Think”. Disponível em: <http://www.drdobbs.com/architecture-and-design/222301141>, acessado em 09 dez 2010.

Fprint, 2010. Main Page – fprint project. Disponível em: <http://www.reactivated.net/fprint/wiki/Main_Page>, acessado em 09 dez 2010.

Google-GSON. Disponível em: <http://code.google.com/p/google-gson/>, acessado em 06 dez 2010.

Griaule Biometrics, 2009. Disponível em:

<http://www.griaulebiometrics.com/page/pt-br/fingerprint_sdk/features>, acessado em 09 dez 2010.

IBG, 2008. Biometrics Market and Industry Report 2004-2008. Disponível em:

<http://www.biometricgroup.com/reports/public/market_report.html>, acessado em 09 dez 2010.

José David e Silva, M.B.R., 2009. Prospecção e Implantação de Tecnologia de Identificação Biométrica. Ano II. vol.01. Nº 02. Revista Edu.Tec. Disponível em: <http://www.faetec.rj.gov.br/desup/images/edutec/num03/biometria_visual.pdf>

Júnior, A.A.B., 2007. Estudo e Desenvolvimento de Aplicação Biométrica em Ambiente de Larga Escala – Reconhecimento de Impressões Digitais. Disponível em:

<http://disciplinas.dcc.ufba.br/pub/MATA67/TrabalhosSemestre20071/monografia-slidesAmadeu.pdf>, acessado em 09 dez 2010.

Levesque, Vincent, 2002. Measurement of Skin Deformation Using Fingerprint Feature Tracking. Disponível em:

<http://www.cs.ubc.ca/~vlev/docs/VL_meng_thesis_2002.pdf>.

Mainguet, Jean-François, 2009. Biometrics: Fingerprint Sensing Techniques. Disponível em: <http://fingerchip.pagesperso-orange.fr/biometrics/types/fingerprint_sensors_physics.htm>, acessado em 09 dez 2010.

MythBusters, 2006. MythBusters – Crimes e Contravenções 2. Disponível em:

<http://dsc.discovery.com/fansites/mythbusters/episode/episode-tab-05.html>, acessado em 09 dez 2010.

Nanavati S., Thieme M., et al., 2002. Biometrics – Identity Verification in a Networked World. Editora John Wiley & Sons Inc.

PHP, Hypertext Preprocessor. Disponível em http://br.php.net/, acesso em 11 out 2010.

Apache Commons. The Apache Software Foundation. Disponível em <http://www.apache.org/>, acessado em 20 nov 2010.

Pinheiro, 2008. Biometria nos Sistemas Computacionais - Você é a Senha. Editora Ciência Moderna.

RFC4627, 2006. Request For Comments – JSON. Disponível em: <http://tools.ietf.org/html/rfc4627>, acessado em 06 dez 2010.

TSE, 2008. Identificação Biométrica do Eleitor – Manual. Disponível em: <http://www.tse.gov.br/downloads/biometria/arquivos/identificacaoBiometricaEleitorAberto.pdf>, acessado em 09 dez 2010.

Ultra-Scan, 2002. The Theory of Live-Scan Fingerprint Imaging. Disponível em:

<http://www.ultra-scan.com/LinkClick.aspx?link=Theory+of+Optical+vs+Ultrasonic+Imaging.pdf&mid=1332>, acessado em 09 dez 2010.