CENTRO UNIVERSITÁRIO DE BRASÍLIA -UniCEUB
CURSO DE ENGENHARIA DE COMPUTAÇÃO

IGOR CARVALHO OLIVEIRA

SISTEMA WEB DE REGISTRO DE PRESENÇA VIA LEITURA BIOMÉTRICA DE IMPRESSÃO DIGITAL PARA INSTITUIÇÕES DE ENSINO

Orientador: Prof. Msc. Francisco Javier De Obaldia Diaz

Brasília
Novembro, 2011

IGOR CARVALHO OLIVEIRA

SISTEMA WEB DE REGISTRO DE PRESENÇA VIA LEITURA BIOMÉTRICA DE IMPRESSÃO DIGITAL PARA INSTITUIÇÕES DE ENSINO

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.

Orientador: Prof. Msc. Francisco Javier

Brasília

Novembro, 2011

IGOR CARVALHO OLIVEIRA

SISTEMA WEB DE REGISTRO DE PRESENÇA VIA LEITURA BIOMÉTRICA DE IMPRESSÃO DIGITAL PARA INSTITUIÇÕES DE ENSINO

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.

Orientador: Prof. Msc. Francisco Javier De Obaldia Diaz

Este Trabalho foi julgado adequado para a obtenção do Título de Engenheiro de Computação, e aprovado em sua forma final pela Faculdade de Tecnologia e Ciências Sociais Aplicadas - FATECS.

_____
Prof. Abiezer Amarilia Fernandez
Coordenador do Curso

Banca Examinadora:

_____
Prof. Francisco Javier de Obaldia Diaz, Msc.
Orientador

_____
Prof. Miguel Arcanjo B. Goes Telles Juni, Dr.
Centro Universitário de Brasília - UniCEUB

_____
Prof. Marco Antônio de Oliveira Araújo, Msc.
Centro Universitário de Brasília - UniCEUB

_____
Prof. João Marcos Souza Costa, Msc.
Centro Universitário de Brasília - UniCEUB

Dedico este trabalho a toda a minha família, em especial a minha avó Maria José, que está hospitalizada, a qual almejo vislumbrar com sua saúde restabelecida muito em breve.

AGRADECIMENTOS

Gostaria de agradecer primeiramente a Deus pelo dom da vida e pela capacidade de aprendizagem.

Gostaria de agradecer também aos meus amigos da empresa Via Appia Informática, onde trabalhei por aproximadamente um ano e meio, aprendendo diversos conceitos e vivenciando várias experiências que me foram indispensáveis para o desenvolvimento deste trabalho. Por todo o carinho, suporte e confiança, agradeço a Geraldo Iraci do Couto, Junior Couto, Leonardo Couto, Cláudia Couto, Wesley Fernandes, Vitor Barbosa, Felipe Esteves, Alberto Yamaguchi e a todos da Via Appia Informática, minha escola de programação.

Ainda, gostaria de agradecer aos meus amigos e atuais companheiros de trabalho da empresa Logus Tecnologia, onde continuo a aprender diariamente e a crescer como programador e como ser humano. Meus humildes agradecimentos a Walace Zloccowick Maia pela oportunidade a mim concedida de ocupar um cargo tão importante a despeito de minha idade e ao mestre, futuro doutor, Alessandro Leite, que sempre se dispôs a tirar minhas dúvidas, servindo de fonte de inspiração profissional.

Agradeço também a minha família e aos meus amigos, em especial, Lucas Zloccowick e Herbert Félix. Por fim, agradeço a minha namorada, amada metade e futura esposa, Letícia Régia Delmondes Vitorio, por todo o apoio, companheirismo e inspiração.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AJP: *Apache Jserv Protocol*;
API: *Application Package Interface*;
CDC: *Connected Device Configuration*;
CLDC: *Connected Limited Device Configuration*;
COM: *Component Object Model*;
CRUD: *Create, Recover, Update, Delete.*
CSS*: Cascading Style Sheets;*
DAO*: Data Access Object;*
DHTML*: Dynamic Hyper Text Markup Language;*
DOM*: Document Object Model;*
DPI: *Dots per Inch*
EJB*: Enterprise Java Bean;*
IDE*: Interface Development Environment;*
HTML: *Hyper Text Markup Language*;
HTTP: *Hyper Text Transfer Protocol*;
MVC: *Model, View, Controller*;
J2EE: *Java 2 Enterprise Edition*;
J2ME: *Java 2 Mobile Edition*;
J2SE: *Java 2 Standard Edition*;
JDBC: *Java Database Connectivity*;
JDK: *Java Development Kit*;
JIT: *Just-in-time*;
JRE: *Java Runtime Environment*;
JSP: *Java Server Page*;
JVM: *Java Virtual Machine*;
SDK: *Standard Development Kit*;
SGBD: Sistema Gerenciador de Banco de Dados;
SQL: *Structured Query Language*;
USB: *Universal Serial Bus*;
VB: *Visual Basic*;
W3C: *World Wide Web Consortium*;
XML: *eXtensible Markup Language*;
ZK: *Zkoss*;

RESUMO

Este trabalho tem o objetivo de proporcionar uma solução alternativa e inovadora para a questão do registro de presença de alunos em instituições de ensino. Tal registro de presença baseia-se na leitura biométrica das respectivas impressões digitais dos alunos através de um *software*, sendo executado remotamente e de maneira centralizada em um servidor da instituição. Após recebidas as informações da presença ou ausência dos alunos, tais registros serão salvos no banco de dados para posteriores operações de contabilização, disponibilização e consulta. O resultado deste trabalho consiste na efetiva implementação do sistema de presença eletrônica, possibilitando a realização do registro de controle de assiduidade dos alunos por parte das entidades responsáveis. Desta forma, observa-se que, através de conceitos de distribuição de sistemas, protocolos de rede, desenvolvimento de *software*, transmissão de sinais e cálculos estatísticos, a implementação e posterior utilização de tal solução nas operações acima citadas é totalmente viável.

Palavras Chave: Desenvolvimento de *software*. Leitura biométrica. Sistemas distribuídos. Protocolos de rede. Chamada por impressão digital.

ABSTRACT

The goal of this work is to provide an alternative and innovative solution for the question of student's presence recording in educational institutions. The presence recording is based on biometrical reading of the respective student's digital fingerprints through a *software* being executed remotely and centrallizedly in a computer server of the educational institution. After receiving the presence or ausence data, these records will be stored in database for posterior accounting, availability and consultation operations. The result of this work consists in an effective implementation of the solution, enabling, through a concrete and evident way, the realization of assiduity recording control of students by the responsable entities. So, it's induced that, through concetps of system distribution, network protocols, signals transmition and statistic calculation, the implementation and aftermost utilization of the solution for the context enunciated above is totally viable.

Keywords: *Software development*. *Biometrical reading*. *Distributed systems*. *Network protocols*. *Presence recording through fingerprint reading*.

CAPÍTULO 1 - INTRODUÇÃO

1.1 - Apresentação do Problema

A automatização de processos rotineiros realizados por parte dos seres humanos em qualquer contexto sempre foi um dos principais fatores motivacionais para os profissionais da área de tecnologia. Diversas operações desse tipo já foram automatizadas e inseridas dentro de um processo mecanizado, moderno e sólido. Dentre tais soluções, pode-se evidenciar a balança com módulo de emissão de etiquetas adesivas contendo valores relativos à pesagem e preço, utilizada em restaurantes, dispensando ao funcionário o trabalho de anotar o subtotal da conta do cliente. Também pode-se utilizar de exemplo as escovas de dentes motorizadas que, de certa forma, dispensam movimentos exagerados no ato da higienização bucal. Além disso, os automóveis mais modernos já estão vindo equipados com uma grande quantidade de sensores, como os de chuva, estacionamento e luminosidade. Por fim, pode-se enumerar também alguns sistemas de aquecimento doméstico que, ao serem configurados para trabalharem sempre com uma temperatura ambiente agradável, automaticamente se condicionam a esfriar ou a esquentar de acordo com a temperatura de entrada captada por seus conjuntos de sensores. Desta forma, pode-se perceber que para a sociedade atual, quaisquer soluções que dispensem a utilização sem sentido de força humana em qualquer atividade são tomadas como válidas e com ramo de aplicação. Esta tem se tornado uma tendência cada vez mais presente no cotidiano das pessoas e cada vez mais as pessoas têm pensado de maneira a vislumbrar um futuro automatizado na medida do possível.

Transportando-se tal premissa, aliada aos exemplos citados, para o contexto das instituições de ensino, percebe-se uma grande quantidade de atividades que poderiam ser automatizadas através de soluções tecnológicas e inovadoras, em especial o processo de realização de controle de assiduidade, a denominada chamada. Se pararmos para pensar em quantas vezes respondemos à chamada, sinalizando nossa presença nas inúmeras aulas e atividades as quais comparecemos durante toda a nossa vida acadêmica, independente do nível, percebemos que realizamos tal processo quase que de maneira condicionada, sem notar que algo poderia ser feito para que tal registro ocorresse de maneira mais rápida e consistente. Velocidade e consistência que, no primeiro momento, acabam sendo desprezadas, devido ao relativo baixo índice de falha e relativo pouco tempo gasto em tal operação, porém, que ao se repetir por várias gerações, inclusive nas gerações

tecnologicamente avançadas e modernas, acaba comprometendo o bom andamento das atividades dos estudantes, dos professores e, por consequência, das instituições de ensino. A quebra de paradigma deve estar presente no pensamento da sociedade moderna em todos os contextos, sendo ele o mais trivial e dispensável ou sendo ele o mais complexo e necessário. Desta forma, analisando a situação sob a óptica da sociedade em que vivemos, será que responder à chamada de maneira oral atesta de fato o comparecimento das pessoas em qualquer atividade, em especial em atividades acadêmicas, e é, mesmo providos de uma infinidade de recursos tecnológicos, a melhor solução a ser adotada?

1.2 - Objetivos do Trabalho

O objetivo geral deste trabalho consiste na elaboração de uma solução segura e coerente para o registro e controle de assiduidade de alunos em instituições de ensino. Tal solução deverá ser completa, de fácil utilização e implantação, e com alto grau de compatibilidade entre os diversos ambientes e contextos em que poderá será aplicada.

Os objetivos específicos consistem na implementação de um sistema capaz de realizar, de maneira automática e consistente, o processo de registro de presença através de leitura de impressão digital, na simulação da execução do mesmo em ambiente próximo ao de uma instituição de ensino e na validação de tal simulação. Para efeitos de evidenciação e constatação dos objetivos enumerados, diversas baterias de testes serão realizadas, cada qual atestando uma funcionalidade específica dentro do contexto de verificação de leituras de impressões digitais.

1.3 - Justificativa e Importância do Trabalho

A incoerência entre os registros de assiduidade existentes nos bancos de dados das instituições de ensino e o que de fato se atestou na realidade é um problema que afeta diretamente o aluno, mas que pode afetar também a própria instituição. No que tange ao aluno, o mesmo pode ser prejudicado por uma falha no canal de comunicação com o professor, ocasionando em um atestado incoerente de comparecimento à atividade em questão. Já no que tange à instituição de ensino, um atestado incoerente de comparecimento a uma determinada atividade pode comprometer a imagem da mesma, pois o aluno pode estar em qualquer outro lugar, se envolvendo em contextos que podem gerar indisposições, como o de um crime, por exemplo. Nessas condições, o aluno possui o registro de presença da instituição, criando uma margem de dúvida sobre onde de fato o mesmo esteve.

Desta forma, mesmo que em um primeiro momento as consequências para incoerência entre fatos sejam relativamente mínimas e desconsideráveis do ponto de vista principalmente da instituição de ensino, um sistema que ateste de fato o comparecimento ou não-comparecimento dos alunos em uma atividade se faz necessário, não só por eficiência, mas também por questões de segurança.

1.4 - Escopo do Trabalho

O trabalho possui o escopo de constituir um sistema que valide a presença ou ausência de alunos em atividades de instituições de ensino através de leituras biométricas de suas respectivas impressões digitais. Desta forma, o produto final objetivado pelo trabalho não abrange um sistema de manutenção de registros de alunos e professores de instituições de ensino para operações referentes as áreas de: biblioteconomia, secretariado, ouvidoria e ensino à distância. Além disso, a integração do sistema de registro de presença via leitura biométrica de impressão digital com os outros sistemas internos e de cunho institucional não faz parte do escopo deste trabalho. A funcionalidade de controle de acesso a áreas da instituição de ensino através de leitura biométrica de impressão digital também não faz parte do escopo deste trabalho. Também não faz parte do escopo deste trabalho a verificação de entrada e saída de pessoas nas dependências da instituição através de catracas com leitores de impressão digital embutidos. Por fim, não faz parte do escopo deste projeto a existência de um módulo gerador e impressor de relatórios de frequências e menções através de filtros de curso, semestre, ano, matrícula, professor ou qualquer outro tipo de filtro desejado. Vale ressaltar que elementos fora do escopo podem ser enumerados como sugestões para projetos futuros ou implementações futuras.

1.5 - Resultados Esperados

Os resultados deverão ser compostos de efetiva execução do sistema em ambiente de simulação devidamente preparado para demonstração, juntamente com o registro de presença via leitura biométrica de impressão digital correspondente com o esperado de acordo com as digitais cadastradas. Tal evidenciação de presença deve ser registrada em banco de dados seguro em tempo de execução, colaborando, desta forma, com a disponibilidade das informações para futuras consultas, publicações e lançamentos.

1.6 - Estrutura do Trabalho

No Capítulo 2, o problema será apresentado. Diversas consequências negativas resultantes do problema serão abordadas de maneira profunda. Além disso, alguns casos reais que geraram consequências imensuráveis devido ao acontecimento do problema serão relatados e explicados. No Capítulo 3, serão enunciadas as bases metodológicas para a resolução do problema, descrevendo-se a relação entre tais bases e as disciplinas cursadas e o porquê da escolha dos métodos. Tais métodos serão explicados detalhadamente com o intuito de proporcionar uma visão simples, porém não pobre e carente de definições, ao leitor. No Capítulo 4, o modelo de resolução do problema será proposto e explicado. Tal modelo conterá referências diretas e claras aos métodos utilizados e apresentados no Capítulo 3, de modo que a estruturação do projeto mantenha a premissa da coerência e da veracidade. No Capítulo 5, serão evidenciados os resultados da aplicação do modelo proposto. Os resultados serão medidos, qualificados e quantificados para o ambiente escolhido para a realização das simulações e posterior demonstração. A definição de tal ambiente irá compor o escopo do Capítulo 4. No Capítulo 6, será realizada a conclusão do trabalho a partir dos resultados obtidos. Além disso, a conclusão também atestará a validade ou não do projeto para um ambiente real.

CAPÍTULO 2 - APRESENTAÇÃO DO PROBLEMA

O processo de realização de controle de assiduidade submetido aos alunos, por parte dos professores, em instituições de ensino, sempre seguiu o mesmo padrão e formalismo. Desde que tal processo foi concebido, sua estrutura e regras básicas nunca foram alteradas. Através de uma evocação oral, normalmente constituída pelo nome do respectivo aluno, e posteriormente, de uma sinalização em resposta por parte de tal aluno, todo o registro de presença dos estudantes vem sendo feito ao longo dos anos. A figura 2.1 mostra este processo.

