UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE CIÊNCIAS EXATAS E DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

API DE SEGURANÇA E ARMAZENAMENTO DE UMA ARQUITETURA MULTIBIOMÉTRICA PARA CONTROLE DE ACESSO COM AUTENTICAÇÃO CONTÍNUA

Adriana Esmeraldo de Oliveira

João Pessoa

2011

ADRIANA ESMERALDO DE OLIVEIRA

API DE SEGURANÇA E ARMAZENAMENTO DE UMA ARQUITETURA MULTIBIOMÉTRICA PARA CONTROLE DE ACESSO COM AUTENTICAÇÃO CONTÍNUA

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal da Paraíba como requisito para a obtenção do título de Mestre em Informática.

Orientador: Prof. Dr. Gustavo Henrique Matos Bezerra Motta

João Pessoa

2011

O48a Oliveira, Adriana Esmeraldo de.
API de segurança e armazenamento de uma arquitetura multibiométrica para controle de acesso com autenticação contínua / Adriana Esmeraldo de Oliveira. - - João Pessoa: [s.n.], 2011.
129f. il.
Orientador: Gustavo Henrique Matos Bezerra Motta.
Dissertação (Mestrado) – UFPB/CCEN.

1. Informática. 2. Arquitetura de software. 3. Segurança de templates. 4. Biometria. 5. Multibiometria.

UFPB/BC CDU: 004(043)

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de Adriana Esmeraldo de Oliveira, candidata ao Título de Mestre em Informática na Área de Sistemas de Computação, realizada em 16 de setembro de 2011.

Aos dezesseis dias do mês de setembro do ano dois mil e onze, às nove horas, no Auditório do CCEN - da Universidade Federal da Paraíba, reuniram-se os membros da Banca Examinadora constituída para examinar o candidato ao grau de Mestre em Informática, na área de "*Sistemas de Computação*", na linha de pesquisa "*Computação Distribuída*", a Sra. Adriana Esmeraldo de Oliveira. A comissão examinadora foi composta pelos professores doutores: Gustavo Henrique Matos Bezerra Motta (DI-UFPB), Orientador e Presidente da Banca Examinadora, Leonardo Vidal Batista (DI-UFPB) e Ed Porto Bezerra (DI-UFPB), como examinadores internos e Tsang Ing Ren (UFPE), como examinador externo. Dando início aos trabalhos, o Prof. Gustavo Motta, cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra à candidata para que o mesmo fizesse, oralmente, a exposição do trabalho de dissertação intitulado "*API de Segurança e Armazenamento de uma Arquitetura Multibiométrica para Controle de Acesso com Autenticação Contínua*". Concluída a exposição, a candidata foi argüido pela Banca Examinadora que emitiu o seguinte parecer: "*aprovado*". Assim sendo, deve a Universidade Federal da Paraíba expedir o respectivo diploma de Mestre em Informática na forma da lei e, para constar, eu, professora Tatiana Aires Tavares, Coordenadora deste Programa, servindo de secretária lavrei a presente ata que vai assinada por mim mesmo e pelos membros da Banca Examinadora. João Pessoa, 16 de setembro de 2011.

Tatiana Aires Tavares

Prof. Dr. Gustavo Henrique Matos Bezerra Motta
Orientador (DI-UFPB)

Prof. Dr. Leonardo Vidal Batista
Examinador Interno (DI-UFPB)

Prof. Dr. Ed Porto Bezerra
Examinador Interno (DI-UFPB)

Prof. Dr Tsang Ing Ren
Examinador Externo (UFPE)

A dois grandes amigos e lutadores, exemplos de vida e perseverança, Gustavo Cavalcanti e JanKees van der Poel, in memoriam.

Agradecimentos

É com imensa satisfação que registro os meus sinceros agradecimentos aos queridos entes a seguir, a quem admiro com muita lucidez.

A Deus.

A meus pais, João Evangelista e Maria Helena, por me darem condições e exemplo para avançar nos estudos.

A meus irmãos, Lílian e Leandro, meus primeiros amigos.

A todos os outros grandes amigos, em especial aqueles que, de certa forma, me acompanharam e incentivaram no período de desenvolvimento deste trabalho: Alexandre Dekker, Anderson Lima, André Brito, André Felipe, Antonio Deusany Junior, Bruno Neiva, Caio Honorato, Eduardo Costa, Felipe Lopes, Gabriel Florentino, Italo Curvelo, Rafael Targino, Raphael D'ávila, Raphael Marques, Ricardo Barbosa, Tássia Belarmina e Thaís Burity, por fazerem parte de minha vida até hoje e fazerem de mim uma pessoa ainda mais feliz.

Ao amigo e professor Sílvio Lago, por acreditar em mim. Suas palavras são de grande incentivo e seu senso crítico lhe dá todas as ferramentas de que precisa para crescer ainda mais.

A Pedro Alves Junior, por acreditar na pesquisa e prover meios para que trabalhos como este possam ser usados fora da academia.

A meus orientadores e amigos, Gustavo Motta e Leonardo Batista, por me guiarem neste trabalho, bem como a todos os outros professores que também lutam pelo Programa de Pós-Graduação em Informática da UFPB a fim de melhorar sua qualidade e elevar seu conceito.

À banca examinadora, em especial os membros Tsang Ing Ren e Ed Porto, por aceitarem o convite e, desta forma, contribuírem para a melhoria deste trabalho e dos trabalhos futuros a ele relacionados.

Sumário

Lista de Figuras e Gráficos

Lista de Tabelas

Lista de Acrônimos

AES	Advanced Encryption Standard
API	Application Programming Interface
ATM	Automated Teller Machine
BaaS	Biometric authentication as a Service
DLL	Dynamic Link Library
EAD	Educação a Distância
FAR	False Acceptance Rate
FRR	False Reject Rate
FTCR	Failure To Capture Rate
FTER	Failure To Capture Rate
ISO	International Organization for Standardization
LAN	Local Area Network
RSA	Rivest Shamir Adleman Algorithm
SDK	Software Development Kit
SGBD	Sistema Gerenciador de Banco de Dados
SPI	Service Provider Interface
SRR	System Response Reliability
SSO	Single Sign-On
UML	Unified Modeling Language

Resumo

OLIVEIRA, A. E. **API de Segurança e Armazenamento de uma Arquitetura Multibiométrica para Controle de Acesso com Autenticação Contínua**. 2011. 129 p. Dissertação (Mestrado) – Departamento de Informática, Universidade Federal da Paraíba, João Pessoa, 2011.

Um sistema biométrico que empregue uma única peculiaridade ou traço característico é restrito. Esta limitação pode ser suavizada pela fusão dos dados apresentados por múltiplas fontes. Um sistema que consolida a evidência apresentada por múltiplas fontes biométricas é conhecido como um sistema multibiométrico.

Nesse contexto, este trabalho propõe a interface de aplicação (API) de segurança e armazenamento de uma arquitetura multibiométrica, com habilidade de empregar uma ou mais modalidades biométricas.

Em aplicações de controle de acesso, um usuário pode ser coagido a se autenticar para permitir um acesso indevido. Como alternativa para este problema, a API utiliza um processo de autenticação contínua, que verifica se o usuário que se identificou no início de uma aplicação de software ainda está apto a continuar no sistema, sem interferências humanas ou paralisações do processo.

Grande parte da literatura sobre projeto de sistemas biométricos tem o foco nas taxas de erro do sistema e na simplificação de equações. No entanto, também é importante que se tenha uma base sólida para progressos futuros no momento em que os processos e a arquitetura da nova aplicação biométrica estiverem sendo projetados. Neste sentido, a arquitetura projetada permitiu a construção de uma API bem definida para sistemas multibiométricos, que deverá auxiliar os desenvolvedores a padronizar, entre outras coisas, sua estrutura de dados, de forma a possibilitar e facilitar a fusão de modelos biométricos e a interoperabilidade.

Deste modo, a API de segurança e armazenamento desenvolvida suporta uma arquitetura multibiométrica de controle de acesso para autenticação contínua extensível, isto é, capaz de receber novas características e processos biométricos com facilidade, permitindo, ainda, o uso de um mecanismo de segurança de *templates* biométricos.

A API foi projetada e implementada. Sua demonstração foi feita através de uma aplicação protótipo, por meio da qual foi possível realizar os testes.

Palavras-chave: Multibiometria, Biometria, Multimodalidade, Controle de acesso, Autenticação contínua, Arquitetura de software, Segurança de *templates*.

Abstract

OLIVEIRA, A. E. **Security and Persistence APIs of a Multi-biometric Access Control Architecture for Continuous Authentication**. 2011. 129 p. Dissertation (Masters) – Departamento de Informática, Universidade Federal da Paraíba, João Pessoa, 2011.

A biometric system that employs one single biometric characteristic is constrained. This limitation can be reduced by fusing the information presented by multiple sources. A system that consolidates the evidence presented by multiple biometric sources is known as a multi-biometric system.

In such a context, this work proposes the security and persistence APIs of a multi-biometric architecture, which is capable of using one or more biometric modalities.

In access control applications, a user might be forced to authenticate in order to give an unauthorized access to a criminal. As an alternative to this problem, the API uses a continuous authentication process, which verifies if the user identified at the start of the software application is still able to remain on the system, without human interferences or breaks in the process.

Much of the literature on biometric system design has focused on system error rates and scaling equations. However, it is also important to have a solid foundation for future progress as the processes and systems architecture for the new biometric application are designed. Hence, the designed architecture made it possible to create a well-defined API for multi-biometric systems, which may help developers to standardize, among other things, their data structure, in order to enable and facilitate templates fusion and interoperability.

Therefore, the developed security and persistence APIs support a multi-biometric access control architecture. This architecture is extensible, that is, capable of easily comprising new biometric characteristics and processes, yet making it possible to use a template security mechanism.

The APIs were designed and implemented. They were demonstrated by a prototype application, through which it was possible to conduct the test experiments.

Keywords: Multi-biometrics, Biometrics, Multimodality, Access control, Continuous authentication, Software architecture, Template security.

Capítulo

1

Introdução

"O que fazemos em vida ecoa pela eternidade."

(Gladiador)

O mundo está vivendo um processo antigo, continuado e cada vez mais intenso de virtualização da economia, onde menos dinheiro circula usando sua representação clássica de papel-moeda. Os computadores exercem um grande papel nesse processo, pois automatizam diversas funções organizacionais, dentre elas, funções bancárias de transferência de ativos.

A oferta de serviços através da automatização já faz parte do contexto de grande parte das organizações nos dias atuais. Ao optar por disponibilizar um serviço que exija algum nível de personalização, uma organização se depara com o seguinte problema: o de estabelecer uma associação entre um indivíduo e uma identidade. Esse problema pode ser dividido em duas categorias: autenticação e identificação. Autenticação refere-se ao problema de confirmar ou negar uma alegada identidade de um indivíduo, enquanto identificação refere-se ao problema de estabelecer a identidade, desconhecida à partida, de um indivíduo (Thian, 2001).

Atualmente, pode-se identificar a existência massiva de serviços Web de autenticação baseados em políticas de senha, mecanismos de criptografia de dados e assinaturas digitais. Além desses, apontam-se também serviços biométricos de identificação e autenticação de usuários, considerados, a princípio, mais seguros, por exigirem a utilização de uma característica física única e, portanto, a presença do usuário ao menos no momento da autenticação.

No entanto, tais métodos fornecem certas vulnerabilidades que já foram exploradas no mundo da criminalidade digital. A metodologia tradicional de autenticação baseada em políticas de senha, por exemplo, permite que qualquer pessoa, munida de um *login* e sua respectiva senha, tenha acesso a um serviço, independente de ser o usuário a quem o *login* e senha se destinam. Por outro lado, os serviços biométricos que utilizam a impressão digital como característica biométrica, por exemplo, podem permitir que uma reprodução da impressão digital de um usuário em material de silicone seja usada para acesso indevido. A adoção de métodos para identificar o uso de dedos falsos (ausência de fluxo sanguíneo, perspiração, odor, etc.) (Antonelli, Raffaele, Maio, & Maltoni, 2006) (Moon, Chen, Chan, So, & Woo, 2005) (Parthasaradhi, Derakhshani, Hornak, & Schuckers, 2005) torna o processo complicado e custoso. Um trabalho recente demonstrou 90% de taxa de falsa aceitação em sistemas modernos de reconhecimento de impressões digitais, utilizando-se dedos de cadáveres, de plástico, de gelatina e de outros materiais de modelagem (Roberts, 2007).

Nesse contexto, e para concretizar ou avançar ainda mais o processo de virtualização da economia, impossibilitando a clonagem de cartões de crédito e o roubo de senhas, por exemplo, faz-se necessário um mecanismo de identificação e autenticação com o mais alto grau de segurança possível.

1.1 Motivação

Com a difusão da implantação de sistemas biométricos em várias aplicações, existem preocupações crescentes a respeito da segurança das tecnologias biométricas e da privacidade de seus usuários (Jain, Nandakumar, & Nagar, 2008). De acordo com (Ross, Nandakumar, & Jain, 2006), essas aplicações podem ser categorizadas em três grupos principais: governamentais (cartão de identificação nacional, carteira de habilitação, registro de eleitor, etc.), comerciais (caixa eletrônico, controle de acesso, telefone celular, *e-commerce*, *Internet banking*, etc.) e forenses (identificação de cadáveres, investigação criminal, crianças desaparecidas, etc.).

Algumas características comumente usadas nos sistemas biométricos são a face, a impressão digital, a geometria da mão, a impressão palmar, a íris, *keystroke* (forma de digitar), a voz e a maneira de andar. Apesar das grandes vantagens oferecidas por tais sistemas, há vários desafios a serem considerados, como ruídos nos dados capturados (e.g., cicatriz no dedo, voz rouca, iluminação ambiente, sujeira no sensor), variações intra-classe (e.g., pose incorreta da face (Hsu, 2002), mudança

biométrica natural), similaridades inter-classe (aumentam a taxa de falsa aceitação), não-universalidade (e.g., extração incorreta de minúcias de uma impressão digital, i.e., extração de informação sem significância), questões de interoperabilidade (e.g., algoritmo de reconhecimento de impressão digital não obtém o mesmo resultado que quando os dados são capturados por sensor de outra marca (Ross & Jain, 2004)), ataques de *spoofing* (mascaramento ou imitação), entre outros. Diante disso, considera-se que um sistema biométrico que empregue uma única peculiaridade ou traço característico é restrito, em especial por fatores intrínsecos. Essa limitação pode ser suavizada pela fusão das informações apresentadas por múltiplas fontes. Um sistema que consolida a evidência apresentada por múltiplas fontes biométricas é conhecido como um sistema multibiométrico (Ross, Nandakumar, & Jain, 2006).

Em aplicações comuns de controle de acesso, o uso da biometria tem, a princípio, a finalidade de garantir a presença do usuário na hora de identificação e autenticação. No entanto, um usuário pode ser coagido a se autenticar para permitir um acesso indevido. Uma alternativa para este problema é utilizar um processo de autenticação contínua. De acordo com (Brosso, 2006), a autenticação contínua deve ser um processo que verifica se o usuário que se identificou no início de uma aplicação de software ainda está apto a continuar no sistema, sem interferências humanas ou paralisações do processo.

Neste contexto, a aceitação pública das tecnologias biométricas irá depender da habilidade dos projetistas de sistemas em demonstrar que esses sistemas são robustos, possuem baixos índices de erros, e são à prova de falsificações (Jain, Nandakumar, & Nagar, 2008). Em sistemas de controle de acesso, isto pode ser feito com o auxílio da autenticação contínua e da multibiometria, onde há a disponibilidade de múltiplas evidências, que fazem com que sistemas multibiométricos sejam considerados mais confiáveis (Hong, Jain, & Pankanti, 1999).

Um outro aspecto relevante refere-se ao desenvolvimento dos sistemas biométricos. Grande parte da literatura sobre projeto de sistemas biométricos tem o foco nas taxas de erro do sistema e na simplificação de equações. No entanto, também é importante que se tenha uma base sólida para progressos futuros no momento em que os processos e a arquitetura da nova aplicação biométrica estiverem sendo projetados (Wayman, Jain, Maltoni, & Maio, 2005). Ainda, uma arquitetura com uma interface de aplicação (API) bem definida para sistemas multibiométricos deverá auxiliar os desenvolvedores a padronizar, entre outras coisas, sua estrutura de dados, de forma a possibilitar e facilitar a fusão de modelos e a interoperabilidade.

Modelos ou *templates* são estruturas que encapsulam dados biométricos. O armazenamento desses modelos deve ser feito com cautela uma vez que, diferentemente das senhas, modelos biométricos roubados não podem ser revogados. Portanto, a segurança de *templates* é uma questão de extrema importância a ser considerada em sistemas biométricos.

Assim, é conveniente que haja uma arquitetura e API de controle de acesso multibiométricas que provejam extensibilidade, isto é, capacidade de abarcar novas características e processos biométricos, permitindo, ainda, o uso de um mecanismo de segurança de *templates*, além de um processo de autenticação contínua.

1.2 Objetivo

O objetivo geral deste trabalho é o de criar e validar as APIs de segurança e armazenamento do BioPass. O projeto BioPass tem o objetivo de construir um *framework* de segurança da informação que use um processo de autenticação multibiométrica contínuo, para trabalhar em ambientes Web e desktop.

Por conseguinte, os objetivos específicos deste trabalho são:

- Objetivo 1. Construir um esquema geral de arquitetura de software multibiométrica para o BioPass;
- Objetivo 2. Criar um padrão de *template* multibiométrico extensível, isto é, capaz de abarcar novas características biométricas sem precisar ser modificado;
- Objetivo 3. Definir a forma como a segurança de *templates* será abordada na API;
- Objetivo 4. Definir a especificação das APIs de segurança e armazenamento do BioPass;
- Objetivo 5. Definir algoritmos ou APIs biométricos de reconhecimento de, no mínimo, dois traços biométricos a serem utilizados no protótipo de validação das APIs de segurança e armazenamento;
- Objetivo 6. Implementar um protótipo para validar as APIs propostas.

1.3 Justificativa

As APIs especificadas neste trabalho fazem parte do projeto BioPass, um produto que deverá ser explorado comercialmente.

O mercado já dispõe de alguns produtos consolidados para controle de acesso biométrico (e.g., (Neurotechnology, 2010), (Suprema Inc., 2010), (Griaule, 2010)). A maioria deles é unibiométrica e começou com o uso de impressão digital como característica biométrica. A Neurotechnology, por exemplo, já possui um produto multibiométrico (por impressão digital e face): o MegaMatcher. No entanto, houve uma mudança considerável na arquitetura do VeriFinger (unibiométrico por impressão digital) e do VeriLook (unibiométrico por face) para a do MegaMatcher, isto é, houve a necessidade da criação de produtos diferentes. Os clientes que já utilizavam VeriFinger tiveram que exportar o banco de dados para um outro, reformulado, e com formato de *templates* diferente, caso quisessem utilizar a multibiometria, oferecida apenas pelo MegaMatcher. Isto aconteceu porque a preocupação inicial era apenas com o algoritmo biométrico, e não com uma arquitetura capaz de evoluir com o mínimo de ônus possível no futuro.

Por esses motivos, é notável que a existência de uma arquitetura e APIs extensíveis e estáveis é de extrema importância. É válido salientar que a arquitetura do BioPass e, consequentemente, as APIs propostas neste trabalho também levaram em consideração outros aspectos, como a segurança de *templates* e a continuidade no controle de acesso, aspectos não explorados nos sistemas multibiométricos mais populares do mercado.

1.4 Metodologia

Após a definição das APIs de segurança e armazenamento, um protótipo foi implementado na linguagem C# para a plataforma .NET, utilizando o Sistema Gerenciador de Banco de Dados (SGBD) PostgreSQL.

Para evoluir com qualidade, um software depende da adoção de um bom processo de software. Este descreve um conjunto de passos para que o objetivo do software seja atingido e é representado por um modelo de processo de software (Sommerville, 2004). Assim, para o desenvolvimento do protótipo, adotou-se um processo iterativo e evolucionário. Tal processo é caracterizado de maneira a permitir que engenheiros de software desenvolvam versões de software cada vez mais completas (Pressman, 2005). Desta forma, a pesquisa para a criação das APIs do presente trabalho se deu por mudanças na estrutura e nas operações da mesma, observando-se as necessidades na prática. Portanto, a construção do protótipo levou à consolidação dessas necessidades, fazendo com que houvesse o desenvolvimento de um software cada vez mais completo.

Como os algoritmos biométricos do BioPass ainda se encontram em desenvolvimento, foram utilizados os algoritmos das DLLs do SDK biométrico de impressão digital e de face desenvolvidos pela Neurotechnology, sob uma licença disponibilizada pela Vsoft Tecnologia.

Os diagramas relacionados às APIs e ao protótipo foram projetados utilizando-se a ferramenta Microsoft Visio 2007 (Microsoft Visio 2007).

1.5 Estrutura do Trabalho

O restante deste trabalho está organizado da seguinte forma.

Capítulo 2 – Fundamentação teórica: apresenta a base teórica necessária para o desenvolvimento do trabalho;

Capítulo 3 – Arquitetura do BioPass: descreve a arquitetura do BioPass, ressaltando sua relação com o domínio do problema e com os objetivos a serem alcançados;

Capítulo 4 – Projeto e Implementação: expõe o processo de projeto e implementação das APIs e do protótipo, bem como os resultados obtidos com sua implementação;

Capítulo 5 – Considerações finais: discute sobre os resultados do trabalho, suas contribuições, trabalhos futuros e conclusões.

Capítulo

2

Fundamentação Teórica

"A teoria sem a prática de nada vale; a prática sem a teoria é cega."

Vladimir Lenin

Este capítulo apresenta, resumidamente, a base teórica necessária para a compreensão dos elementos que constituem a arquitetura de software projetada e as APIs propostas, bem como dos processos de identificação e autenticação multibiométricas a serem utilizados. Neste sentido, as próximas seções apresentam os conceitos essenciais relacionados a sistemas biométricos, multibiometria, segurança de *templates*, sistemas distribuídos e *clusters*.

2.1 Sistemas biométricos

Tecnologias biométricas são métodos automatizados de verificar ou reconhecer a identidade de uma pessoa viva baseados em uma característica fisiológica ou comportamental (Wayman, Jain, Maltoni, & Maio, 2005). Um sistema biométrico de verificação (autenticação) compara a característica biométrica capturada com um modelo de referência de um indivíduo, já armazenado no sistema. Por sua vez, um sistema biométrico de identificação reconhece um indivíduo quando, ao fazer uma busca no banco de modelos de referência, comparando-os com a característica biométrica capturada, encontra seu modelo de referência. Portanto, diz-se que um sistema de verificação conduz uma comparação um-para-um para confirmar que um indivíduo é quem ele diz ser, retornando verdadeiro se sim ou falso caso contrário e, de forma semelhante, diz-se que um sistema de identificação conduz comparações um-para-muitos para estabelecer se um indivíduo está presente

no banco de dados, retornando o identificador de seu registro de referência caso esteja registrado na base (Maltoni, Maio, Jain, & Prabhakar, 2009). A Figura 1 ilustra os processos de registro, verificação e identificação biométricos.

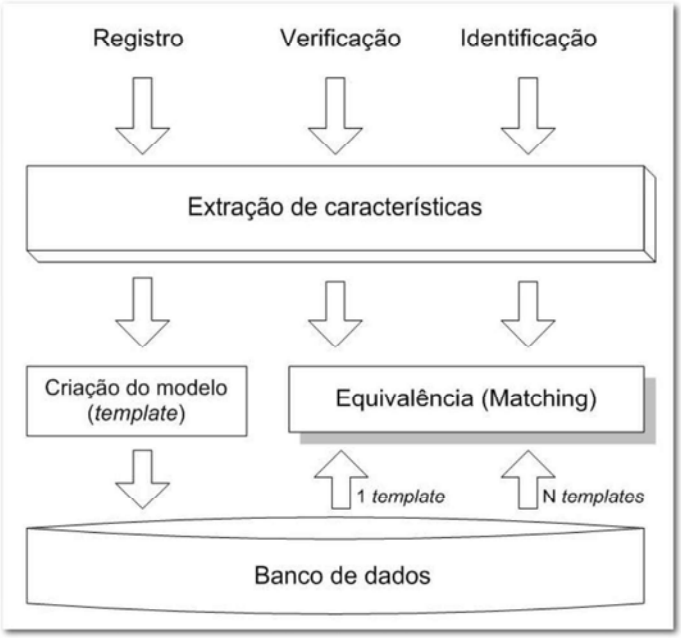

Figura 1. Processos de registro, verificação e identificação biométricos

Qualquer traço humano anatômico ou comportamental pode ser usado em um sistema de identificação biométrica, desde que satisfaça os seguintes requisitos: universalidade, singularidade, permanência e contabilidade (Tabela 1). Na prática, outras questões devem ser consideradas: desempenho, aceitabilidade e evasão (Ross, Nandakumar, & Jain, 2006) (Maltoni, Maio, Jain, & Prabhakar, 2009).

Tabela 1. Propriedades importantes a serem consideradas para a escolha de uma característica na construção de um sistema de identificação biométrica

Propriedade	**Definição**
Universalidade	Cada pessoa deve possuir o traço biométrico.
Singularidade	Quaisquer duas pessoas devem ser suficientemente diferentes em termos do traço biométrico.
Permanência	O traço deve ser invariante (em relação ao critério de *matching*) com o tempo.
Contabilidade	O traço biométrico pode ser medido quantitativamente.
Desempenho	Precisão do reconhecimento, velocidade, requisitos dos recursos, robustez aos fatores operacionais e ambientais.

Aceitabilidade	A que ponto os usuários estão dispostos a aceitar o identificador biométrico em suas vidas diárias.
Evasão	Facilidade com que o sistema biométrico pode ser contornado por meio de métodos fraudulentos.

Dentre as características biométricas mais comuns, temos: impressão digital, geometria da mão, face, íris, voz e assinatura.

2.1.1 Impressão digital

Ainda hoje não há uma explicação científica para a questão da individualidade das impressões digitais (Moenssens, 1971) (Lee & Gaensslen, 2001). Por isso, apesar de "impressão digital" ser popularmente sinônimo de individualidade, esta é apenas uma observação empírica. O conhecimento cientificamente comprovado que se tem sobre as impressões digitais é que elas se formam quando o feto tem aproximadamente sete meses e que suas configurações de cristas não mudam durante a vida do indivíduo, exceto por acidente, como, por exemplo, cortes no dedo. A característica estrutural mais evidente de uma impressão digital é um padrão de intercalação de cristas e vales (Ashbaugh, 1999).

Figura 2. Cristas e vales em uma impressão digital

Hoje, os scanners de impressão digital estão bastante acessíveis e muitas vezes o custo de embarcar um sensor de impressão digital em um sistema (por exemplo, em um computador portátil) tem se tornado cada vez mais baixo para uma grande variedade de aplicações.

Os sistemas de reconhecimento de impressões digitais atuais são adequados para sistemas de autenticação em muitas aplicações, em especial na forense. Múltiplas impressões digitais de uma pessoa proveem informação adicional para permitir identificação em larga escala envolvendo milhões de identidades. Uma desvantagem de sistemas de reconhecimento por impressões digitais em larga escala é a grande quantidade de recursos

computacionais necessários, principalmente para a operação de identificação (Ross, Nandakumar, & Jain, 2006).

Por fim, impressões digitais de uma pequena parte da população podem não servir para identificação automática por fatores genéticos, envelhecimento, razões ambientais ou ocupacionais (através de cortes frequentes nos dedos de um pedreiro, por exemplo).

2.1.2 Geometria da mão

Sistemas de reconhecimento por geometria da mão se baseiam em um número de medidas tiradas da mão humana, incluindo sua forma, tamanho da palma, e comprimento e largura dos dedos [Zunkel, 1999]. Tais características são relativamente invariantes e peculiares a um indivíduo. No entanto, não são muito distintivas.

O sistema de aquisição da imagem requer cooperação do indivíduo para capturar imagens de vista frontal e lateral da palma horizontalmente posicionada em um painel, com os dedos esticados. O espaço de armazenamento do *template* da mão é muito pequeno, sendo este um atrativo para sistemas com limitação de banda e de memória. Entretanto, sistemas baseados na geometria da mão são usados geralmente para verificação, não sendo muito adequados para aplicações de identificação, uma vez que as características extraídas da geometria da mão não são muito distintivas.

Além disso, as informações retiradas da geometria da mão variam durante o período de crescimento do indivíduo. Ainda, a presença de jóias (e.g. anéis) na mão ou deformações ocasionadas por uma artrite, por exemplo, podem se apresentar como desafios para a extração da informação correta da geometria da mão. Outra desvantagem é que a mão é fisicamente grande e, por isso, o *scanner* não pode ser embarcado em certos tipos de dispositivos como *laptop* PCs (Ross, Nandakumar, & Jain, 2006).

2.1.3 Face

A face é um dos traços biométricos mais aceitos porque é um dos métodos mais comuns de reconhecimento que o ser humano usa em suas interações visuais diárias. Além disso, o reconhecimento facial é um método não intrusivo, em termos da forma como o traço é capturado. O desafio de tal método está no desenvolvimento de técnicas tolerantes aos efeitos de envelhecimento, expressões faciais, variações no ambiente da captura da imagem (e.g., variações de iluminação), e pose facial com relação à câmera (Maltoni, Maio, Jain, & Prabhakar, 2009).

As abordagens mais populares ao reconhecimento de face (Li & Jain, 2005) baseiam-se na localização e formato de atributos faciais tais como olhos, sobrancelhas, nariz,

lábios e queixo, e suas relações espaciais, ou na análise global da imagem da face que representa uma face como uma combinação ponderada de um número de faces.

Na literatura especializada é questionado se a face, isoladamente e sem nenhuma informação contextual, é suficiente para reconhecer uma pessoa dentre um alto número de identidades com um nível de confiança extremamente elevado.

Para que um sistema de reconhecimento facial funcione corretamente na prática, ele deve automaticamente (Ross, Nandakumar, & Jain, 2006):

1. Detectar se a face está presente na imagem adquirida;
2. Localizar a face, caso haja uma;
3. Reconhecer a face de um ponto de vista geral, isto é, de várias poses.

2.1.4 Íris

A íris é a região anular do olho, limitada pela pupila na parte interna e pela esclera (parte branca do olho) na parte externa. A textura visual da íris humana é determinada pelos processos morfogenéticos durante o desenvolvimento do embrião e é diferente em cada pessoa e em cada olho (Daugman, 1999). Esta textura se estabiliza durante os dois primeiros anos de vida. No entanto, a pigmentação continua a variar por um extenso período de tempo (Wasserman, 1974). A textura da íris é complexa e traz consigo informações muito distintivas (Daugman, 2004).

O processo de captura da imagem da íris normalmente não necessita que haja qualquer contato. Este processo geralmente envolve a cooperação do usuário, tanto para registrar a imagem da íris centralizada na área de imagem quanto para certificar que a íris está a uma distância predeterminada do plano focal da câmera (Maltoni, Maio, Jain, & Prabhakar, 2009).

O reconhecimento por meio da íris tem se mostrado extremamente preciso e rápido, em imagens de íris de alta resolução e bem capturadas, sendo viável para ser usado em sistemas de identificação de larga escala. Além disso, é possível detectar lentes de contato com íris falsas impressas (Daugman, 1999).

Embora os sistemas de reconhecimento baseados em íris fossem inicialmente caros e necessitassem de uma participação considerável do usuário, os sistemas mais atuais têm se tornado mais amigáveis do ponto de vista da usabilidade e com melhor custo-benefício (Fancourt, et al., 2005). Enquanto tais sistemas têm uma taxa de falsa aceitação (FAR) baixa em comparação aos outros traços biométricos, a taxa de falsa rejeição (FRR) desses sistemas pode ser bastante elevada (Ross, Nandakumar, & Jain, 2006).

2.1.5 Voz

A voz é uma combinação de características biométricas físicas e comportamentais (Campbell, 1997). As características físicas da voz de um indivíduo baseiam-se na forma e tamanho dos elementos usados para sintetizar o som (e.g. aparelho vocal, boca, cavidades nasais e lábios). Essas características físicas não variam em um indivíduo. No entanto, as características comportamentais variam com o tempo devido a diversos fatores como, por exemplo, idade, condições de saúde (e.g., gripe), estado emocional, stress, entre outros.