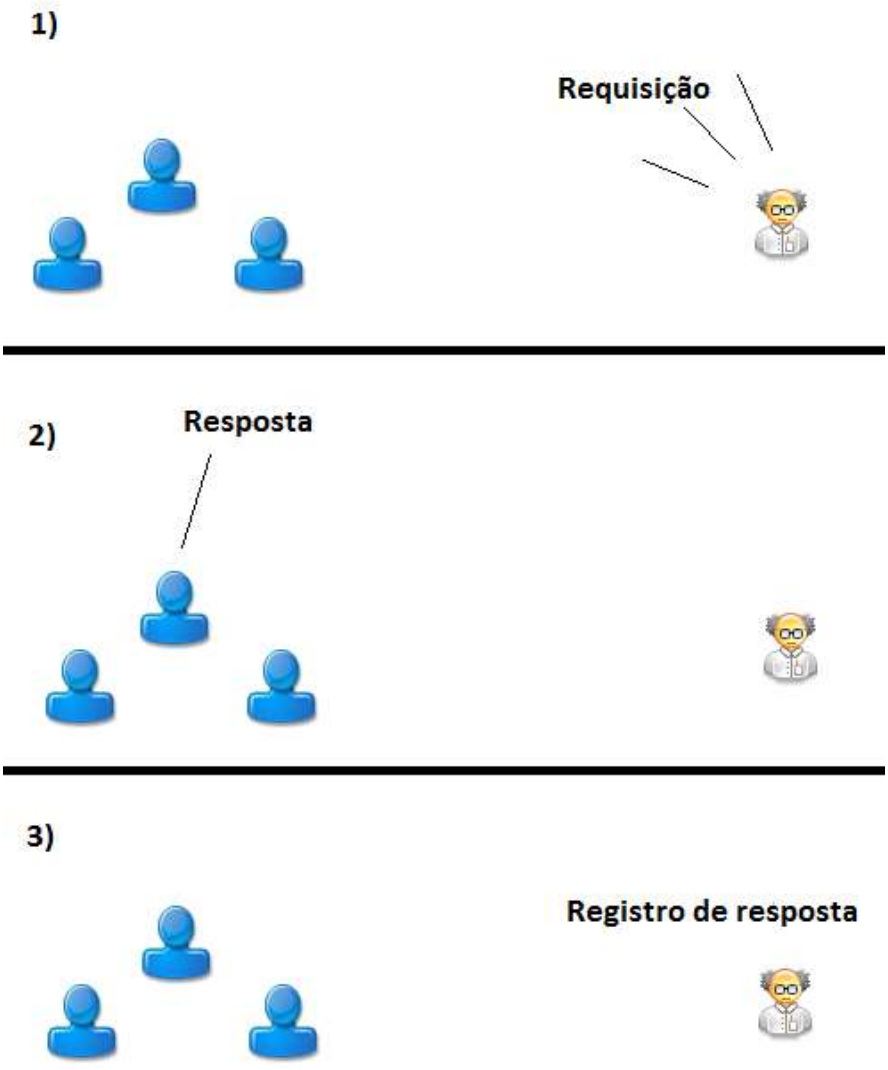


Figura 2.1 - Método atual para realização de chamada

Todavia, diversos problemas decorrentes de tal processo poderiam ser evitados se uma solução que se utilizasse de validação através de identificadores únicos de indivíduos, como o caso da impressão digital, fosse aplicada dentro do contexto de constatação de presença em atividades de cunho acadêmico. A princípio, tais problemas podem ser considerados mínimos em relação ao que o processo é responsável por propiciar, contudo, em uma sociedade cada vez mais informatizada e que possui como principal premissa a realização de atividades no menor intervalo de tempo e com maior segurança possíveis, tais problemas passam a sofrer uma alteração de magnitude considerável à medida de tal evolução. Além disso, qualquer incoerência entre os fatos registrados e os fatos que ocorrem na vida real resulta em potenciais chances de gerar inúmeras indisposições, que podem ir deste a reprovação de um aluno por excesso de faltas de maneira injusta ou até a expulsão de um professor de uma instituição de ensino renomada. A figura 2.2 explicita uma situação que pode ocorrer em decorrência da incoerência entre os fatos registrados e os que ocorrem na vida real.

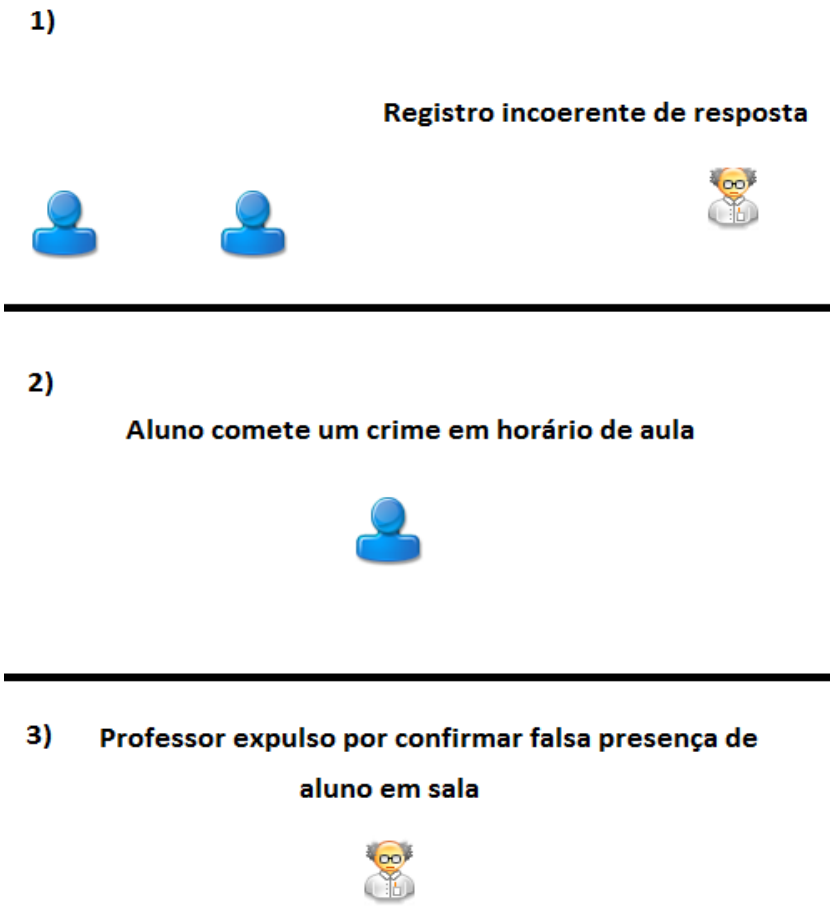


Figura 2.2 - Consequência de chamada incoerente

Primeiramente, pode-se enumerar como problema a questão do tempo que se leva para realizar o processo de chamada de maneira tradicional. Dependendo do tamanho da turma, tal processo pode levar mais de 10 minutos, simplesmente para se constatar quais alunos compareceram e quais deixaram de comparecer. Vale ressaltar que a questão aqui não está relacionada diretamente com o tempo que se gasta na execução de tal processo, mas sim relacionada com o tempo que se poderia economizar ao se ter um sistema centralizado que fosse capaz de proporcionar a realização de tal procedimento através de comparações entre identificadores únicos, como a impressão digital, visto que a mesma já é aplicada de maneira eficiente em diversos outros contextos. A figura 2.3 mostra um exemplo de leitor de impressão digital utilizado, principalmente, em instituições e recintos que possuem controle de acesso e trânsito de pessoas.

Mais um ponto negativo a ser apresentado relativamente à forma como se é realizado o processo de chamada atualmente é a questão dos problemas de recebimento e/ou transmissão de informações. De maneira similar a um protocolo de comunicação, o transmissor, no caso o professor, envia uma mensagem através do meio de comunicação esperando ouvir uma resposta de confirmação/sinalização de mensagem recebida. Desta forma, pode-se ter problemas tanto para se ouvir a mensagem transmitida (receptor) quanto para se ouvir a mensagem de resposta (professor). Normalmente esse tipo de problema ocorre por dois motivos. O primeiro deles é a questão do barulho normalmente existente em salas de aula devido a conversas paralelas ou movimentação demasiada. De maneira análoga, é como se o meio (canal de transmissão) estivesse cheio demais para a interpretação de uma transmissão específica por parte do professor ou por parte do aluno em questão. A figura 2.4 mostra uma situação de incoerência de registro de chamada devido a barulho.

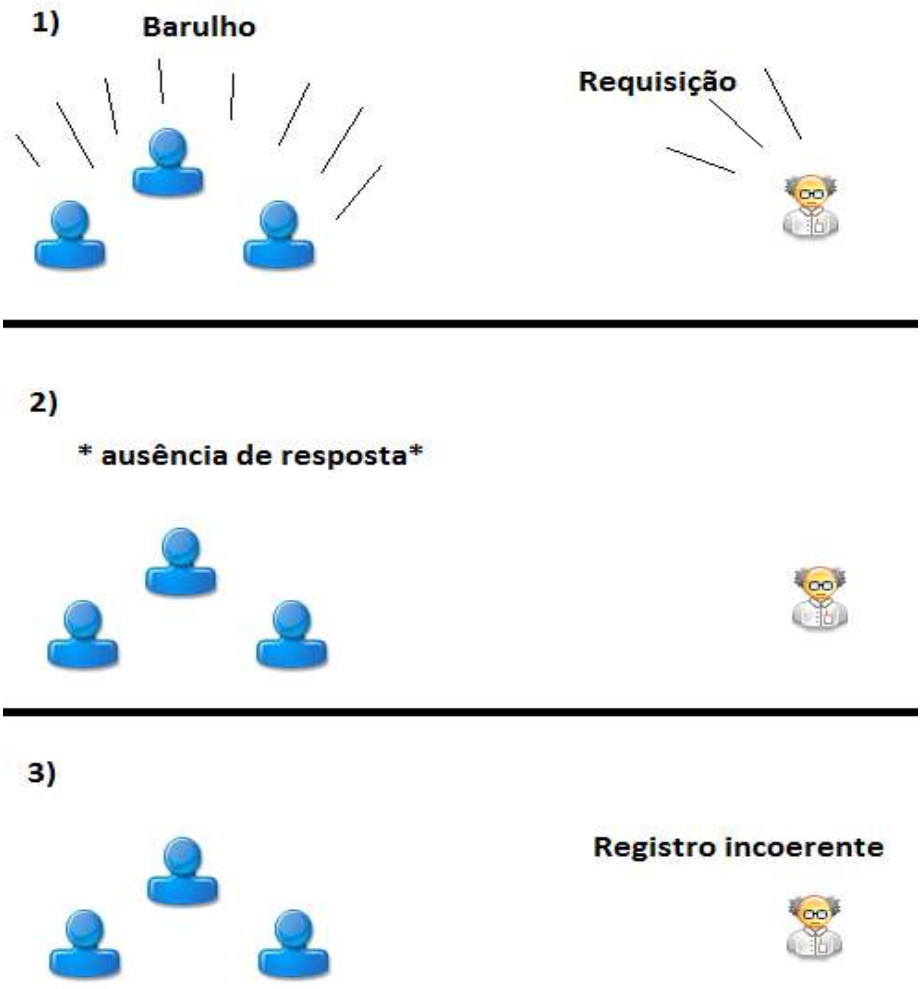


Figura 2.4 – Chamada incoerente devido a barulho

O outro ponto que pode acarretar esse problema é a quantidade exagerada de alunos em sala de aula. Algumas turmas chegam a possuir 60 alunos matriculados, dificultando-se ainda mais o processo de comunicação entre professor e aluno específico, um por um, devido a, mais uma vez, conversas paralelas e movimentações demasiadas que acabam sendo inevitáveis na maioria das instituições de ensino por uma questão cultural. A figura 2.5 mostra uma situação de incoerência de registro de chamada devido a grande quantidade de alunos em sala.

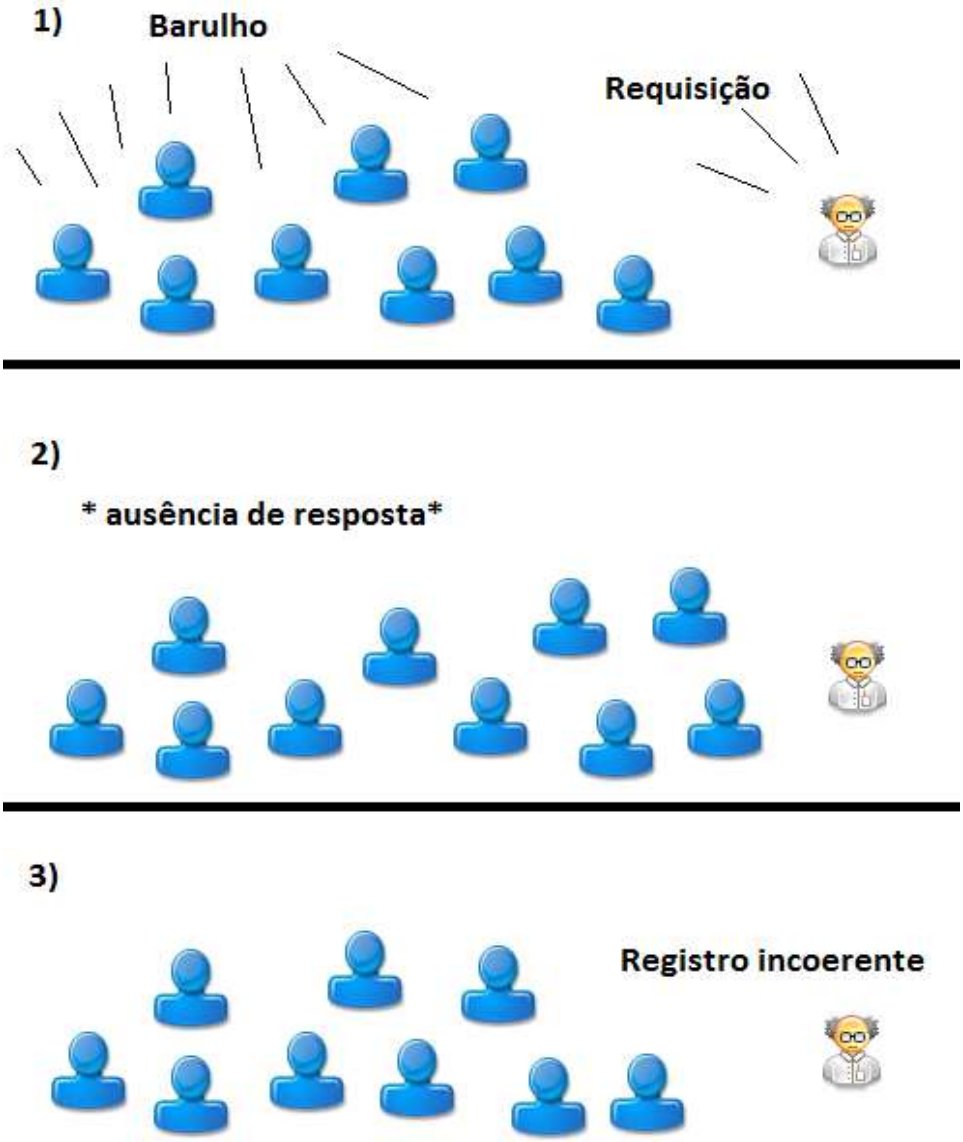


Figura 2.5 - Chamada incoerente devido à grande quantidade de alunos

Já em relação à validade do processo de chamada contestada diretamente com o fato real da presença ou ausência do aluno em questão, existe abertura para três tipos de falhas. A primeira delas está relacionada à má-fé dos alunos associada à boa-fé do professor. Por repetidas vezes, alunos combinam entre si de responderem à chamada uns pelos outros, de maneira que se é criado um rodízio logístico entre tais elementos, com a finalidade de proporcionar possibilidade de falta sem que seja contabilizada em registro oficial acadêmico. Desta forma, o professor, muitas vezes cansado, desatento ou por não ter decorado a voz dos respectivos alunos, acaba marcando como presentes alunos que muitas vezes sequer estão em sala de aula. A figura 2.6 mostra uma situação de incoerência de registro de chamada devido a prática costumeira entre alunos de responderem as chamadas uns pelos outros.

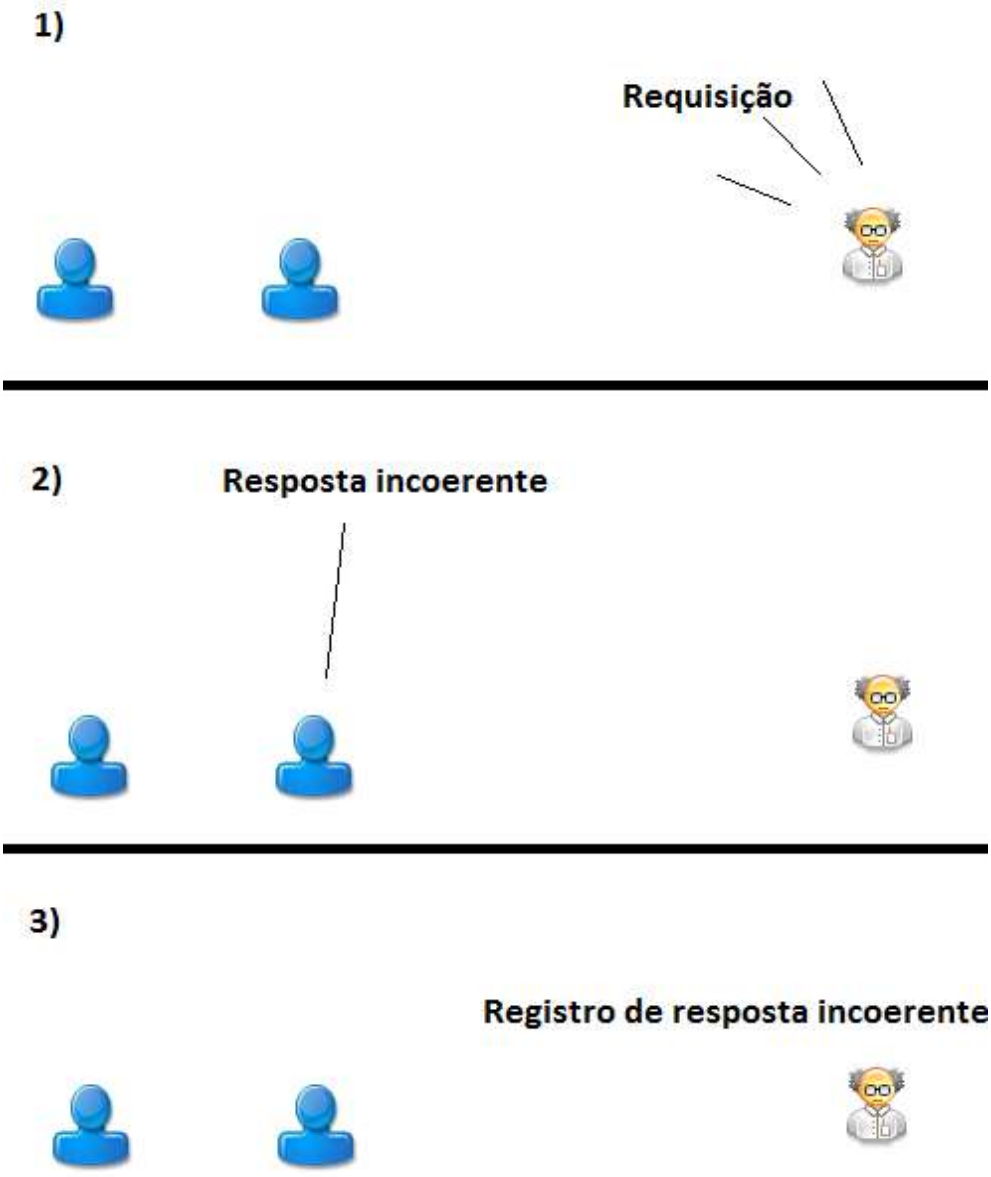


Figura 2.6 - Chamada incoerente devido à má-fé dos alunos

O segundo caso de tal situação ocorre em momentos em que professores, por diversos motivos, ao invés de realizarem o procedimento de chamada de maneira usual e padrão, distribuem listas de presença, onde os alunos que estão em sala de aula devem assinar para constatarem que assistiram à respectiva aula. Neste caso, os alunos acabam assinando uns pelos outros, visto que não há uma evocação direta um a um e que, na maioria das vezes, os professores não conferem o quantitativo presente com a quantidade de assinaturas obtidas no documento que foi tramitado pela sala de aula.

A figura 2.7 mostra uma situação de incoerência de registro de chamada devido à utilização de listas de presença.

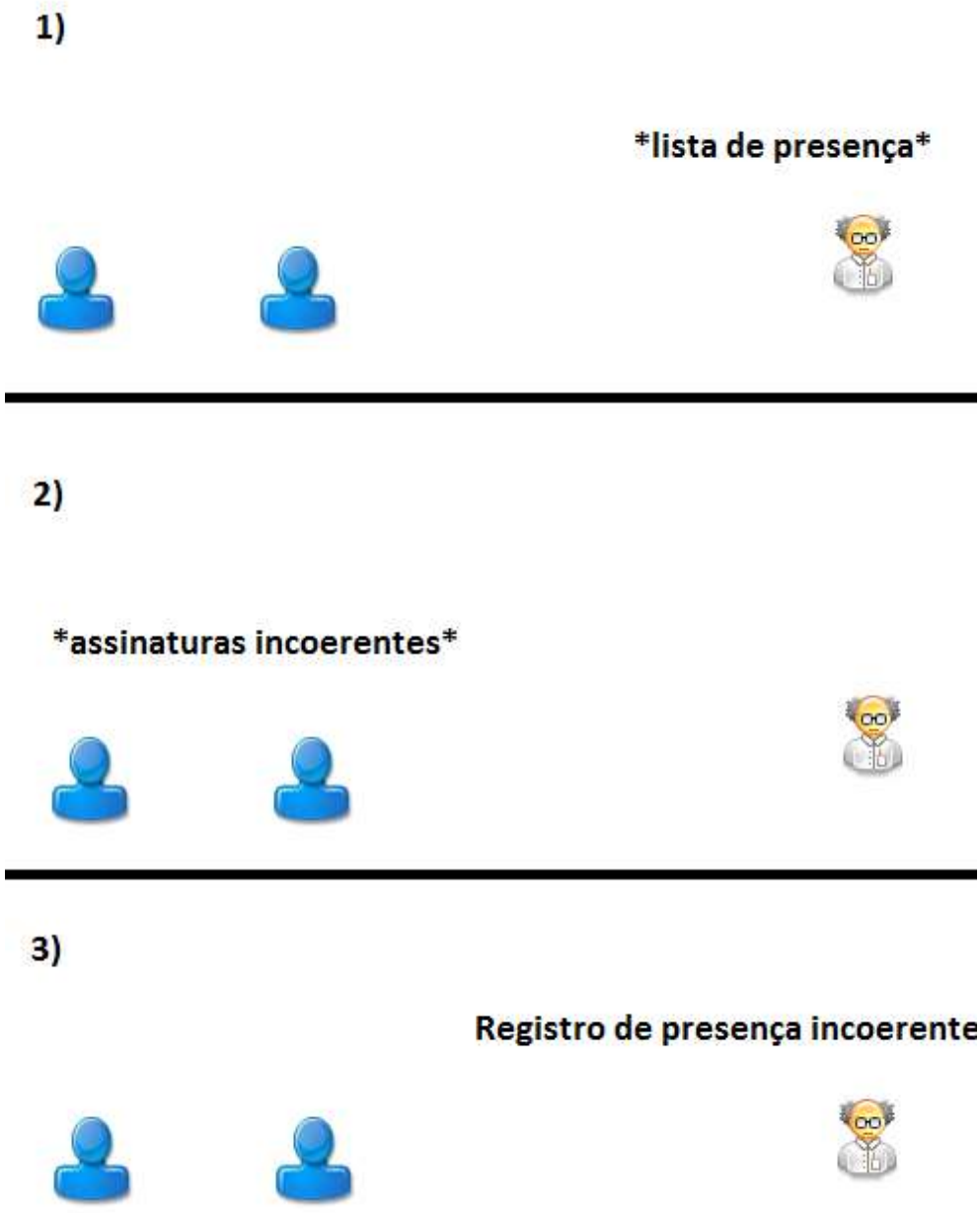


Figura 2.7 - Chamada incoerente devido à utilização de listas de presença

O terceiro e último caso levantado referente à validade do processo de chamada está relacionado com a política de dispensa de chamada adotada por alguns professores. Por todos

saberem que o procedimento padrão de realização de chamada é algo rotineiro, desgastante e muitas vezes gera muito mais indisposições quando realizado do que quando não realizado, alguns professores optam por descartar tal procedimento. A figura 2.8 mostra uma situação de incoerência de registro de chamada devido à dispensa da mesma.

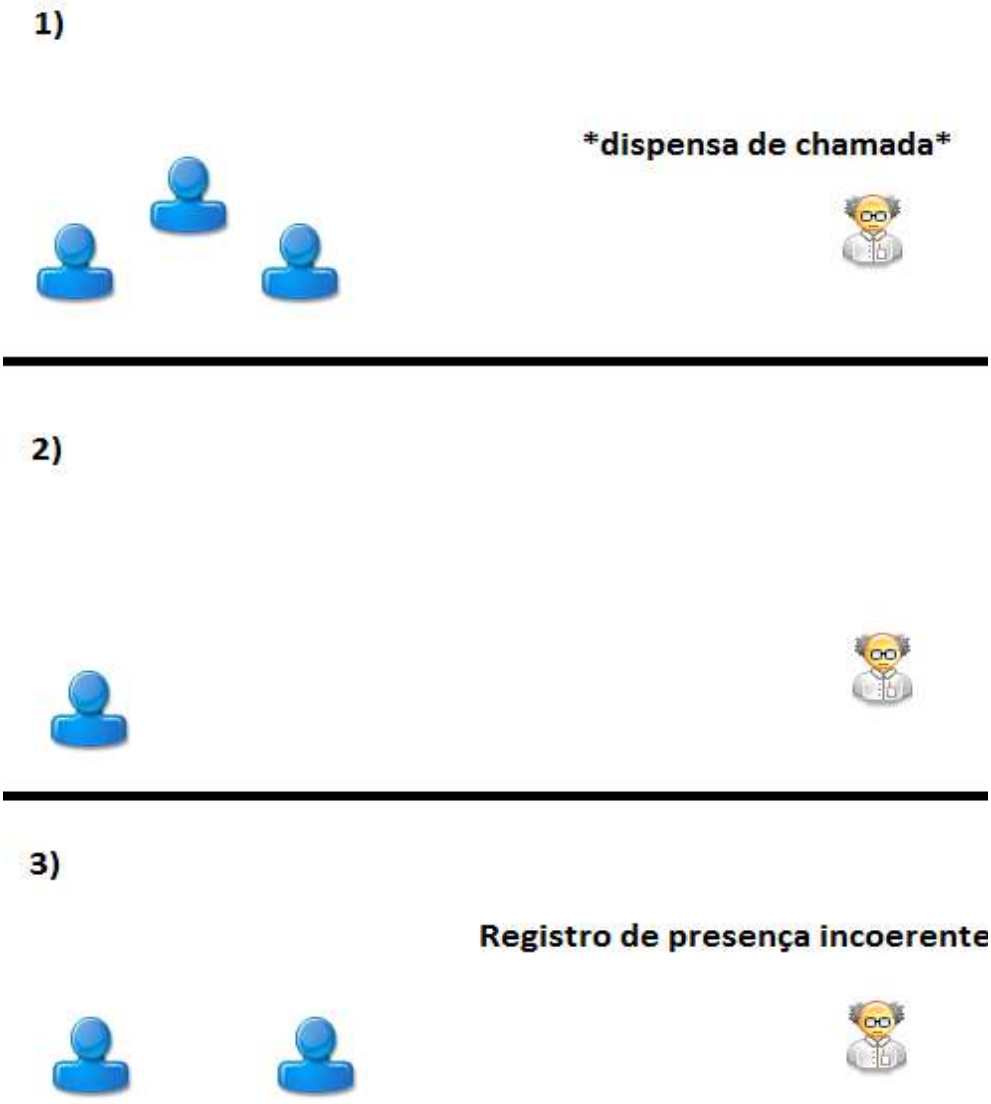


Figura 2.8 - Chamada incoerente devido à dispensa de chamada

Tendo em vista os fatos que foram expostos neste capítulo, infere-se que uma solução para que o processo de realização de controle de assiduidade dos alunos em instituições de ensino é mais do que necessária, sendo bastante viável sua implementação. Para que tal solução vigore com sucesso, alcançando os objetivos desejados, a mesma deverá possuir meios de comunicação tanto com o servidor de banco de dados da instituição quanto com as demais salas de aula. Nas salas de aulas deverão existir leitores de impressão digital em cada uma delas. Desta forma, o acesso à aplicação se dará de maneira remota ao servidor de aplicações, que ficará responsável por gerir todas as operações de registro de chamada via impressão digital da instituição de ensino. A figura 2.9 mostra a topologia de uma solução para o registro de presença de forma automatizada com uso de biometria em uma instituição de ensino, e a figura 2.10 mostra o leitor biométrico utilizado.

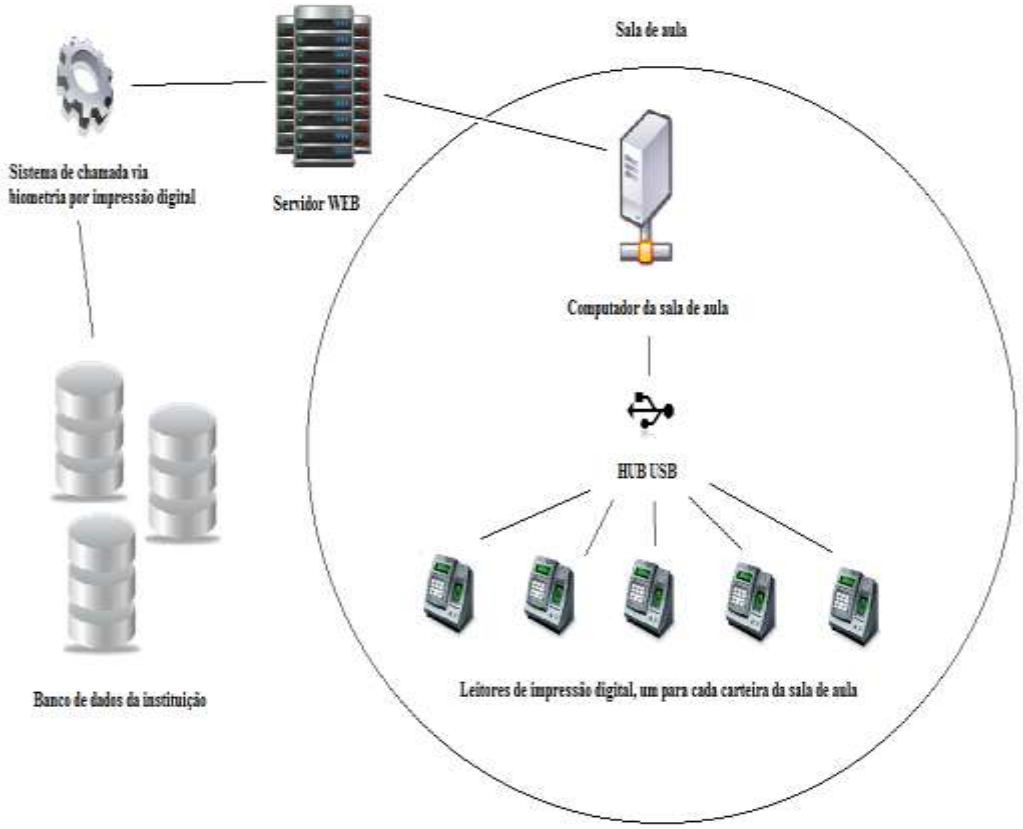


Figura 2.9 - Modelo genérico de implementação de solução para o problema

Algumas características do leitor utilizado *Microsoft Fingerprint Reader* que podem ser enumeradas são: compatibilidade, rejeição de imagens latentes, bom funcionamento com dedos úmidos e secos, resolução de 512 DPI (*Dots per Inch*), área de captura de 14.16 x 18.1 e 256 tons de cinza.