Nesse contexto, a voz não é um traço muito distintivo e, por isso, não é indicado para uso em um sistema de identificação em larga escala. Além disso, um sinal digital de voz é tipicamente degradado pela qualidade do microfone ou canal de comunicação. Ainda, algumas pessoas possuem o talento de imitar a voz de outras extremamente bem (Maltoni, Maio, Jain, & Prabhakar, 2009).

2.1.6 Assinatura

A forma como uma pessoa assina seu nome é conhecida como uma característica deste indivíduo (Nalwa, 1997). Mesmo que as assinaturas demandem o contato com o instrumento de escrita e um esforço por parte do usuário, elas têm sido aceitas em transações legais, comerciais e do governo como método de autenticação por muito tempo.

A assinatura é uma característica biométrica que muda com o tempo e é influenciada por condições físicas e emocionais do signatário. As assinaturas de algumas pessoas variam substancialmente, uma vez que mesmo sendo adquiridas sucessivamente, elas são significantemente diferentes. Ainda, falsificadores profissionais podem reproduzir assinaturas de outras pessoas, conseguindo enganar o sistema (Harrison, 1981) (Maltoni, Maio, Jain, & Prabhakar, 2009) (Ross, Nandakumar, & Jain, 2006).

2.1.7 Comparação entre as características biométricas

As características biométricas mais comuns, descritas nas seções 2.1.1 a 2.1.6, foram comparadas por (Maltoni, Maio, Jain, & Prabhakar, 2009) em termos das propriedades apresentadas na Tabela 1. A avaliação das características considerou três níveis: alto, médio e baixo. Esses níveis indicam o grau em que uma característica biométrica satisfaz uma determinada propriedade. A Tabela 2 apresenta a comparação feita por (Maltoni, Maio, Jain, & Prabhakar, 2009). Essa comparação é útil na decisão de qual traço deve ser usado em um sistema de reconhecimento biométrico.

Tabela 2. Comparação de traços biométricos comumente usados

Característica Biométrica	Universalidade	Singularidade	Permanência	Contabilidade	Desempenho	Aceitabilidade	Evasão
Impressão digital	Médio	Alto	Alto	Médio	Alto	Médio	Médio
Geometria da mão	Médio	Médio	Médio	Alto	Médio	Médio	Médio
Face	Alto	Baixo	Médio	Alto	Baixo	Alto	Alto
Íris	Alto	Alto	Alto	Médio	Alto	Baixo	Baixo
Voz	Médio	Baixo	Baixo	Médio	Baixo	Alto	Alto
Assinatura	Baixo	Baixo	Baixo	Alto	Baixo	Alto	Alto

2.2 Multibiometria

Sistemas multibiométricos consolidam a evidência apresentada por múltiplas fontes de informação biométrica. Essas fontes incluem múltiplos sensores (multi-sensor), múltiplas representações e algoritmos de *matching* (multi-algoritmo), múltiplas amostras do mesmo traço biométrico (multi-amostra), múltiplas instâncias de um traço biométrico (multi-instância) e múltiplos traços biométricos (multi-modal) (Nandakumar, 2008).

O uso de múltiplos sensores significa que um traço biométrico deve ser capturado por diferentes tipos de sensores (por exemplo, sensor ótico e sensor de distância para captura de face). Múltiplas representações e algoritmos de *matching* referem-se à extração de características. Por exemplo, minúcias e textura são diferentes características que podem ser extraídas de uma mesma imagem de impressão digital. Uma forma de ter múltiplas amostras de um mesmo traço biométrico é capturá-lo mais de uma vez. Exemplos de múltiplas instâncias de um traço biométrico são as íris dos olhos esquerdo e direito de um indivíduo. O uso de múltiplos traços biométricos é associado com a captura de mais de uma característica biométrica.

De acordo com (Nandakumar, 2008), as vantagens que os sistemas multibiométricos apresentam sobre os sistemas unibiométricos incluem:

- Melhora da precisão global do sistema biométrico, por causa da combinação das evidências obtidas de fontes diferentes;
- Redução da FTER (*Failure To Enroll Rate* – Taxa de Falha de Registro) e da FTCR (*Failure To Capture Rate* – Taxa de Falha de Captura) (um feri-

mento no dedo, por exemplo, não vai proibir que o usuário se identifique porque outro traço biométrico ainda pode ser usado);

- Resistência a ataques de *spoofing,* pelo fato de ser difícil fraudar simultaneamente múltiplas fontes biométricas.

Sistemas multibiométricos também têm desvantagens em comparação aos sistemas unibiométricos. Como exemplo, eles geralmente exigem um registro mais longo, são mais caros, e precisam de recursos computacionais e de armazenamento melhores.

Há vários fatores que devem ser considerados ao se projetar um sistema multibiométrico. Esses fatores incluem a escolha do número de comportamentos biométricos e tipos de sensores, o nível no sistema biométrico em que as informações fornecidas pelas múltiplas características biométricas devem ser integradas, a sequência em que essa informação é obtida e processada, a metodologia adotada para fundir a informação, e a questão custo versus desempenho de *matching* (Anwar, Rahman, & Azad, 2009) (Nandakumar, 2008).

2.2.1 Sequência de aquisição e processamento

A sequência de aquisição em um sistema multibiométrico diz respeito à ordem em que as várias fontes de evidência serão adquiridas de um indivíduo. Essa questão não é considerada no caso de sistemas que usam múltiplos algoritmos, pois uma única amostra biométrica se faz necessária.

A aquisição pode ocorrer sequencialmente ou simultaneamente. A forma sequencial, serial ou em cascata é a mais comum. Nela, cada fonte de evidência é obtida independentemente com um curto intervalo de tempo entre as aquisições. A aquisição simultânea ocorre quando duas ou mais informações biométricas podem ser obtidas simultaneamente. A face e a íris, por exemplo, podem ser capturadas simultaneamente por duas câmeras alojadas em uma mesma unidade (Ross, Nandakumar, & Jain, 2006).

A sequência de processamento é a ordem em que a informação adquirida é processada para gerar uma decisão. No modo serial ou sequencial, o processamento das informações acontece sequencialmente. Neste modo, o tempo de processamento pode ser efetivamente reduzido se uma decisão é feita antes de seguir por todos os subsistemas biométricos.

Por outro lado, no modo paralelo, cada subsistema processa sua informação independentemente ao mesmo tempo e a informação processada é combinada usando-se um esquema de fusão apropriado.

2.2.2 Níveis de fusão

Uma das questões fundamentais no projeto de um sistema multibiométrico é determinar o tipo de informação que deve ser fundida (Nandakumar, 2008). Dependendo do tipo de informação, o esquema de fusão pode ser classificado mais amplamente como fusão antes do *matching* e fusão depois do *matching*.

No nível de fusão antes do *matching*, a informação de múltiplas fontes biométricas é integrada em nível de sensor ou em nível de característica.

A fusão em nível de sensor só pode acontecer se as fontes forem amostras do mesmo traço biométrico obtidas de múltiplos sensores compatíveis ou se forem múltiplas instâncias do mesmo traço biométrico obtidas pelo mesmo sensor. Múltiplas imagens 2D de uma face obtidas de diferentes pontos de vista, por exemplo, podem se juntar para formarem um único modelo 3D da face (Liu & Chen, 2003).

A fusão em nível de característica é a combinação de diferentes conjuntos de características extraídas de múltiplas fontes biométricas. Como exemplo, temos a fusão de características extraídas de múltiplas impressões de um mesmo dedo.

O nível de fusão depois do *matching* pode ser dividido em quatro categorias: seleção dinâmica do classificador, fusão em nível de decisão, fusão em nível de *score* e fusão em nível de classificação (Ross, Nandakumar, & Jain, 2006) (Nandakumar, 2008).

Na seleção dinâmica do classificador, é escolhida a fonte biométrica que mais provavelmente retornará a decisão correta para o padrão de entrada específico.

Na fusão em nível de decisão, apenas as decisões de cada fonte biométrica estão disponíveis. Os métodos para a fusão em nível de decisão podem incluir regras de "AND" e "OR" (Daugman, 2010), votação pela maioria ponderada (Kuncheva, 2004), fusão por decisão Bayesiana (Xu, Krzyzak, & Suen, 1992), entre outros.

Na fusão em nível de *score*, *scores* de *matching* provenientes da saída de diferentes algoritmos de *matching* são consolidados para que se chegue a uma decisão final de reconhecimento. *Score* de *matching* é a medida de similaridade resultante do *matching* entre a característica biométrica de entrada e o *template* armazenado.

A fusão é feita em nível de classificação quando a saída de cada sistema biométrico é um subconjunto de possíveis identidades em ordem decrescente de coincidência.

A maioria dos sistemas multibiométricos faz a fusão da informação em nível de *score* ou de decisão.

2.2.3 Desafios no projeto de sistemas multibiométricos

Projetar um sistema multibiométrico não é uma tarefa fácil (Nandakumar, 2008) devido aos seguintes fatores: heterogeneidade das fontes de informação, complexidade de fusão, capacidade discriminativa variada e a correlação entre as fontes.

Em um sistema típico de reconhecimento biométrico, a quantidade de informação disponível para o sistema vai diminuindo, isto é, se comprimindo, à medida que a informação biométrica é capturada e vai passando pelo extrator de características até o momento da decisão, quando é feito o *matching*. Neste sentido, acredita-se que a integração em estágios iniciais de processamento é mais eficiente. No entanto, a fusão em nível de sensor, por exemplo, nem sempre é possível devido à incompatibilidade ou heterogeneidade do conteúdo da informação.

A complexidade do algoritmo de fusão pode chegar a anular as vantagens da fusão, mesmo quando as fontes de informação são compatíveis (e.g., duas impressões do mesmo dedo). Dependendo do nível onde a fusão ocorre, será necessário um maior processamento devido à complexidade gerada como, por exemplo, um algoritmo mais complexo – e menos eficiente – de *matching* para os dados fundidos.

A quantidade de informação discriminatória de cada fonte biométrica pode ser muito diferente. Assim, se a fusão for feita com pesos iguais para cada fonte, por exemplo, a precisão do sistema multibiométrico provavelmente diminuirá se for feita a fusão entre uma fonte da qual é possível extrair muitos dados discriminatórios e outra da qual são extraídos apenas poucos dados discriminatórios.

As fontes biométricas de um sistema multibiométrico podem não ser estatisticamente independentes. Sistemas que usam traços fisicamente relacionados (e.g., fala e movimento labial), múltiplos algoritmos de *matching* para o mesmo traço biométrico ou mesmo conjunto de características, ou múltiplas amostras do mesmo traço, são exemplos de sistemas multibiométricos em que as fontes de informação estão correlacionadas. A fusão de evidências independentes geralmente fornece uma maior precisão com relação à fusão de fontes correlacionadas.

Além dos quatro fatores descritos acima, é importante ressaltar que bom desempenho e boa precisão são os dois requisitos mais importantes dos sistemas uni- e multibiométricos. Enquanto nos sistemas multibiométricos a utilização de mais fontes de evidência proporciona mais precisão de reconhecimento, ela poderá também diminuir o desempenho do sistema. Por isso, encontrar um equilíbrio entre esses dois requisitos é uma tarefa árdua.

2.3 Segurança de Templates

A segurança de *templates* é uma área de estudo muito importante no projeto de sistemas biométricos, pois o ataque contra *templates* é um dos mais potencialmente onerosos. Diferentemente das senhas, *templates* roubados não podem ser revogados.

Os ataques a *templates* podem conduzir a quatro vulnerabilidades (Nandakumar, 2008) a seguir.

1. O *template* pode ser substituído pelo *template* de um impostor, a fim de se obter um acesso não autorizado;
2. Um mascaramento físico pode ser criado através do *template*, a fim de se obter um acesso não autorizado;
3. O *template* roubado pode ser repassado diretamente para o sistema de *matching*, a fim de se obter acesso não autorizado;
4. Os *templates* podem ser usados para *cross-matching* em bancos de dados diferentes de forma a disfarçadamente rastrear uma pessoa sem seu consentimento.

Neste contexto, a fim de armazenar os *templates* com segurança, tal que tanto a segurança da aplicação quanto a privacidade dos usuários não sejam comprometidos por ataques de adversários, *templates* biométricos ou seus dados brutos (e.g. imagem da impressão digital) não devem ser armazenados na forma *plaintext* (não criptografada) e técnicas à prova de falhas devem ser empregadas.

Sistemas multibiométricos que usam a evidência de múltiplas fontes podem melhorar a precisão de reconhecimento. No entanto, eles exigem o armazenamento de múltiplos *templates* de diferentes fontes biométricas para o mesmo usuário. Por este motivo, a segurança de *templates* é ainda mais crítica em sistemas multibiométricos, onde é essencial a segurança de múltiplos *templates* de um usuário.

2.3.1 Esquemas de proteção de templates

A utilização de abordagens genéricas como, por exemplo, sistemas criptográficos de chave pública como o RSA (RSA Laboratories, 1999) ou criptografia simétrica como o AES (NIST, 2001) (NIST, 2001) (NIST, 2001) não é uma boa solução para proteger *templates* devido às duas razões (Nandakumar, 2008) descritas a seguir.

A primeira razão está no fato de a criptografia não ser uma função suave. Assim, pequenas diferenças nos valores dos conjuntos de características extraídas do dado biométrico puro levariam a uma grande diferença nas características criptografadas resultantes. Portanto, como a extração de características de várias aquisições diferentes do mesmo tra-

ço biométrico não resulta os mesmos valores, não seria possível fazer a comparação dos valores no domínio da criptografia e seria necessário descriptografar o *template* previamente armazenado para executar a operação de *matching*. No entanto, em um sistema de identificação que conduz comparações um-para-muitos, a descriptografia de *templates* afetaria consideravelmente a velocidade da identificação. Além disso, o *template* descriptografado ficaria exposto na memória durante todas as tentativas de autenticação.

A segunda razão é que a segurança do esquema de criptografia depende de uma chave de descriptografia. Esta chave precisa ser armazenada com segurança no sistema e, se ela for comprometida, o *template* não mais estará seguro, ou seja, a segurança do *template* depende da segurança da chave.

É por causa das razões apresentadas que algoritmos padrão de criptografia, sozinhos, não são adequados para proteger *templates* biométricos. Logo, fazem-se necessárias técnicas projetadas para considerar especificamente a variação intra-usuário dos dados biométricos.

Os esquemas de proteção de *templates* propostos na literatura podem ser amplamente classificados em duas categorias (Jain, Nandakumar, & Nagar, 2008): transformação de característica e criptossistema biométrico.

Na abordagem de transformação de característica, uma função de transformação F é aplicada ao *template* biométrico T e apenas o *template* transformado (F(T, K)) é armazenado na base de dados. A chave K pode ser randômica ou derivada de uma senha. A mesma função de transformação é aplicada ao *template* T_2 de entrada, isto é, àquele gerado para tentar autenticar o usuário no sistema ($F(T_2, K)$). Assim, os *templates* transformados (F(T, K) e $F(T_2, K)$) são comparados diretamente.

Dependendo das características da função de transformação F, os esquemas de transformação de características podem ser categorizados como transformações *salting* (inversíveis) ou não-inversíveis.

Na transformação *salting*, F é inversível, isto é, se um atacante obtiver acesso à chave e ao *template* transformado, ele pode recuperar o *template* biométrico original ou algo próximo a ele. Assim, a segurança da transformação *salting* baseia-se na segurança da chave ou senha. Por outro lado, transformações não-inversíveis tipicamente aplicam uma função de um caminho (somente ida) no *template*, sendo computacionalmente difícil de inverter um *template* transformado, mesmo conhecendo a chave.

Os criptossistemas biométricos foram desenvolvidos originalmente com o propósito de prover segurança à chave criptográfica usando características biométricas ou diretamente gerando uma chave criptográfica a partir das características biométricas. No entanto, os criptossistemas biométricos também podem ser usados como um mecanismo de

proteção do *template*. Em tais sistemas, algumas informações públicas do *template* biométrico são armazenadas. Essas informações são consideradas como dados de ajuda. Elas não revelam informação significante sobre o *template* biométrico original, mas são necessárias durante o processo de *matching*, para extrair uma chave criptográfica das características biométricas de entrada. O *matching* é feito indiretamente pela verificação da validade da chave extraída. Geralmente, técnicas de correção de erros são usadas para controlar as variações intra-usuário.

Os criptossistemas biométricos podem ser classificados em sistemas de amarração de chave e sistemas de geração de chave, dependendo de como os dados de ajuda são obtidos.

Quando eles são obtidos pela amarração de uma chave com o *template* biométrico, temos um criptossistema biométrico de amarração de chave. É importante salientar que apenas com os dados de ajuda é computacionalmente difícil recuperar tanto a chave quanto o *template*. A operação de *matching* em sistemas de amarração de chave envolve a recuperação da chave pelos dados de ajuda usando-se as características biométricas de entrada. Se os dados de ajuda resultarem do *template* biométrico e a chave criptográfica for diretamente gerada dos dados de ajuda e das características biométricas de entrada, temos um sistema biométrico de geração de chave.

Com exceção da transformação *salting*, nenhum dos outros esquemas de proteção de *templates* necessita de uma informação secreta, como uma chave, que deva ser armazenada com segurança ou apresentada durante o *matching*.

2.4 Sistemas distribuídos e Clusters

Um sistema distribuído é uma coleção de computadores autônomos que se mostram a seus usuários como um único sistema coerente (Tanenbaum & Steen, 2002). Em outras palavras, um sistema distribuído é formado por um conjunto de máquinas autônomas, mas seus usuários acreditam estar usando um único sistema.

Um sistema distribuído deve: (1) conectar usuários e recursos facilmente; (2) ser transparente, isto é, esconder o fato de os recursos estarem distribuídos pela rede; (3) ser aberto, ou seja, oferecer serviços de acordo com regras padrão que descrevem a sintaxe e semântica desses serviços; e (4) ser escalável, isto é, ser capaz de abarcar novos usuários e recursos, geograficamente distribuídos ou não, e ser fácil de gerenciar, mesmo que compreenda várias organizações administrativas autônomas (Tanenbaum & Steen, 2002).

Considere-se um banco brasileiro que possui filial na Argentina. O sistema de autoatendimento (ATM) deste banco permite que usuários do Brasil acessem suas contas na Argentina e vice-versa (1). Quando um brasileiro tenta se autenticar no ATM argentino, o

serviço local consulta o serviço brasileiro para permitir ou recusar o acesso. Caso a autenticação ocorra com sucesso, o brasileiro pode consultar seu saldo (convertido para pesos argentinos), sacar pesos, transferir dinheiro para uma conta do mesmo banco (brasileira ou argentina), etc. (2). Isto é possível porque existe uma interface bem definida, disponibilizada por ambos os serviços, que permite a recuperação dos dados necessários às operações bancárias (3). Se uma filial do banco for criada no Uruguai, sem maiores esforços, o critério de funcionamento continuará o mesmo, podendo brasileiros, argentinos e uruguaios acessarem suas contas em quaisquer dos três países (4).

Um *cluster* é um tipo de sistema de processamento paralelo ou distribuído, que consiste em um conjunto de computadores autônomos interconectados trabalhando juntos como um recurso computacional único e integrado. Um nó pode ser um sistema uni- ou multiprocessado (computadores pessoais, estações de trabalho ou um sistema de multiprocessadores simétricos) com memória, facilidades de entrada e saída, e um sistema operacional. Um *cluster* geralmente se refere a dois ou mais computadores (nós) conectados. Os nós podem existir em um único cômodo ou estarem fisicamente separados e conectados por uma LAN (*Local Area Network*). Um *cluster* de computadores interconectados pode aparecer como um único sistema para usuários e aplicações (Buyya, 1999).

Os nós em um *cluster* usam o mesmo sistema operacional e geralmente não são controlados individualmente; ao invés disso, deve-se usar um escalonador especial de tarefas (Rauber & Rünger, 2010).

Os *clusters* podem oferecer as seguintes funcionalidades a um custo relativamente baixo: alto desempenho, expansibilidade e escalabilidade, alta taxa de transferência, e alta disponibilidade.

Um sistema de reconhecimento (multi) biométrico, por exemplo, que conduza comparações um-para-muitos e que possua um banco de dados com milhares ou até milhões de *templates* pode operar em *cluster* a fim de elevar o desempenho do sistema, distribuindo o processamento das operações de *matching* entre os nós. Além disso, caso um dos nós venha a falhar, o escalonador de tarefas do *cluster* pode, por exemplo, redistribuir as tarefas do nó que falhou entre os outros nós, reforçando a disponibilidade de tal sistema.

2.5 Trabalhos relacionados

Diversos trabalhos abordando o reconhecimento multibiométrico podem ser encontrados na literatura (Abate, Nappi, Riccio, & Marsico, 2007) (Lee, Kim, Kwak, Kim, & Yoon, 2007) (Varchol, Levicky, & Juhar, 2008). No entanto, eles geralmente descrevem a análise e projeto de um sistema multibiométrico específico e limitado a determinados pro-

cessos ou características biométricas, ou mantêm o foco em algoritmos multibiométricos restritos a determinadas características. Neste contexto, a arquitetura e APIs desenvolvidas no presente trabalho se propõem a compreender características não abordadas em conjunto pela literatura, como a facilidade na inserção de novos traços biométricos, a preocupação com a segurança específica para *templates* e o controle de acesso contínuo.

Considerando os aspectos mencionados, poucos – porém significantes – trabalhos relacionados puderam ser encontrados na literatura. As subseções a seguir descrevem cada um desses trabalhos.

2.5.1 BioAPI

O Consórcio da BioAPI (*BioAPI Consortium*) foi fundado visando ao desenvolvimento de uma API capaz de prover independência de plataforma e dispositivo para desenvolvedores de aplicações e provedores de serviços biométricos.

A especificação da BioAPI tem o objetivo de prover um modelo de autenticação biométrica genérico adequado para qualquer forma de tecnologia biométrica. Ela define a interface de programação de aplicativos (API) e a interface do provedor de serviço (SPI) para uma interface de tecnologia biométrica padrão.

Há duas versões da BioAPI. Uma delas é um padrão nacional americano (ANSI/INCITS 358-2002, também conhecida como BioAPI 1.1). A outra é um padrão internacional (ISO/IEC 19784-1:2005, também conhecida como BioAPI 2.0). O padrão BioAPI 1.1 contem a mesma especificação da BioAPI 1.1 originalmente desenvolvida pelo *BioAPI Consortium*. O padrão BioAPI 2.0, por sua vez, é uma versão completamente nova, desenvolvida pelo comitê de padrões internacionais para biometria da ISO (*International Organization for Standardization*) (ISO/IEC JTC1 SC37).

A especificação da BioAPI 2.0 descreve um modelo de arquitetura que permite que os componentes de um sistema biométrico sejam fornecidos por diferentes fabricantes e interajam por meio de APIs totalmente definidas.

Como se pode notar, a BioAPI está bem consolidada. No entanto, deve-se atentar para certas questões. Primeiramente, definir requisitos de segurança para aplicações biométricas e provedores de serviços está fora do escopo desta especificação, embora algumas informações relacionadas à forma como a API pretende suportar boas práticas de segurança estejam inclusas. Além disso, nenhum suporte explícito a biometria multimodal é fornecido. Por fim, as interfaces da BioAPI estão especificadas na linguagem C, e não de uma forma mais independente.

2.5.2 Verificação Contínua Usando Biometria Multimodal

(Sim, Zhang, Janakiraman, & Kumar, 2007) propõem e implementam um sistema multimodal de verificação biométrica que continuamente verifica a presença de um usuário em um computador protegido. Tal sistema é embutido em um Sistema Operacional (Linux kernel 2.4.26), deixando-o sensível à presença do usuário e, por conseguinte, permitindo-o protejer suas sessões de login.

As modalidades utilizadas no sistema são face e impressão digital, sendo a verificação contínua feita para ambas as características, uma vez que se utiliza uma câmera convenientemente posicionada para a verificação facial e o mouse *SecuGen* com sensor de impressão digital ergonomicamente integrado para a verificação da impressão digital – dois sensores passivos, isto é, não intrusivos.

Os autores apontam que sua teoria pode ser prontamente estendida para incluir mais modalidades no futuro, levando em consideração que utilizam um novo modelo de fusão em nível de *score* por eles proposto. Este modelo é chamado de modelo Holístico e utiliza um Modelo Oculto de Markov que considera uma sequência de estados (Seguro, Atacado), emitindo observações a cada instante no tempo. Por isso, afirmam que modalidades adicionais podem ser incorporadas bastando apenas que se adicione o verificador da nova modalidade ao sistema.

No entanto, além deste método de fusão, não é definida ou apresentada uma arquitetura ou API que revele o nível de facilidade e/ ou estensibilidade para a adição de novas características. Além disso, o método proposto visa à integração de características desde que estas venham a fazer parte da verificação contínua.

2.5.3 MUBAI

(Abate, Marsico, Riccio, & Tortora, 2010) propõem uma arquitetura biométrica multiagente para Inteligência Ambiental (Multiagent Biometrics for Ambient Intelligence – MUBAI), que especifica a composição de módulos biométricos em um sistema de reconhecimento.

Um agente é considerado um fragmento de software que age em nome de um usuário ou outro programa, tendo a autoridade de escolher a ação correta de tempos em tempos. Assim, agentes não serão necessariamente invocados por uma tarefa ou usuário, mas podem se ativar, por exemplo, de acordo com técnicas de sensibilização de contexto. Em geral, agentes não são capazes de atingir um objetivo sozinhos e precisam comunicar-se entre si, formando um sistema multiagente. Nesse trabalho, são usados os agentes autô-

nomos, subsistemas que pegam a entrada e extraem características salientes a serem utilizadas em seus próprios processos de reconhecimento.

Na prática, nem todos esses agentes, sozinhos, são igualmente confiáveis. Desta forma, os índices de Confiabilidade de Resposta do Sistema (SRR – *System Response Reliability*) permitem a avaliação da confiabilidade de cada resposta de um agente, sendo uma resposta considerada para a fusão apenas se o valor SRR correspondente for maior do que um limiar de característica do agente.

A arquitetura do MUBAI considera um sistema multibiométrico como um sistema multiagente. Ela possui dois tipos de agentes: os Agentes Classificadores e o Agente Supervisor. Os Agentes Classificadores capturam dados biométricos, executam procedimentos de reconhecimento específicos que demandam comunicação extensiva com os outros agentes classificadores, além de produzirem uma saída de suas próprias listas de candidatos e, eventualmente, realizarem a atualização de *templates*. O Agente Supervisor, por sua vez, recebe a lista produzida pelos Classificadores e as usa para produzir um resultado de reconhecimento e eventualmente atualizar os parâmetros dos Classificadores. Cada agente precisa de um procedimento de configuração para se alinhar com os outros agentes em um sistema inteiramente operacional. Atualização de *templates* e treinamento em geral são aspectos a serem considerados para este alinhamento.

Os autores implementaram uma instância da arquitetura do MUBAI com quatro módulos implementando diferentes técnicas de reconhecimento facial (agentes classificadores), e um módulo supervisor (agente supervisor). Os resultados obtidos mostraram que a arquitetura do MUBAI pode ser regulada dinamicamente, de forma que novos agentes podem ser adicionados de tempos em tempos ao sistema e efetivamente treinados online.

2.5.4 HUMABIO

O projeto HUMABIO (*Human Monitoring and Authentication using Biodynamic Indicators and Behavioural Analysis*) (HUMABIO) combina novos tipos de características biométricas com os sensores mais atuais para elevar a segurança em várias aplicações. Portanto, o objetivo do projeto é desenvolver um sistema de monitoramento e autenticação biométrica modular, robusta e multimodal, que utilize perfil fisiológico biodinâmico, isto é, único para cada indivíduo, além de avançar no estado da arte da biometria (face, voz, forma de andar e antropometria baseada no assento).

(Damousis, Tzovaras, & Bekiaris, 2008) afirmam que a natureza dessas características fisiológicas permite a autenticação contínua de uma pessoa em um ambiente controlado, apresentando, assim, um maior desafio para potenciais atacantes. Além disso, o HUMABIO permite checagem de vivacidade do usuário e autenticação não intrusiva, uma

vez que características biométricas como a forma de andar sugerem autenticação ou até mesmo identificação em movimento, construindo um ambiente de inteligência. Neste sentido, os autores apontam que o HUMABIO envolve dois conceitos:

1. A melhoria da autenticação biométrica através do uso de novos tipos de biometria que descrevem a fisiologia própria do usuário, sua cooperação com as características biométricas comportamentais existentes, e a melhora de soluções de sistemas largamente usados;
2. A melhora da segurança através da minimização de acidentes relacionados a operadores humanos em operações críticas.

Os autores apontam, ainda, que a arquitetura de sistema projetada é modular, aberta e eficiente, podendo ser usada em diferentes aplicações e sistemas do HUMABIO. Neste sentido, três planos-pilotos foram projetados de forma a demonstrar versatilidade e modularidade extensiva do sistema, bem como prover a avaliação de desempenho em cenários de aplicação realísticos.

O projeto não foi implementado como um todo e encontra-se em desenvolvimento, devendo utilizar novas modalidades biométricas com o objetivo de superar as desvantagens das soluções biométricas atuais, principalmente os protocolos que os usuários são obrigados a seguir. Por fim, afirma-se que, dentre outras inovações, o HUMABIO irá suportar autenticação contínua de indivíduos e o monitoramento de parâmetros fisiológicos para certificar o estado normal de operadores de processos críticos.

2.5.5 BioID

A tecnologia multimodal do BioID (BioID, 2010) (BioID, 2010) utiliza face, íris e voz como características biométricas, sendo a autenticação realizada por qualquer combinação dos três traços. Afirma-se que o BioID oferece o primeiro serviço de autenticação biométrica multimodal baseado em *cloud computing* para Web e dispositivos móveis do mundo. (BioID, 2010) (BioID, 2010) aponta também que o BioID é uma identidade online universal, que está em conformidade com os padrões globais de identidade da Internet e provê uma forma de autenticação conveniente e altamente segura.

Na vida real, os usuários possuem apenas um nome. No entanto, na Internet, eles possuem várias identidades, que acabam sendo numerosas e difíceis de ser controladas por um humano. Para resolver este problema, surgiram as tecnologias de gerenciamento de identidades. OpenID (OpenID) é uma dessas tecnologias, que atua reduzindo o número de identidades de um indivíduo.

Neste contexto, o BioID emprega o conceito do OpenID para interoperabilidade, enquanto gerencia a informação biométrica armazenada através de sua tecnologia biomé-

trica. Assim, através do OpenID e outros sistemas de identificação, o BioID permite uma autenticação biométrica na Web do tipo *single sign-on* (SSO), isto é, possibilidade de autenticar-se apenas uma vez de forma que o acesso é então autorizado automaticamente às diversas aplicações externas, sem a necessidade de recordar *login* e senha de cada sistema.

Segundo (BioID, 2010), a plataforma de autenticação biométrica como serviço (*Biometric authentication as a Service* – BaaS) do BioID oferece alta disponibilidade e confiança a seus usuários.

Com relação à parte interna do sistema, o BioID é compatível com a BioAPI 1.1, suportando, entre outras, as seguintes funções principais: captura, criação de *templates* e registro. Neste sentido, o BioID herda as limitações da BioAPI 1.1.

Capítulo

3

Arquitetura do BioPass

"Se os fatos não se adéquam à teoria, modifique os fatos."

Albert Einsten

Com o objetivo de permitir a construção e evolução de sistemas uni e multi-biométricos sem maiores complicações, uma arquitetura multibiométrica para controle de acesso foi projetada (Oliveira, Motta, & Batista, 2010) (Oliveira, Motta, & Batista, 2010). Esta arquitetura visa à construção de sistemas biométricos de controle de acesso extensíveis, isto é, prontos para a inserção de novos sensores e características biométricas sem a necessidade de serem remodelados ou adaptados. Remodelagem e adaptação de software tornam sua evolução um processo custoso, podendo ainda ocasionar perdas consideráveis em desempenho.

Outra característica da arquitetura é a de permitir a construção de sistemas distribuídos e, também, de sistemas que possam ser usados em *clusters* de estações de trabalho a fim de melhorar o desempenho das comparações um-para-muitos do processo de identificação.

A Figura 3 ilustra o diagrama UML de componentes da arquitetura do BioPass (Oliveira, Motta, & Batista, 2010) (Oliveira, Motta, & Batista, 2010). Há várias definições para componentes. Algumas são até mais antigas do que a Engenharia de Software (McIlroy, 1968). Neste trabalho, considera-se que componentes são artefatos de software ou unidades de composição com interfaces bem definidas e dependência de contexto explícita.