CAPÍTULO 3 - BASES METODOLÓGICAS PARA RESOLUÇÃO DO PROBLEMA

Por possuir como objetivo principal a implementação de um sistema *web* de registro de presença via leitura biométrica de impressão digital para instituições de ensino, este trabalho se relaciona com as seguintes disciplinas cursadas: linguagens e técnicas de programação 1 e 2, estrutura de dados, compiladores, arquitetura de computadores 1 e 2, redes 1 e 2, arquitetura de sistemas distribuídos, banco de dados e sistemas operacionais. Além disso, este trabalho está diretamente relacionado com técnicas, padrões arquiteturais, conceitos e tecnologias de desenvolvimento de sistemas *web*.

3.1 - Arquitetura *Web*

Com o advento da *Internet*, aliado as evoluções que vêm ocorrendo na área de telecomunicações, novos padrões arquiteturais para o desenvolvimento de *softwares* têm surgido e sido colocados em prática ao redor do mundo inteiro por profissionais e cientistas da desenvolvimento de sistemas. Cada novo conceito representa um novo passo à frente no que diz respeito à quebra de paradigmas da computação, trazendo consigo novos desafios e novas possibilidades.

Em relação ao desenvolvimento de aplicações *web*, ou seja, aplicações que rodam de maneira centralizada em um ou em vários servidores, sendo acessados em clientes em diferentes máquinas e em diferentes locais, os padrões mais comum para tal implementação são: cliente leve da *web*, cliente pesado da *web* e liberação pela *web*. Este projeto utiliza o padrão cliente pesado da *web*, que será explicado a seguir.

3.1.1 - Cliente pesado da *Web*

O padrão de arquitetura Cliente pesado da *web* amplia o padrão Cliente leve da *web* ao usar *scripts* e objetos personalizados no cliente como, por exemplo, controles *ActiveX* e *applets Java*. O padrão Cliente pesado da *web* tem esse nome porque o cliente pode realmente executar uma parte da lógica do negócio do sistema e, portanto, torna-se mais que um simples contêiner genérico da

interface do usuário (http://www.wthreex.com/rup/process/workflow/ana_desi/co_warchpatt.htm , acessado em 26/10/2011).

O padrão de arquitetura Cliente pesado da *web* é mais adequado a aplicativos da *web* que possam determinar uma versão de navegador e configuração do cliente, que requeiram uma interface do usuário sofisticada e/ou que possam executar uma parte da lógica do negócio no cliente. Muitas diferenças entre os padrões Cliente leve da *web* e Cliente pesado da *web* estão no papel que o navegador desempenha na execução da lógica de negócio do sistema.

As duas motivações mais fortes para a utilização do Cliente pesado da *web* são: capacidade aprimorada da interface do usuário e execução pelo cliente da lógica do negócio. Uma interface do usuário sofisticada deve ser usada para visualizar e modificar modelos tridimensionais, ou para incluir animações em um gráfico financeiro, por exemplo. Em alguns casos, pode-se usar o controle *ActiveX* para fazer a comunicação com o equipamento de monitoração do cliente. Por exemplo, um equipamento médico que possa medir a pressão sanguínea, fazer a contagem de açúcar no sangue e medir outros sinais vitais pode ser usado por uma instituição que precise monitorar diariamente pacientes situados em diferentes locais, reduzindo dessa forma a frequência de visitas pessoais a duas vezes por semana.

Às vezes, a lógica do negócio pode ser executada somente no cliente. Nesse caso, todos os dados necessários para executar o processo devem estar disponíveis no cliente. A lógica pode ser tão simples quanto validar dados inseridos. Pode-se verificar a exatidão das datas ou compará-las com outras datas (por exemplo, a data de nascimento deve ser anterior à data da primeira internação no hospital). Com base nas regras de negócios do sistema, alguns campos do sistema podem ou não estar ativados, dependendo dos valores inseridos.

Os usos mais óbvio de *scripts*, *applets*, controles e *plug-ins* pelo cliente estão relacionados à *Internet*, na forma de interfaces aprimoradas do usuário. Os *scripts* Java são normalmente usados para alterar a cor ou a imagem de um botão ou item de menu em páginas *Hyper Text Markup Language* (HTML). As *applets Java* e os controles *ActiveX* são normalmente usados para criar controles de visualização de árvores hierárquicas sofisticadas.

O controle e o *plug-in Shockwave ActiveX* compõem um dos componentes mais conhecidos de interface do usuário atualmente em uso na Internet. Ele ativa animações interativas e é usado principalmente para tornar mais interessantes os sites na *Internet*, adicionando gráficos atrativos.

Entretanto, também vem sendo usado para exibir simulações e monitorar eventos esportivos. O controle de agente da *Microsoft* é usado por vários sites na *Internet* para aceitar comandos de voz e executar, no navegador, ações que ajudam o usuário a navegar no site da *web*.

Fora da *Internet*, uma empresa de *software* médico desenvolveu um aplicativo de *Intranet* baseado na *web* para gerenciar registros de pacientes e faturamento. A interface do usuário baseada na *web* usa intensamente *scripts* cliente para executar validações de dados e ajudar o usuário a navegar no site. Além dos *scripts*, o aplicativo utiliza vários controles *ActiveX* para gerenciar conteúdos *eXtensible Markup Language* (XML) usados como o esquema principal de codificação de informações.

Toda a comunicação entre cliente e servidor, assim como ocorre no padrão Cliente leve da *web*, é feita com o protocolo *Hyper Text Transfer Protocol* (HTTP). Como o HTTP é um tipo de protocolo "sem conexão", na maior parte do tempo não há uma conexão aberta entre cliente e servidor. O cliente só envia informações durante as solicitações de páginas. Isso significa que os *scripts*, os controles *ActiveX* e os *applets Java* do cliente estão limitados à interação com objetos somente no cliente.

O padrão Cliente pesado da *Web* utiliza determinados recursos do navegador como controles *ActiveX* e *applets Java* para executar a lógica do negócio no cliente. Os controles *ActiveX* são executáveis binários compilados, cujo *download* no cliente pode ser feito via *HTTP* e que podem ser chamados pelo navegador. Como os controles *ActiveX* são essencialmente objetos *Component Object Model* (COM), eles têm total domínio sobre os recursos do cliente. Podem interagir tanto com o navegador como com o próprio sistema cliente. Por esse motivo, os controles *ActiveX*, especialmente os da *Internet*, são normalmente "autenticados" por terceiros confiáveis.

As versões mais recentes dos navegadores HTML mais conhecidos também permitem que o cliente produza *scripts*. As páginas HTML podem ser incorporadas com *scripts* escritos em *Java script* ou VB *script*. Essa capacidade de criação de *scripts* permite que o próprio navegador execute (ou interprete) o código que pode ser parte da lógica do negócio do sistema. O termo "pode ser" é usado porque é muito comum que os *scripts* cliente contribuam somente com os aspectos externos da interface do usuário, não fazendo realmente parte da lógica do negócio. Nos dois casos, há

elementos potencialmente importantes do ponto de vista da arquitetura (isto é, *scripts*), incorporados às páginas HTML, que precisam ser expressos como tal.

Na medida em que o padrão Cliente pesado da *web* é realmente uma simples extensão do padrão Cliente leve da *web*, a maioria dos elementos significativos do ponto de vista da arquitetura são os mesmos. Os elementos adicionais apresentados pelo Cliente pesado da *web* são:

Script cliente**:** *JavaScript* ou VB (*Visual Basic*) *script* incorporado às páginas HTML formatadas. O navegador interpreta o *script.* O *World Wide Web Consortium* (W3C), uma organização de padrões para a *Internet*, definiu a interface *Document Object Model* (DOM) e HTML que o navegador oferece aos *scripts* cliente.

Documento XML**:** Um documento formatado por meio de *eXtensible Markup Language* (XML). Os documentos XML representam o conteúdo (os dados) sem formatação da interface do usuário.

Controle ActiveX: Um objeto COM que pode ser referenciado em um *script* cliente e cujo *download* pode ser feito para o cliente, se necessário. Assim como qualquer objeto COM, ele tem total acesso aos recursos do cliente. Os princípios fundamentais do mecanismo de segurança para proteção dos equipamentos cliente são obtidos por meio de autenticação e assinatura. Pode-se configurar os navegadores de *Internet* para não aceitar o *download* de controles *ActiveX* no cliente ou para avisar ao usuário que isso será feito. Os mecanismos de autenticação e assinatura basicamente determinam a identidade do autor do controle por meio de terceiros confiáveis.

Applet Java: Componente independente e compilado, que é executado no contexto de um navegador. Por motivos de segurança, ele tem acesso limitado aos recursos no lado do cliente. Os *Applets Java* são usados como elementos sofisticados da interface do usuário e também para finalidades não referentes à interface do usuário como, por exemplo, análise de documentos XML ou encapsulamento de lógica de negócio complexa.

*Java Bean***:** Componente *Java* que implementa um determinado conjunto de interfaces, permitindo que seja facilmente incorporado a sistemas mais complexos e de maior porte. O termo *bean* (grão) reflete a natureza simples e a finalidade única que o componente deve ter. Uma xícara

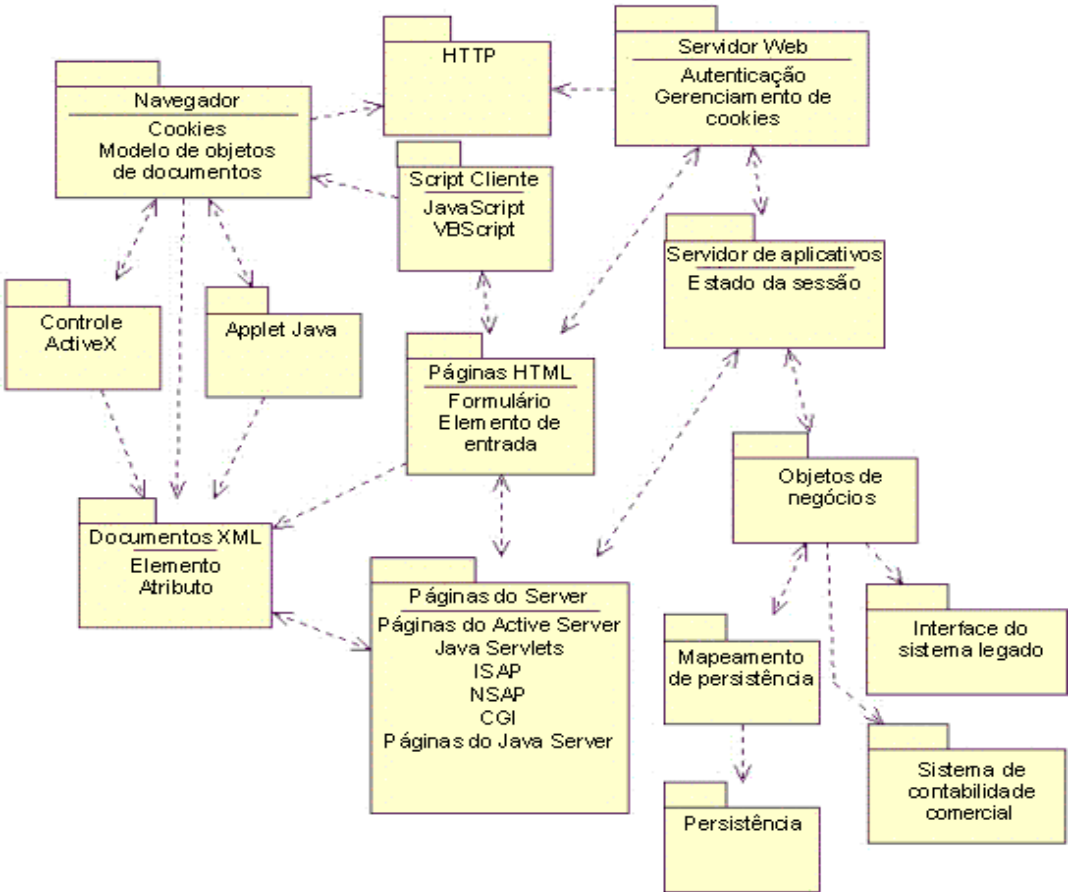


Figura 3.1: Visão lógica do padrão de arquitetura Cliente pesado da web
http://www.wthreex.com/rup/process/workflow/ana_desi/co_warchpatt.htm

cheia de café geralmente requer mais de um grão. O *ActiveX*, nas arquiteturas centralizadas da *Microsoft*, é análogo ao *Java Bean* (Kai Qian; Richard Allen; Mia Gan; Robert Brown, 2010).

A principal dinâmica do padrão Cliente pesado da *web* inclui o padrão Cliente leve da *web*, mais a capacidade de executar a lógica do negócio no cliente. Assim como acontece no padrão Cliente leve da Web, toda a comunicação entre cliente e servidor é feita durante as solicitações de páginas. A lógica do negócio, porém, pode ser parcialmente executada no cliente, por meio de *scripts*, controles ou *applets*. A figura 3.1 mostra, de maneira genérica, como se dá a interação entre as partes constituintes da arquitetura Cliente pesado da *web*.

Quando uma página é enviada para um navegador cliente, ela pode conter *scripts*, controles e *applets*. Elas podem ser usadas simplesmente para aprimorar a interface do usuário ou para contribuir com a lógica do negócio. A utilização mais simples da lógica do negócio está relacionada às validações de campo. É possível usar os *scripts* cliente para validar as entradas, não apenas em um único campo, mas também em todos os campos de uma determinada página da *web*. Por

exemplo, um aplicativo de comércio eletrônico que permita aos usuários configurar seus próprios sistemas de computador pode usar *scripts* para evitar a especificação de opções incompatíveis.

Para que os *applets Java* e os controles *ActiveX* possam ser usados, é necessário especificá-los no conteúdo da página HTML. Esses controles e applets podem funcionar independentemente de qualquer *script* na página, ou podem ser controlados por *scripts* na página. Os *scripts* em uma página HTML podem responder a eventos especiais enviados pelo navegador. Esses eventos podem indicar que o navegador acabou de concluir o carregamento da página da *web* ou que o mouse do usuário acabou de ser movido sobre uma região específica da página (Décio Heinzelmann Luckow; Alexandre Altair de Melo, 2010).

Eles têm acesso à interface *Document Object Model* (DOM). Essa interface é um padrão W3C que dá aos *scripts*, controles e *applets* acesso ao navegador e ao conteúdo HTML das páginas. A implementação da *Microsoft* e do *Netscape* quanto a esse modelo é o HTML Dinâmico (DHTML). O DHTML é mais que uma simples implementação da interface DOM. Ele inclui eventos que não faziam parte da especificação de Nível 1 do DOM no momento em que este documento estava sendo elaborado.

O núcleo do DOM é um conjunto de interfaces que manipula especificamente os documentos XML. XML é uma linguagem flexível, que permite que os designers criem suas próprias marcas para fins especiais. A interface DOM permite que os *scripts* cliente acessem documentos XML.

O uso de componentes especiais no cliente possibilita a utilização de XML como um mecanismo padrão de troca de informações entre cliente e servidor. Os controles *ActiveX* ou os *applets Java* podem ser implementados no cliente para solicitar e enviar, de modo independente, documentos XML. Por exemplo, um *applet Java* incorporado a uma página HTML pode fazer uma solicitação HTTP de um documento XML do servidor de Web. O servidor de *web* localiza e processa as informações solicitadas e envia um documento formatado em XML, e não um documento HTML. O *applet* ainda em execução na página HTML, no lado do cliente, aceita o documento XML, analisa-o e interage com o documento HTML atualmente no navegador, a fim de exibir o seu conteúdo para o usuário. Toda a seqüência ocorre no contexto de uma única página HTML no navegador cliente (Steve Soulders, 2007).