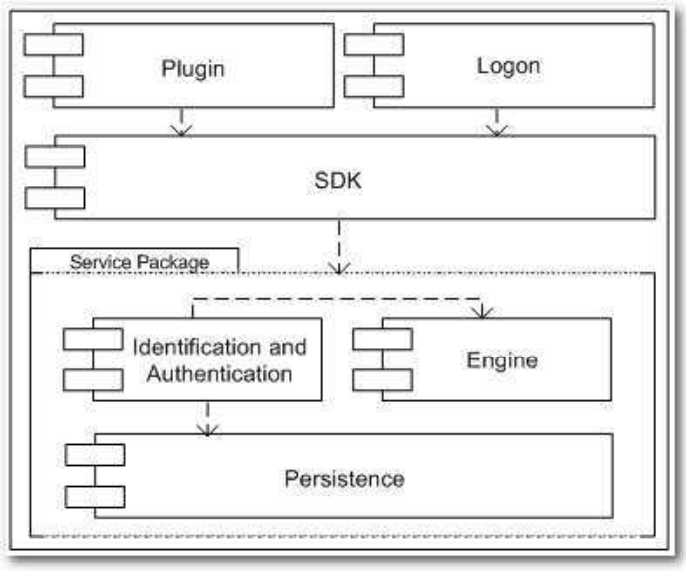

Figura 3. Arquitetura multibiométrica do BioPass

As subseções a seguir descrevem os componentes da arquitetura apresentada na Figura 3, bem como as causas das relações de dependência, quando estas existirem. Ainda, é apresentada a API dos componentes Engine (segurança) e Persistence (armazenamento), descrevendo seus tipos e formatos de dados, interfaces e suas operações.

3.1 Persistence

O componente de persistência é responsável pelo armazenamento e recuperação de dados biométricos. Este componente abstrai a forma de armazenamento dos dados (e.g., em arquivos ou usando um SGBD) e fornece meios para realização de consultas customizadas de recuperação e armazenamento de dados.

A customização de consultas leva em consideração opções de definição e recuperação de registros por classe biométrica, e de seleção de faixas de registros a fim de auxiliar no processo de divisão de tarefas entre os nós de um *cluster* de estações de trabalho.

É importante salientar que este componente considera o armazenamento dos dados biométricos separado do armazenamento dos dados da aplicação, uma vez que o controle de acesso independe da aplicação, por ambos executarem tarefas distintas.

3.1.1 Persistence API

O componente de persistência provê a interface *IBioPassPersistence*, que possui operações de registro, recuperação, remoção e atualização de *templates* multibiométricos. Estas operações abstraem a forma de armazenamento dos dados. A Figura 4 apresenta a interface *IBioPassPersistence*, com suas operações.

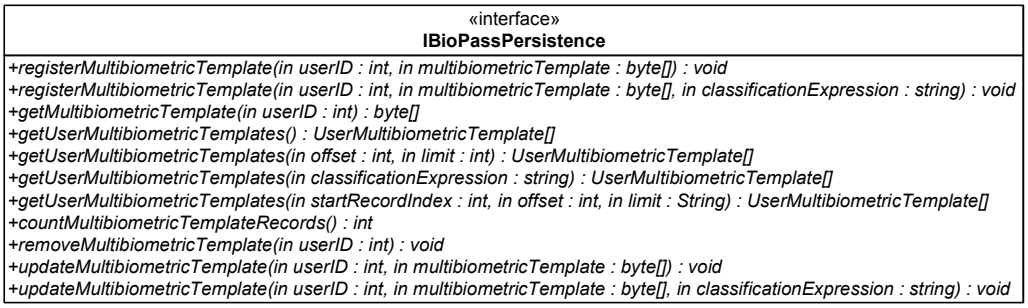

Figura 4. Interface IBioPassPersistence

A operação de recuperação de *templates* (*getUserMultibiometricTemplates*) apresenta versões que auxiliam a implementação de *clusters* de estações de trabalho a fim de elevar o desempenho do sistema de identificação por facilitar a distribuição do processamento das operações de *matching* entre os nós. Esta operação permite obter subconjuntos de registros do banco de dados. Cada nó do *cluster* seria, então, responsável pela execução do *matching* no subconjunto a ele designado.

Por questões de organização, a interface *IBioPassPersistence* é detalhada no Apêndice A.1.

3.1.1.1 Formatos e tipos de dados

A fim de padronizar o formato dos *templates* multibiométricos, bem como de permitir e até mesmo facilitar a evolução dos sistemas através da adição de novas características biométricas, foi criado o padrão de formato apresentado na Figura 5. Este padrão é dividido em duas partes principais: o cabeçalho (*header*) e o corpo (*body*).

O cabeçalho do *template* deve possuir as informações gerais a respeito deste. São elas:

- Tamanho (*size*): corresponde aos primeiros quatro bytes, espaço suficiente reservado ao armazenamento do tamanho (em bytes) total do *template*, incluindo esses quatro bytes, ou seja, o valor armazenado pelo tama-

nho corresponderá ao tamanho do cabeçalho somado ao tamanho do corpo;

- Versão (*version*): a versão do *template* deve ser armazenada no byte seguinte ao tamanho.

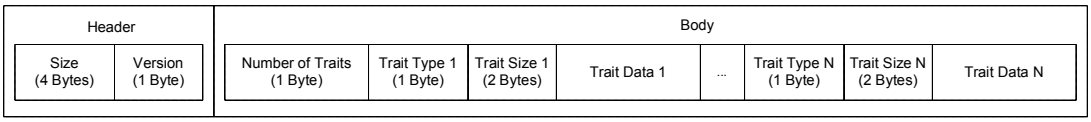

Figura 5. Formato dos templates multibiométricos

O corpo do *template* multibiométrico é formado pelo número de traços mais três partes, que devem ser repetidas atomicamente n vezes, sendo n o número de traços. O número de traços (*number of traits*) armazena, em um byte, o número de características biométricas cujos *templates* unibiométricos estão armazenados no corpo do *template* multibiométrico. As três partes restantes, que representam informações relevantes a um *template* unibiométrico, são descritas a seguir.

- Tipo de traço (*trait type*): armazena, em um byte, um valor que corresponderá ao tipo de traço;
- Tamanho do traço (*trait size*): dois bytes correspondentes ao espaço, em bytes, ocupado pelo *template* unibiométrico em questão, isto é, o tamanho dos dados do traço (*trait data*);
- Dados do traço (*trait data*): dados do *template* unibiométrico.

Neste contexto, os *templates* multibiométricos devem ser considerados como *arrays* de bytes, organizados de acordo com o padrão descrito acima. Além dos *arrays* de bytes, o componente Persistence usa de um tipo de dado chamado *UserMultibiometricTemplate*. Este tipo é ilustrado na Figura 6.

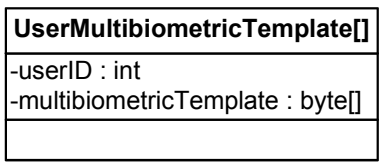

Figura 6. Tipo de dado UserMultibiometricTemplate

O tipo *UserMultibiometricTemplate* encapsula o identificador do usuário (*userID*) e o *template* multibiométrico (*multibiometricTemplate*). O identificador do

usuário corresponde ao identificador no banco de dados da aplicação, uma vez que este componente de persistência comporta apenas dados relacionados a controle de acesso.

3.2 Engine

O componente Engine compreende a parte multibiométrica da arquitetura. Ele inclui algoritmos de processamento de imagens e de reconhecimento uni e multibiométricos. Este componente permite, ainda, a adição de novas características sem necessidade de mudanças ou reengenharia no código já existente, para a construção de sistemas multimodais. A partir dele é possível, também, criar *templates* uni e multibiométricos, obter classes de dados biométricos e realizar o *matching* entre *templates*. Além disso, o Engine dá suporte à autenticação contínua, através da detecção de traços biométricos, e à segurança específica para *templates*, desde que não utilize um algoritmo de transformação *salting*.

O componente Engine possui limiares configuráveis de aceitação e de rejeição uni e multibiométricos, valor do peso exercido pelas características no *matching* e limiares de qualidade, que convêm ao processo de fusão depois do *matching*.

O Engine é o componente mais importante da arquitetura, pois nele estão concentrados os algoritmos biométricos, que são essenciais para o reconhecimento e, portanto, para a identificação e autenticação de usuários no controle de acesso.

3.2.1 Engine API

O componente Engine provê três interfaces: *IBioPassEngine*, *IBioPassEngineTrait* e *IBioPassEngineTraitDetector*.

Além das operações básicas de criação, validação e *matching* de *templates* multibiométricos, a interface *IBioPassEngine* possui, ainda: operações de configuração de limiar de reconhecimento multibiométrico; uma operação de detecção biométrica para auxílio na autenticação contínua; e uma operação utilizada para carregar implementações unibiométricas de traços a serem utilizados para o reconhecimento multibiométrico. A Figura 7 apresenta a interface *IBioPassEngine*, com suas operações.

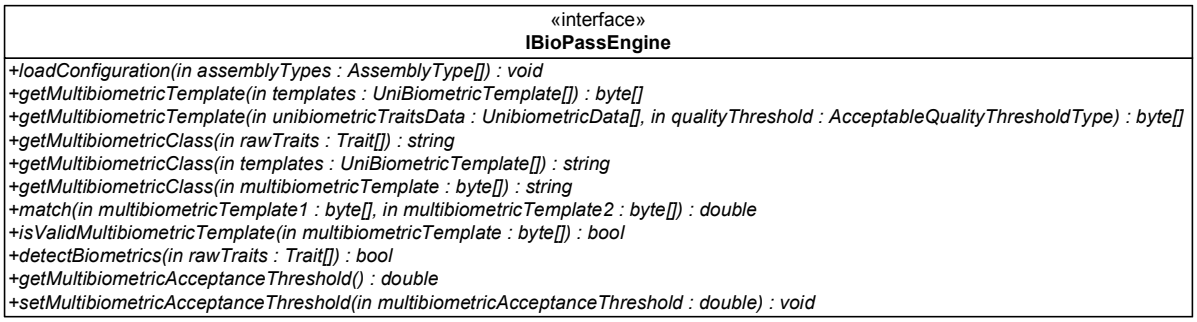

Figura 7. Interface IBioPassEngine

Por questões de organização, a interface *IBioPassEngine* é detalhada no Apêndice B.1.

Já a interface *IBioPassEngineTrait* possui, basicamente, operações de criação, validação e *matching* de *templates* unibiométricos. Assim, sempre que uma nova característica biométrica for adicionada, deve-se implementar a interface *IBioPassEngineTrait* para esta nova característica. A Figura 8 apresenta a interface *IBioPassEngineTrait*, com suas operações.

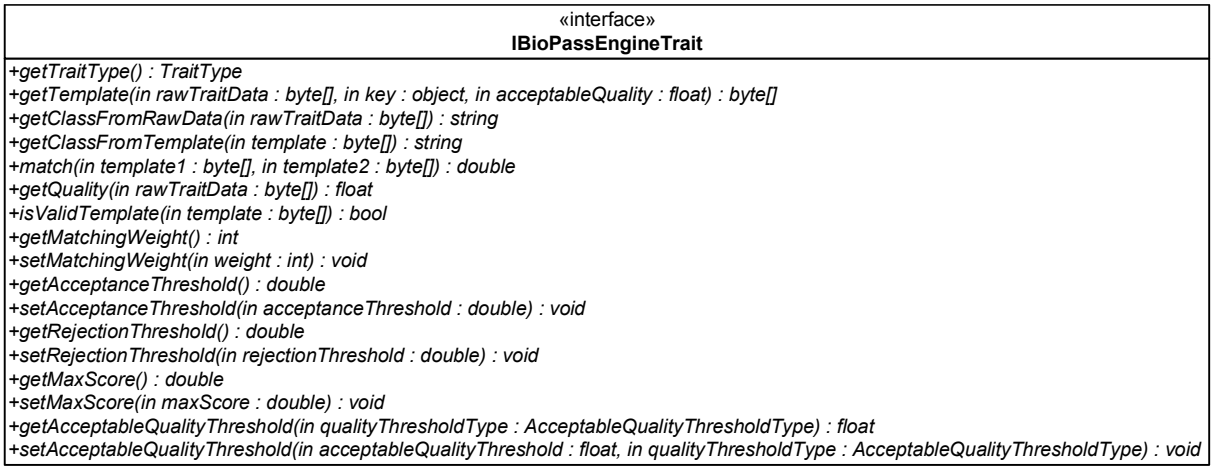

Figura 8. Interface IBioPassEngineTrait

Por questões de organização, a interface *IBioPassEngineTrait* é detalhada no Apêndice B.2.

Por fim, a interface *IBioPassEngineTraitDetector* define operações relativas à detecção da característica biométrica. Esta interface deve ser implementada caso a

característica biométrica venha a ser utilizada para a autenticação contínua. A Figura 9 apresenta a interface *IBioPassEngineTraitDetector*, com suas operações.

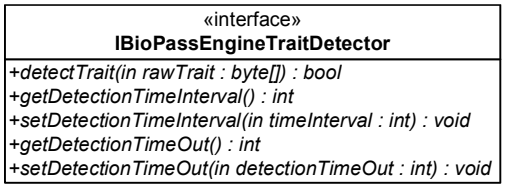

Figura 9. Interface IBioPassEngineTraitDetector

A interface *IBioPassEngineTraitDetector* deverá ser implementada apenas se desejado ou se o traço em questão for usado para autenticação contínua. Por questões de organização, a interface *IBioPassEngineTraitDetector* é detalhada no Apêndice B.3.

3.2.1.1 Tipos de dados

O componente *Engine* utiliza enumerações e outros tipos de dados.

O tipo de dado *TraitType* é uma enumeração de tipos de características biométricas a serem usadas na aplicação. O valor correspondente aos tipos deve ser o mesmo usado no *Trait type* do *template* multibiométrico. Este tipo é ilustrado na Figura 10.

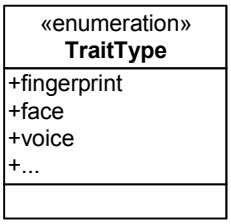

Figura 10. Tipo de dado (enumeração) TraitType

O tipo de dado *AcceptableQualityThresholdType* é uma enumeração de tipos de limiar de qualidade aceitável, isto é, a qualidade que o sinal do traço capturado precisa ter para que o *template* seja criado. Desta forma, cada tipo de limiar poderá ser usado para determinar a qualidade mínima necessária ao sinal capturado, dependendo da finalidade de seu uso: registro, verificação, identificação ou desconhecido. A Figura 11 apresenta este tipo.

Além das enumerações descritas, o componente Engine utiliza mais quatro tipos, a serem apresentados a seguir.

O tipo *AssemblyType* encapsula um *Assembly*, isto é, um bloco independente e reutilizável de software, e seus tipos ou classes a serem utilizados. Ele é usado apenas na configuração do Engine, com o intuito de apontar as implementações da interface unibiométrica *IBioPassEngineTrait* que farão parte do processo multibiométrico. A Figura 12 apresenta este tipo.

O tipo *Trait* representa um traço biométrico. Ele encapsula o tipo do traço e um *array* de bytes que armazena o próprio traço. A Figura 13 ilustra este tipo.

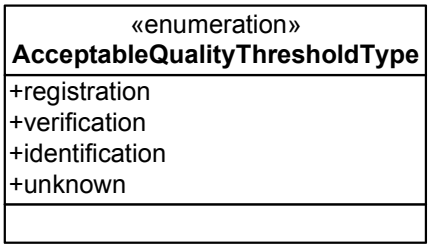

Figura 11. Tipo de dado (enumeração) AcceptableQualityThresholdType

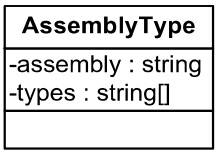

Figura 12. Tipo de dado AssemblyType

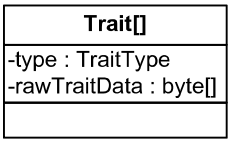

Figura 13. Tipo de dado Trait

O tipo *UnibiometricData* encapsula o tipo do traço, o próprio traço e a chave criptográfica. Esta deve pertencer a uma classe geral de tipos chamada *object*, de modo que a chave de cada algoritmo unibiométrico pode ser de um tipo específico a seu algoritmo, bastando apenas que este tipo seja um subtipo de *object*. O tipo *UnibiometricData* é ilustrado na Figura 14.

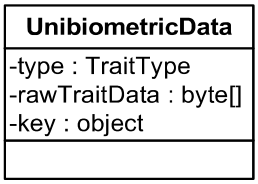

Figura 14. Tipo de dado UnibiometricData

Por fim, o tipo *UnibiometricTemplate* representa um *template* unibiométrico. Ele encapsula o tipo do traço e um *array* de bytes que armazena o *template*. A Figura 15 ilustra este tipo.

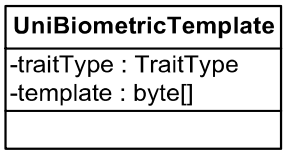

Figura 15. Tipo de dado UnibiometricTemplate

3.3 Identification and Authentication

O componente de identificação e autenticação deve responder a requisições de registro, identificação e autenticação de usuários. Ele deve permitir a construção de sistemas distribuídos e, também, de sistemas que possam ser usados em *clusters* de estações de trabalho. Para isto, deverá possuir um escalonador de tarefas, que irá gerenciar os processamentos de *matching* entre os nós, como ilustrado na Figura 16. A utilização de *clusters* de estações de trabalho visa a elevar o desempenho da operação de identificação para situações envolvendo, por exemplo, a execução sobre um banco de dados com milhares ou até milhões de indivíduos cadastrados.

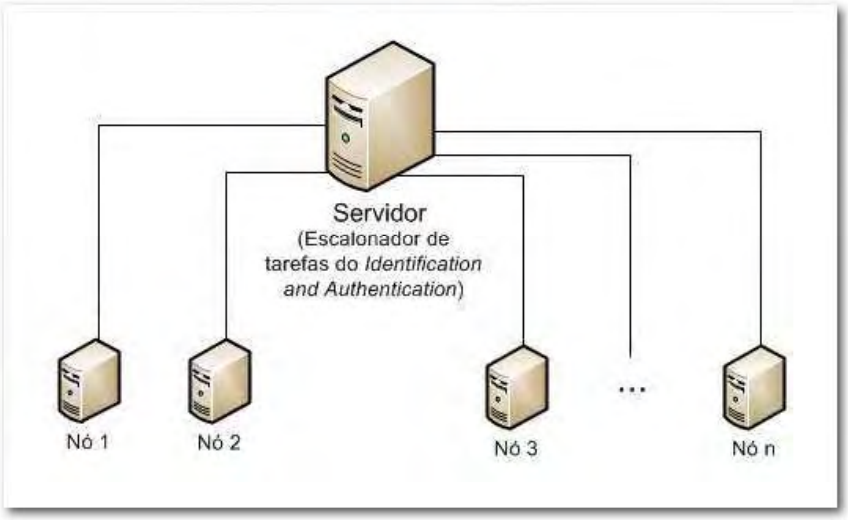

Figura 16. Utilização de clusters para elevar a eficiência da identificação

Ainda, uma interface *Web Services* (Stal, 2002) com métodos de identificação e autenticação de usuários deve ser disponibilizada. *Web Services* proveem interoperabilidade entre plataformas e aplicações desenvolvidas em diferentes linguagens de programação ou para diferentes dispositivos. Este componente acessa os componentes Engine e Persistence diretamente e deve usar mecanismos de segurança para a transmissão e validação de dados, a fim de garantir a autenticidade, integridade e confidencialidade do processo e dos dados envolvidos.

3.4 SDK (Software Development Kit)

O SDK deve possuir bibliotecas para comunicação com os dispositivos de hardware, isto é, os sensores biométricos.

O SDK também compreende outras bibliotecas de comunicação com os componentes do pacote de serviço (*Service Package*), para que os desenvolvedores possam programar aplicações Web ou desktop, para ambientes locais ou distribuídos. Em outras palavras, o SDK pode possibilitar que os desenvolvedores combinem: (1) ao menos um traço biométrico; (2) um processo simples ou contínuo de autenticação; (3) paradigma de sistemas local ou distribuído; (4) tipo de ambiente desktop ou Web.

3.5 Logon

A fim de identificar e autenticar usuários em seus computadores pessoais e dispositivos móveis, o componente Logon se mostra como uma aplicação cliente a ser integrada com o sistema operacional. Portanto, ao invés de usar o mecanismo de controle de acesso tradicional, identificação e autenticação multibiométricas podem ser usadas para se fazer o *login*. Ainda, um processo de autenticação contínua deve bloquear o acesso na ausência do usuário. O componente Logon é uma aplicação pronta para ser usada que deve utilizar o componente SDK em sua implementação.

3.6 Plugin

O Plugin deve ser responsável pela interação entre o componente de identificação e autenticação, e *websites* e outras aplicações. O Plugin tem o objetivo de facilitar o trabalho do desenvolvedor através do encapsulamento do processo de registro biométrico e questões de segurança computacional, bem como a comunicação com *Web Services*. Assim, os desenvolvedores podem focar no domínio da aplicação e abstrair toda a funcionalidade de controle de acesso. Este componente deve usar o componente SDK em sua implementação.

Capítulo

4

Projeto

"Podem morrer as pessoas, mas nunca suas ideias."

Ernesto Che Guevara

Este capítulo descreve o projeto de uma aplicação protótipo que utiliza os componentes Persistence e Engine definidos neste trabalho. As APIs destes componentes foram implementadas conforme a especificação desenvolvida e apresentada no capítulo 3, de forma que a aplicação protótipo utiliza estes dois componentes a fim de demonstrar a execução de operações de cadastro, verificação, identificação convencional e com autenticação contínua, bem como a utilização das outras operações das APIs que permitem, dentre outras funcionalidades, o ajuste de limiares e parâmetros de configuração. As características biométricas utilizadas nesta aplicação foram impressão digital e face. A Figura 17 apresenta o esquema geral da aplicação protótipo, que acessa os componentes Engine e Persistence a fim de realizar operações biométricas e de armazenamento dos dados, respectivamente.

As seções a seguir detalham especificidades de projeto das APIs de segurança e armazenamento do BioPass, bem como da aplicação protótipo como um todo.

4.1 Componente Persistence

Como mencionado no capítulo 3 (seção 3.1.1), a API do componente Persistence possui basicamente quatro tipos de operações: registro, recuperação, atualização e remoção de dados. Todas essas operações estão presentes na mesma interface, a *IBioPassPersistence*, implementada pela classe *BioPassPersistence*.

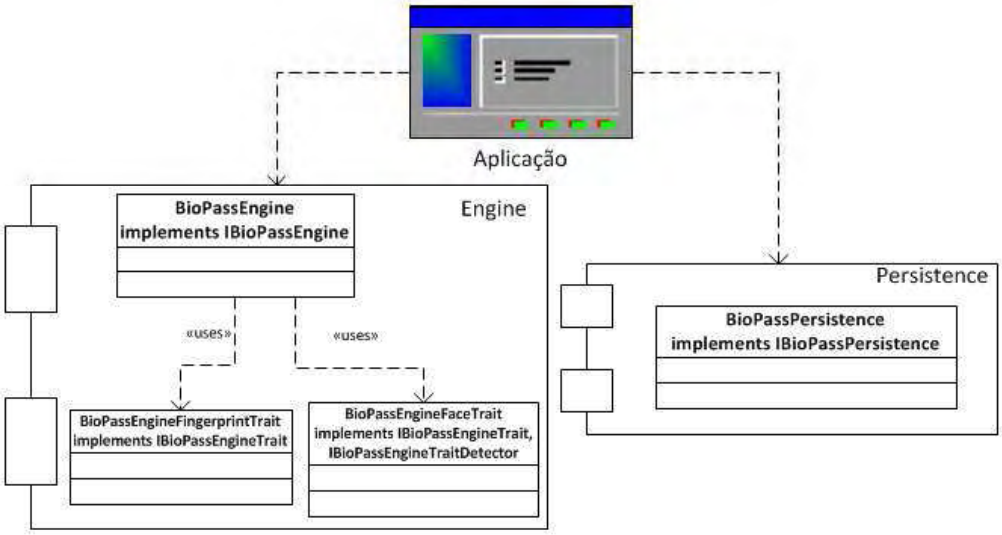

Figura 17. Esquema geral da aplicação protótipo

A Figura 18 ilustra o diagrama de classes do componente Persistence implementado. As operações são omitidas a fim de ressaltar a relação entre a classe e a interface.

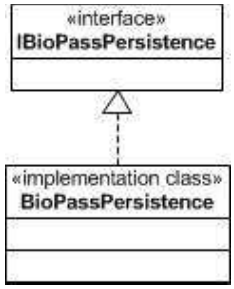

Figura 18. Diagrama de classes do componente Persistence implementado

4.2 Componente Engine

Como mencionado no capítulo 3, a API do componente Engine compreende três interfaces: *IBioPassEngine, IBioPassEngineTrait e IBioPassEngineTraitDetector*.

A interface *IBioPassEngineTrait* inclui operações que devem ser implementadas para cada característica biométrica do sistema, neste caso, impressão digital e face. Foram utilizados os algoritmos desenvolvidos por (Neurotechnology, 2010).

A interface *IBioPassEngineTraitDetector*, por sua vez, deve ser implementada apenas visando à autenticação contínua, para as características biométricas que participarão deste processo. Neste trabalho, a autenticação contínua foi implementada para a face, sendo, portanto, implementada pela classe *BioPassEngineFaceTrait*.

Por fim, a interface *IBioPassEngine,* responsável pelas operações multibiométricas, foi implementada pela classe *BioPassEngine*. Sua implementação utiliza reflexão, permitindo que o comportamento do software seja modificado ou definido em tempo de execução. Assim, quando novas características biométricas forem adicionadas ao sistema, não haverá a necessidade de mudanças no código atual.

O diagrama de classes do componente Engine implementado é apresentado na Figura 19. Suas operações foram removidas do diagrama para simplificar sua visualização e entendimento, mantendo o foco nas relações entre as classes e interfaces.

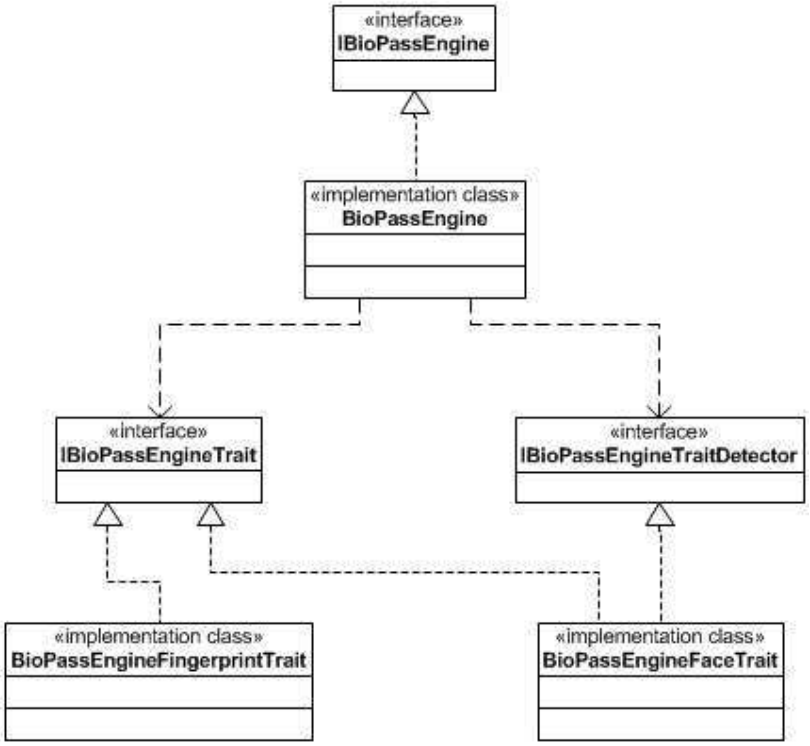

Figura 19. Diagrama de classes do componente Engine implementado

As figuras a seguir (Figura 20 a Figura 31) expõem os principais diagramas UML de sequência das operações implementadas pela classe *BioPassEngine*. Algumas delas possuem um funcionamento básico e local. No entanto, a maioria delas acessa as classes que implementam as interfaces *IBioPassEngineTrait* e *IBioPassEngineTraitDetector*, em busca da união dos comportamentos unibiométricos a fim de gerar um resultado de comportamento multibiométrico. É importante ressaltar que os diagramas contemplam todas as possibilidades de retorno de cada operação. Além disso, a especificação dessas operações pode ser consultada no Apêndice B.

A primeira operação apresentada (Figura 24) é a operação *loadConfiguration* cujo código foi apresentado na .

As operações dos diagramas seguintes (Figura 21 a Figura 25) chamam operações das classes unibiométricas que implementam a interface *IBioPassEngineTrait*, para gerar um resultado. As classes a serem instanciadas são reveladas em tempo de execução.

A Figura 21 e a Figura 22 ilustram a implementação das duas versões da operação *getMultibiometricTemplate*. A primeira recebe os *templates* unibiométricos (array de bytes) e constrói o *template* multibiométrico. Para isso, ela checa se os arrays de bytes correspondentes a cada *template* unibiométrico são válidos. Se sim, eles se unem para formar o *template* multibiométrico, conforme o padrão definido (Figura 5). Caso contrário, é lançada a exceção *InvalidTemplateFormatException*.

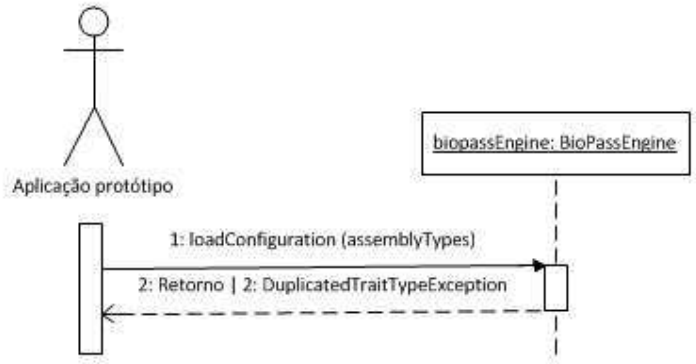

Figura 20. Diagrama de sequência da operação loadConfiguration (assemblyTypes)

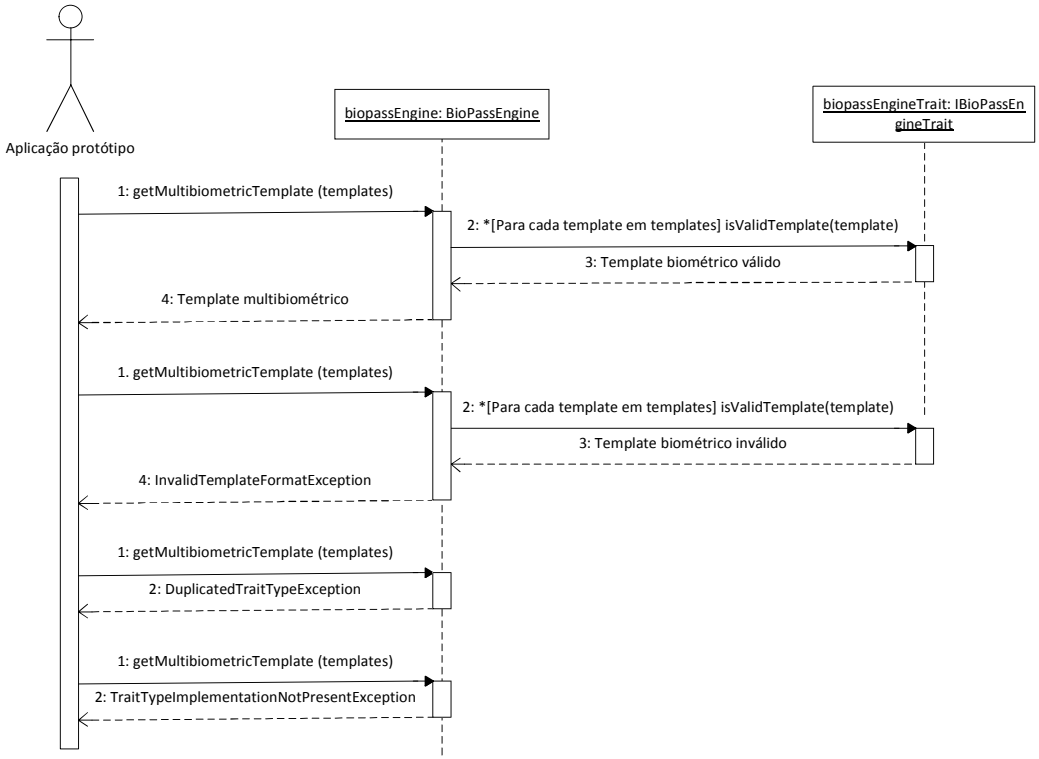

Figura 21. Diagrama de sequência da operação getMultibiometricTemplate(templates)

Além disso, caso haja mais de um template unibiométrico para a mesma característica ou se a implementação de alguma das características passadas como parâmetro não tiver sido carregada, a operação lança uma exceção. A exceção deste último caso

é lançada por quase todas as operações, uma vez que em quase todas é preciso que as implementações necessárias das características biométricas estejam presentes.