3.2 - Linguagem de programação *Java*

Basicamente, *Java* constitui-se de uma linguagem de programação e um programa para execução chamado de máquina virtual ou *virtual machine* (JVM). Quando programa-se em *Java* usa-se a linguagem de programação *Java* e um ambiente de desenvolvimento *Java* para gerar um *software* que será executado em um ambiente de distribuição *Java*. Tudo isso é a tecnologia *Java*.

A tecnologia *Java* começou a ser criada em 1991 com o nome de *Green Project*. O projeto era esperado como a próxima geração de *software* embarcado. Nele trabalhavam James Grosling, Mike Sheridan e Patrik Naughton. Em 1992 surge a linguagem *Oak*, a primeira máquina virtual implementada. Várias tentativas de negócio foram feitas para vender o *Oak*, mas nenhuma com sucesso (http://devjr.wordpress.com/tecnologia-java/ , acessado em 27/10/2011).

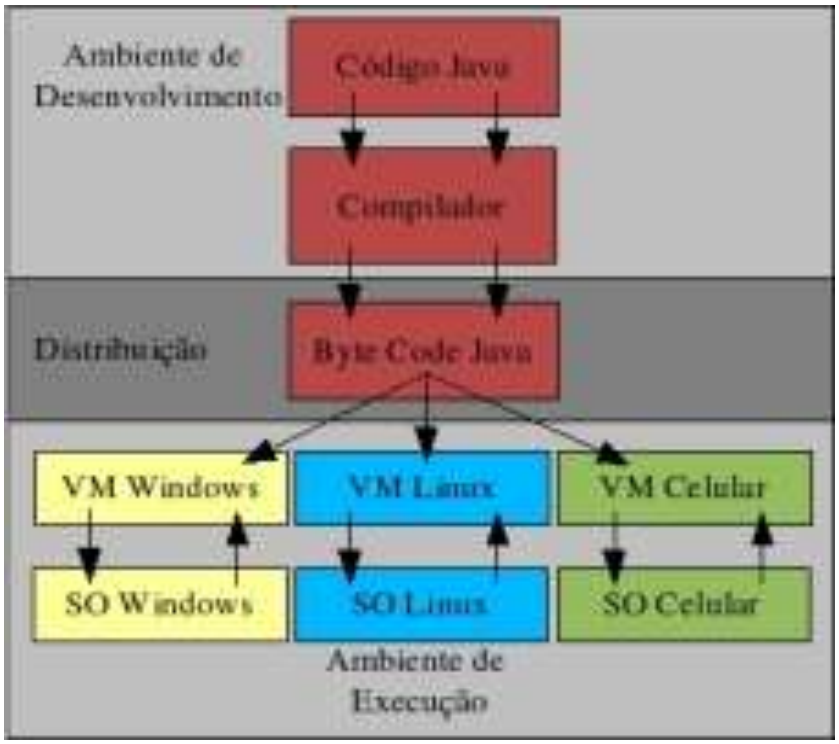
Em 1994 surge a *Internet*, a *Sun* vê uma nova possibilidade para o *Green Project* e cria uma linguagem para construir aplicativos *web* baseada na *Oak*, a *Java*. Em 23 de maio de 1995 a linguagem *Java* é oficialmente lançada na *SunWorld Expo* 95 com a versão *Java Development Kit* (JDK) 1.0 *alpha*. A *Netscape* aposta na idéia e inicia a implementação de interpretadores *Java* em seu navegador, possibilitando a criação de *Java applets*. A partir desta etapa o *Java* começa a crescer muito. Em 1997 é lançado a prova da maturidade da tecnologia. De 1998 até hoje a tecnologia evoluiu muito possuindo um dos maiores repositórios de projetos livres do mundo, o java.net. Em 1999 surgiu a plataforma para desenvolvimento e distribuição corporativa batizado de *Java 2 Enterprise Edition* (J2EE) e a plataforma *Java 2 Mobile Edition* (J2ME) para dispositivos móveis, celulares, e outros aparelhos limitados. Atualmente *Java* é uma das linguagens mais usadas e serve para qualquer tipo de aplicação, entre elas: *web*, *desktop*, servidores, *mainframes*, jogos, aplicações móveis, *chips* de identificação, etc.

Java é multiplataforma. Quando um programa *Java* é compilado um código intermediário é gerado, chamado de *bytecode*. Este *bytecode* é interpretado pelas máquinas virtuais java (JVMs) para a maioria dos sistemas operacionais. A máquina virtual é a responsável por criar um ambiente multiplataforma, ou seja, se alguém construir um sistema operacional novo, basta criar uma máquina virtual java que traduza os *bytecodes* para código nativo.

Entre outras funções, a máquina virtual java também é responsável por carregar de forma segura todas as classes do programa, verificar se os *bytecodes* aderem a especificação JVM e se eles não violam a integridade e a segurança do sistema.

De um código *Java*, que está em um arquivo *.java*, o compilador *javac* gera o *bytecode*: um arquivo *.class*. Após isso uma máquina virtual java executa o *bytecode* e roda o programa.

Como existe um programa traduzindo um código a cada execução do sistema, poderia se dizer que *Java* sempre será mais lenta que as linguagens que geram código nativo do sistema operacional como *Delphi*, *Visual Basic* ou *C++*. Isso era fato até 1996 quando a *Sun* criou o compilador *Just-in-time* (JIT) que analisa e retira códigos desnecessários aumentando consideravelmente a velocidade da execução. Atualmente o *Java* é mais rápido que o próprio *C* em vários aspectos (Harvey M. Deitel, 2010). A figura 3.2 mostra como a máquina virtual *Java* interage com os diferentes sistemas operacionais e códigos *Java*.



Java se divide em três grandes edições:

Java 2 Standard Edition (J2SE): É a tecnologia *Java* para computadores pessoais, notebooks e arquiteturas com poder de processamento e memória consideráveis. Várias *Application Package Interfaces* (APIs) acompanham esta versão e tantas outras podem ser baixadas opcionalmente no

site da *Sun*. É com elas que a maioria das aplicações são construídas e executadas. O J2SE possui duas divisões: o *Java Development Kit* (JDK) ou *Standard Development Kit* (SDK), que é um conjunto para desenvolvimento em *Java* e deveria ser instalado apenas pelos desenvolvedores por possuir ferramentas para tal tarefa, e o *Java Runtime Edition* (JRE), que é uma versão mais leve da JDK pois é preparada para o ambiente de execução, ou seja, é esta versão que executará os sistemas construídos com a SDK (http://javafree.uol.com.br/artigo/871498/ , acessado em 27/10/2011).

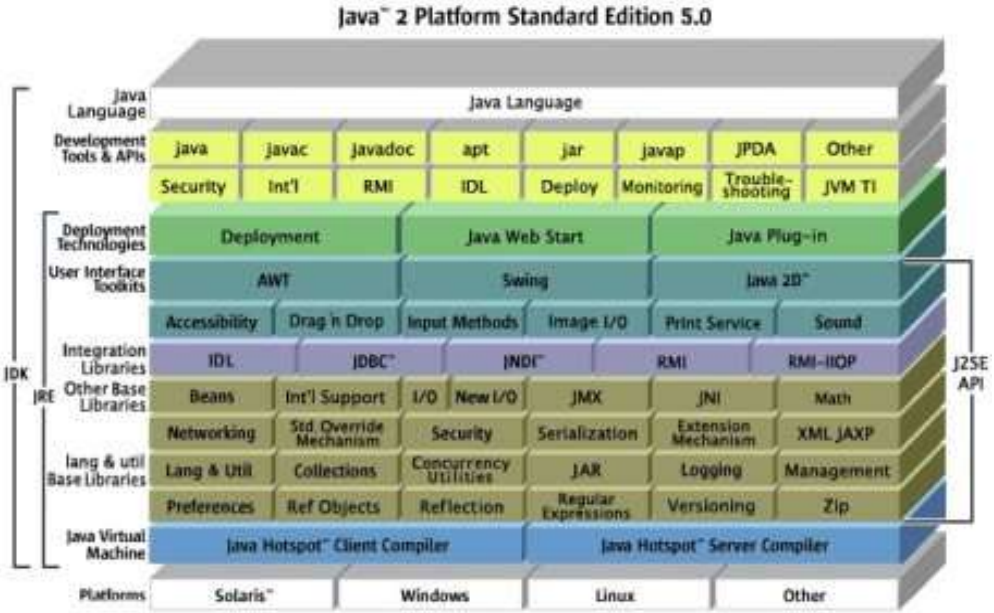
Java 2 Mobile Edition (J2ME): É a tecnologia *Java* para dispositivos móveis com limitações de memória ou processamento. Possui APIs bem simples e leves para economizar espaço, memória e processamento. São utilizadas para sistemas em celulares, *palmtops*, *pocket pcs*, *smartphones*, *javacards* e demais dispositivos. O J2ME se divide em dois grupos de bibliotecas. É dividida em dois grupos: o *Connected Limited Device Configuration* (CLDC), para celulares e *smartphones*, que são mais limitados, o *Connected Device Configuration* (CDC), para *palmtops* e *pocket pcs* e alguns dispositivos mais poderosos.

Java 2 Enterprise Edition (J2EE): É a tecnologia *Java* para aplicações corporativas que podem estar na internet ou não. Possui um grande número de APIs onde a segurança é a principal preocupação. É ideal para a construção de servidores de aplicação, integração de sistemas ou distribuição de serviços para terceiros. Este projeto se baseia na edição J2EE do *Java*. A figura 3.3 possibilita uma visão panorâmica de toda abrangência da arquitetura J2SE.

Desta forma, a linguagem escolhida foi *Java*, primeiramente pela questão de ser multiplataforma, isto é, independente do sistema operacional do computador ou dispositivo de acesso à aplicação, o programa conseguirá ser executado normalmente. Isto acontece graças à máquina virtual *Java*, que converte todo o código do programa para *bytecode*, e após isso, executa tal *bytecode* em si mesma. Os clientes que acessarão o programa deverão possuir apenas a máquina virtual instalada (JRE). Outro ponto que motivou a escolha da linguagem *Java* é a questão da robustez, principalmente no que diz respeito ao desenvolvimento de aplicações *web.* As diferentes edições da linguagem *Java* permitem a existência de inúmeras bibliotecas específicas para todo e qualquer tipo de comunicação, seja ele entre computadores, dispositivos móveis ou qualquer outro aparelho que possua a máquina virtual *Java* instalada. E como é uma linguagem já consolidada no mercado, sendo utilizada nos mais diversos contextos, *Java* foi escolhida para ser a linguagem de escrita do projeto referenciado por este trabalho.

3.3 - *Applets*

Applets são pequenos programas de computador, escritos na linguagem *Java*, que destinam-se a uma execução objetiva no lado do cliente em uma comunicação cliente servidor. Dentro do contexto de uma aplicação *web*, normalmente, *applets* são utilizados para se executar alguma validação, extração de dados ou operação do lado do dispositivo do cliente que o servidor não teria



possibilidade de realização.

No caso específico deste trabalho, uma solução em *applet* foi utilizada para se poder extrair o conteúdo em dados binários da impressão digital de um aluno no momento de seu cadastro e/ou no momento da realização do processo de chamada. Este tipo de solução foi adotado pois o(s) leitor(es) de impressão digital estarão conectados em uma máquina cliente, que possuirá os respectivos *drivers* instalados localmente, proporcionando a leitura local com fins de posterior envio remoto de dados para a aplicação desenvolvida rodando em um servidor. Desta maneira, todas as vezes que uma página que contenha uma declaração da *applet* de leitura de impressão digital for chamada, uma solicitação para que o *applet* seja baixado será feita. Ao se confirmar tal solicitação,

a *applet* baixada para a máquina cliente, proporcionando tal leitura e posterior comunicação com o servidor. Vale lembrar que os *applets* devem possuir assinatura digital para que qualquer tipo de bloqueio de *download* nesse sentido seja evitado.

Em uma aplicação, o ponto de partida para a sua execução é realizada no método *main()*. Em um *applet*, porém, este não é o caso. O primeiro código que um *applet* executa é o código definido para sua inicialização, e seu construtor, através do método *init()* que executa a inicialização básica do *applet*. Geralmente usado para iniciar as variáveis globais, obter recursos de rede e configurar a interface. O método *init()* é chamado apenas uma vez, quando o código do *applet* é carregado pela primeira vez, ao iniciar.

Depois que o *init ()* é completado, o *browser* chama outro método chamado *start()*. Esse método é chamado sempre que uma página *web* contendo o *applet* torna-se ativo no *browser* ou para iniciar várias vezes o mesmo *thread* que será comentado com mais detalhe posteriormente.

A exibição de um *applet* é criada a partir do método *paint()*. O método *paint()* é chamado pelo ambiente de *browser* sempre que a exibição do *applet* precise ser atualizado, isto pode acontecer quando a janela de *browser* é aparecida depois de ser minimizado, por exemplo. (http://cavalcante.us/Programacao/ , acessado em 27/10/2011)

Se o usuário sair da página que contém o *applet*, o *browser* chama o método *stop*. No caso da finalização de uma *thread*, o método *stop()* é acionado automaticamente. O método *destroy()* é chamado para otimizar a alocação de memória, liberando espaço para ela antes do *applet* ser excluído. A figura 3.4 mostra uma tela de verificação de segurança de *applet* baixada da *Internet*.

Figura 3.4: Exemplo de solicitação de download de uma applet qualquer
http://exploit.co.il/wp-content/uploads/2010/07/java3.jpg

3.4 - *Servlets*

Servlets são pequenos programas de computador, escritos na linguagem *Java*, que destinam-se a uma execução objetiva no lado do servidor em uma comunicação cliente servidor. Dentro do contexto de uma aplicação *web*, normalmente, *servlets* são utilizados para se processar requisições e respostas, proporcionando, dessa maneira, novos recursos ao servidor.

Servlets utilizam uma API própria (*Java Servlet* API – javax.servlet) para realizar tais operações de processamento de requisições e respostas. Tais componentes são considerados como extensões de um servidor *web*, utilizados com uma função específica, quando o processamento direto pelo servidor não é viável ou não pode ser realizado.

Os *servlets* podem ser acessados via GET e POST. Basicamente, o método GET trabalha com parâmetros passados na URL de acesso e conexão ao *servlet*, sendo tais parâmetros recuperados internamente através da chamada do método *getParameter()* para posterior processamento e resposta. O método POST consiste também na passagem de parâmetros, porém esse procedimento é realizado através do envio de formulários. A recuperação dos parâmetros também se dá de maneira semelhante ao método GET. Um *servlet* deve, obrigatoriamente, implementar um ou outro método de passagem de parâmetros, ficando também facultada a implementação de ambos, porém, nesse último caso, a sobrescrita do método *service()* fica obrigatória.

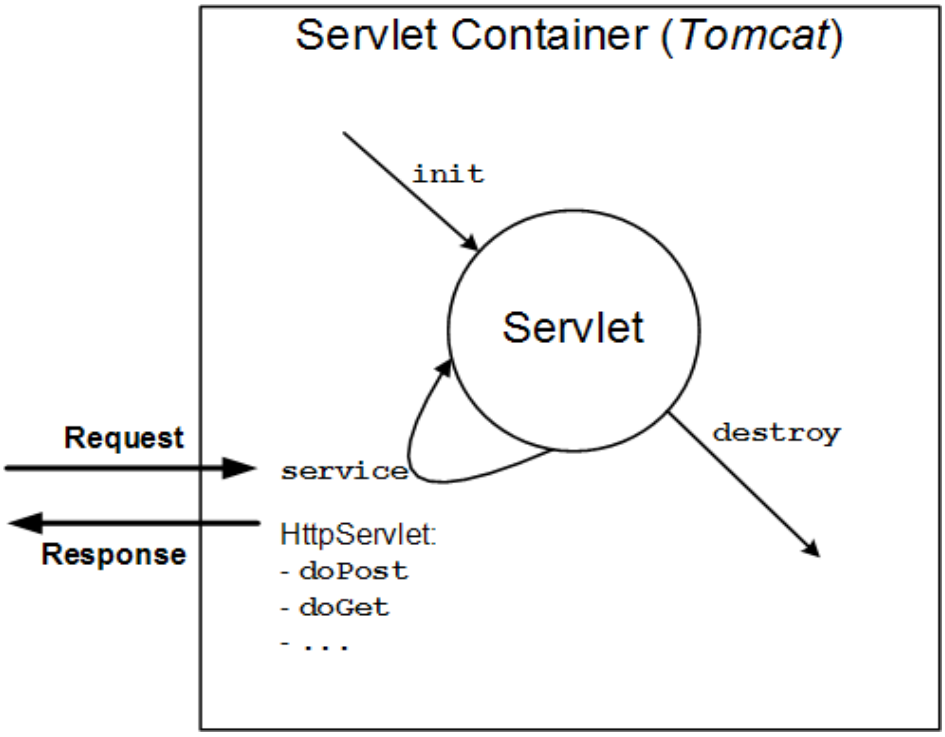


Figura 3.5: Modelo de um servlet genérico

http://disciplinas.ist.utl.pt/leic-sod/2008-2009/labs/03-web-applications/servlets-jsps/servlet-life-cycle.png

Para o contexto deste projeto, um *servlet* foi utilizado para fazer a intermediação da comunicação entre a aplicação de fato, e o *applet* de leitura de impressão digital. Tal intermediação é feita através do registro de *cookies*, que nada mais são do que estruturas de dados do tipo chave/valor, que sinalizam e armazenam valores de transmissão entre cliente e servidor. A figura 3.5 mostra uma representação genérica de comunicação entre um contâiner *web* e um *servlet*.

3.5 - Servidor *Web*

Um servidor *web* é um programa de computador responsável por aceitar pedidos HTTP de clientes, geralmente os navegadores, e servi-los com respostas HTTP, incluindo, opcionalmente, dados, que geralmente são páginas *web*, tais como documentos HTML com objetos embutidos (imagens, por exemplo).

Os pedidos HTTP que se referem habitualmente a paginas HTML são normalmente feitos através de navegadores. O processo se inicia com a conexão entre o computador onde está instalado o servidor *web* e o computador ou dispositivo cliente. Como na *web* não é possível prever a hora em que se dará a conexão, os servidores *web* precisam estar disponíveis dia e noite. A partir daí, é processado o pedido do cliente, conforme as restrições de segurança e a existência da informação solicitada, o servidor devolve os dados em forma de resposta.

Atualmente há cada vez mais programas que fazem pedidos HTTP (leitores *Really Simple Sindication* – RSS, por exemplo) e quase desde o início da *web*, as páginas servidas pelo servidor *web* vão além de páginas HTML, incluindo imagens, arquivos de áudio, e outros tipos de mídias. Genericamente, tudo que possa ser convertido em código HTML ou codificado em uma página HTML pode ser enviado a um servidor *web*. Os servidores *web* também podem executar *scripts*, interagindo ainda mais com o usuário.

No caso específico deste trabalho, escolheu-se o servidor *web Tomcat 6*, da *Apache Software Foundation* (ASF). O *Tomcat* não é um servidor *web* de fato. Na verdade, o *Tomcat* é um *servlet* contâiner, isto é, um componente de *software* direcionado a armazenar e realizar o processamento de *Java servlets e Java Server Pages* (JSPs), que realiza comunicação com os clientes utilizando-se dos protocolos HTTP 1.1 e *Apache Jserv Protocol* (AJP) 1.3. Funciona de maneira muito similar a um servidor *web*, sendo muitas vezes confundindo com o servidor *web* da *Apache*, porém, uma de suas diferenças está no fato de não possuir suporte ao padrão *Enterprise Java Bean* (EJB).

O *Tomcat* realiza o processamento das páginas da aplicação, codificando-as e decodificando-as em arquivos HTML, possuindo diversos *plugins* que podem ser utilizados de acordo com o objetivo desejado e de fácil configuração. Também possui total suporte à linguagem *Java*, implementando especificações da própria *Oracle*. A figura 3.6 mostra a tela inicial do *Apache Tomcat* ao ser acessado localmente quando inicializado.

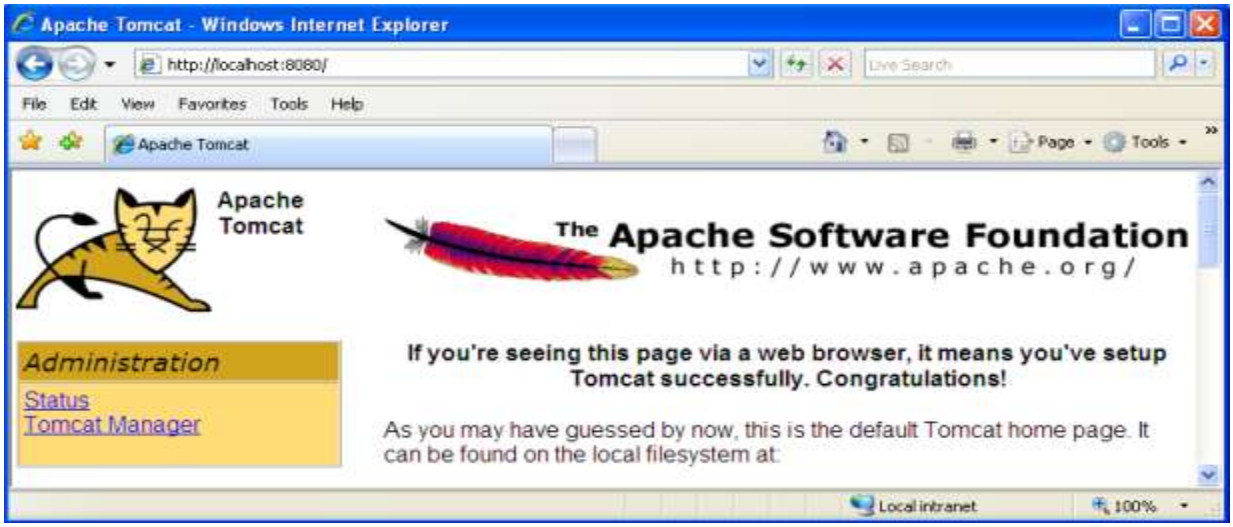


Figura 3.6: Página inicial de demonstração do Tomcat, após instalado
http://workshops.opengeo.org/postgis-spatialdbtips/_images/tomcat_install_20.png

3.6 - IDE

Uma *Interface Development Environment* (IDE) é um programa de computador que reúne características e ferramentas de apoio o desenvolvimento de *software*, com o objetivo de agilizar tal processo. Alguns componentes das IDEs são:

Editor de texto: onde de fato o código será escrito. Vários *plug-ins* de formatação e seleção de trechos de código são desenvolvidos para tal componente.

Compilador: responsável por compilar o código-fonte de uma linguagem específica e transformá-lo em linguagem de máquina.

Depurador: auxilia o programador na depuração do código, permitindo executar a aplicação de maneira pausada, de acordo com os pontos determinados para tal pausa.

A IDE escolhida para o desenvolvimento do projeto em questão foi a chamada *Eclipse Helios 32bits*. Tal escolha se pautou na excelente compatibilidade com a linguagem *Java*, visto que a IDE também foi desenvolvida em tal linguagem. Além disso, existe uma inúmera quantidade de *plug-ins* já disponíveis e desenvolvidos a todo momento para tal programa. E por fim, a questão da sua rapidez, leveza de execução, boa aceitação por parte de programadores e interface intuitiva

também influenciaram na escolha da IDE a ser utilizada para a implementação deste trabalho. A figura 3.7 mostra a tela inicial da IDE *Eclipse Helios 32bits*.

3.7 - *Framework* de componentes gráficos

Frameworks são soluções reutilizáveis de *software*, com a finalidade de poupar o desenvolvedor de começar uma aplicação do zero e reescrever códigos que são comuns nos modelos arquiteturais de programação. Um *framework* pode incluir programas de suporte, bibliotecas de código, linguagens de *script*, e outros *softwares* com a finalidade de auxiliar no desenvolvimento da aplicação propriamente dita.

Neste trabalho, utilizou-se um *framework* chinês de componentes gráficos para o desenvolvimento *web* chamado *Zkoss* (ZK). A proposta do ZK é o desenvolvimento rápido, com a menor quantidade possível de códigos *javascript* explícitos na página e com intuitiva utilização. Os componentes também são bastante simples de serem manipulados, visto que o próprio *framework* se encarrega do processamento bruto, porém extremamente poderosos quando utilizados de maneira consciente e organizada (http://zkbrasil.blogspot.com/, acessado em 10/11/2011). O ZK é um *framework* relativamente novo e, por isso, pouco difundido dentro da comunidade de programadores brasileira. Contudo, vários clientes e grandes empresas ao redor do mundo já utilizam e atestam o potencial do ZK. Sua instalação também é bastante simples, visto que já existe

um *plug-in* de instalação automatizada para a IDE *Eclipse*. A renderização dos componentes gráficos do ZK, é bastante rápida e, acima de tudo, consiste em um projeto ativo que a cada mês disponibiliza novas versões, com inovações, correções e propostas para desenvolvimentos futuros. A figura 3.8 apresenta uma lista de algumas empresas que estão utilizando ZK.

3.8 - Sistema de gerenciamento de banco de dados (SGBD) e modelagem

Um SGBD é um programa de computador cujo objetivo principal é gerenciar o acesso e a correta manutenção dos dados armazenados em um banco de dados. Suas funções básicas consistem em: métodos de acesso, integridade semântica, segurança, concorrência e independência.

Para este trabalho, não se fez necessária a utilização de um SGBD robusto, completo, e com funcionalidades além das básicas. Desta forma, foi-se utilizado o SGBD *PgAdmin*, gerenciador de banco de dados *PostgreSQL*. Tais *softwares* atuando conjuntamente constituem uma ferramenta de manutenção de dados bastante simples e eficaz. As interfaces gráficas são bastante intuitivas, e sua licença é gratuita. Este conjunto de ferramentas é utilizado em várias soluções corporativas e empresariais, possuindo boa aceitação do mercado no geral.

Ainda se tratando de banco de dados, a modelagem da estrutura do banco foi feita utilizando-se o programa *PowerDesigner*, que também possui uma boa aceitação no mercado, com interface amigável, e com ferramentas de geração de documentação, engenharia reversa e conversão

entre *Structured Query Languages* (SQL) de vários fabricantes diferentes de bancos de dados. A figura 3.9 mostra a tela inicial do *PgAdmin*, e a figura 3.10 mostra um modelo genérico feito com a ferramenta *PowerDesigner*.

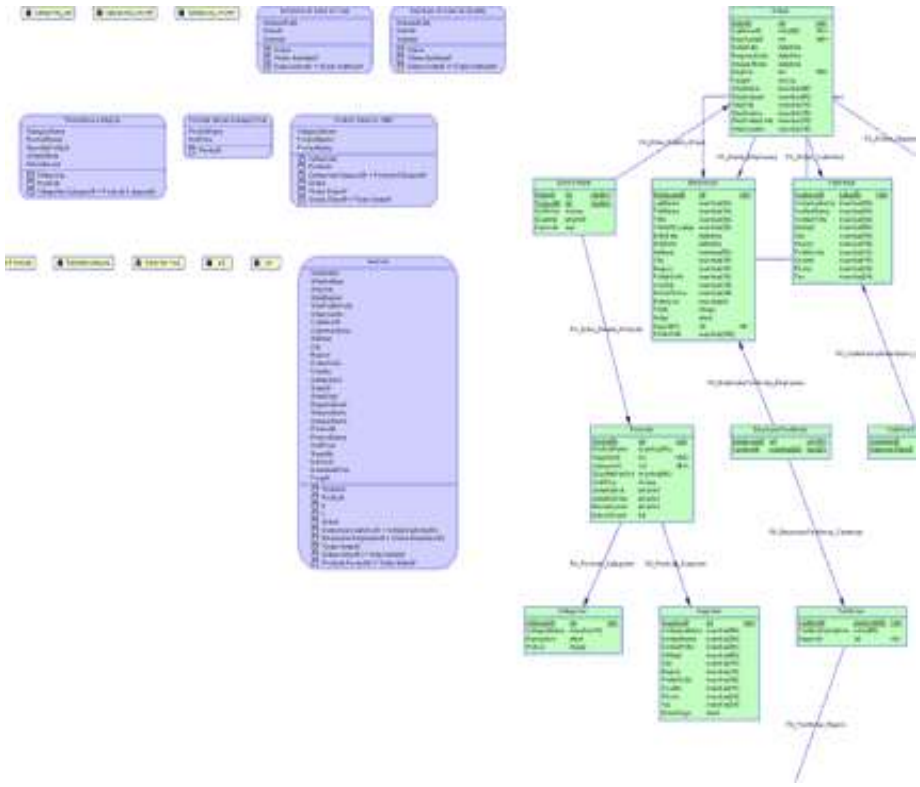


Figura 3.10: Modelo gerado no PowerDesigner

3.9 - Biometria por impressão digital

As biometrias de digitais são amplamente conhecidas como um método preciso de identificação e verificação biométrica. A maior parte dos sistemas de digitais um-paramuitos (1:n) e um-para-um (1:1) analisa pequenos atributos únicos na imagem da digital, que são conhecidos como minúcias. Elas podem ser definidas como os contornos das linhas papilares ou bifurcações (ramificações das linhas papilares). Outros sistemas de impressões digitais analisam os pequenos poros no dedo que, assim como as minúcias, são posicionados de forma única para diferenciar uma pessoa de outra. A densidade da imagem digital ou a distância entre as linhas papilares também podem ser analisadas.

Certas condições podem afetar as impressões de diferentes indivíduos. Por exemplo, sujeira, dedos secos ou rachados podem reduzir a qualidade da captura da imagem. Idade, sexo e etnia também podem impactar a qualidade das imagens digitais. A forma como um usuário interage com um *scanner* de digitais é outra consideração importante. Pressão muito forte na superfície do *scanner* , por exemplo, pode distorcer uma imagem. Os fornecedores estão trabalhando para

solucionar esses problemas, tanto que alguns *scanners* são ergonomicamente desenhados para otimizar o processo de captura de impressões digitais.

Uma diferença chave entre as várias tecnologias de digitais no mercado é a forma de captura da imagem. Sistemas de verificação um-para-um (1:1) usam quatro técnicas principais de captura: óptica, termal ou tátil, captação e ultra-som. Produtos um-para-muitos (1:n) capturam imagens digitais usando a técnica óptica ou por varredura eletrônica das imagens de um papel. No caso deste projeto, foi utilizada a verificação um-para-um da imagem digital, devido ao próprio contexto da aplicação. As leituras foram realizadas através de dispositivos de leitura de impressão digital com barramento universal (USB) *Microsoft Fingerprint Reader*. Tal escolha foi condicionada pela alta compatibilidade com a *applet* de leitura de impressão digital, devido à *drivers* já homologados e pela disponibilidade de tal leitor (Douglas Vigliazzi, 2006).