A segunda operação (Figura 22), por sua vez, recebe os dados unibiométricos e a chave (encapsulados pelo tipo *UnibiometricData*) e o tipo de qualidade necessária para a criação de seus respectivos *templates* para gerar o *template* multibiométrico. Neste sentido, os valores deste tipo de qualidade são recuperados e utilizados, juntamente com os dados biométricos brutos e a chave, para a criação de *templates* unibiométricos. A operação anterior (Figura 21) é chamada e o *template* multibiométrico é criado a partir dos *templates* unibiométricos. Caso algum dos dados brutos não possua qualidade suficiente para execução da operação ou, até mesmo, seja inválido, uma exceção é lançada.

A Figura 23, a Figura 24 e a Figura 25 apresentam a implementação da operação *getMultibiometricClass*, que obtém a classificação multibiométrica através dos dados unibiométricos brutos, de seus *templates* unibiométricos ou do *template* multibiométrico, obtidas por suas respectivas operações unibiométricas. A operação que obtém a classificação através do *template* multibiométrico recorre à operação que extrai a classificação a partir dos *templates* unibiométricos, que, por sua vez, são extraídos do mesmo. Assim como ocorre em várias operações da classe *BioPassEngine*, a exceção *DuplicatedTraitTypeException* é lançada caso haja mais de um template unibiométrico para a mesma característica. As operações unibiométricas chamadas por *getMultibiometricClass* foram implementadas apenas para a impressão digital.

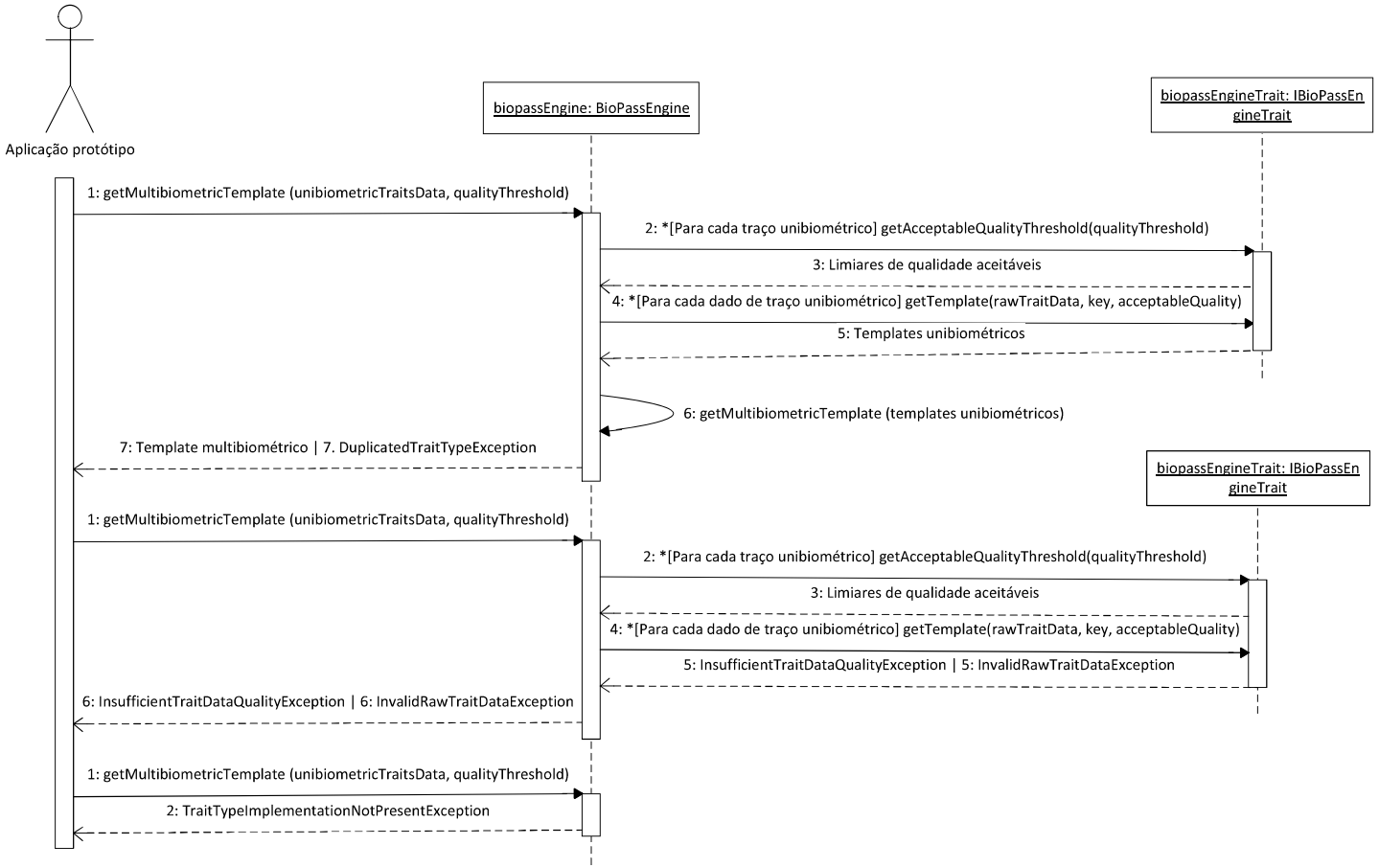

Figura 22. Diagrama de sequência da operação *getMultibiometricTemplate (unibiometricTraitsData, qualityThreshold)*

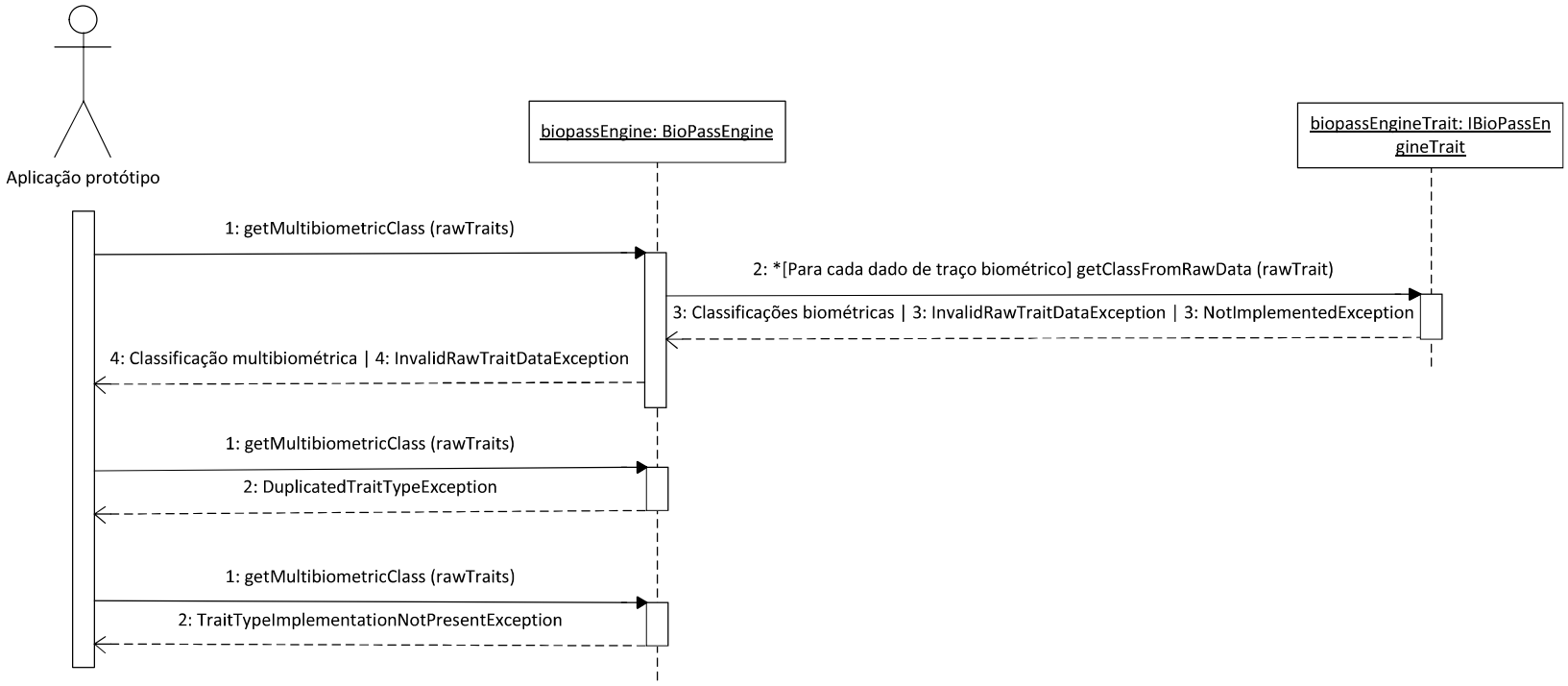

Figura 23. Diagrama de sequência da operação *getMultibiometricClass (rawTraits)*

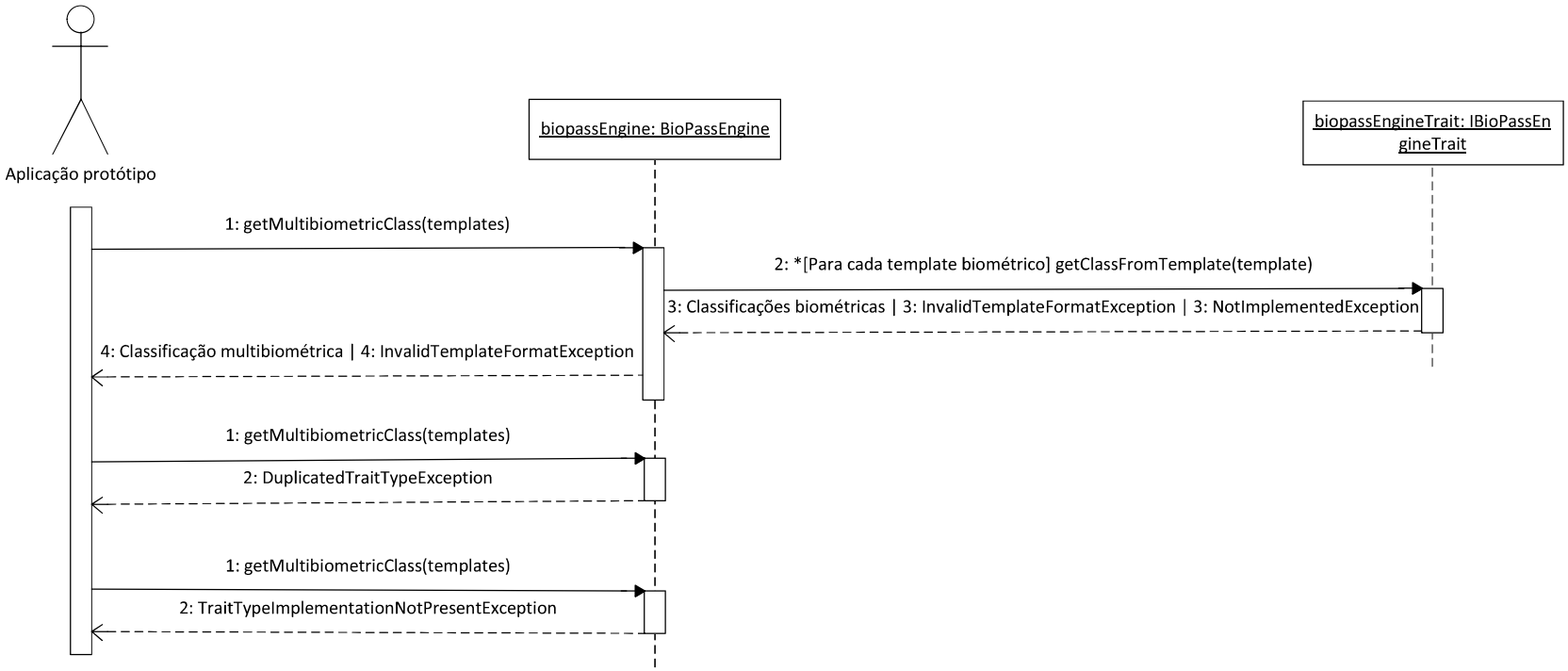

Figura 24. Diagrama de sequência da operação *getMultibiometricClass (templates)*

Figura 25. Diagrama de sequência da operação *getMultibiometricClass (multibiometricTemplate)*

Os diagramas de sequência da operação de *matching* multibiométrico (Figura 26 a Figura 30) apresentados ilustram, separadamente, cada caso (anormais e de sucesso) que pode ocorrer. A operação primeiramente confere se os *templates*, tanto multi quanto unibiométricos, são válidos. Em seguida ela obtém os *scores* de *matching* unibiométricos, os *scores* máximos, os pesos exercidos pelas características biométricas e os limiares de aceitação e de rejeição de cada característica para decidir o resultado do *matching* multibiométrico.

Como os valores máximos de *score* do *matching* entre as características biométricas não assumem um valor exato na prática, tem-se que, a partir de certo valor de *score*, o *matching* entre dois *templates* atingiu o valor máximo. Assim, o algoritmo multibiométrico implementado considera que, caso o *score* unibiométrico resulte em um valor maior do que o valor definido para o *score* máximo, é considerado que o resultado desta operação de *matching* foi igual ao do valor de *score* máximo definido para aquela característica.

Os pesos exercidos pelas características biométricas são utilizados no cálculo do *score* multibiométrico final.

Por fim, os limiares de aceitação e rejeição foram utilizados da seguinte forma. Caso o *score* unibiométrico seja menor do que o limiar de rejeição, a operação retorna um *score* multibiométrico igual a 0. Caso o *score* unibiométrico seja maior do que o limiar de aceitação, a operação retorna o *score* multibiométrico máximo definido nesta implementação, isto é, 100.

O último diagrama apresentado (Figura 31) ilustra a operação de detecção de traços biométricos. Esta operação, diferentemente das anteriores demonstradas, instancia as classes unibiométricas que implementam a interface *IBioPassEngineTraitDetector*. Ela pode ser usada como auxílio à autenticação contínua uma vez que a detectação de traços biométricos pode ser realizada repetidas vezes para checar a presença do usuário evitando, assim, sucessivas chamadas à operação de *matching*, que notadamente exigiriam maior processamento, reduzindo o desempenho do sistema. A operação unibiométrica *detectTrait* foi implementada apenas para face.

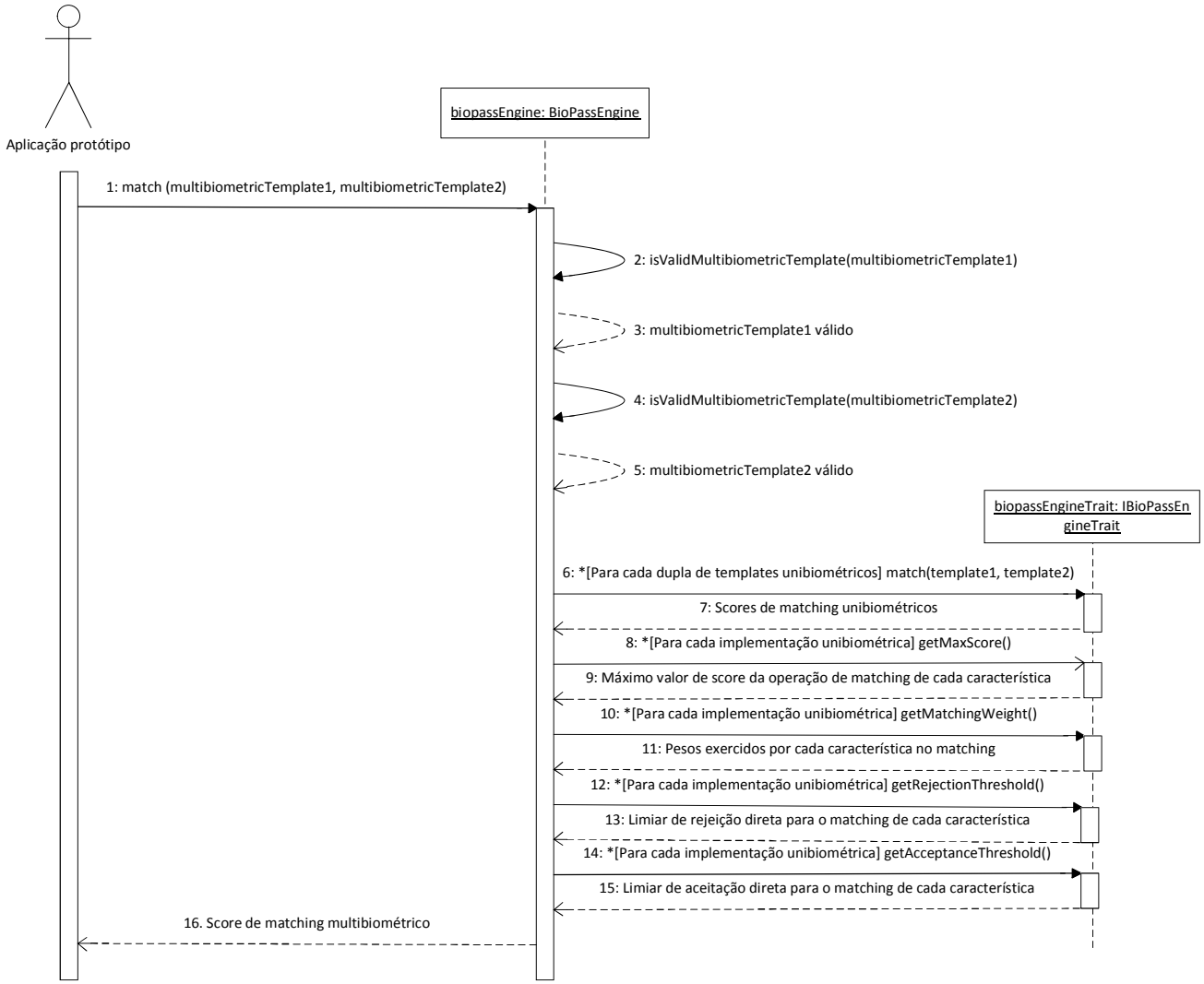

Figura 26. Diagrama de sequência da operação match (multibiometricTemplate1, multibiometricTemplate2) (1)

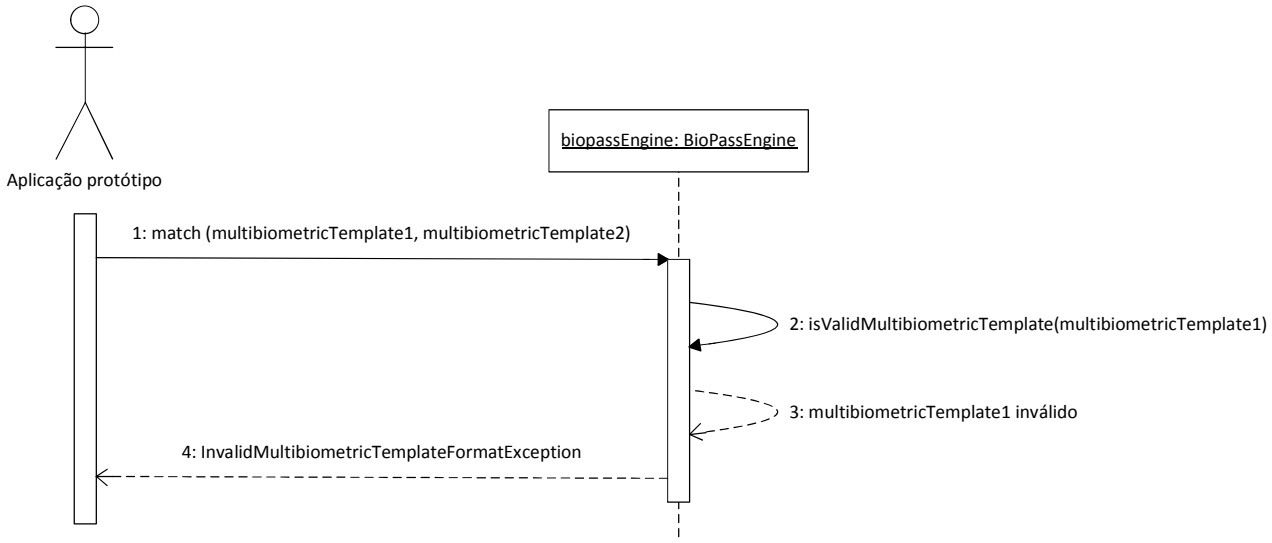

Figura 27. Diagrama de sequência da operação match (multibiometricTemplate1, multibiometricTemplate2) (2)

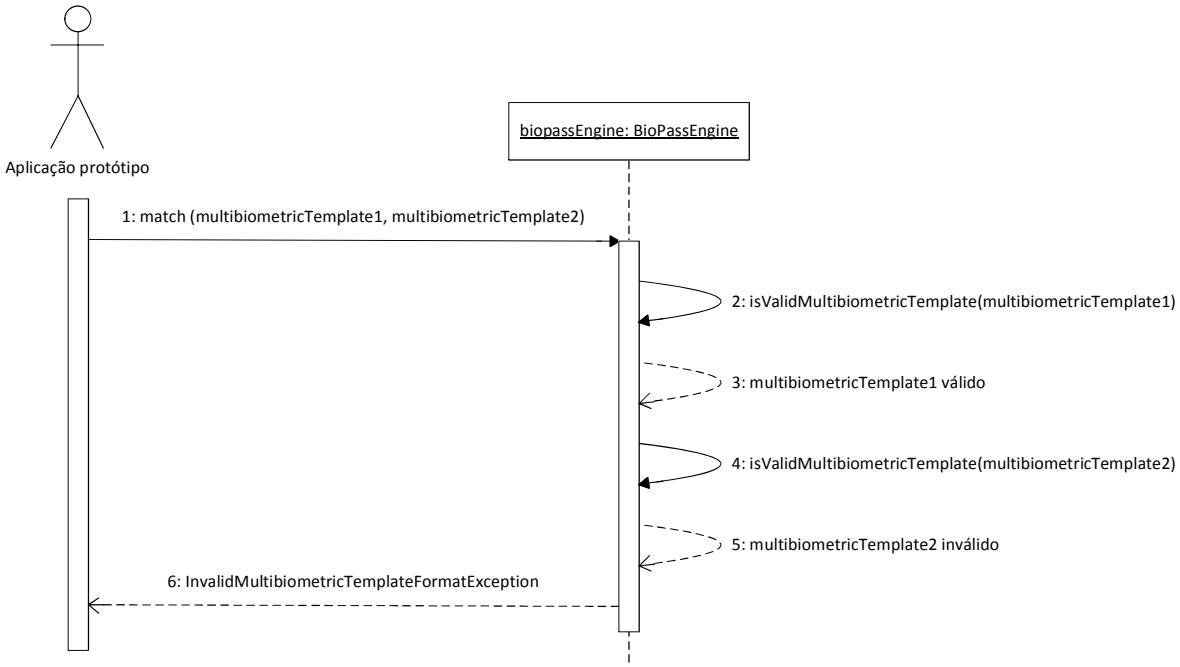

Figura 28. Diagrama de sequência da operação match (multibiometricTemplate1, multibiometricTemplate2) (3)

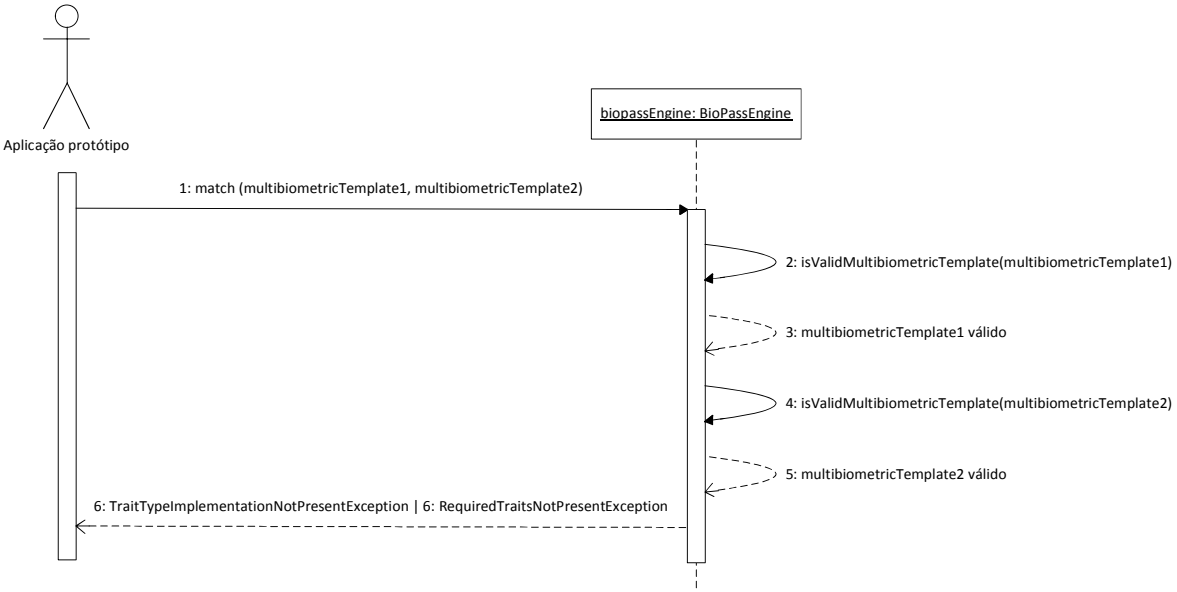

Figura 29. Diagrama de sequência da operação match (multibiometricTemplate1, multibiometricTemplate2) (4)

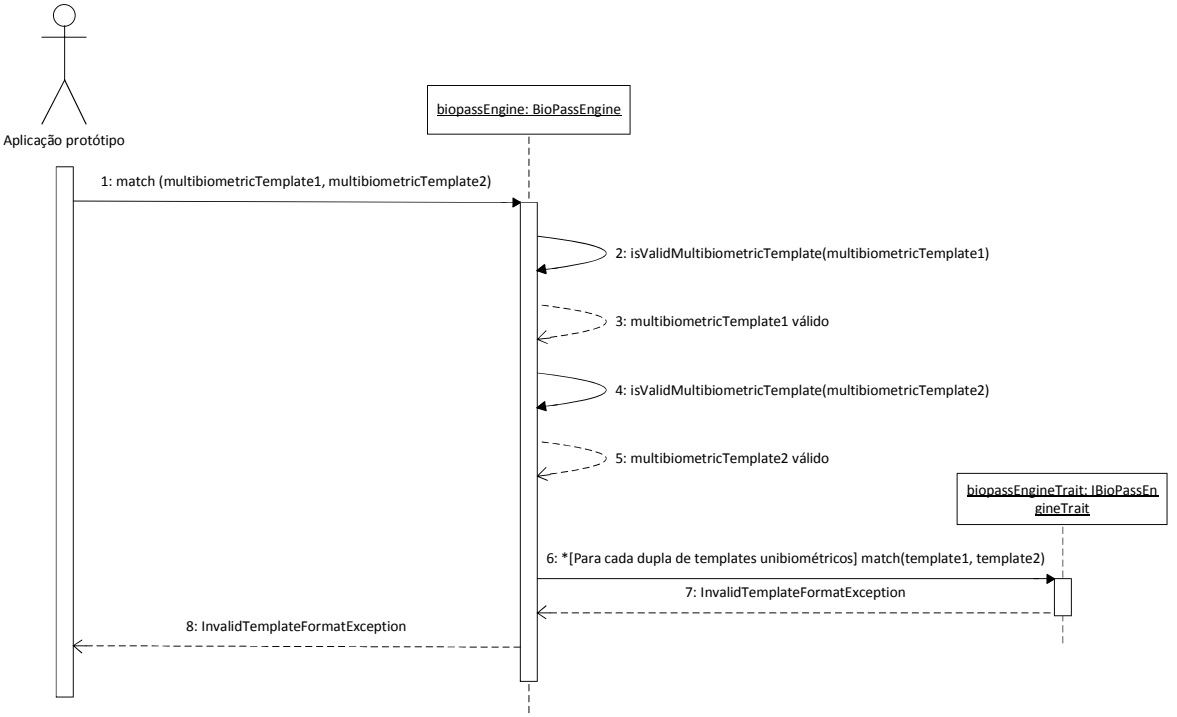

Figura 30. Diagrama de sequência da operação match (multibiometricTemplate1, multibiometricTemplate2) (5)

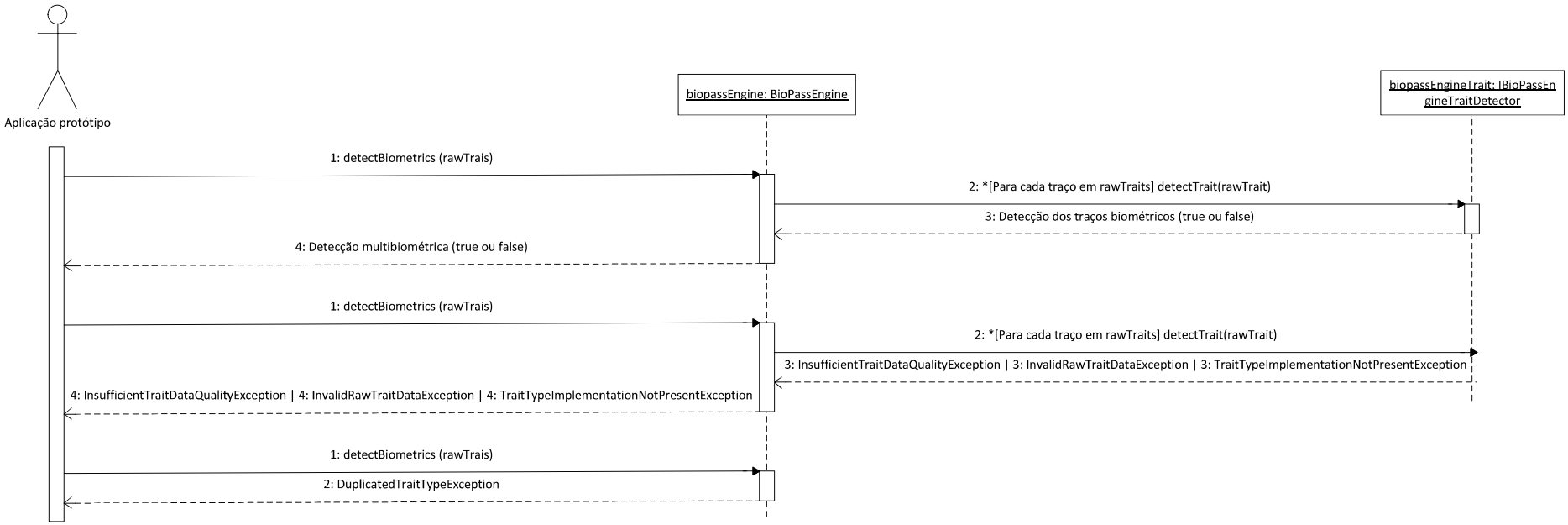

Figura 31. Diagrama de sequência da operação *detectBiometrics (rawTraits)*

Capítulo

5

Implementação e Testes

"O sucesso é uma consequência e não um objetivo."

Gustave Flaubert

Este capítulo descreve a implementação da aplicação protótipo, que utiliza os componentes Persistence e Engine definidos neste trabalho. As seções a seguir detalham especificidades de implementação das APIs de segurança e armazenamento do BioPass, bem como da aplicação final desenvolvida. Por fim, são apresentados os testes realizados na aplicação.