3.9.1 - Verificação da Imagem Digital

O processo de verificação um-para-um (1:1) é o seguinte:

(http://www.portalsaofrancisco.com.br/alfa/biometria/biometria-digital.php/ , acessado em 28/10/2011)

– Captura: A técnica de imagem óptica normalmente envolve a geração de uma fonte de luz, a qual é refracionada através de um prisma, em cuja superfície de vidro o dedo é colocado A luz brilha na ponta do dedo e a impressão é feita pela imagem do dedo que é capturada. Técnicas táteis ou termais utilizam uma sofisticada tecnologia de *chip* para conseguir os dados da imagem digital. O usuário posiciona um dedo no sensor que sentirá o calor ou a pressão do dedo e o dado será capturado;

– Sensores de captação medem as cargas elétricas e dão um sinal elétrico quando o dedo é colocado em sua superfície. O elemento chave da técnica de captação, assim como dos métodos táteis e termais, é o sensor. Usando a captação, as mínimas elevações e aprofundamentos das linhas papilares e os vales na ponta do dedo são analisados. Um sinal elétrico é dado quando as linhas papilares entram em contato com o sensor. Nenhum sinal é gerado pelos vales. Essas variações na carga elétrica produz a imagem digital;

– A captura de imagem por ultra-som utiliza ondas de som abaixo do limite de audição humano. Um dedo é colocado no *scanner* e ondas acústicas são usadas para medir a densidade do padrão da imagem digital;

– Extração: O equipamento biométrico extrai os dados contidos na imagem digital. Uma representação matemática única é então armazenada na forma de um *template;*

– Comparação: Durante o processo de comparação, um novo exemplo é comparado com o *template;*

– Dependendo da base que está configurada para a aplicação, pode-se obter um par como resultado ou não.

Os aspectos tecnológicos abordados neste capítulo serão utilizados em todo o desenvolvimento do projeto. Os mesmos são utilizados como referência e serão citados nos itens de implementação dos próximos capítulos.

CAPÍTULO 4 - MODELO PROPOSTO

Neste capítulo são apresentadas a proposta de solução para o problema de inconsistência no processo de realização de chamada e o passo a passo do desenvolvimento, assim como os requisitos para criação dos ambientes necessários.

4.1 - Apresentação Geral do Modelo Proposto

O projeto consiste, basicamente, na implementação de uma solução que possibilite o cadastro, recuperação e leitura de impressões digitais com fins de realização de chamadas em instituições de ensino. Para isto, dois fluxos de ações são necessários: o fluxo de cadastro de alunos e o fluxo de realização de chamada. A figura 4.1 mostra o modelo do projeto. A seguir são descritos os passos do desenvolvimento do projeto.

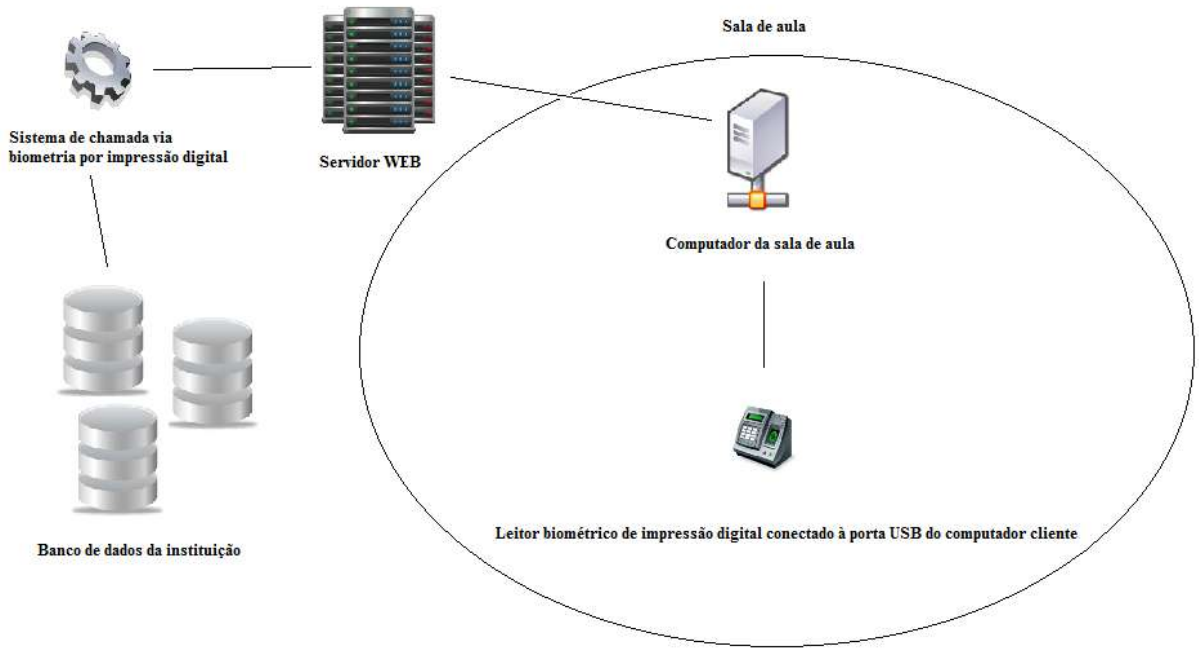


Figura 4.1: Modelo do projeto

4.1.1 - Fluxo de cadastro de alunos

Este fluxo consiste na operação a ser realizada no momento de registro da pessoa como aluno integrante da instituição de ensino. Tal operação inicia-se com o acesso à tela de manutenção de alunos, existente no sistema definido por este projeto. Posteriormente, deve-se preencher os dados relativos ao cadastro do aluno, respeitando-se os campos obrigatórios e, por fim, realizar a leitura de impressão digital do mesmo, com a finalidade de se manter uma cópia binária salva no banco de dados para posteriores consultas e validações. A figura 4.2 mostra o fluxo de cadastro de alunos através do sistema.

Figura 4.2: Fluxo de cadastro de alunos

4.1.2 - Fluxo de realização de chamada

Este fluxo consiste na operação de realização de chamada em si. Com os alunos cadastrados e matriculados nas respectivas turmas, o professor deve acessar o sistema utilizando seu *login* e senha com o fim de realizar a chamada da aula corrente. O sistema automaticamente carrega a aula corrente de acordo com as disciplinas ministradas pelo professor, dia da semana e hora. O professor deve selecionar a aula que deseja realizar a chamada e habilitar tal operação. Os alunos devem posicionar o dedo utilizado no registro de impressão digital no(s) leitor(es) até que seus nomes tenham sido retirados da lista de alunos faltantes, mostrada na tela do sistema. O sistema fará a leitura das impressões digitais informadas, comparando, validando e limpando a lista de alunos faltantes enquanto a chamada estiver habilitada. Tendo todos os alunos presentes registrado seu comparecimento, o professor deve finalizar a chamada, terminando o processo. A figura 4.3 mostra o fluxo de realização de chamada através do sistema.

No próximo capítulo é apresentado o modelo na sua aplicação prática, o que encoraja sua implementação em uma instituição de ensino, mostrando sua viabilidade. O desenvolvimento até a demonstração de funcionamento além de dar-se em caráter acadêmico, como requisito para a conclusão do curso de engenharia de computação, também apresenta caráter de produto completo, podendo vir a ser utilizado por instituições de ensino.

CAPÍTULO 5 - APLICAÇÃO PRÁTICA DO MODELO PROPOSTO

Neste capítulo serão explanados aspectos relativos à aplicação do projeto, dentro de um modelo real, com o objetivo de se constatar a viabilidade do produto final, bem como permitir que novos conhecimentos sejam incorporados.

5.1 - Apresentação da área de Aplicação do modelo

A área de aplicação do modelo é acadêmica, abrangendo aulas normais ou outras atividades que suportem verificação de presença por parte de alunos. As condições para aplicação de tal modelo são puramente técnicas, e serão abordadas neste capítulo.

5.1.1 - Relevância para a Engenharia de Computação

Este projeto encontra sua relevância para a engenharia de computação no sentido de constituir uma inovação tecnológica que se baseia em conceitos vastamente estudados e explorados no decorrer do curso. Disciplinas como redes, linguagens e técnicas de programação e arquitetura de computadores foram de total relevância para a implementação deste projeto e a obtenção dos resultados esperados. Além disso, tal projeto dá margem para futuros estudos referentes à infraestrutura tecnológica a ser adotada na implantação do mesmo, contribuindo, assim para a engenharia.

5.2 - Descrição da Aplicação do Modelo

5.2.1 - Descrição das Etapas do Modelo

O modelo do projeto consiste, essencialmente, em operações realizadas através do sistema com a finalidade de se proporcionar a realização da chamada via impressão digital. As operações identificadas no modelo são as de: acesso, cadastro (incluindo leitura de impressão digital e administração), e realização de chamada de fato.

5.2.1.1 - Acesso

O acesso ao sistema se dá através de computadores cliente que devem possuir navegadores instalados, como, por exemplo, *Internet Explorer, Mozilla Firefox*, *Google Chrome* e *Safari*. Tais clientes acessam o servidor onde está devidamente instalado o sistema por meio de chamadas via estes navegadores. Essas requisições são processadas pelo servidor, que envia as respostas aos clientes, caracterizando uma comunicação cliente servidor, porém dentro do contexto da *web*, visto que o tráfego de dados se dá a partir de máquinas diferentes que se comunicam através da rede. Por se tratar de um sistema desenvolvido na linguagem *Java*, deve-se possuir o ambiente *Java* devidamente instalado nas máquinas cliente (JRE) para que tal acesso possa ser concretizado. A aplicação deve estar rodando em um servidor que possua um contâiner *web*, como o *Tomcat 6*, e o acesso a este servidor deve estar liberado para as outras máquinas da rede.

Tendo o cliente conseguido acessar o servidor de maneira bem sucedida, o mesmo deve informar o seu *login* e senha para, de fato, acessar o sistema, conseguindo permissão sobre as funcionalidades do mesmo. A figura 5.1 mostra a tela inicial do sistema, sendo acessada através do navegador *Google Chrome*.

5.2.1.2 - Cadastro, Leitura de Impressão Digital e Administração

Tendo o usuário concretizado seu acesso ao sistema, vários menus de funcionalidades serão disponibilizados ao mesmo para que os devidos cadastros sejam realizados, com a finalidade de se constituir uma base de informações que serão posteriormente consultadas e manipuladas no decorrer da utilização do sistema. O menu de cadastro é composto das opções: instituição de ensino, faculdade, curso, disciplina, turma, professor e aluno, respectivamente. A figura 5.2 mostra o menu de cadastro do sistema.

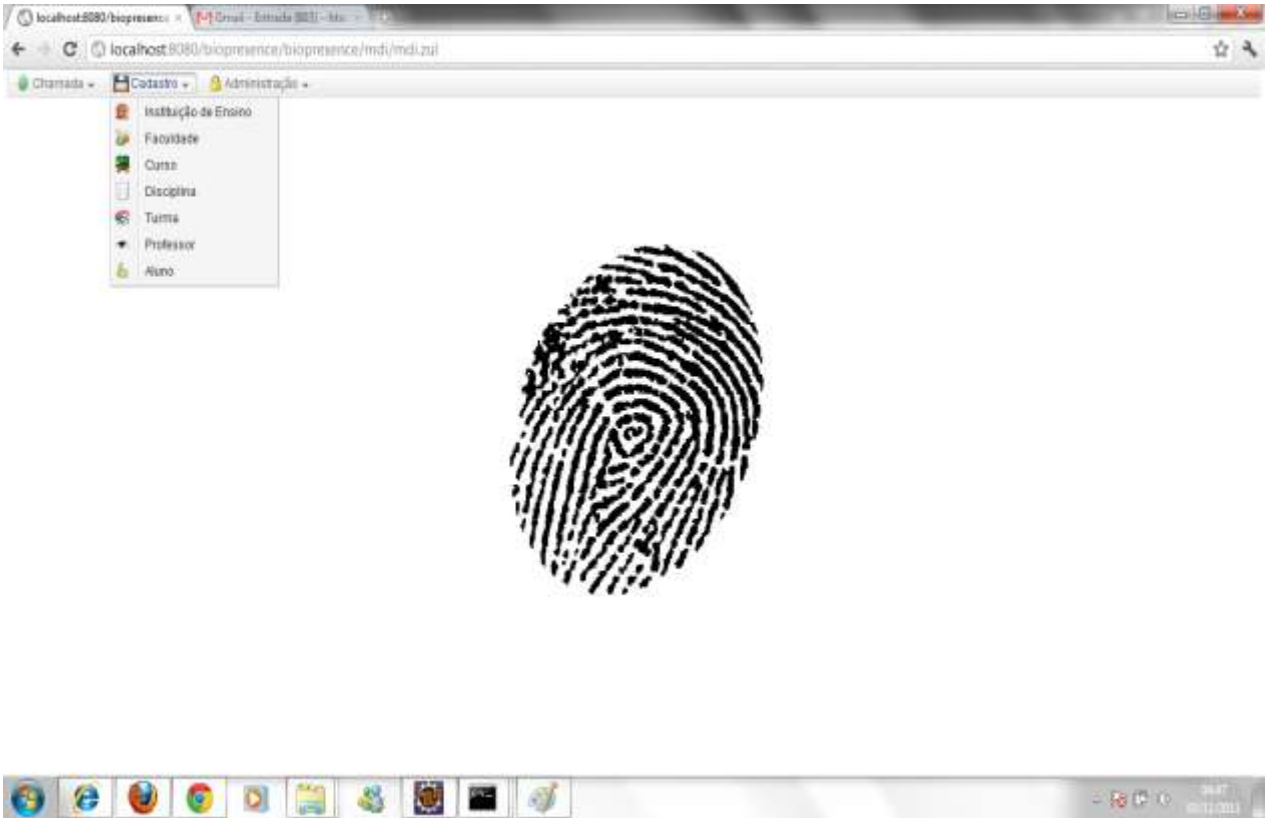


Figura 5.2: Menu de cadastro do sistema

Cada opção do menu de cadastro dá ao usuário responsável a possibilidade de se manter os registros das entidades que são utilizadas pelo sistema. Através do mesmo, é possível manipular as turmas em que os alunos estão matriculados, quais disciplinas os professores serão responsáveis por ministrar e quais instituições de ensino serão abrangidas pelo sistema, por exemplo. As dependências entre cada entidade podem ser ilustradas através do modelo de dados feito utilizando-se a ferramenta *PowerDesigner* e que, posteriormente, foi importado através do SGBD *PgAdmin* do banco de dados *PostgreSQL*. A figura 5.3 mostra a modelagem do banco de dados feito através da ferramenta *PowerDesigner*. Para informações adicionais acerca do modelo, ver Apêndice A – RELATÓRIO FÍSICO DO MODELO DE DADOS.

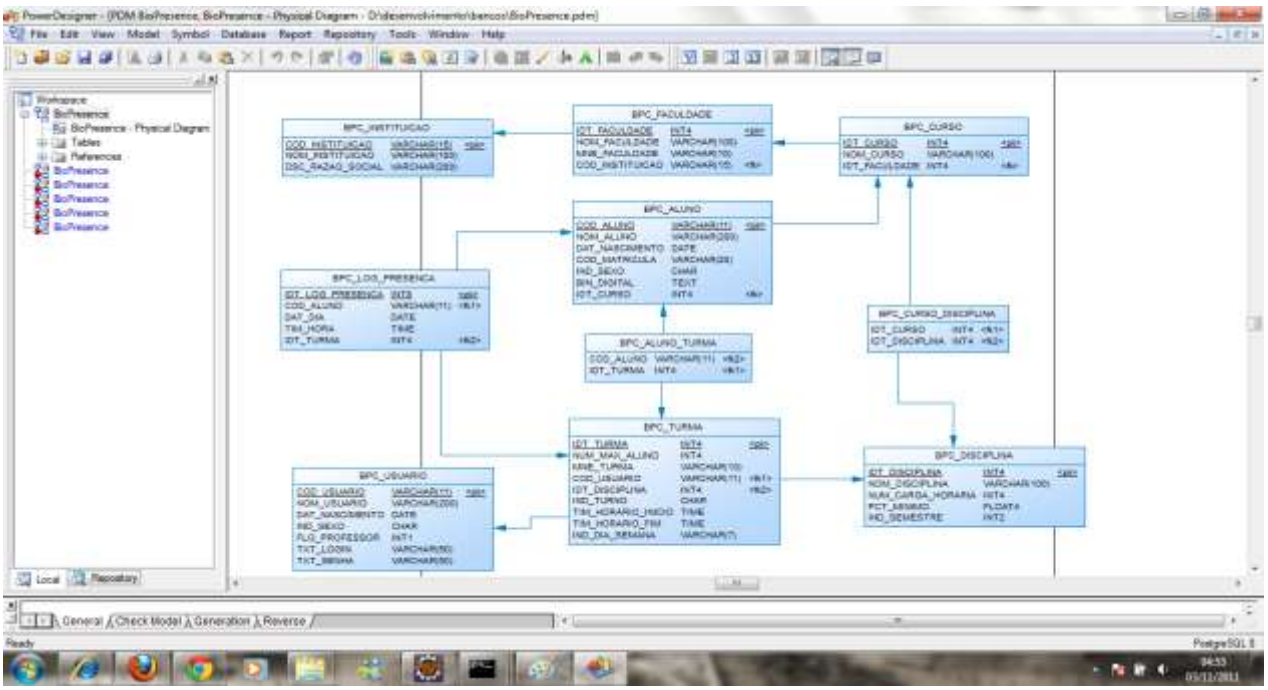


Figura 5.3: Modelo de dados do sistema feito utilizando-se o programa PowerDesigner

Todo o sistema foi desenvolvido sob o padrão arquitetural *Model, View, Controller* (MVC), onde são definidas camadas delimitadoras de atuação e comunicação entre os objetos *Java*.