5.1 Implementação do componente Persistence

A implementação dos métodos do componente Persistence possui, basicamente, chamadas diretas ao banco de dados e suas especificações podem ser encontradas no Apêndice A. A Figura 32 apresenta o código C# da operação *registerMultibiometricTemplate* como exemplo de implementação dos métodos deste componente.

```
//Connection string
private static string connStr = @"Server=127.0.0.1;Port=5432;Database=biopassdb;
                                  User Id=postgres;Password=      ;";
//Criação da conexão
NpgsqlConnection con = new NpgsqlConnection(connStr);

public void registerMultibiometricTemplate(int userID, byte[] multibiometricTemplate)
{
    con.Open(); //Abrir conexão com o banco de dados
    //Criação de comando SQL
    NpgsqlCommand command = new NpgsqlCommand(@"INSERT INTO tblbiodata (dbid, template)
                            VALUES (@userID, @multibiometricTemplate)", con);
    command.Parameters.AddWithValue("@userID", userID);
    command.Parameters.AddWithValue("@multibiometricTemplate", multibiometricTemplate);
    command.ExecuteNonQuery(); //Execução do comando SQL
    con.Close(); //Fechar conexão com o banco de dados
}
```

Figura 32. Exemplo de implementação (operação registerMultibiometricTemplate)

Como citado no capítulo 1 (seção 1.4), o componente Persistence utiliza o SGBD PostegreSQL em sua implementação. O banco de dados criado possui apenas uma tabela, formada pelas colunas ID (inteiro: chave primária), DBID (inteiro: valores únicos e não nulos referentes ao ID do usuário na aplicação), *template* (array de *bytes*: dados biométricos) e *class* (inteiro: classe biométrica). As classes biométricas armazenadas no campo *class* contemplam os tipos de impressão digital (espirais, arcos, loops, desconhecidos), representados por uma enumeração. A ilustra esta base de dados.

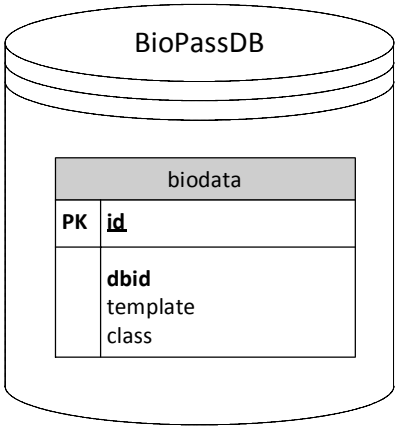

Figura 33. Banco de dados criado para a aplicação

5.2 Implementação do componente Engine

De acordo com as implementações unibiométricas que foram carregadas na inicialização da classe *BioPassEngine* (pela operação *loadConfiguration*), as operações da interface *IBioPassEngine* acessam, em tempo de execução (por meio da reflexão), as classes que implementaram as interfaces *IBioPassEngineTrait* e *IBioPassEngineTraitDetector*, de acordo com suas necessidades. A ilustra o código da operação *loadConfiguration*, demonstrando como a reflexão foi usada para instanciar as classes unibiométricas, isto é, aquelas que implementam a interface *IBioPassEngineTrait*.

```
public void loadConfiguration(AssemblyTypes[] assemblyTypes)
{
    //Esvazia a lista global engineTraits
    this.engineTraits.Clear();

    //Lista de tipos de traços para checagem de
    //implementações repetidas para o mesmo traço
    List<TraitType> traitTypes = new List<TraitType>();

    //Percorre os assemblies passados como parâmetro
    for (int i = 0; i < assemblyTypes.Length; i++)
    {
        Assembly o = Assembly.Load(assemblyTypes[i].Assembly);

        //Para cada assembly, percorre as classes indicadas,
        //que implementam a interface IBioPassEngineTrait
        for (int j = 0; j < assemblyTypes[i].Types.Length; j++)
        {
            //Uso da reflexão para incializar um objeto da classe t (Assembly.Tipo)
            Type t = Type.GetType(assemblyTypes[i].Assembly + "."
                     + assemblyTypes[i].Types[j]);
            IBioPassEngineTrait trait = (IBioPassEngineTrait)o.CreateInstance(t.FullName);

            //Se a implementação do traço corrente estiver duplicada,
            //esvazia a lista global engineTraits e a exceção é lançada
            if (traitTypes.Contains(trait.getTraitType()))
            {
                this.engineTraits.Clear();
                throw new DuplicatedTraitTypeException();
            }

            //Caso contrário, o tipo do traço é armazenado
            //e a operação continua normalmente
            traitTypes.Add(trait.getTraitType());

            //Adição da instância de implementação de IBioPassEngineTrait
            //na lista global engineTraits
            this.engineTraits.Add(trait);
        }
    }
}
```

Figura 34. Implementação da operação loadConfiguration

Como já mencionado, o componente Engine foi implementado para face e impressão digital como características biométricas. A face é uma característica de baixa singularidade, como pôde ser visto na Tabela 1. Por isso, sua taxa de falsa aceitação (FAR) tende a ser alta, especialmente no reconhecimento de uma pessoa dentre um alto número de identidades. A taxa de falsa rejeição (FRR), por outro lado, tende a ser baixa. Neste sentido, uma configuração de identificação que estabelece que a operação de *matching* multimodal pare caso o resultado do *matching* facial entre dois *templates* seja de rejeição pode elevar o desempenho geral de identificação da aplicação se a face for verificada primeiramente na ordem de *matching* das características biométricas desse sistema. Assim, como ilustra a , a classe *BioPas-*

sEngine foi inicializada considerando esta ordem. O construtor da classe *BioPassEngine* chama a operação *loadConfiguration*. As classes *BioPassEngineFaceTrait* e *BioPassEngineFingerprintTrait* foram colocadas dentro do mesmo *assembly*, denominado Engine.

```
public static BioPassEngine engine =
    new BioPassEngine(new AssemblyTypes[] { new AssemblyTypes("Engine",
    new string[] { "BioPassEngineFaceTrait", "BioPassEngineFingerprintTrait" }) });
```

Figura 35. Inicialização da classe BioPassEngine

5.3 Aplicação protótipo desenvolvida

A aplicação protótipo desenvolvida é uma aplicação desktop com quatro operações: cadastro, verificação, identificação convencional e com autenticação contínua de usuários. A aplicação acessa os componentes Engine e Persistence, e foi desenvolvida com o intuito de validar as APIs definidas para estes componentes.

Com o objetivo de popular o banco de dados, foram utilizadas as bases de dados de faces GTAV (GTAV) e de impressão digital U.are.U Sample Database (U.are.U). Cada nome de imagem de face do banco de dados GTAV começa com o rótulo 'ID' seguido do número de identificação do indivíduo, '_' e três caracteres correspondentes ao número da amostra. Por exemplo, 'ID24_007' refere-se à sétima pose capturada do indivíduo de número de identificação 24. A Figura 36 ilustra, como exemplo, as poses capturadas para o sujeito 24 do banco de dados GTAV.

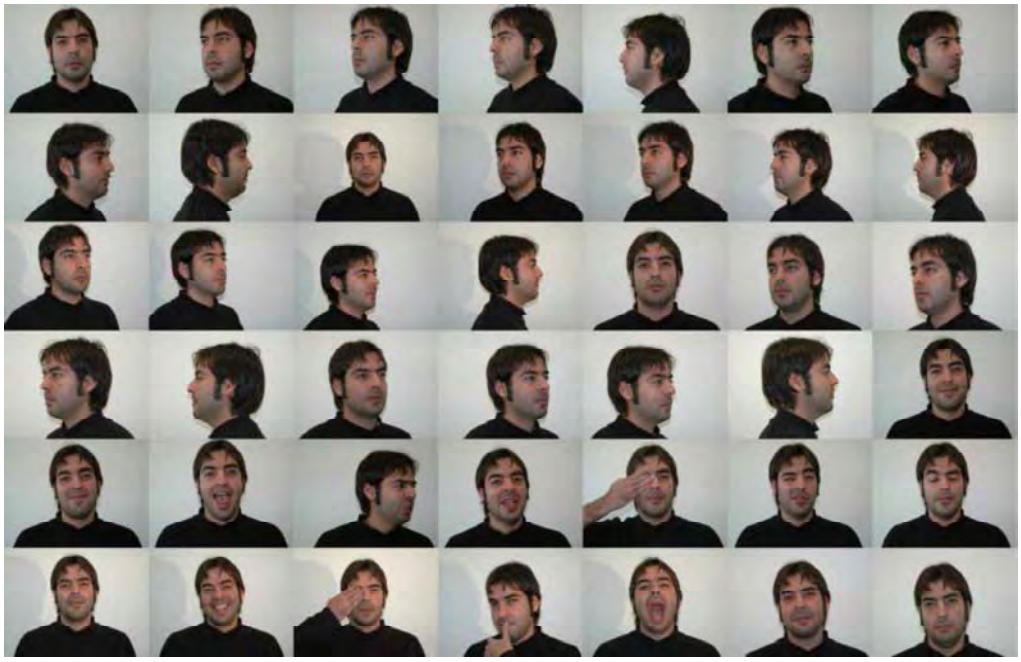

Figura 36. Exemplo de poses capturadas para um indivíduo do GTAV

As imagens de impressões digitais do banco de dados U.are.U utilizado foram renomeadas com o objetivo de padronizar a nomenclatura seguindo o modelo utilizado pelo banco de imagens faciais GTAV. Desta forma, a título de simulação, cada identificador de impressão digital e suas digitalizações foram atribuídos a cada indivíduo do banco GTAV. A Figura 37 ilustra as impressões digitais capturadas para o sujeito 24 do banco de dados U.are.U.

Figura 37. Impressões digitais capturadas para um indivíduo do U.are.U

O banco de dados da aplicação protótipo foi populado com 40 identidades. A Figura 38 ilustra a tela inicial da aplicação.

Na aba *configurations*, há vários parâmetros que são determinados por operações de recuperação (get) e atribuição (set) definidas nas interfaces do componente Engine. Os valores padrão para estes parâmetros foram configurados considerando-se a calibragem feita para o universo de dados em questão (bancos de faces GTAV e de digitais U.are.U). Estes valores podem ser vistos na Tabela 3.

Os valores dos parâmetros de qualidade foram calibrados considerando-se os máximos e mínimos obtidos, bem como a quantidade de itens (instâncias de traços capturados) por qualidade. A Figura 39 apresenta os valores de qualidade alcançados para todas as capturas de traços presentes nos bancos de dados utilizados. A qualidade varia entre os limites 0 e 100 e corresponde à qualidade da característica biométrica capturada, calculada a partir de um algoritmo específico ao traço. Para impressão digital, o valor máximo extraído foi 100, e o mínimo 36. Para face, o máximo foi 68 e o mínimo 0. Assim, foram definidos valores mais altos para o cadastro do usuário, de forma que o *template* armazenado no banco seja o mais confiável possível, e valores menores para verificação e identificação.

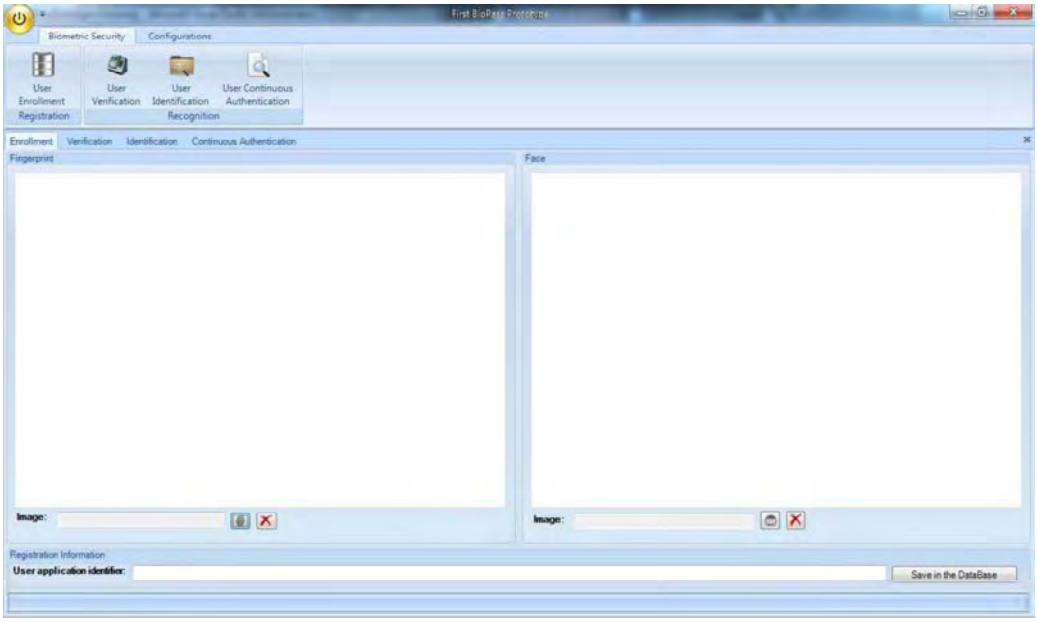

Figura 38. Tela inicial da aplicação protótipo

Tabela 3. Valores padrão definidos para os parâmetros de configuração da aplicação

Qualidade		
Parâmetro	Impressão digital	Face
Cadastro	90	50
Verificação	40	20
Identificação	50	30
Equivalência (*Matching*)		
Parâmetro	Impressão digital	Face
Peso	50	50
Limiar de aceitação	550	150
Limiar de rejeição	48	20
Score máximo	600	180
Limiar de aceitação multibiométrica	80	
Detecção facial		
Parâmetro	Face	
Intervalo de tempo (ms)	1000	
Tempo limite (ms)	4000	

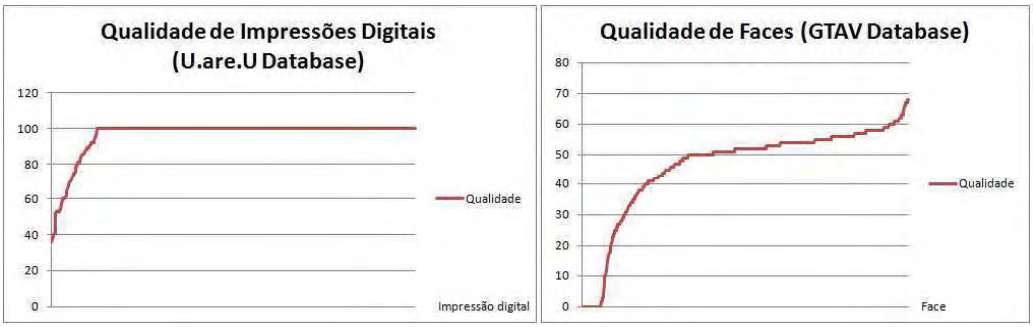

Figura 39. Qualidade dos traços dos bancos de faces (GTAV) e de impressões digitais (U.are.U)

Para determinar os valores dos *scores* máximos, a operação de *matching* foi realizada entre todos os itens correspondentes dos bancos de dados desconsiderando, é claro, o *matching* de uma imagem com ela mesma. O *score* máximo obtido para a equivalência de impressões digitais foi 1427 e para a equivalência de faces, 180. Por conseguinte, os mínimos foram 45 para impressões digitais e 0 para faces. De forma semelhante, a operação de *matching* foi realizada entre todos os itens não correspondentes do banco de dados, isto é, entre cada imagem com todas as outras imagens que não correspondem a seu indivíduo. O *score* máximo obtido para impressões digitais foi 59 e para faces, 79. Esses valores auxiliaram na escolha dos limiares de aceitação e rejeição.

O limiar de aceitação multibiométrica foi configurado para 80, sendo o *score* de aceitação multibiométrica uma normalização dos scores unibiométricos para uma escala de 0 a 100, observando-se os parâmetros para eles definidos.

Os parâmetros de detecção facial foram definidos de forma que a detecção facial ocorra de 1 em 1 segundo (intervalo de tempo). Caso a face não seja detectada no tempo limite de 4 segundos, uma nova autenticação será necessária. Estes parâmetros são usados no auxílio à autenticação contínua.

As subseções a seguir apresentam a utilização da ferramenta desenvolvida nas operações de cadastro, verificação, identificação e autenticação contínua.

5.3.1 Cadastro

Para ilustrar a operação de cadastro da ferramenta desenvolvida, a Figura 40 apresenta esta operação realizada para um novo usuário (usuário 41). A imagem de impressão digital utilizada foi a primeira do sujeito 41 do banco de impressões digitais U.are.U (ID41_001).

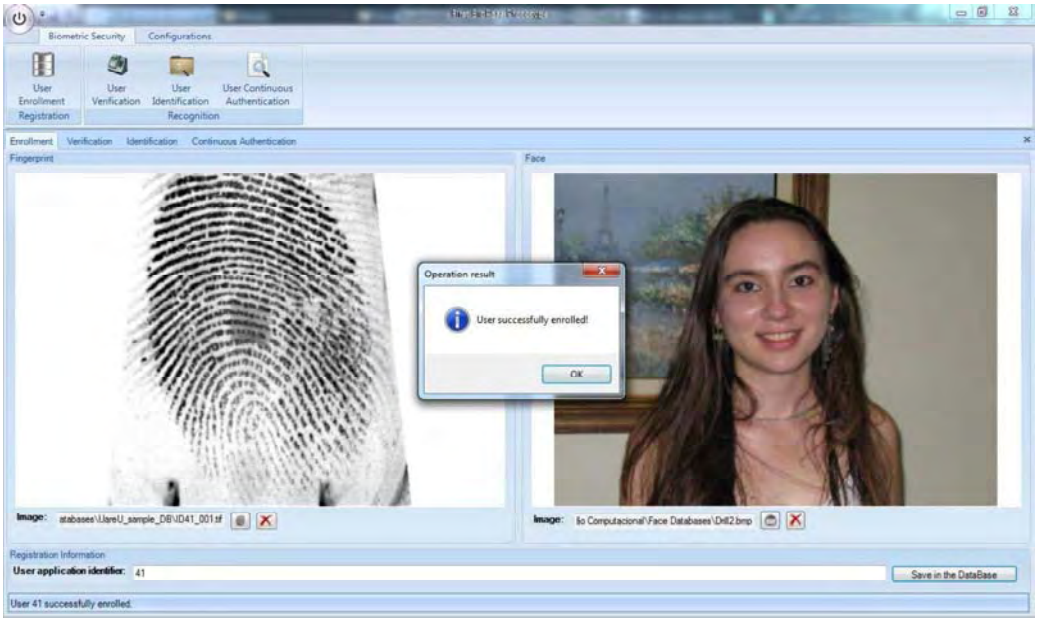

Figura 40. Tela da operação de cadastro

A operação de cadastro basicamente transforma as imagens da face e da impressão digital de entrada em arquivos binários (arrays de *bytes*) e as passa como objetos *UnibiometricData* para a operação *getMultibiometricTemplate* implementada pela classe *BioPassEngine*, usando o limiar de qualidade de cadastro das características biométricas, que foi definido na aba de configurações. Em seguida, o identificador de usuário juntamente com o *template* multibiométrico resultante da chamada à operação *getMultibiometricTemplate* é passado para a operação *registerMultibiometricTemplate* implementada pela classe *BioPassPersistence*. A Figura 41 apresenta o diagrama UML de sequência do caso de sucesso da operação de cadastro implementada na aplicação protótipo.

A segurança de *templates* não foi utilizada na aplicação protótipo porque os algoritmos biométricos utilizados não a implementam. Assim, a chave passada nos objetos *UnibiometricData* consistiu em objetos nulos. Ela poderia ser uma chave gerada e/ ou fornecida pelo sistema, ou, ainda, entrada pelo usuário. Neste último caso, o sistema, além de guardar os *templates* com segurança e ser multimodal, é considerado também multifator, pois o usuário teria que entrar com uma senha (algo que ele tem ou sabe) para se identificar/ autenticar. Isto poderia ser usado para elevar o desempenho do sistema na identificação. Por exemplo, caso o código *hash* da senha entrada coincidisse com o código *hash* da senha armazenado, a operação seguiria para o *matching* (multi) biométrico. Caso contrário, aquele registro seria

automaticamente descartado para o reconhecimento, evitando o processamento da operação de *matching* para o registro em questão.

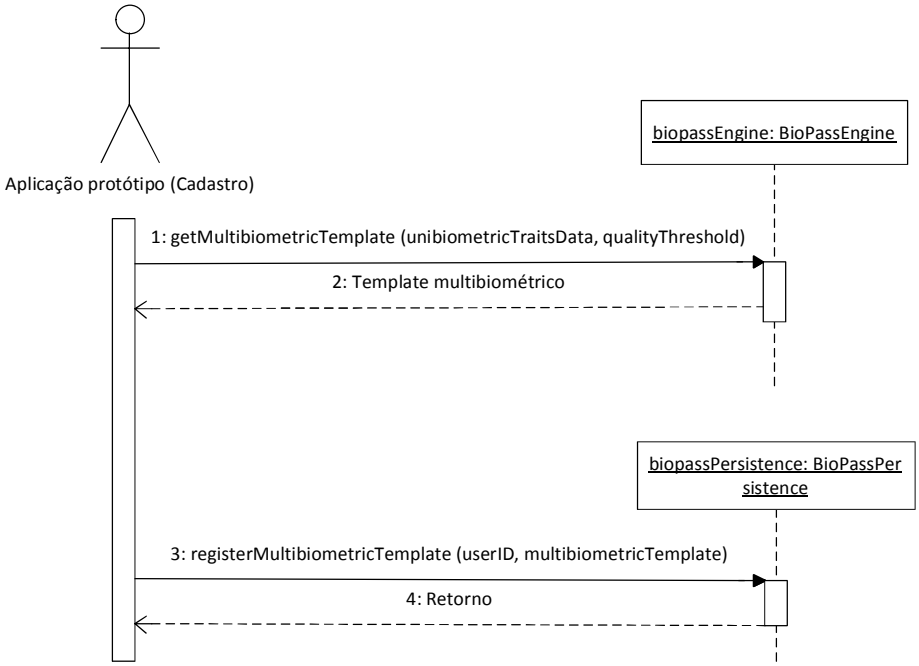

Figura 41. Diagrama de sequência da operação de Cadastro

Os *templates* multibiométricos gerados pela aplicação protótipo (mais especificamente, pela operação *getMultibiometricTemplate* implementada pela classe *BioPassEngine*) possuem a estrutura ilustrada na Figura 42. É importante notar que o formato dos templates unibiométricos pode ser abstraído. Desta forma, cada desenvolvedor que implementar a interface unibiométrica *IBioPassEngineTrait* poderá utilizar o formato de *template* unibiométrico que desejar.

5.3.2 Verificação

A operação de verificação considera um identificador de entrada, que corresponde ao usuário cadastrado no banco de dados que será comparado às imagens de impressão digital e face de entrada. Esta operação é ilustrada na Figura 43.

Header		Body					
Size = Header Size + Body Size	Version = 1.0	Number of Traits = 2	Trait Type 1 = Face	Trait Size 1 = Face Template Size	Trait Data 1 = Face Template	Trait Type 2 = Fingerprint	Trait Size 2 = Fingerprint Template Size	Trait Data 2 = Fingerprint Template

Figura 42. Estrutura dos templates multibiométricos da aplicação desenvolvida

Figura 43. Tela da operação de verificação

O caso de sucesso desta operação é ilustrado na Figura 44 e descrito a seguir. Assim como a operação de cadastro, a operação de verificação inicia transformando as imagens de face e impressão digital de entrada em arquivos binários (arrays de *bytes*) e os passa como objetos *UnibiometricData* para a operação *getMultibiometricTemplate* implementada pela classe *BioPassEngine*, usando, desta vez, o limiar de qualidade de verificação das características biométricas, que foi definido na aba de configurações. Em seguida, o identificador de usuário é utilizado para recuperar o *template* multibiométrico deste identificador que foi armazenado no banco de dados, através da operação *getMultibiometricTemplate*, implementada pela classe *BioPassPersistence*. Por fim, os dois templates multibiométricos (resultantes das chamadas às operações *getMultibiometricTemplate* implementadas pelas classes *BioPassEngine* e *BioPassPersistence*, respectivamente) são passados para a operação de *matching* implementada pela classe *BioPassEngine*, que retorna o *score* de *matching* multibiométrico destes dois modelos.

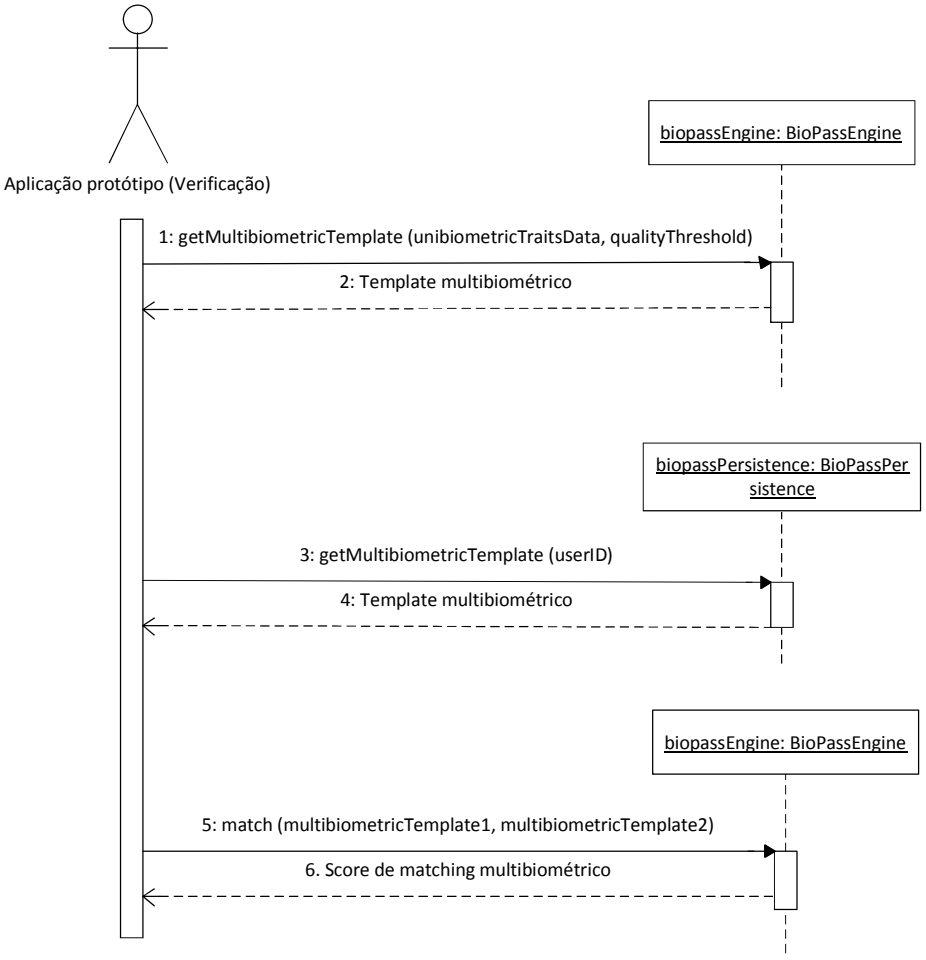

Figura 44. Diagrama de sequência da operação de Verificação

A fim de verificar o funcionamento e os resultados multibiométricos das operações da API de segurança, alguns testes foram realizados para diferentes parâmetros.

A Tabela 4 apresenta alguns resultados obtidos para a operação de verificação quando realizada para diferentes entradas. Observando-se o funcionamento da operação de *matching* multibiométrico, a Tabela 4 exibe, também, os valores intermediários, i.e., aqueles obtidos para o *matching* unibiométrico, a fim de demonstrar os resultados finais da verificação. Nota-se que, como o *matching* da face é realizado primeiro, o *matching* da impressão digital não será realizado caso o *score* para a face não esteja entre os limiares de aceitação e de rejeição, ocasionando em uma aceitação ou rejeição direta. Por exemplo, o *score* de *matching* do usuário 27 com sua pró-

pria face de *scan* número 29 (ID27_029) é menor do que o limiar de aceitação definido (150, vide Tabela 3). Por este motivo, a operação de *matching* também foi executada para impressão digital. A operação de verificação ocorreu com sucesso (*successful*) e o *score* multibiométrico foi o máximo (100), visto que o *score* de impressão digital está acima do máximo definido e, ainda, do limiar de aceitação definido para esta característica. Em contrapartida, no caso em que o score da verificação para face é maior do que o limiar definido, como é o caso da primeira linha da Tabela 4, a operação de *matching* não será realizada para impressão digital, resultando em um score multibiométrico máximo (100).

Tabela 4. Resultados de testes realizados sobre a operação de verificação

ID	Face	*Score*	Impressão digital	*Score*	Resultado da Verificação
20	ID20_010	180	ID20_002	-	Successful. Score: 100.
20	ID20_033	-	ID20_003	-	Exception: Insufficient Face data quality exception.
27	ID27_029	122	ID27_007	1118	Successful. Score: 100.
32	ID20_010	23	ID20_002	0	Successful. Score: 0.
20	ID32_028	0	ID32_007	-	Successful. Score: 0.
20	ID10_033	11	ID10_005	-	Successful. Score: 0.
10	ID20_019	15	ID20_002	-	Successful. Score: 0.
20	ID20_010	180	ID08_003	-	Successful. Score: 100.
20	ID08_029	0	ID20_006	-	Successful. Score: 0.
08	ID20_036	0	ID20_003	-	Successful. Score: 0.
20	ID24_036	0	ID20_002	-	Successful. Score 0.
13	ID25_010	0	ID07_004	-	Successful. Score 0.
40	ID42_019	0	ID42_008	-	Successful. Score 0.

A Tabela 5 exibe os mesmos resultados dos testes realizados para a operação de verificação na Tabela 4, com a configuração dos limiares de rejeição de impressão digital e de face para 0 e os limiares de aceitação para os valores de *score* máximos. É importante perceber que, neste caso, independente do *score* obtido, a operação de *matching* será realizada para todas as características e só então o resultado final é decidido. Agora, independente do *score* de *matching* do usuário 27 com sua própria face de *scan* número 29 (ID27_029), a operação de *matching* para impressão digital também será realizada, já que os limiares de aceitação e rejeição foram alterados para os valores de *score* máximo e o mínimo, respectivamente. Como nenhum *score* conseguirá passar do máximo, todos os que não forem ambos (face e impressão digi-

tal) iguais ao máximo resultarão em uma média ponderada dos valores normalizados. É importante ressaltar que o fato de uma verificação ter ocorrido com sucesso não significa que o usuário tenha sido reconhecido, mas, apenas, que o *score* multibiométrico foi obtido com sucesso.

Tabela 5. Resultados de testes realizados sobre a operação de verificação, com limiares de rejeição reduzidos para 0 e limiares de aceitação elevados para os valores de score máximos

ID	Face	*Score*	Impressão digital	*Score*	Resultado da Verificação
20	ID20_010	180	ID20_002	1080	Successful. Score: 100.
20	ID20_033	-	ID20_003	-	Exception: Insufficient Face data quality exception.
27	ID27_029	122	ID27_007	1118	Successful. Score: 83,89.
32	ID20_010	23	ID20_002	0	Successful. Score: 6,39.
20	ID32_028	0	ID32_007	11	Successful. Score: 0,92.
20	ID10_033	11	ID10_005	11	Successful. Score: 3,97.
10	ID20_019	15	ID20_002	2	Successful. Score: 4,33.
20	ID20_010	180	ID08_003	6	Successful. Score: 50,50.
20	ID08_029	0	ID20_006	692	Successful. Score: 50.
08	ID20_036	0	ID20_003	17	Successful. Score: 1,42.
20	ID24_036	0	ID20_002	1080	Successful. Score 50.
13	ID25_010	0	ID07_004	3	Successful. Score 0,25.
40	ID42_019	0	ID42_008	8	Successful. Score 0,67.

A Tabela 6 e a Tabela 7 mostram, através dos mesmos testes da Tabela 5, como os *scores* de *matching* multibiométrico são afetados modificando-se os pesos que cada característica exerce no *matching*. Deve-se salientar que o resultado da verificação, isto é, da operação de *matching* multibiométrico, é uma média ponderada dos valores de *score* unibiométricos normalizados. Nas tabelas de teste anteriores, os pesos de cada característica biométrica estavam definidos para o mesmo valor (50, vide Tabela 3).

Tabela 6. Resultados de testes realizados sobre a operação de verificação quando a face e a impressão digital possuem pesos 25 e 75, respectivamente

ID	Face	*Score*	Impressão digital	*Score*	Resultado da Verificação
20	ID20_010	180	ID20_002	1080	Successful. Score: 100.
20	ID20_033	-	ID20_003	-	Exception: Insufficient Face data quality exception.
27	ID27_029	122	ID27_007	1118	Successful. Score: 91,94.
32	ID20_010	23	ID20_002	0	Successful. Score: 3,19.
20	ID32_028	0	ID32_007	11	Successful. Score: 1,38.
20	ID10_033	11	ID10_005	11	Successful. Score: 2,90.
10	ID20_019	15	ID20_002	2	Successful. Score: 2,33.
20	ID20_010	180	ID08_003	6	Successful. Score: 25,75.
20	ID08_029	0	ID20_006	692	Successful. Score: 75.
08	ID20_036	0	ID20_003	17	Successful. Score: 2,13.
20	ID24_036	0	ID20_002	1080	Successful. Score 75.
13	ID25_010	0	ID07_004	3	Successful. Score 0,38.
40	ID42_019	0	ID42_008	8	Successful. Score 1.

Tabela 7. Resultados de testes realizados sobre a operação de verificação quando a face e a impressão digital possuem pesos 75 e 25, respectivamente

ID	Face	*Score*	Impressão digital	*Score*	Resultado da Verificação
20	ID20_010	180	ID20_002	1080	Successful. Score: 100.
20	ID20_033	-	ID20_003	-	Exception: Insufficient Face data quality exception.
27	ID27_029	122	ID27_007	1118	Successful. Score: 75,83.
32	ID20_010	23	ID20_002	0	Successful. Score: 9,58.
20	ID32_028	0	ID32_007	11	Successful. Score: 0,46.
20	ID10_033	11	ID10_005	11	Successful. Score: 5,04.
10	ID20_019	15	ID20_002	2	Successful. Score: 6,33.
20	ID20_010	180	ID08_003	6	Successful. Score: 75,25.
20	ID08_029	0	ID20_006	692	Successful. Score: 25.
08	ID20_036	0	ID20_003	17	Successful. Score: 0,71.
20	ID24_036	0	ID20_002	1080	Successful. Score 25.
13	ID25_010	0	ID07_004	3	Successful. Score 0,13.
40	ID42_019	0	ID42_008	8	Successful. Score 0,33.

5.3.3 Identificação

A operação de identificação busca, no banco de dados, um usuário que seja similar ao usuário cujos dados (impressão digital e face) foram passados na entrada. A Figura 45 ilustra esta operação.

Assim como as operações de cadastro e verificação, a operação de identificação também inicia transformando as imagens de face e impressão digital de entrada em arquivos binários (arrays de *bytes*) e os passa como objetos *UnibiometricData* para a operação *getMultibiometricTemplate* implementada pela classe *BioPassEngine*, usando, desta vez, o limiar de qualidade de identificação das características biométricas, também definido na aba de configurações. Em seguida, dependendo das configurações escolhidas para a classificação, a expressão de classificação é obtida. Por conseguinte, os registros de templates armazenados no banco de dados são recuperados através da operação *getUserMultibiometricTemplates*. Ela retorna objetos do tipo *UserMultibiometricTemplate*, que encapsulam o ID e o *template* de cada indivíduo. Todos os registros do banco de dados são recuperados a não ser que haja classificação. Neste caso, é chamada a versão da operação que recupera apenas os registros que sejam da mesma classe biométrica do indivíduo. Por fim, a operação de *matching* multibiométrico é realizada entre o *template* multibiométrico gerado a partir dos dados de entrada do indivíduo e os *templates* recuperados do banco de dados. O maior *score* é exibido para o usuário na aba da operação de identificação. Caso seja maior ou igual ao valor configurado para *score* multibiométrico, considera-se que o usuário foi identificado. O diagrama de sequência do caso de sucesso da operação de identificação é ilustrado na Figura 46.

Figura 45. Tela de identificação

É importante observar que, se o componente Persistence for implementado para *clusters* de estações de trabalho, a versão da operação *getUserMultibiometricTemplates* a ser usada é aquela que retorna um subconjunto de registros, entre o *offset* e o *limit* (vide Tabela 17 e Tabela 19, no Apêndice A). Assim, cada nó do *cluster* realizaria a operação de *matching* apenas em um subconjunto de registros.

Com o objetivo de certificar o correto funcionamento da operação de identificação, e das operações das APIs de segurança e de armazenamento que ela utiliza, alguns testes foram realizados. Com relação à variação dos parâmetros, os testes se assemelham àqueles realizados na operação de verificação da subseção anterior.

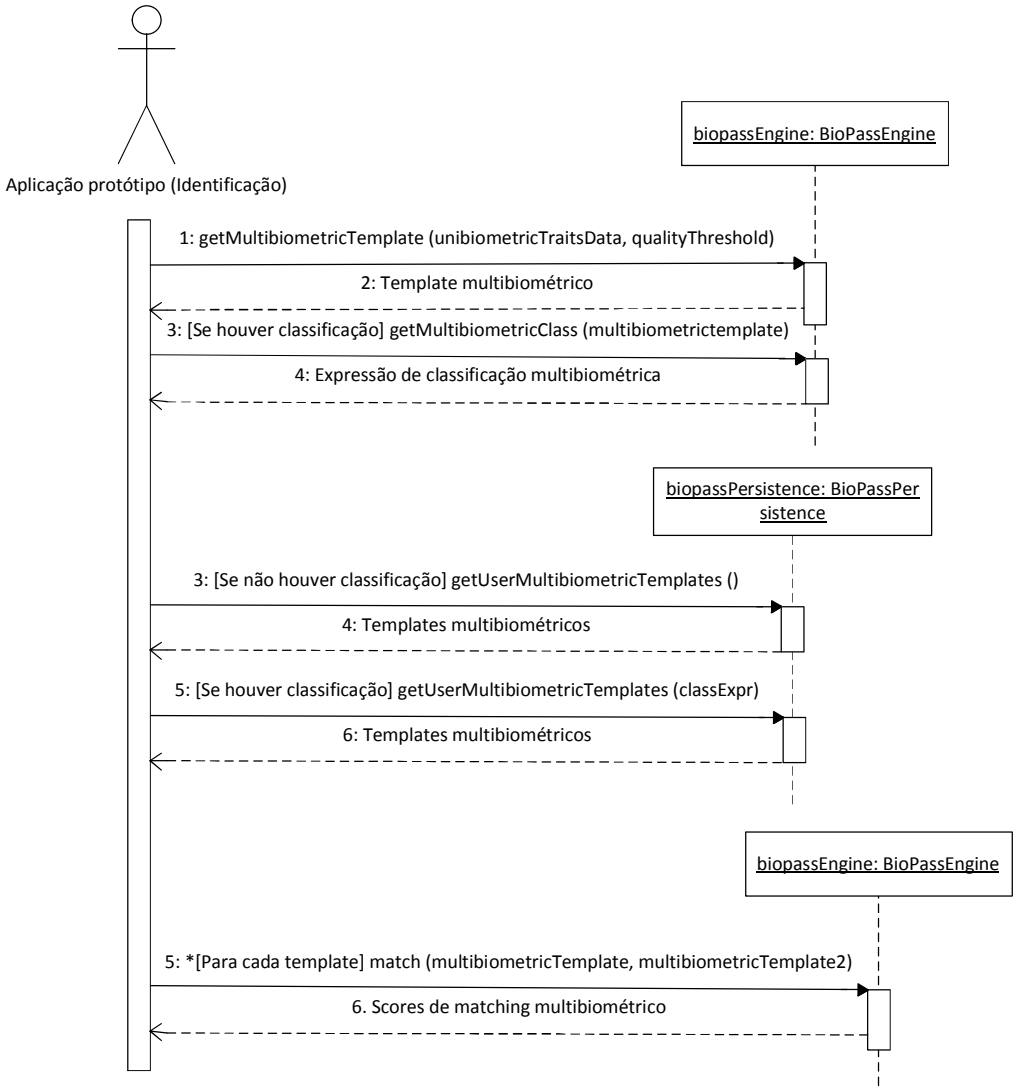

Figura 46. Diagrama de sequência da operação de Identificação

A Tabela 8 apresenta alguns resultados obtidos para esta operação quando realizada para diferentes entradas, inclusive para o tipo de entrada em que há a combinação de face e impressão digital de suspeitos diferentes, porém ambos cadastrados no banco de dados. Os *scores* de *matching* unibiométricos também são indicados. Os valores de configuração são os valores definidos na Tabela 3. Os testes para a operação de identificação seguem a mesma linha de raciocínio dos testes realizados para a operação de verificação quanto à execução das operações de *matching* unibiométrico respeitando-se os limiares definidos. No entanto, o resultado da operação de identificação, quando esta ocorre com sucesso, é o identificador do usuário

identificado (caso em que o *score* multibiométrico é maior ou igual ao seu limiar de aceitação) ou o identificador do usuário com maior *score* multibiométrico, isto é, aquele que mais se assemelha aos dados biométricos de entrada.

Tabela 8. Resultados de testes realizados sobre a operação de identificação

Face	***Score***	**Impressão digital**	***Score***	***Score* Mult.**	**Resultado da Identificação**
ID20_010	180	**ID20_002**	-	100	Successful. Identified. UserID: 20.
ID23_010	97	**ID23_002**	339	55,19	Successful. Not identified. Approx. UserID: 23.
ID20_010	180	**ID24_007**	-	100	Successful. Identified. UserID: 20.
ID28_029	-	**ID20_003**	-	-	Successful. User not identified.
ID43_019	-	**ID43_001**	-	-	Successful. User not identified.
ID01_015	-	**ID01_004**	-	-	InsufficientTraitDataQualityException

A Tabela 9 a seguir ilustra os mesmos testes da Tabela 8, exibindo a quantidade de itens utilizados para a identificação quando há a classificação por impressões digitais. Sem a classificação, o componente Persistence retorna todos os 41 registros do banco de dados. Nota-se que um resultado diferiu. Isto ocorreu porque a classificação não incluiu o registro ID20 (espiral), uma vez que a impressão digital ID24 (*left loop*) possui classe diferente da impressão digital ID20. Assim, como a face ID20 não fez parte da operação de *matching*, não houve interferência do limiar de aceitação de face neste caso.

Tabela 9. Resultados de testes realizados sobre a operação de identificação com classificação por impressões digitais

Qtde de Registros	Face	*Score*	Impressão digital	*Score*	*Score* Mult.	Resultado da Identificação
10	**ID20_010**	180	**ID20_002**	-	100	Successful. Identified. UserID: 20.
16	**ID23_010**	97	**ID23_002**	339	55,19	Successful. Not identified. Approx. UserID: 23.
16	**ID20_010**	20	**ID24_007**	404	39,22	Successful. Not identified. Approx. UserID: 24.
10	**ID28_029**	-	**ID20_003**	-	-	Successful. User not identified.
16	**ID43_019**	-	**ID43_001**	-	-	Successful. User not identified.
13	**ID01_015**	-	**ID01_004**	-	-	InsufficientTraitDataQualityException

A classificação por face não foi implementada porque o algoritmo de faces utilizado não define propriedades de classificação para esta característica biométrica.

Os mesmos testes iniciais foram executados depois de configurados os limiares de rejeição de impressão digital e de face para 0 e os limiares de aceitação para os valores de *score* máximos (Tabela 10). Em seguida, são mostrados os resultados quando modificados os pesos que cada característica exerce no *matching* (Tabela 11 e Tabela 12).

Tabela 10. Resultados de testes realizados sobre a operação de identificação, com limiares de rejeição reduzidos para 0 e limiares de aceitação elevados para os valores de score máximos

Face	*Score*	Impressão digital	*Score*	*Score* Mult.	Resultado da Identificação
ID20_010	180	**ID20_002**	1080	100	Successful. Identified. UserID: 20.
ID23_010	97	**ID23_002**	339	55,19	Successful. Not identified. Approx. UserID: 23.
ID20_010	180	**ID24_007**	17	51,42	Successful. Not identified. Approx. UserID: 20.
ID28_029	0	**ID20_003**	825	50	Successful. Not identified. Approx. UserID: 20.
ID43_019	28	**ID43_001**	12	8,78	Successful. Not identified. Approx. UserID: 19.
ID01_015	-	**ID01_004**	-	-	InsufficientTraitDataQualityException

Tabela 11. Resultados de testes realizados sobre a operação de identificação quando a face e a impressão digital possuem pesos 25 e 75, respectivamente

Face	*Score*	Impressão digital	*Score*	*Score* Mult.	Resultado da Identificação
ID20_010	180	**ID20_002**	1080	100	Successful. Identified. UserID: 20.
ID23_010	97	**ID23_002**	339	55,85	Successful. Not identified. Approx. UserID: 23.
ID20_010	20	**ID24_007**	404	53,28	Successful. Not identified. Approx. UserID: 24.
ID28_029	0	**ID20_003**	825	75	Successful. Not identified. Approx. UserID: 20.
ID43_019	28	**ID43_001**	12	5,39	Successful. Not identified. Approx. UserID: 19.
ID01_015	-	**ID01_004**	-	-	InsufficientTraitDataQualityException

Tabela 12. Resultados de testes realizados sobre a operação de identificação quando a face e a impressão digital possuem pesos 75 e 25, respectivamente

Face	***Score***	**Impressão digital**	***Score***	***Score* Mult.**	**Resultado da Identificação**
ID20_010	180	**ID20_002**	1080	100	Successful. Identified. UserID: 20.
ID23_010	97	**ID23_002**	339	54,54	Successful. Not identified. Approx. UserID: 23.
ID20_010	180	**ID24_007**	17	75,71	Successful. Not identified. Approx. UserID: 20.
ID28_029	83	**ID20_003**	14	35,17	Successful. Not identified. Approx. UserID: 28.
ID43_019	28	**ID43_001**	12	12,29	Successful. Not identified. Approx. UserID: 35.
ID01_015	-	**ID01_004**	-	-	InsufficientTraitDataQualityException

5.3.4 Autenticação contínua

Nesta subseção, a operação de autenticação contínua é ilustrada com um cenário de uso, descrito a seguir. A Figura 47 ilustra a tela da operação de autenticação contínua. Os limiares de aceitação e rejeição foram configurados para os valores de *score* máximos e mínimos respectivamente e o limiar de aceitação multibiométrico para 50.

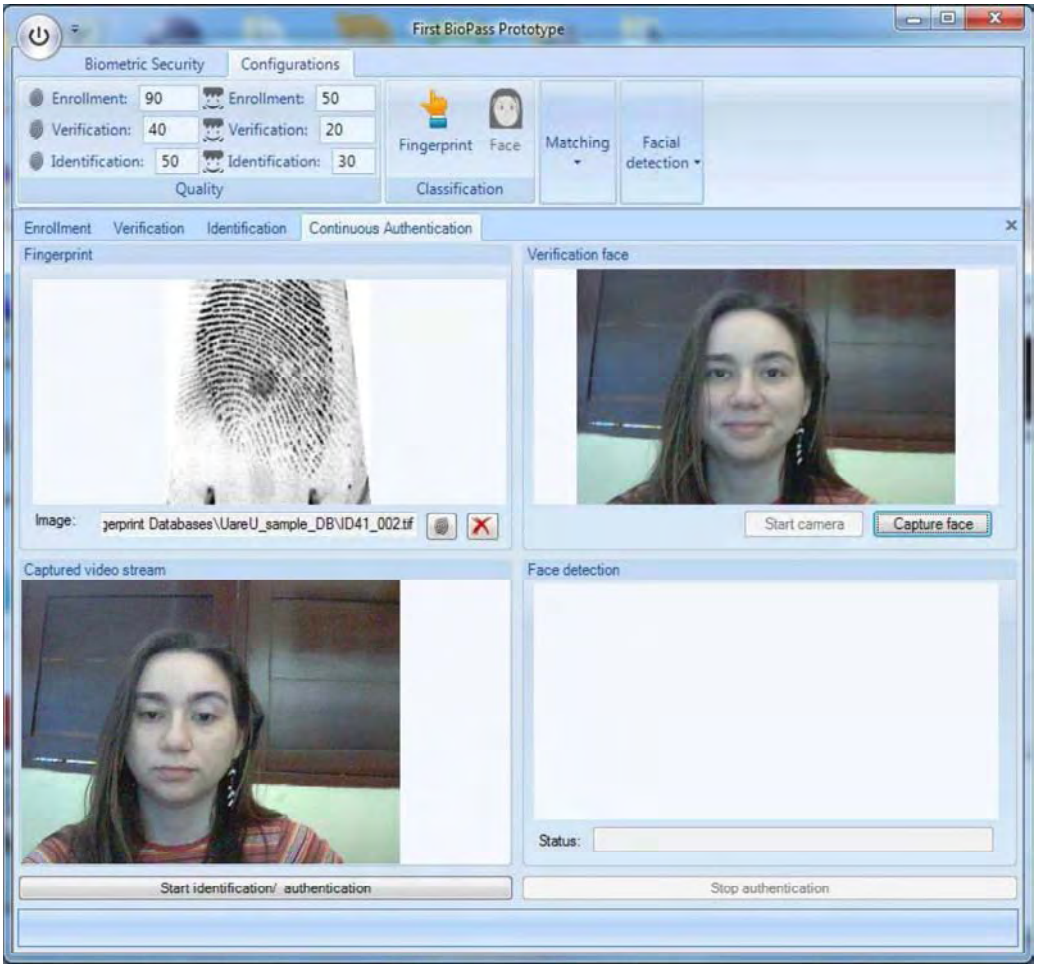

Figura 47. Tela de autenticação contínua

A caixa “Fingerprint” foi carregada com uma imagem de impressão digital do banco de dados U.are.U, a segunda digitalização do usuário 41 (ID41_002). A caixa “Captured video stream” mostra o que está sendo capturado pela webcam. Quando pressionado o botão “Start camera”, um *stream* de vídeo se inicia. A face usada para identificação é a face mostrada na caixa “Verification face”, que foi capturada pressionando-se o botão “Capture face”. Assim, há uma identificação inicial, que utiliza a imagem de impressão digital de entrada e a imagem capturada pela webcam mostrada na caixa “Verification face”. Feita a identificação, a operação de autenticação contínua se inicia. A Figura 48 ilustra a situação descrita.

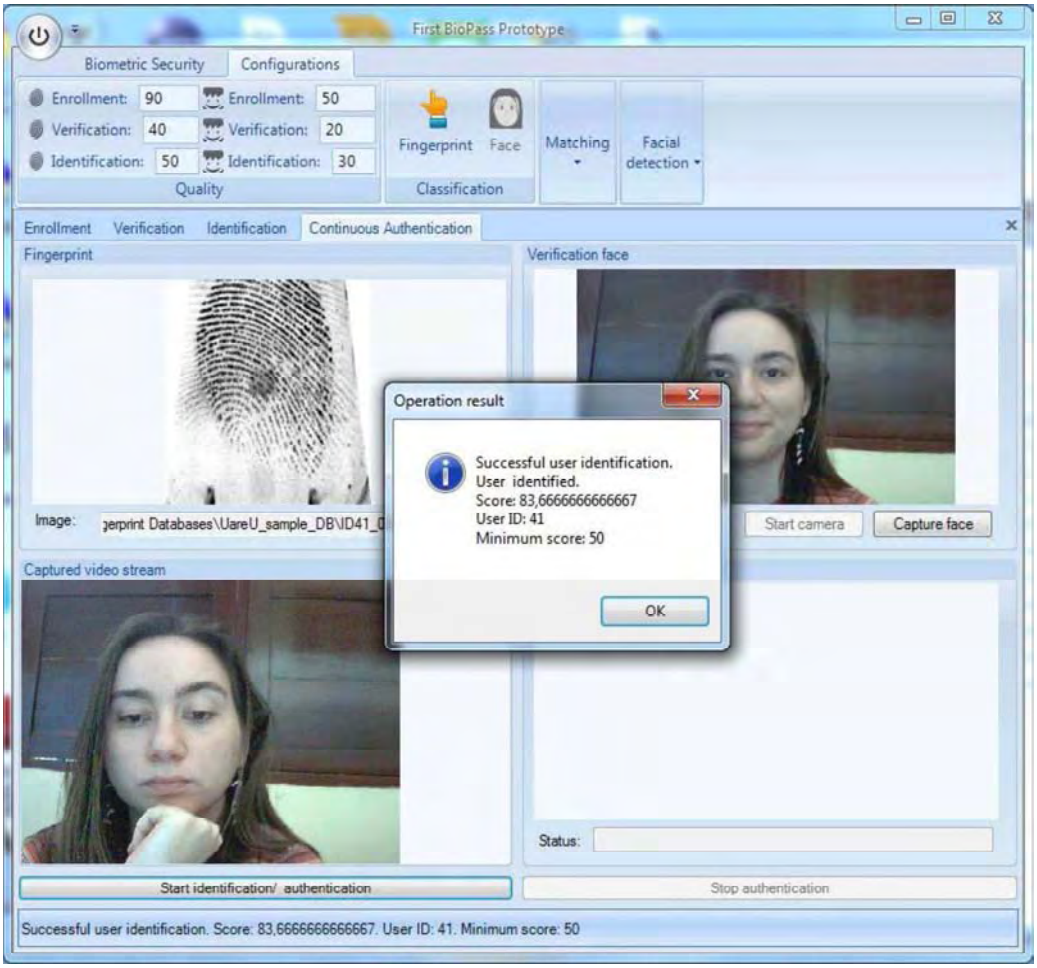

Figura 48. Usuário identificado para a operação de autenticação contínua

A caixa “Face detection” é carregada, de tempos em tempos (de acordo com o intervalo de tempo configurado), com a última face capturada do stream de vídeo. A Figura 49 ilustra este aspecto.

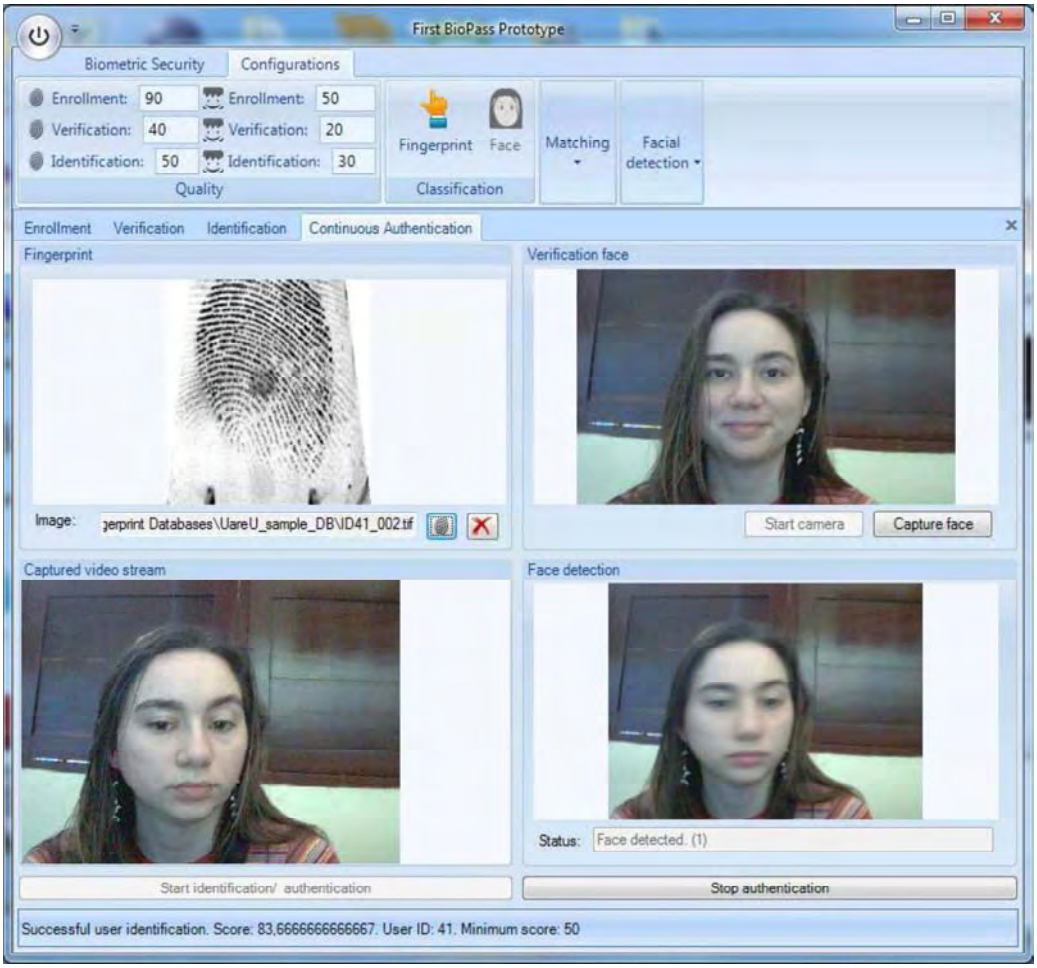

Figura 49. Captura de faces na caixa "Face Detection"

Se nenhuma face for detectada dentro do tempo limite (*time out*) configurado, a operação é terminada. No entanto, se o usuário erguer a cabeça, virá-la para o lado ou incliná-la de forma que a face não possa ser detectada, por exemplo, mas novamente posicioná-la corretamente em frente à webcam logo em seguida, antes que seja atingido o tempo limite, a operação de autenticação contínua irá continuar normalmente. A caixa de texto "Status" é configurada com informação textual sobre a detecção, isto é, se a face foi detectada ou não. Caso não tenha sido detectada mais de uma vez, uma contagem é exibida ao lado da informação textual. As Figuras Figura 50, Figura 51 e Figura 52 mostram estas situações.

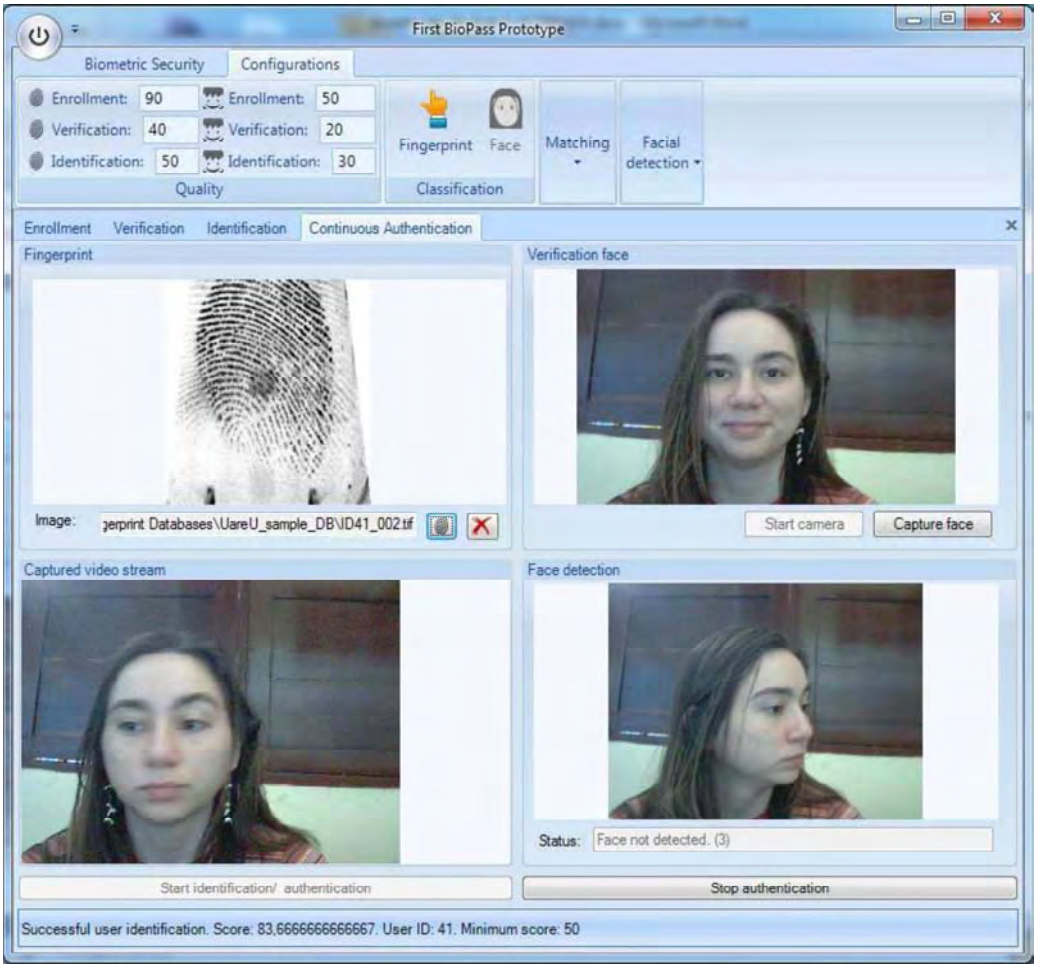

Figura 50. Face não detectada pela terceira vez

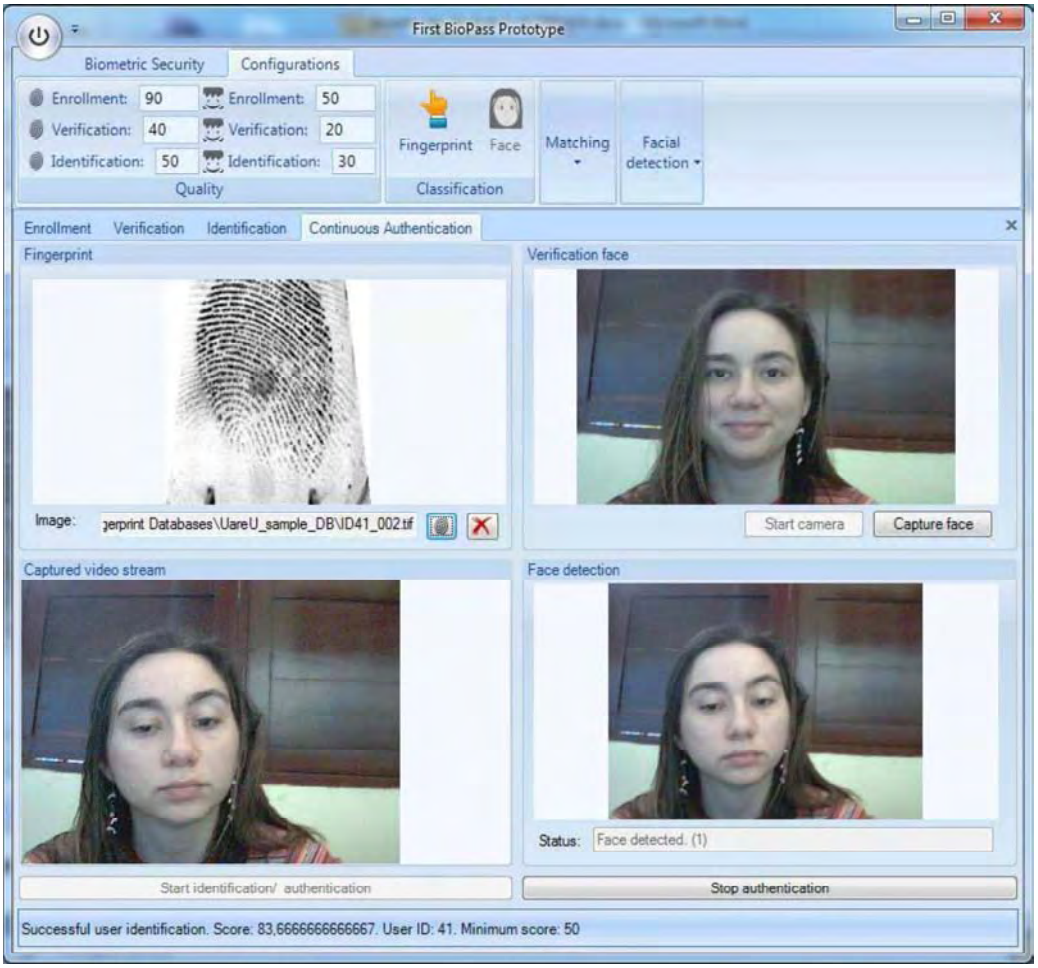

Figura 51. Face reposicionada em frente à webcam antes de o tempo limite ser atingido

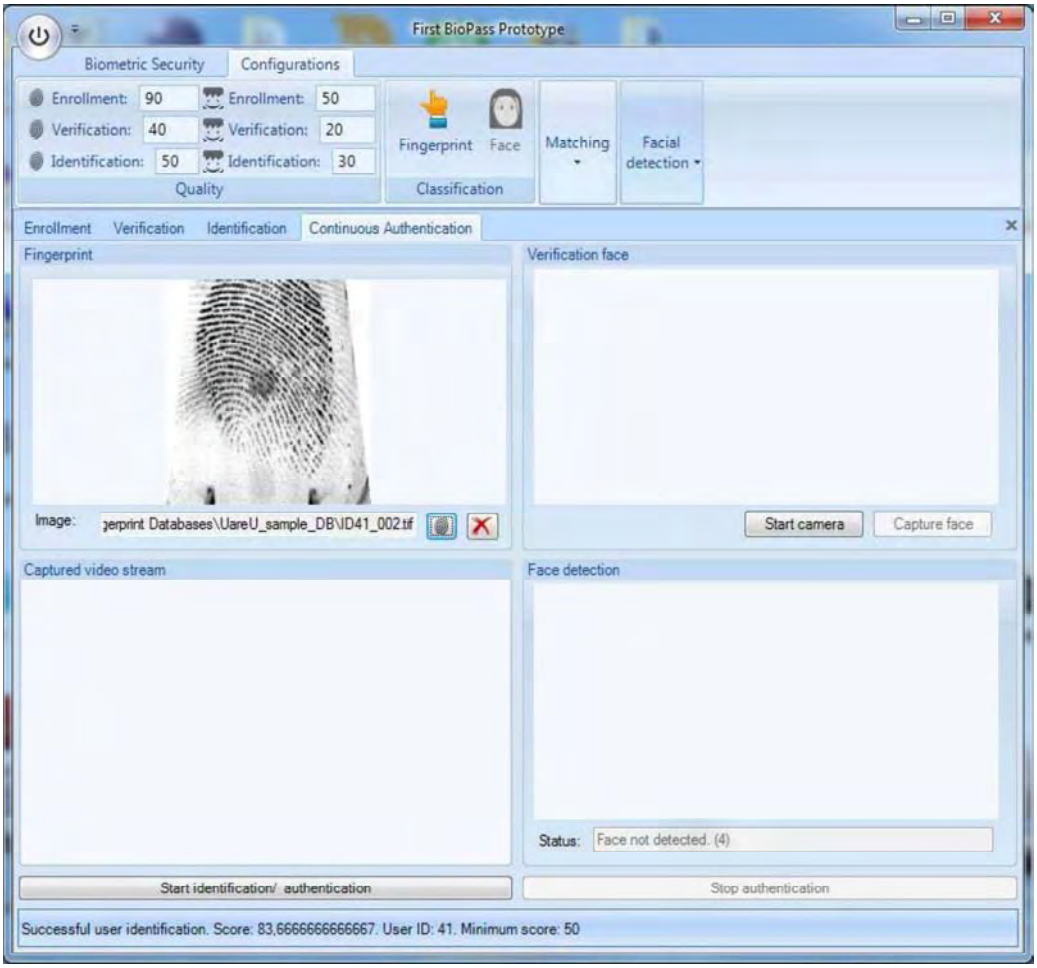

Figura 52. Tempo limite atingido sem detectar a face; operação de autenticação contínua terminada

A autenticação contínua pode ser bastante útil em diversos cenários e aplicações.

Em aplicações bancárias, por exemplo, ela pode ser utilizada para elevar nível de garantia de que é realmente o cliente legítimo quem realiza as transações, reduzindo as chances de fraude por roubos de *tokens* (e. g., cartões), senhas ou uso de sessões abertas remanescentes. Neste caso, o processo de autenticação contínua poderia fazer a verificação mais frequentemente, para que o assaltante não tivesse tempo para concluir com sucesso as operações de seu acesso indevido.

Outro exemplo de aplicação onde a autenticação contínua poderia ser usada seria para o controle de acesso em sistemas de organizações em geral, como aeroportos, serviços internos de instituições financeiras e governo. Assim, quando o funcionário, por algum motivo, tivesse que se ausentar do computador, a sessão seria tra-

vada automaticamente e seria reativada quando ele novamente se posicionasse em frente à câmera do computador. Isto evitaria que usuários não autorizados utilizassem o sistema em sua ausência ou, até mesmo, que um funcionário utilizasse o sistema na conta de outro funcionário, passando-se por ele.

Além dessas, há também as aplicações de EAD (Educação a Distância). A autenticação contínua poderia ser usada, dentre outras funcionalidades, para a aplicação das provas, a fim de certificar continuamente que o aluno está fazendo a prova, e não que outra pessoa esteja fazendo por ele. Isto poderia funcionar com sucesso em presídios que permitem que o prisioneiro participe de tais programas, por exemplo.

O ponto relevante do suporte a autenticação contínua oferecido pela API do Engine é que os parâmetros (intervalo de tempo e tempo limite) e a operação de detecção de características podem ser usados para implementar um processo de autenticação contínua conforme desejado e não necessariamente como foi implementado na aplicação protótipo.

Dependendo da aplicação e da característica biométrica a ser usada, a autenticação em si pode ser feita sempre, logo em seguida à detecção. Por exemplo, a forma de digitar (*keystroke*) poderia ser checada (detectada) e logo em seguida verificada. Caso aquela forma de digitar não pertencesse ao usuário que incialmente fez o *login* no sistema, o acesso ao sistema poderia ser bloqueado até que um novo *login* fosse feito. Caso a digitação não seja detectada, o sistema pode também bloquear o acesso após o tempo limite definido.

Além disso, a presença de mais características biométricas a serem utilizadas para a detecção pode permitir a construção de um método de autenticação contínua mais robusto e menos inconveniente ou intrusivo.

Capítulo

6

Considerações finais

"O fim nunca é o fim. Um novo desafio o aguarda."

Kid Cudi

Neste capítulo são apresentadas as contribuições do trabalho bem como discussões com relação aos resultados obtidos até o presente momento. Neste sentido, são realizados comentários com relação às dificuldades encontradas, suas soluções e trabalhos futuros. Por fim, as conclusões são apresentadas.

6.1 Discussão

A primeira dificuldade encontrada na construção da arquitetura do BioPass e, consequentemente, das APIs desenvolvidas e apresentadas neste trabalho está relacionada ao domínio da aplicação. A arquitetura foi projetada observando-se as aplicações comerciais existentes, suas tendências e o problema a ser resolvido, isto é, ter uma arquitetura capaz de representar sistemas uni e multibiométricos de maneira genérica e segura. A maior dificuldade, entretanto, foi o projeto e especificação das APIs. A API de armazenamento possui operações simples e que precisaram considerar alguns aspectos como, por exemplo, suporte a *clusters* de estações de trabalhos e classificação biométrica, além da existência de um formato para os *templates* multibiométricos, um ponto significativo desta API. A API de segurança, por sua vez, mostrou-se como o maior desafio relacionado ao domínio da aplicação, pois suas operações devem refletir o que o problema precisa para ser resolvido, ou seja, qual a

melhor forma de se construir sistemas biométricos considerando-se, ainda, características nem sempre implementadas pelos sistemas existentes, como é o caso da autenticação contínua e da segurança de *templates*. As APIs disponibilizadas pelos desenvolvedores de aplicações biométricas nem sempre seguem um padrão e, muitas vezes, possuem operações e características próprias e específicas a seus sistemas. Portanto, definir operações para o componente Engine se mostrou um grande desafio neste trabalho.

A fim de validar e realizar testes nas APIs definidas, foi necessário obter algoritmos biométricos de, no mínimo, duas características biométricas, para que se pudesse demonstrar a multimodalidade da arquitetura. Há muitos trabalhos apresentados na literatura, porém, poucas implementações gratuitas ou completas estão disponíveis. Neste sentido, outra dificuldade encontrada foi a obtenção de implementações adequadas para que os testes fossem realizados. Depois de implementada a primeira versão das APIs, observou-se que, na prática, algumas funcionalidades eram necessárias e outras poderiam ser modificadas para facilitar a codificação. Com isso, as APIs foram iterativamente modificadas, até a obtenção de uma especificação estável e melhor utilizável na prática.

Outro problema, além das dificuldades para se obter implementações de algoritmos biométricos adequadas, foi o de encontrar uma implementação aberta para que fosse possível a adição de um método de segurança de *templates* ou, até mesmo, uma implementação que já incluísse um método de segurança de *templates*. Desta forma seria possível fazer a validação das APIs considerando-se este aspecto. Os algoritmos utilizados (Neurotechnology, 2010) possuem código fechado, o que impossibilitou a implementação de esquemas de proteção de *templates*, uma vez que tais esquemas estão integrados às operações de criação de *templates* e de *matching*.

Embora a segurança de *templates* seja um diferencial das APIs desenvolvidas neste trabalho, ela não foi utilizada pelos algoritmos biométricos adotados. De forma semelhante, o suporte a *clusters* de estações de trabalho também não foi utilizado na prática. Em contrapartida, foi demonstrado como o suporte a segurança de *templates* e a *clusters* foi abordado, e como poderia ser explorado na aplicação.

A autenticação contínua também pode ser considerada um diferencial do trabalho. Ela é um fator importante e que auxilia fortemente a segurança das aplicações no que diz respeito ao controle de acesso. Há várias aplicações em que o uso desta funcionalidade seria extremamente adequado, como no caso de aplicações bancárias, sistemas aeroportuários e de EAD, exemplos dados na Seção 5.3.4.

Mesmo com as dificuldades encontradas, apesar de suas limitações, o trabalho desenvolvido também apresenta outros diferenciais a respeito dos trabalhos correlatos encontrados na literatura.

A BioAPI, por exemplo, é um trabalho complexo, que considera fatores de implementação biométrica e os explora em seu baixo nível. Apesar de estar bem consolidada e de prover um modelo de autenticação biométrica genérico adequado para qualquer forma de tecnologia biométrica, ela não é multimodal. Por isso, ela pode ser usada em conjunto ao BioPass, na implementação interna dos algoritmos de reconhecimento biométrico de cada característica a ser usada. Além disso, a BioAPI não define requisitos de segurança. A API desenvolvida neste trabalho, por outro lado, é multimodal e oferece suporte a segurança de *templates*, embora este suporte não tenha sido utilizado na aplicação protótipo desenvolvida.

(Sim, Zhang, Janakiraman, & Kumar, 2007), por sua vez, implementaram um sistema que precisa de sensores especiais para evitar a intrusão, uma vez que todas as características necessariamente devem fazer parte da autenticação contínua. O sistema é embutido em um sistema operacional, sendo dependente de plataforma.

O MUBAI (Abate, Marsico, Riccio, & Tortora, 2010) apresenta uma arquitetura simples e utiliza os conceitros de agentes classificadores e agente supervisor. Se comparado ao trabalho desenvolvido, os agentes classificadores corresponderiam às classes que implementam as interfaces unibiométricas (*BioPassEngineFaceTrait* e *BioPassEngineFingerprintTrait* no protótipo desenvolvido), e o agente supervisor corresponderia à classe que implementa a interface multibiométrica (*BioPassEngine* no protótipo desenvolvido). A idéia parece ser semelhante, mas (Abate, Marsico, Riccio, & Tortora, 2010) não apresentam um trabalho de engenharia de sotware demonstrando a API utilizada na arquitetura do MUBAI, que é apresentada de maneira rudimentar.

O HUMABIO (HUMABIO) ainda está em desenvolvimento. Ele explora várias características biométricas, mas o foco está principalmente nos algoritmos de reconhecimento biométrico. Além disso, várias das características exploradas ainda possuem baixa aceitabilidade principalmente devido ao alto custo dos sensores na atualidade.

Por fim, o BioID (BioID, 2010) foi desenvolvido especialmente para a Web e emprega a idéia de autenticação biométrica como serviço. Ele utiliza o conceito do OpenID (OpenID) para interoperabilidade e implementa a BioAPI 1.1. Apesar de herdar as limitações desta, o BioID se mostra bastante eficaz e coerente com as ne-

cessidades da atualidade. Além disso, seus objetivos estão alinhados aos objetivos do BioPass. No entanto, a arquitetura e API do sistema não são apresentadas. Desta forma seria possível avaliar e comparar o BioID ao trabalho aqui desenvolvido mais criteriosamente, principalmente com relação à forma como a multimodalidade é abordada.

6.2 Contribuições

Considera-se que os objetivos definidos para este trabalho foram atingidos, pois foi possível criar e validar as APIs de segurança e armazenamento do BioPass, deixando, portanto, as contribuições a seguir:

1. A arquitetura multibiométrica do BioPass, que é um esquema geral baseado nas aplicações (multi) biométricas existentes no mercado e apresentadas na literatura, considerando as necessidades atuais deste tipo de aplicação;
2. A API de armazenamento (componente Persistence da arquitetura), que define operações gerais de armazenamento para sistemas (multi) biométricos. Ela inclui operações que dão suporte a *clusters* de estações de trabalho definindo limites de registros para que cada nó do *cluster* possa realizar parte das tarefas (*matching*), a fim de elevar a eficiência geral do sistema. Além disso, um padrão estrutural extensível para *templates* multibiométricos é definido, permitindo que novas características sejam adicionadas quando necessário;
3. A API de segurança (componente Engine da arquitetura), que define operações relacionadas ao reconhecimento (multi) biométrico. Ela dá suporte à autenticação contínua através da detecção da característica biométrica em intervalos de tempo, podendo-se, a partir daí, definir quando uma sessão deve expirar, por exemplo. Ainda, o componente Engine dá suporte à segurança de *templates* (esquemas não-inversíveis) utilizando uma abordagem em que as chaves são encapsuladas por um objeto genérico, o qual o sistema de criação de *templates* será capaz de "decifrar" e utilizar. Além disso, a API de segurança suporta a adição de novas características biométricas sem necessidade de mudanças no código, isto é, bastando-se apenas implementar as interfaces unibiométricas e adicioná-las na inicialização da classe multibiométrica;
4. A aplicação protótipo, que utiliza algoritmos unibiométricos para validar e demonstrar na prática as APIs especificadas, ilustrando as operações princi-

pais dos sistemas biométricos e as características relevantes das APIs, como os parâmetros de configuração multibiométricos e a operação que utiliza autenticação contínua.

6.3 Trabalhos futuros

Um dos pontos positivos e mais significativos deste trabalho para o âmbito acadêmico é que seus trabalhos futuros não consistem em uma mera descrição de pontos que poderiam melhorar o trabalho expressivamente, mas que muitas vezes não são alcançados ou implementados, sendo ele mais um dentre tantos trabalhos descontinuados; o trabalho deverá ser utilizado comercialmente no futuro e, portanto, terá uma continuidade.

Considerando-se o projeto BioPass como um todo, há uma variedade de direções a serem seguidas visando à expansão do trabalho ou conclusão do projeto em si. Quanto a este trabalho especificamente, podem-se considerar como trabalhos futuros mais relevantes e necessários em curto ou médio prazo:

1. Realizar experimentos e testes a fim de obter resultados relevantes relacionados a como a API multibiométrica influencia o desempenho do sistema de maneira geral, além das taxas de erro multibiométricas para diferentes algoritmos;
2. Pesquisar sobre algoritmos de detecção de vivacidade, isto é, algoritmos que detectam se a característica biométrica inserida provém de um ser vivo e não de uma reprodução “sem vida” do traço. A Figura 53 e a Figura 54 a seguir demonstram, utilizando a operação de autenticação contínua, como a ausência de um método de detecção de vivacidade pode afetar a segurança de um sistema biométrico.

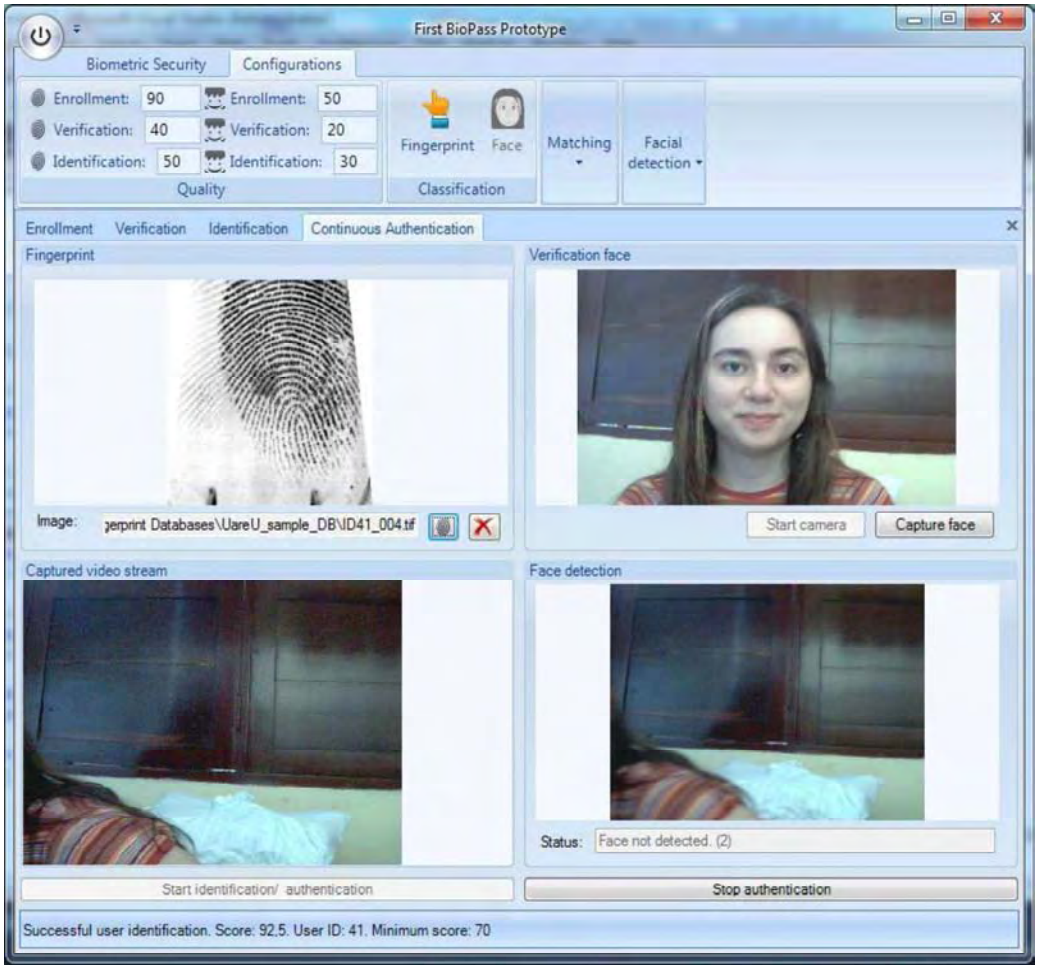

Figura 53. Tela de autenticação contínua: face não detectada

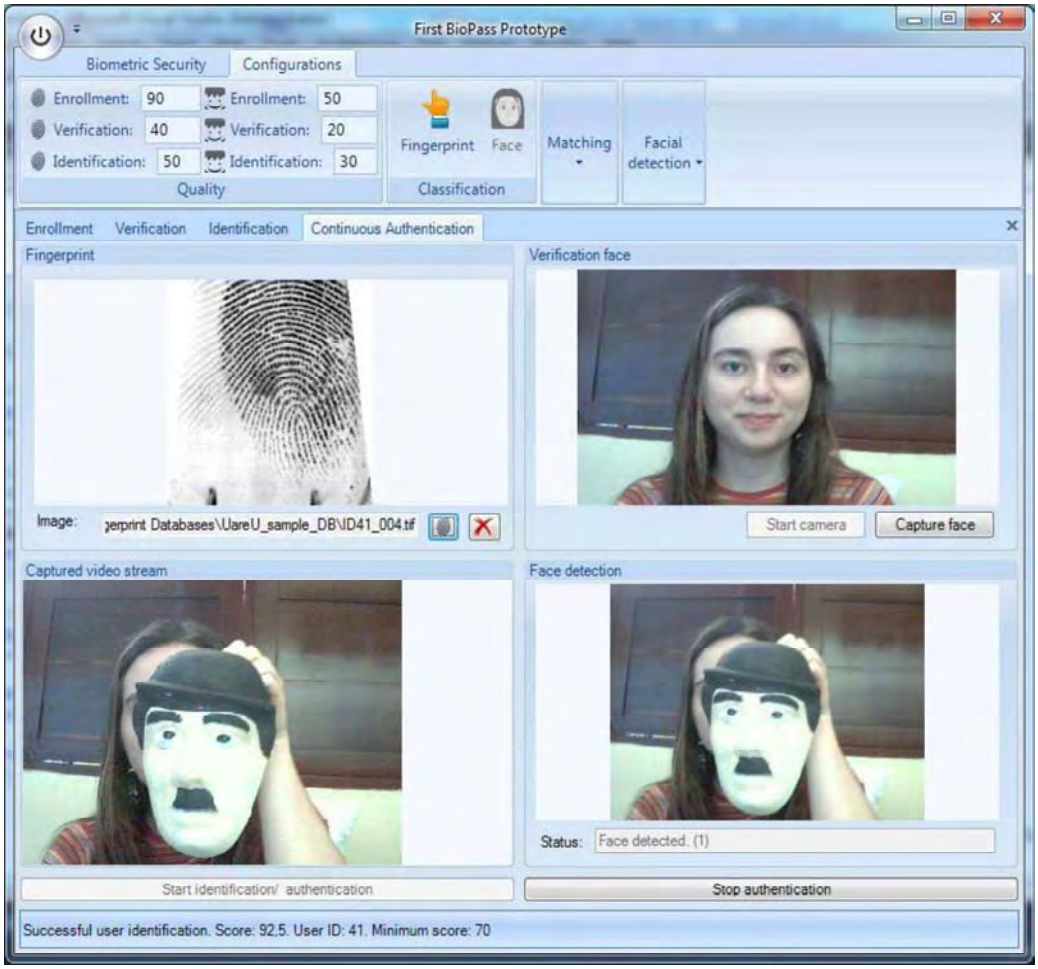

Figura 54. Tela de autenticação contínua: face detectada; desvantagem da não detecção de vivacidade

A partir daí, decidir como a API de segurança pode suportar tais algoritmos, mantendo a simplicidade e facilidade de uso para os desenvolvedores, e como a API poderia suportar a detecção de vivacidade quando feita diretamente através do hardware biométrico;

3. Implementar para a API multibiométrica um algoritmo biométrico que suporte segurança de *templates* (possivelmente o algoritmo de reconhecimento por impressões digitais do BioPass, que se encontra em desenvolvimento) a fim de verificar como a API se comporta e quais as dificuldades (se existirem) para seu uso no desenvolvimento de software;
4. Especificar a API do componente Identification and Authentication e implementá-lo para fazer testes de integração com os componentes Engine e

Persistence e verificar se o suporte a *clusters* de estações de trabalhos foi especificado de maneira conveniente do ponto de vista da codificação;

5. Realizar testes para verificar o desempenho do uso das APIs em larga escala, especialmente da operação de identificação multibiométrica, considerando um banco de dados com um número significativo (milhares ou milhões) de usuários cadastrados;
6. Realizar testes com desenvolvedores, a fim de certificar que as APIs são autoexplicativas e fáceis de usar. Pode-se verificar também se há alguma necessidade não prevista e que se mostre como tendência no desenvolvimento de sistemas biométricos.

6.4 Conclusões

As APIs de segurança e armazenamento do BioPass foram especificadas e validadas. Juntamente com o esquema geral de arquitetura que foi definido, pode-se dizer que os objetivos (Seção 1.2) foram alcançados, fundamentando sua justificativa descrita na seção 1.3.

O resultado do trabalho se mostrou consistente e coerente com o problema a ser resolvido. Com a ferramenta protótipo que foi implementada, foi possível perceber que as APIs responderam satisfatoriamente, facilitando o processo de desenvolvimento de quem utiliza as interfaces definidas. O trabalho desenvolvido se mostra como um avanço na área de segurança biométrica, especialmente se comparado aos trabalhos a ele relacionados (Seção 2.5). Estes apresentam soluções mais restritas do ponto de vista da arquitetura em si, suas funcionalidades e sua escalabilidade (capacidade de crescer). Além disso, eles se mostram menos seguros no que diz respeito à utilização de métodos de criptografia específicos, como é o caso da segurança específica para *templates*, e a outras preocupações com o acesso indevido, como é o caso da autenticação contínua.

A utilização da linguagem UML para modelagem dos componentes e a sequência de chamada às operações facilitou o processo de projeto e evolução das APIs. As tecnologias utilizadas apresentaram-se eficientes. O uso de uma linguagem de alto nível e orientada a objetos (C#) acelerou o trabalho de codificação, evolução e testes. A reflexão possibilitou a criação de um software que se adapta em tempo de execução, permitindo que as novas características biométricas sejam adicionadas sem necessidade de mudanças no código já existente.

Existem várias possibilidades de trabalhos futuros visando à consolidação das APIs através da realização de testes e pesquisa para adição de novas funcionalidades à especificação. Pelo fato de fazer parte de um produto a ser explorado comercialmente, o BioPass, fica evidente a evolução e consolidação destas APIs no futuro para sua utilização na prática.

O futuro das aplicações (multi) biométricas se mostra bastante promissor. Percebe-se que a utilização de tais softwares vem crescendo exponencialmente nos últimos anos, sendo utilizada para controle de acesso em empresas de pequeno, médio e grande porte. No entanto, pouca importância – por parte de quem compra o software e de quem o utiliza para reconhecimento – tem sido dada à segurança dos dados biométricos, dados estes que permanecem praticamente os mesmos durante a vida do indivíduo, não podendo, portanto, ser revogados em caso de roubo. Por isso, acredita-se que a pesquisa na área de segurança de *templates* biométricos deve ganhar grande enfoque nos próximos anos, uma vez que os trabalhos atuais reduzem o desempenho e/ ou eficácia do reconhecedor biométrico, muitas vezes elevando suas taxas de erro. Por fim, vislumbra-se uma atividade notória de pesquisa na área de reconhecimento de sinais e padrões com foco em características biométricas menos utilizadas. Isto contribuirá para a consolidação de outros traços biométricos, diferentemente da impressão digital e da face, por exemplo, que já se consideram consolidados atualmente. Assim, a crescente utilização de sistemas multibiométricos multimodais vem colocando em foco, também, a questão da existência de uma arquitetura e infraestrutura do sistema em si, de forma que haja uma base sólida para progressos futuros no momento em que novas características biométricas sejam adicionadas.

Apêndice

A

Persistence API

A.1. Interface IBioPassPersistence

As operações que fazem parte da interface *IBioPassPersistence*, que constitui a API do componente Persistence, são apresentadas a seguir.

Tabela 13. Especificação da operação registerMultibiometricTemplate (userID, multibiometricTemplate)

void	registerMultibiometricTemplate (userID, multibiometricTemplate)
Esta operação registra, no banco de dados, o *template* multibiométrico de um usuário relacionado ao identificador deste usuário no banco de dados da aplicação (*userID*).	
Parâmetros:	**userID**: valor inteiro correspondente ao identificador do usuário no banco de dados da aplicação. **multibiometricTemplate**: *array* de bytes com o *template* multibiométrico do usuário.
Retorno:	Esta operação não retorna nenhum valor (*void*).
Exceções:	

Tabela 14. Especificação da operação registerMultibiometricTemplate (userID, multibiometricTemplate,classificationExpression)

void	registerMultibiometricTemplate (userID, multibiometricTemplate, classificationExpression)
Esta operação registra, no banco de dados, o *template* multibiométrico de um usuário relacionado ao identificador deste usuário no banco de dados da aplicação (*userID*). Além disto, as características de classificação do usuário são armazenadas fora do *template*, a fim de elevar a velocidade de identificação.	
Parâmetros:	**userID**: valor inteiro correspondente ao identificador do usuário no banco de dados da aplicação.

	multibiometricTemplate: *array* de bytes com o *template* multibiométrico do usuário. **classificationExpression**: *string* representando a expressão com a classificação do *template*.
Retorno:	Esta operação não retorna nenhum valor (*void*).
Exceções:	**InvalidClassificationExprException**: expressão de classificação inválida.

Tabela 15. Especificação da operação getMultibiometricTemplate (userID)

byte[]	getMultibiometricTemplate (userID)
Esta operação recupera o *template* multibiométrico de um usuário específico.	
Parâmetros:	**userID**: valor inteiro correspondente ao identificador do usuário no banco de dados da aplicação.
Retorno:	Esta operação retorna um *array* de bytes, correspondente ao template multibiométrico do usuário.
Exceções:	**InvalidUserIDException**: o identificador do usuário é inválido.

Tabela 16. Especificação da operação getUserMultibiometricTemplates ()

UserMultibiometricTemplate[]	getUserMultibiometricTemplates ()
Esta operação recupera todos os *templates* multibiométricos registrados no banco de dados.	
Parâmetros:	
Retorno:	Esta operação retorna um *array* de elementos do tipo *UserMultibiometricTemplate*, correspondente a todos os registros de *templates* multibiométricos armazenados no banco de dados.
Exceções:	

Tabela 17. Especificação da operação getUserMultibiometricTemplates (offset, limit)

UserMultibiometricTemplate[]	getUserMultibiometricTemplates (offset, limit)
Seja n o valor de *limit* e m o valor de *offset*. Esta operação recupera o subconjunto formado pelos n *templates* multibiométricos registrados no banco de dados a partir do m-ésimo *template*.	
Parâmetros:	**offset**: valor inteiro correspondente à posição do primeiro *template* em um conjunto de templates. **limit**: valor inteiro correspondente ao número de *templates* a serem recuperados a

	partir de *offset*.
Retorno:	Esta operação retorna um *array* de elementos do tipo *UserMultibiometricTemplate*, correspondente a um conjunto com 1 ≤ elementos ≤ *limit*, onde o primeiro elemento está na posição *offset* de um conjunto maior.
Exceções:	**InvalidOffsetException**: índice de registro inicial, *offset*, inválido ou fora dos limites.

Tabela 18. Especificação da operação getUserMultibiometricTemplates (classificationExpression)

UserMultibiometricTemplate[]	getUserMultibiometricTemplates (classificationExpression)
Esta operação recupera os *templates* multibiométricos registrados no banco de dados cuja classificação esteja de acordo com a expressão de classificação (*classificationExpression*).	
Parâmetros:	**classificationExpression**: *string* representando a expressão com a classificação dos *templates* a serem recuperados.
Retorno:	Esta operação retorna um *array* de elementos do tipo *UserMultibiometricTemplate*, correspondente aos registros de *templates* multibiométricos armazenados no banco de dados cuja classificação esteja de acordo com a expressão de classificação.
Exceções:	**InvalidClassificationExprException**: expressão de classificação inválida.

Tabela 19. Especificação da operação getUserMultibiometricTemplates (offset, limit, classificationExpression)

UserMultibiometricTemplate[]	getUserMultibiometricTemplates (offset, limit, classificationExpression)
Seja n o valor de *limit* e m o valor de *offset*. Esta operação recupera o subconjunto formado pelos n *templates* multibiométricos registrados no banco de dados a partir do m-ésimo *template* cuja classificação esteja de acordo com a expressão de classificação (*classificationExpression*).	
Parâmetros:	**offset**: valor inteiro correspondente à posição do primeiro *template* em um conjunto de templates. **limit**: valor inteiro correspondente ao número de *templates* a serem recuperados a partir de *offset*. **classificationExpression**: *string* representando a expressão com a classificação dos *templates* a serem recuperados.
Retorno:	Esta operação retorna um *array* de elementos do tipo *UserMultibiometricTemplate*, correspondente a um conjunto com 1 ≤ elementos ≤ *limit*, onde o primeiro elemento está na posição *offset* de um conjunto maior, constituído por elementos cuja classificação esteja de acordo com a expressão de classificação (*classificationExpression*).
Exceções:	**InvalidClassificationExprException**: expressão de classificação inválida. **InvalidOffsetException**: índice de registro inicial, *offset*, inválido ou fora dos limites.

Tabela 20. Especificação da operação countMultibiometricTemplateRecords ()

int	countMultibiometricTemplateRecords ()
Esta operação recupera a quantidade de registros armazenados no banco de dados.	
Parâmetros:	
Retorno:	Esta operação retorna um valor inteiro, correspondente à quantidade de registros de *templates* multibiométricos armazenados no banco de dados.
Exceções:	

Tabela 21. Especificação da operação removeMultibiometricTemplate (userID)

void	removeMultibiometricTemplate (userID)
Esta operação remove do banco de dados o *template* multibiométrico do usuário que possui o identificador *userID* no banco de dados da aplicação.	
Parâmetros:	**userID**: valor inteiro correspondente ao identificador do usuário no banco de dados da aplicação.
Retorno:	Esta operação não retorna nenhum valor (*void*).
Exceções:	**InvalidUserIDException**: o identificador do usuário é inválido.

Tabela 22. Especificação da operação updateMultibiometricTemplate (userID, multibiometricTemplate)

void	updateMultibiometricTemplate (userID, multibiometricTemplate)
Esta operação atualiza, no banco de dados, o *template* multibiométrico do usuário cujo identificador no banco de dados da aplicação é *userID*.	
Parâmetros:	**userID**: valor inteiro correspondente ao identificador do usuário no banco de dados da aplicação. **multibiometricTemplate**: *array* de bytes com o novo *template* multibiométrico do usuário.
Retorno:	Esta operação não retorna nenhum valor (*void*).
Exceções:	**InvalidUserIDException**: o identificador do usuário é inválido.

Tabela 23. Especificação da operação updateMultibiometricTemplate (userID, multibiometricTemplate, classificationExpression)

void	updateMultibiometricTemplate (userID, multibiometricTemplate, classificationExpression)
Esta operação atualiza, no banco de dados, o *template* multibiométrico e a expressão de classificação (*classificationExpression*) do usuário cujo identificador no banco de dados da aplicação é *userID*.	
Parâmetros:	**userID**: valor inteiro correspondente ao identificador do usuário no banco de dados da aplicação. **multibiometricTemplate**: *array* de bytes com o novo *template* multibiométrico do usuário. **classificationExpression**: *string* representando expressão com a classificação do *template*.
Retorno:	Esta operação não retorna nenhum valor (*void*).
Exceções:	**InvalidClassificationExprException**: expressão de classificação inválida. **InvalidUserIDException**: o identificador do usuário é inválido.

Apêndice

B

Engine API

B.1. Interface IBioPassEngine

As operações que fazem parte da interface *IBioPassEngine* são apresentadas a seguir.

Tabela 24. Especificação da operação loadConfiguration (assemblyTypes)

void	loadConfiguration (assemblyTypes)
Esta operação carrega as classes de reconhecimento unibiométrico que implementam a interface *IBioPassEngineTrait* a serem utilizadas para o reconhecimento multibiométrico. Ela deve ser a primeira operação a ser chamada.	
Parâmetros:	**assemblyTypes**: *array* de *assemblies* e respectivas classes a serem utilizadas no reconhecimento multibiométrico (*AssemblyType*).
Retorno:	Esta operação não retorna nenhum valor (*void*).
Exceções:	**DuplicatedTraitTypeException**: Há mais de uma implementação para o mesmo tipo de traço no conjunto de *assemblyTypes* de entrada.

Tabela 25. Especificação da operação getMultibiometricTemplate (templates)

byte[]	getMultibiometricTemplate (templates)
Esta operação cria um *template* multibiométrico a partir dos *templates* unibiométricos de entrada.	
Parâmetros:	**templates**: *array* de *templates* unibiométricos (*UnibiometricTemplate*).
Retorno:	Esta operação retorna um *array* de bytes correspondente ao *template* multibiométrico.
Exceções:	**DuplicatedTraitTypeException**: Há mais de um *template* do mesmo traço no conjunto de *templates* de entrada. **InvalidTemplateFormatException**: O formato de algum dos *templates* unibiométricos de entrada é inválido.

	TraitTypeImplementationNotPresentException: O *assembly* de algum dos traços de entrada não foi encontrado.

Tabela 26. Especificação da operação getMultibiometricTemplate (unibiometricTraitsData, qualityThreshold)

byte[]	getMultibiometricTemplate (unibiometricTraitsData, qualityThreshold)
Esta operação cria um *template* multibiométrico a partir das características biométricas de entrada, desde que os sinais capturados possuam qualidade igual ou superior à definida para o limiar do parâmetro *qualityThreshold*.	
Parâmetros:	**unibiometricTraitsData**: *array* de traços e seus dados unibiométricos (*UnibiometricData*). **qualityThreshold**: tipo de limiar de qualidade aceitável para que o *template* multibiométrico seja criado (*AcceptableQualityThresholdType*).
Retorno:	Esta operação retorna um *array* de bytes correspondente ao *template* multibiométrico.
Exceções:	**DuplicatedTraitTypeException**: Há mais de um *template* do mesmo traço no conjunto de *templates* de entrada. **InsufficientTraitDataQualityException**: Um ou mais traços de entrada não possuem qualidade suficiente para serem analisados. **InvalidRawTraitDataException**: Um ou mais traços de entrada são inválidos. **TraitTypeImplementationNotPresentException**: O *assembly* de algum dos traços de entrada não foi encontrado.

Tabela 27. Especificação da operação getMultibiometricClass (rawTraits)

string	getMultibiometricClass (rawTraits)
Esta operação constrói uma expressão com uma classificação multibiométrica, de acordo com as características biométricas de entrada.	
Parâmetros:	**rawTraits**: *array* de traços unibiométricos (*Trait*).
Retorno:	Esta operação retorna uma *string* correspondente à expressão de classificação multibiométrica, de acordo com os traços biométricos entrados.
Exceções:	**DuplicatedTraitTypeException**: Há mais de um elemento do mesmo traço no conjunto de traços de entrada. **InvalidRawTraitDataException**: Um ou mais traços de entrada são inválidos. **TraitTypeImplementationNotPresentException**: O *assembly* de algum dos traços de entrada não foi encontrado.

Tabela 28. Especificação da operação getMultibiometricClass(templates)

string	getMultibiometricClass(templates)

Esta operação constrói uma expressão com uma classificação multibiométrica, extraída dos *templates* unibiométricos de entrada.	
Parâmetros:	**templates**: *array* de *templates* unibiométricos (*UnibiometricTemplate*).
Retorno:	Esta operação retorna uma *string* correspondente à expressão de classificação multibiométrica, extraída dos *templates* unibiométricos de entrada.
Exceções:	**DuplicatedTraitTypeException**: Há mais de um elemento do mesmo traço no conjunto de traços de entrada. **InvalidTemplateFormatException**: O formato de pelo menos um dos *templates* unibiométricos que fazem parte de algum dos *templates* de entrada é inválido. **TraitTypeImplementationNotPresentException**: O *assembly* de algum dos traços de entrada não foi encontrado.

Tabela 29. Especificação da operação getMultibiometricClass (multibiometricTemplate)

string	getMultibiometricClass (multibiometricTemplate)
Esta operação constrói uma expressão com uma classificação multibiométrica, extraída do *template* multibiométrico de entrada.	
Parâmetros:	**multibiometricTemplate**: *array* de bytes com um *template* multibiométrico.
Retorno:	Esta operação retorna uma *string* correspondente à expressão de classificação multibiométrica, extraída do *template* multibiométrico de entrada.
Exceções:	**InvalidMultibiometricTemplateFormatException**: O formato do *template* multibiométrico de entrada é inválido. **DuplicatedTraitTypeException**: Há mais de um elemento do mesmo traço no conjunto de traços de entrada. **InvalidTemplateFormatException**: O formato de pelo menos um dos *templates* unibiométricos que fazem parte do *template* multibiométrico é inválido. **TraitTypeImplementationNotPresentException**: O *assembly* de algum dos traços de entrada não foi encontrado.

Tabela 30. Especificação da operação match (multibiometricTemplate1, multibiometricTemplate2)

double	match (multibiometricTemplate1, multibiometricTemplate2)
Esta operação executa o *matching* entre dois *templates* multibiométricos e retorna o seu *score* de equivalência.	
Parâmetros:	**multibiometricTemplate1**: *array* de bytes com um *template* multibiométrico. **multibiometricTemplate2**: *array* de bytes com outro *template* multibiométrico.
Retorno:	Esta operação retorna um valor *double* com o *score* de *matching*.
Exceções:	**InvalidMultibiometricTemplateFormatException**: O formato de pelo menos um dos *templates* multibiométricos de entrada é inválido. **InvalidTemplateFormatException**: O formato de pelo menos um dos *templates* unibiométricos que fazem parte de algum dos *templates* multibiométricos é inválido.

	RequiredTraitsNotPresentException: Algum dos traços necessários não estão presentes em pelo menos um dos *templates* multibiométricos. **TraitTypeImplementationNotPresentException**: O *assembly* de algum dos traços de entrada não foi encontrado.

Tabela 31. Especificação da operação isValidMultibiometricTemplate (multibiometricTemplate)

boolean	isValidMultibiometricTemplate (multibiometricTemplate)
Esta operação verifica se um *template* multibiométrico é válido.	
Parâmetros:	**multibiometricTemplate**: *array* de bytes com um *template* multibiométrico.
Retorno:	Esta operação retorna um valor *boolean*, sendo *true* caso o formato do *template* multibiométrico de entrada seja válido e *false*, caso contrário.
Exceções:	

Tabela 32. Especificação da operação detectBiometrics (rawTrais)

boolean	detectBiometrics (rawTrais)
Esta operação detecta que os traços biométricos de entrada foram capturados, isto é, se eles estão diante dos sensores.	
Parâmetros:	**rawTraits**: *array* de traços unibiométricos (*Trait*).
Retorno:	Esta operação retorna um valor boolean, sendo true caso os traços estejam presentes e foram detectados e false, caso contrário.
Exceções:	**DuplicatedTraitTypeException**: Há mais de um elemento do mesmo traço no conjunto de traços de entrada. **InsufficientTraitDataQualityException**: Um ou mais traços de entrada não possuem qualidade suficiente para serem detectados. **InvalidRawTraitDataException**: Um ou mais traços de entrada são inválidos. **TraitTypeImplementationNotPresentException**: O *assembly* de algum dos traços de entrada não foi encontrado.

Tabela 33. Especificação da operação getMultibiometricAcceptanceThreshold ()

double	getMultibiometricAcceptanceThreshold ()
Esta operação recupera o limiar de aceitação para a operação de *matching* multibiométrico.	
Parâmetros:	
Retorno:	Esta operação retorna um valor *double* correspondente ao limiar de aceitação para a operação de *matching* multibiométrico.
Exceções:	

Tabela 34. Especificação da operação setMultibiometricAcceptanceThreshold (multibiometricAcceptanceThreshold)

void	setMultibiometricAcceptanceThreshold (multibiometricAcceptanceThreshold)
Esta operação estabelece o limiar de aceitação para a operação de *matching* multibiométrico.	
Parâmetros:	**multibiometricAcceptanceThreshold**: valor *double* correspondente ao limiar de aceitação para a operação de *matching* multibiométrico.
Retorno:	Esta operação não retorna nenhum valor (*void*).
Exceções:	**InvalidParameterValue**: O valor passado no parâmetro é inválido ou não está dentro da faixa de valores permitidos.

B.2. Interface IBioPassEngineTrait

As operações que fazem parte da interface *IBioPassEngineTrait* são apresentadas a seguir.

Tabela 35. Especificação da operação getTraitType ()

TraitType	getTraitType ()
Esta operação recupera o tipo de traço unibiométrico da implementação corrente desta interface.	
Parâmetros:	
Retorno:	Esta operação retorna o tipo de traço unibiométrico (enumerado em *TraitType*) da implementação corrente desta interface.
Exceções:	

Tabela 36. Especificação da operação getTemplate (rawTraitData, key, acceptableQuality)

byte[]	getTemplate (rawTraitData, key, acceptableQuality)
Esta operação cria um *template* unibiométrico.	
Parâmetros:	**rawTraitData**: *array* de *bytes* com o traço unibiométrico capturado. **key**: chave criptográfica. **acceptableQuality**: valor do tipo *float* correspondente à qualidade mínima que o traço capturado deve ter para que o *template* seja criado.
Retorno:	Esta operação retorna um *array* de bytes corresponde ao *template* unibiométrico.
Exceções:	**InsufficientTraitDataQualityException**: O traço de entrada não possui qualidade suficiente para ser analisado ou não possui qualidade aceitável. **InvalidRawTraitDataException**: O traço de entrada é inválido.

Tabela 37. Especificação da operação getClassFromRawData (rawTrait)

string	getClassFromRawData (rawTrait)
Esta operação constrói uma expressão de classificação biométrica, de acordo com a característica biométrica de entrada.	
Parâmetros:	**rawTrait**: *array* de bytes com o traço capturado.
Retorno:	Esta operação retorna uma *string* correspondente à expressão de classificação biométrica, de acordo com o traço biométrico de entrada.
Exceções:	**InvalidRawTraitDataException**: O traço de entrada é inválido. **NotImplementedException**: A operação não foi implementada para o tipo de característica biométrica em questão.

Tabela 38. Especificação da operação getClassFromTemplate (template)

string	getClassFromTemplate (template)
Esta operação constrói uma expressão de classificação biométrica, de acordo com o *template* biométrico de entrada.	
Parâmetros:	**template**: *array* bytes com o *template* unibiométrico.
Retorno:	Esta operação retorna uma *string* correspondente à expressão de classificação biométrica, de acordo com o *template* biométrico de entrada.
Exceções:	**InvalidTemplateFormatException**: O formato do *template* de entrada é inválido. **NotImplementedException**: A operação não foi implementada para o tipo de característica biométrica em questão.

Tabela 39. Especificação da operação match (template1, template2)

double	match (template1, template2)
Esta operação executa o *matching* entre dois *templates* unibiométricos e retorna o seu *score* de equivalência.	
Parâmetros:	**template1**: *array* de bytes com um *template* unibiométrico. **template2**: *array* de bytes com outro *template* unibiométrico.
Retorno:	Esta operação retorna um valor *double* com o *score* de *matching*.
Exceções:	**InvalidTemplateFormatException**: O formato de pelo menos um dos *templates* de entrada é inválido.

Tabela 40. Especificação da operação getQuality (rawTrait)

float	getQuality (rawTrait)
Esta operação calcula a qualidade de um traço biométrico.	

Parâmetros:	**rawTrait**: *array* de bytes com o traço capturado.
Retorno:	Esta operação retorna um valor *float* com a qualidade do traço de entrada.
Exceções:	**InvalidRawTraitDataException**: O traço de entrada é inválido.

Tabela 41. Especificação da operação isValidTemplate (template)

boolean	isValidTemplate (template)
Esta operação verifica se um *template* unibiométrico é válido.	
Parâmetros:	**template**: *array* de bytes com um *template* unibiométrico.
Retorno:	Esta operação retorna um valor *boolean*, sendo *true* caso o formato do *template* de entrada seja válido e *false*, caso contrário.
Exceções:	

Tabela 42. Especificação da operação getMatchingWeight ()

int	getMatchingWeight ()
Esta operação recupera o peso que o tipo de característica biométrica em questão exerce na fusão.	
Parâmetros:	
Retorno:	Esta operação retorna um valor *inteiro* correspondente ao peso que o tipo de característica biométrica em questão exerce na fusão.
Exceções:	

Tabela 43. Especificação da operação setMatchingWeight (weight)

void	setMatchingWeight (weight)
Esta operação estabelece o peso que o tipo de característica biométrica em questão exerce na fusão.	
Parâmetros:	**weight**: valor *inteiro* correspondente ao peso que o tipo de característica biométrica em questão deve exercer na fusão.
Retorno:	Esta operação não retorna nenhum valor (*void*).
Exceções:	**InvalidParameterValue**: O valor passado no parâmetro é inválido ou não está dentro da faixa de valores permitidos.

Tabela 44. Especificação da operação getAcceptanceThreshold ()

double	getAcceptanceThreshold ()

Esta operação recupera o limiar de aceitação direta para a operação de *matching*.	
Parâmetros:	
Retorno:	Esta operação retorna um valor *double* correspondente ao limiar de aceitação direta para a operação de *matching*.
Exceções:	

Tabela 45. Especificação da operação setAcceptanceThreshold (acceptanceThreshold)

void	setAcceptanceThreshold (acceptanceThreshold)
Esta operação estabelece o limiar de aceitação direta para a operação de *matching*.	
Parâmetros:	**acceptanceThreshold**: valor *double* correspondente ao limiar de aceitação direta para a operação de *matching*.
Retorno:	Esta operação não retorna nenhum valor (*void*).
Exceções:	**InvalidParameterValue**: O valor passado no parâmetro é inválido ou não está dentro da faixa de valores permitidos.

Tabela 46. Especificação da operação getRejectionThreshold ()

double	getRejectionThreshold ()
Esta operação recupera o limiar de rejeição direta para a operação de *matching*.	
Parâmetros:	
Retorno:	Esta operação retorna um valor *double* correspondente ao limiar de rejeição direta para a operação de *matching*.
Exceções:	

Tabela 47. Especificação da operação setRejectionThreshold (rejectionThreshold)

void	setRejectionThreshold (rejectionThreshold)
Esta operação estabelece o limiar de rejeição direta para a operação de *matching*.	
Parâmetros:	**rejectionThreshold**: valor *double* correspondente ao limiar de rejeição direta para a operação de *matching*.
Retorno:	Esta operação não retorna nenhum valor (*void*).
Exceções:	**InvalidParameterValue**: O valor passado no parâmetro é inválido ou não está dentro da faixa de valores permitidos.

Tabela 48. Especificação da operação getMaxScore ()

double	getMaxScore ()
Esta operação recupera um valor de *score* considerado o valor máximo possível obtido na operação de *matching*.	
Parâmetros:	
Retorno:	Esta operação retorna um valor *double* correspondente ao *score* considerado o valor máximo possível obtido na operação de *matching*.
Exceções:	

Tabela 49. Especificação da operação setMaxScore (maxScore)

void	setMaxScore (maxScore)
Esta operação estabelece o valor de *score* a ser considerado o valor máximo possível obtido na operação de *matching*.	
Parâmetros:	**maxScore**: valor *double* correspondente ao *score* a ser considerado o valor máximo possível obtido na operação de *matching*.
Retorno:	Esta operação não retorna nenhum valor (*void*).
Exceções:	**InvalidParameterValue**: O valor passado no parâmetro é inválido ou não está dentro da faixa de valores permitidos.

Tabela 50. Especificação da operação getAcceptableQualityThreshold (qualityThresholdType)

float	getAcceptableQualityThreshold (qualityThresholdType)
Esta operação recupera o limiar de qualidade aceitável para o tipo de limiar de qualidade passado como parâmetro.	
Parâmetros:	**qualityThresholdType**: tipo de limiar de qualidade (enumerado em *AcceptableQualityThresholdType*).
Retorno:	Esta operação retorna um valor *float* correspondente ao limiar de qualidade aceitável para o tipo de limiar de qualidade passado como parâmetro.
Exceções:	

Tabela 51. Especificação da operação setAcceptableQualityThreshold (acceptableQualityThreshold, qualityThresholdType)

void	setAcceptableQualityThreshold (acceptableQualityThreshold, qualityThre-

	sholdType)
Esta operação estabelece o limiar de qualidade aceitável para o tipo de limiar de qualidade passado como parâmetro.	
Parâmetros:	**acceptableQualityThreshold**: valor *float* correspondente ao limiar de qualidade aceitável para o tipo de limiar de qualidade passado como parâmetro. **qualityThresholdType**: tipo de limiar de qualidade (enumerado em *AcceptableQualityThresholdType*).
Retorno:	Esta operação não retorna nenhum valor (*void*).
Exceções:	**InvalidParameterValue**: O valor *acceptableQualityThreshold* passado no parâmetro é inválido ou não está dentro da faixa de valores permitidos.

B.3. Interface IBioPassEngineTraitDetector

As operações que fazem parte da interface *IBioPassEngineTraitDetector* são mostradas a seguir.

Tabela 52. Especificação da operação detectTrait (rawTrait)

boolean	detectTrait (rawTrait)
Esta operação detecta que o traço biométrico foi capturado, isto é, se há um traço biométrico diante do sensor.	
Parâmetros:	**rawTrait**: *array* de *bytes* com o traço unibiométrico capturado.
Retorno:	Esta operação retorna um valor *boolean*, sendo *true* caso o traço esteja presente e foi detectado e *false*, caso contrário.
Exceções:	**InsufficientTraitDataQualityException**: O traço de entrada não possui qualidade suficiente para ser detectado. **InvalidRawTraitDataException**: O traço de entrada é inválido.

Tabela 53. Especificação da operação getDetectionTimeInterval ()

int	getDetectionTimeInterval ()
Esta operação recupera o valor em ms do intervalo de tempo entre o qual as chamadas à operação de detecção da característica biométrica devem ocorrer.	
Parâmetros:	
Retorno:	Esta operação retorna um valor *int* correspondente ao intervalo de tempo a ser usado entre a realização das chamadas à operação de detecção da característica biométrica.
Exceções:	

Tabela 54. Especificação da operação setDetectionTimeInterval (detectionTimeInterval)

void	setDetectionTimeInterval (detectionTimeInterval)
Esta operação estabelece o valor em ms do intervalo de tempo entre o qual as chamadas à operação de detecção da característica biométrica devem ocorrer.	
Parâmetros:	**detectionTimeInterval**: valor *int* correspondente ao intervalo de tempo em ms entre o qual as chamadas à operação de detecção da característica biométrica devem ocorrer.
Retorno:	Esta operação não retorna nenhum valor (*void*).
Exceções:	**InvalidParameterValue**: O valor *detectionTimeInterval* passado no parâmetro é inválido ou não está dentro da faixa de valores permitidos.

Tabela 55. Especificação da operação getDetectionTimeOut ()

int	getDetectionTimeOut ()
Esta operação recupera o valor em ms do tempo limite para que as chamadas à operação de detecção sejam cessadas, considerando-se que o traço biométrico não esteja sendo detectado.	
Parâmetros:	
Retorno:	Esta operação retorna um valor *int* correspondente ao tempo limite em ms para que as chamadas à operação de detecção sejam cessadas.
Exceções:	

Tabela 56. Especificação da operação setDetectionTimeOut (detectionTimeOut)

void	setDetectionTimeOut (detectionTimeOut)
Esta operação estabelece o valor em ms do tempo limite para que as chamadas à operação de detecção sejam cessadas, considerando-se que o traço biométrico não esteja sendo detectado.	
Parâmetros:	**detectionTimeOut**: valor *int* correspondente ao tempo limite em ms para que as chamadas à operação de detecção sejam cessadas.
Retorno:	Esta operação não retorna nenhum valor (*void*).
Exceções:	**InvalidParameterValue**: O valor *detectionTimeOut* passado no parâmetro é inválido ou não está dentro da faixa de valores permitidos.

Referências

Abate, A. F., Marsico, M., Riccio, D., & Tortora, G. (Setembro de 2010). MUBAI: multiagent biometrics for ambient intelligence. J Ambient Intell Human Comput .

Abate, A. F., Nappi, M., Riccio, D., & Marsico, M. (2007). Face, Ear and Fingerprint: Designing Multibiometric Architectures. 14th IEEE International Conference on Image Analysis and Processing (ICIAP).

American National Standards Institute. (2007). NIST Special Public Report: 500–271. Data Format for the Interchange of Fingerprint Facial, & Other Biometric Information (revision of ANSI/NIST–ITL 1–2000) .

Antonelli, A., Raffaele, C., Maio, D., & Maltoni, D. (2006). Fake finger detection by skin distortion analysis. IEEE Transactions on Information Forensics and Security , 1 (3), pp. 360-373.

Anwar, F., Rahman, A., & Azad, S. (2009). Multibiometric Systems Based Verification Technique. European Journal of Scientific Research , 34 (2), pp. 260-270.

Ashbaugh, D. (1999). Quantitative–Qualitative Friction Ridge Analysis: An Introduction to Basic and Advanced Ridgeology. Boca Raton, FL: CRC Press.

BioAPI. (s.d.). Acesso em Agosto de 2010, disponível em http://www.bioapi.org/

BioID. (2010). Acesso em Novembro de 2010, disponível em http://www.bioid.com

Bleichenbacher, D., & Nguyen, P. Q. (2000). Noisy Polynomial Interpolation and Noisy Chinese Remaindering. Proceedings of Nineteenth IACR Eurocrypt, (pp. 53-69). Bruges, Belgium.

Brosso, M. (2006). Autenticação Contínua de Usuários em Redes de Computadores (Tese de Doutorado). Escola Politécnica da Universidade de São Paulo, Departamento de Engenharia da Computação e Sistemas Digitais, São Paulo.

Buyya, R. (1999). High Performance Cluster Computing: Architectures and Systems (1ª ed., Vol. 1). Prentice Hall.

Campbell, J. P. (1997). Speaker Recognition: a Tutorial. Proceedings of the IEEE , 85 (9), pp. 1437-1462.

Damousis, I. G., Tzovaras, D., & Bekiaris, E. (2008). Unobtrusive Multimodal Biometric Authentication: The HUMABIO Project Concept. EURASIP Journal on Advances in Signal Processing , 2008.

Daugman, J. (2010). Combining Multiple Biometrics. Fonte: http://www.cl.cam.ac.uk/users/jgd1000/combine/combine.html

Daugman, J. (2004). How Iris Recognition Works? IEEE Transactions on Circuits and Systems for Video Technology , 14, pp. 21-30.

Daugman, J. (1999). Recognizing persons by their iris patterns. In: A. Jain, R. Bolle, & S. Pankanti, Biometrics: Personal Identification in a Networked Society. Norwell, MA: Kluwer.

Fancourt, C. L., Bogoni, L., Hanna, K. J., Quo, Y., Wildes, R. P., Takahashi, N., et al. (2005). Iris Recognition at a Distance. Fifth International Conference on Audio- and Video-based Biometric Person Authentication (AVBPA) (pp. 1-13). USA: Rye Brook.

Griaule. (2010). Acesso em 2010, disponível em http://www.griaule.com/

GTAV. (s.d.). Acesso em 2010, disponível em GTAV Face Database: http://gps-tsc.upc.es/GTAV/ResearchAreas/GTAVDatabase.htm

Harrison, W. R. (1981). Suspect Documents, their Scientific Examination. Nelson-Hall Publishers.

Henry, E. (1900). Classification and Uses of Finger Prints. London: Routledge.

Hong, L., Jain, A. K., & Pankanti, S. (1999). Can Multibiometrics Improve Performance? Proceedings of IEEE Workshop on Automatic Identification Advanced Technologies (AutoID), (pp. 59-64). New Jersey .

Hsu, R. L. (2002). Face Detection and Modeling for Recognition (PhD thesis). Michigan State University, Department of Computer Science and Engineering.

HUMABIO. (s.d.). Acesso em Novembro de 2010, disponível em Human Monitoring and Authentication using Biodynamic Indicators and Behavioural Analysis: http://www.humabio-eu.org

IBM. (2001). IBM Annual Report.

Jain, A., Nandakumar, K., & Nagar, A. (2008). Biometric Template Security. EURASIP Journal on Advances in Signal Processing .

Juels, A., & Sudan, M. (2002). A Fuzzy Vault Scheme., (p. 408). Lausanne, Switzerland.

Kuncheva, L. (2004). Combining Pattern Classifiers - Methods and Algorithms. Wiley.

Lee, H., & Gaensslen, R. (2001). Advances in Fingerprint Technology (2 ed.). New York: Elsevier.

Lee, J., Kim, D., Kwak, K.-C., Kim, H.-J., & Yoon, H.-S. (2007). Integrating Evidences of Independently Developed Face and Speaker Recognition Systems by Using Discrete Probability Density Function. 16th IEEE International Conference on Robot & Human Interactive Communication.

Li, S. Z., & Jain, A. K. (2005). Handbook of Face Recognition. Springer-Verlag.

Liu, X., & Chen, T. (2003). Geometry-assisted Statistical Modeling for Face Mosaicing. IEEE International Conference on Image Processing (ICIP), 2, pp. 883-886. Barcelona, Spain.

Maltoni, D., Maio, D., Jain, A. K., & Prabhakar, S. (2009). Handbook of fingerprint recognition (2 ed.). London, England: Springer Science.

McCabe, R. (2004). Fingerprint interoperability standards. In: N. Ratha, & R. Bolle, Automatic Fingerprint Recognition Systems (pp. 433–451). New York: Springer.

McIlroy, M. D. (1968). Mass produced software components. In: P. Naur, & B. Randell, Conference of the NATO Science Committee (pp. 138-150).

Microsoft Visio 2007. (s.d.). Acesso em Agosto de 2010, disponível em http://office.microsoft.com/pt-br/visio/

Moenssens, A. (1971). Fingerprint Techniques. London: Chilton Book Company.

Moon, Y., Chen, J., Chan, K., So, K., & Woo, K. (2005). Wavelet-based fingerprint liveness detection. Electronic Letters , 41 (20).

Nalwa, V. S. (1997). Automatic On-Line Signature Verification. Proceedings of the IEEE , 85 (2), pp. 215-239.

Nandakumar, K. (2008). Multibiometric Systems: Fusion Strategies and Template Security (Ph.D. Dissertation). Michigan State University, Department of Computer Science and Engineering.

Neurotechnology. (2010). Acesso em Agosto de 2010, disponível em http://www.neurotech.com/

NIST. (Novembro de 2001). Announcing the Advanced Encryption Standard. Federal Information Processing Standards Publication.

Object Management Group (OMG). (2004). Common Request Broker Architecture: Core Specification, Version 3.0.3.

Oliveira, A. E., Motta, G. H., & Batista, L. V. (2010). A multibiometric access control architecture for continuous authentication. IEEE Intelligence and Security Informatics (ISI). Vancouver, Canada.

OpenID. (s.d.). Acesso em Novembro de 2010, disponível em http://openid.net/

Parthasaradhi, S., Derakhshani, R., Hornak, L., & Schuckers, S. (2005). Time series detection of perspiration as a liveness test in fingerprint scanners. IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews , 35, pp. 335-343.

Pressman, R. S. (2005). Software Engineering: A Practitioner's Approach. New York: McGraw-Hill.

Rauber, T., & Rünger, G. (2010). Parallel Programming For Multicore and Cluster Systems. Springer-Verlag Berlin Heidelberg.

Roberts, C. (2007). Biometric attack vectors and defenses. Computers and Security Journal , 26, pp. 14-25.

Ross, A. A., Nandakumar, K., & Jain, A. (2006). Handbook of multibiometrics. New York: Springer Science.

Ross, A., & Jain, A. (2004). Biometric Sensor Interoperability: A Case Study in Fingerprints. LNCS 3087, pp. 134-145. Prague, Czech Republic: Springer.

RSA Laboratories. (1999). PKCS #5: Password-Based Criptography Standard, Version 2.0. RSA Laboratories.

Sim, T., Zhang, S., Janakiraman, R., & Kumar, S. (Abril de 2007). Continuous Verification Using Multimodal Biometrics. IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE , 29 (4).

Sommerville, I. (2004). Software Engineering (7th Edition ed.). Addison Wesley.

Stal, M. (2002). Web Services: Beyond Component Bases Computing. Communications of the ACM , 45 (110), pp. 71-76.

Sun Microsystems Inc. (2006). EJB Core Contracts and Requirements, JSR 220: Enterprise Java Beans, Version 3.0.

Sun Microsystems. (s.d.). RMI specification. Acesso em 2010, disponível em http://download.oracle.com/docs/cd/E17476_01/javase/1.4.2/docs/guide/rmi/spec/rmiTOC

Suprema Inc. (2010). Acesso em 2010, disponível em http://www.supremainc.com/

Tanenbaum, A. S., & Steen, M. (2002). Distributed Systems: Principles and Paradigms. Prentice Hall.

The Open Group. (1997). CDE 1.1: Remote Procedure Call.

Thian, N. (2001). Biometric Authentication System (Master Thesis). USM, Penang, Malásia.

U.are.U. (s.d.). Acesso em 2009, disponível em U.are.U Fingerprint Sampla Database: http://www.neurotechnology.com/download/UareU_sample_DB.zip

Varchol, P., Levicky, D., & Juhar, J. (2008). Multimodal biometric authentication using speech and hand geometry fusion. 15th International Conference on Systems, Signals and Image Processing (IWSSIP).

Wasserman, H. P. (1974). Ethnic Pigmentation. New York, USA: Elsevier.

Wayman, J., Jain, A., Maltoni, D., & Maio, D. (2005). Biometric Systems: technology, design and performance evaluation. London: Springer Science.

Xu, L., Krzyzak, A., & Suen, C. Y. (1992). Methods for Combining Multiple Classifiers and their Applications to Handwriting Recognition. IEEE Transactions on Systems, Man, and Cybernetics , pp. 418-435.

Zunkel, R. (1999). Hand Geometry Based Authentication. In: A. K. Jain, R. Bolle, & S. Pankanti, Biometrics: Personal Identification in Networked Society (pp. 87-102). London, UK: Kluwer Academic Publishers.