Na camada de modelo (*model)*, existem os objetos que proporcionam a comunicação de fato com o banco de dados. Tais objetos são acessados com a finalidade de se realizar operações sob o banco de dados, como uma consulta, por exemplo. O acesso ao banco de dados através da aplicação se dá a partir da API *Java Database Connectivity (JDBC)*, que consiste em um conjunto de classes que servem como ponte entre a aplicação e o banco de dados, utilizando os respectivos *drivers* e os dados de conexão do mesmo para estabelecer tal comunicação. Também nesta camada existem os objetos que representam os dados em si. Estes objetos são populados com as informações existentes nos bancos de dados e são manipulados pela aplicação de acordo com objetivos específicos, atuando como pivôs entre a aplicação e o banco. A figura 5.4 mostra um exemplo de objeto de acesso a dados de uma tabela no banco de dados.

Na camada de visão (*view)*, existem os objetos que proporcionam todo o arcabouço de *interface*s gráficas do sistema. É nesta camada que se situa o *framework Zkoss* utilizado para compor todo o ambiente gráfico do sistema, oferecendo ao usuário a melhor visualização dos dados possível dentro do contexto *web*. Vale lembrar que toda a parte gráfica do sistema pode ser editada e customizada através de tecnologias de *Cascading Style Sheets* (CSS), visto que o *Zkoss* suporta objetos HTML e possui uma gramática de conversação própria com os mesmos. A figura 5.5 mostra o código de uma página do sistema.

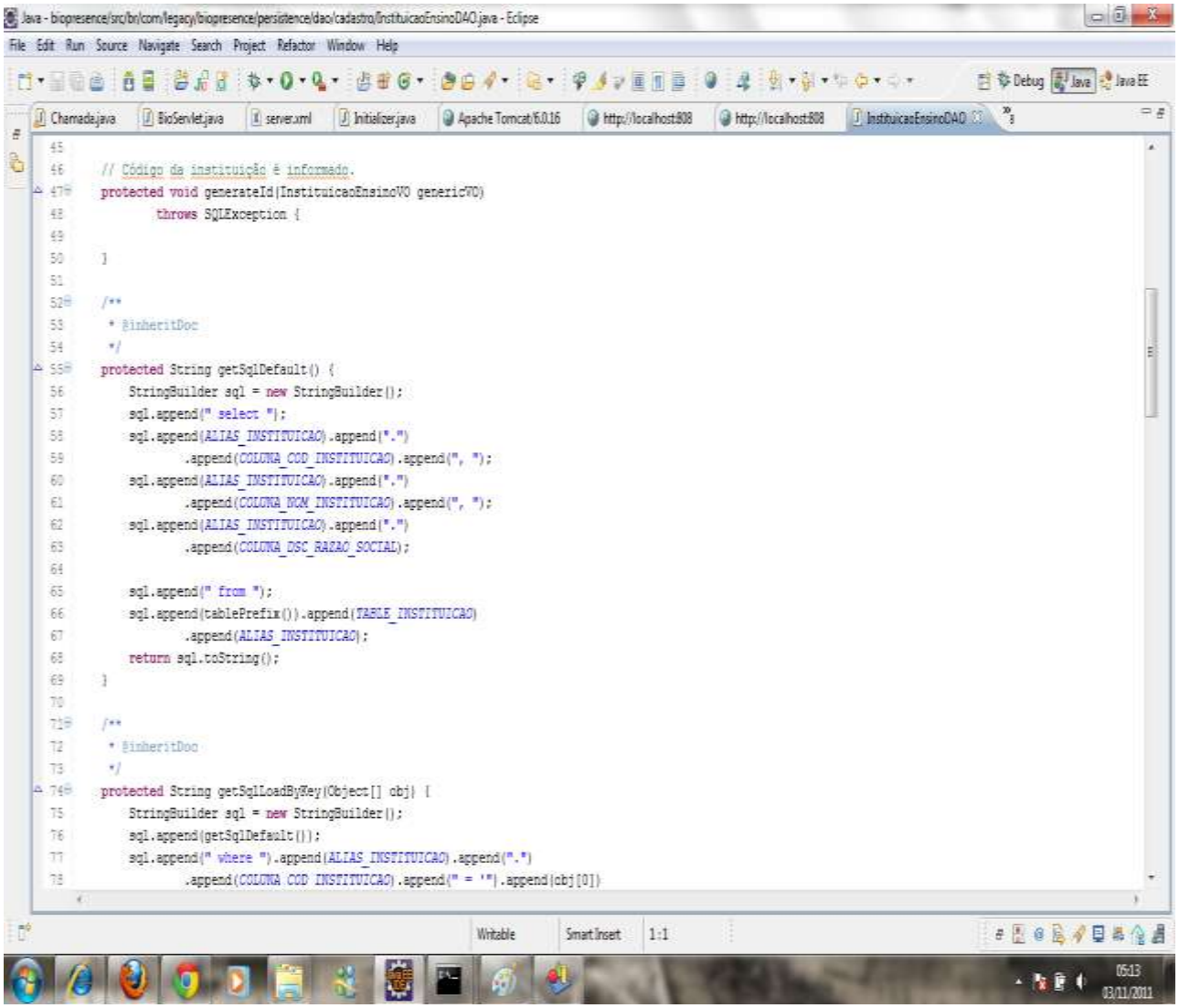
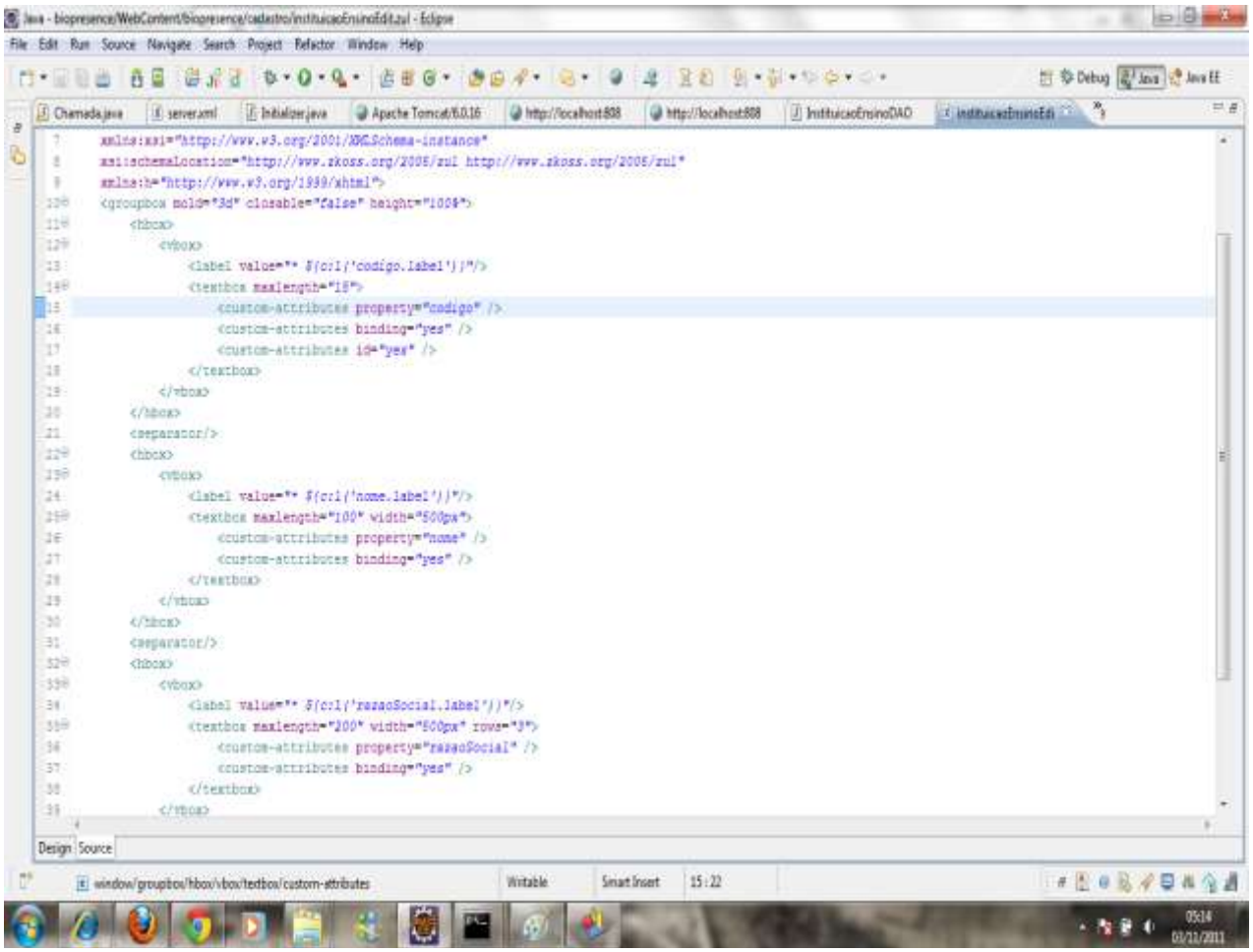


Figura 5.4: Exemplo de objeto que é responsável pela interação com a tabela de Instituição de Ensino e geração de objetos Instituição de Ensino.



Por fim, na camada de controle (*controller)*, existem todas as classes *Java* que atuam como manipuladoras dos objetos, orquestrando todo o funcionamento do sistema via codificação. É nesta camada que é definido o funcionamento do sistema, assim como a implementação dos mecanismos utilizados pelo mesmo na realização de suas operações. A camada de controle consiste, basicamente, no motor que dá a operacionalização ao sistema. É nesta camada também que se encontram definições de pequenas aplicações atuando paralelamente ao sistema, como *applets* e *servlets*. Um *applet* de leitura de impressão digital foi utilizado no sistema para captura e validação de tal leitura, enviando estas informações ao *servlet*. Um *servlet* foi escrito apenas para intermediar a comunicação entre cliente e servidor, processando *cookies* responsáveis por carregar uma *string* binária com os dados da impressão digital do usuário em questão. A figura 5.6 mostra o código de uma classe manipuladora de uma página do sistema.

A parte administrativa do sistema consiste apenas em algoritmos de mudança e recuperação de senha, e *logoff*. Tais algoritmos foram implementados baseados nos conceitos previamente expostos e constituem funcionalidades dispensáveis para o objetivo final do sistema. Contudo, proporcionam maior qualidade de navegação ao usuário. A figura 5.7 apresenta a classe principal da *applet* de leitura de impressão digital. A figura 5.8 apresenta o código da *servlet* responsável por intermediar a comunicação entre a *applet* e o sistema. A figura 5.9 apresenta a visão do menu de administração do sistema.

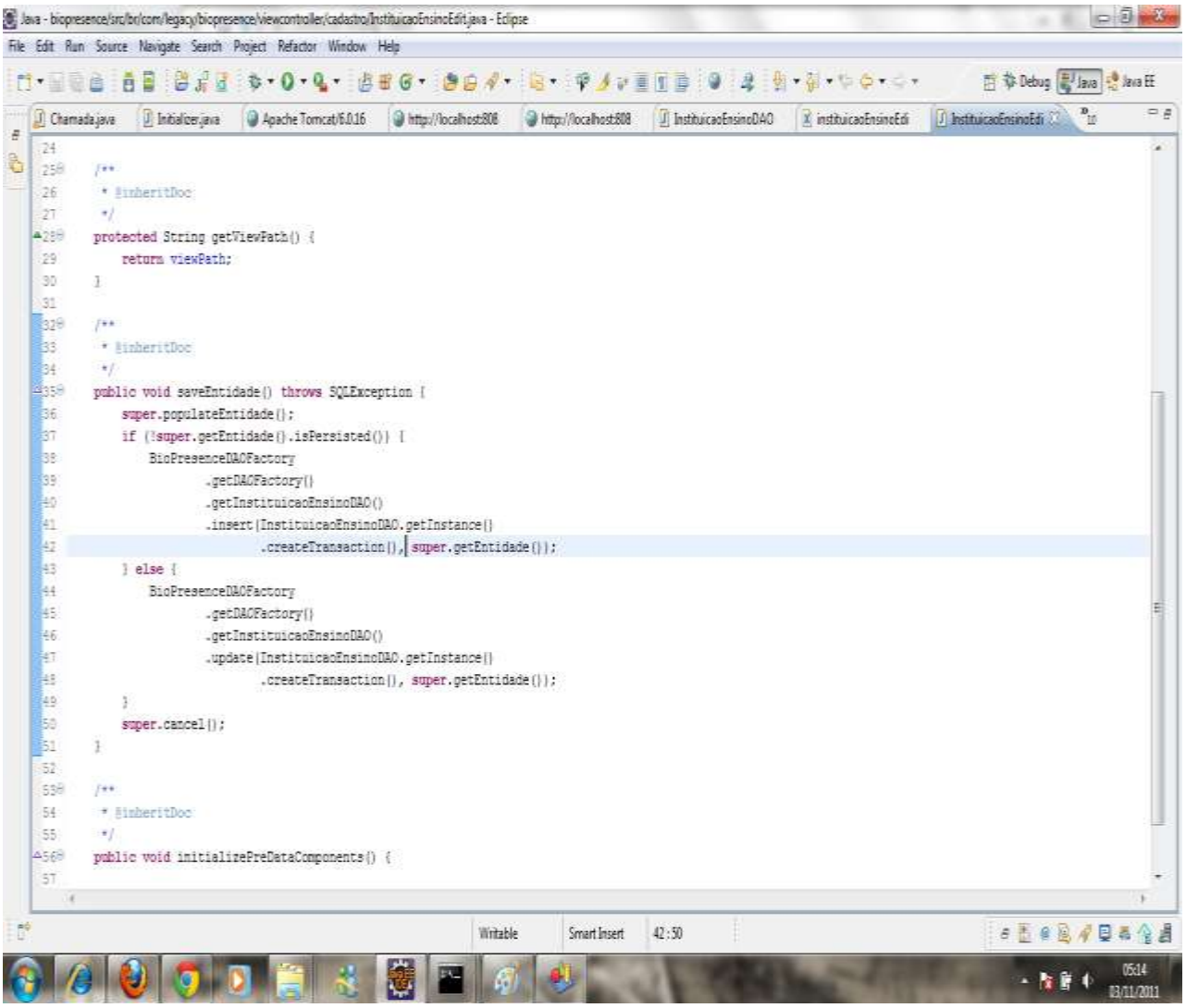
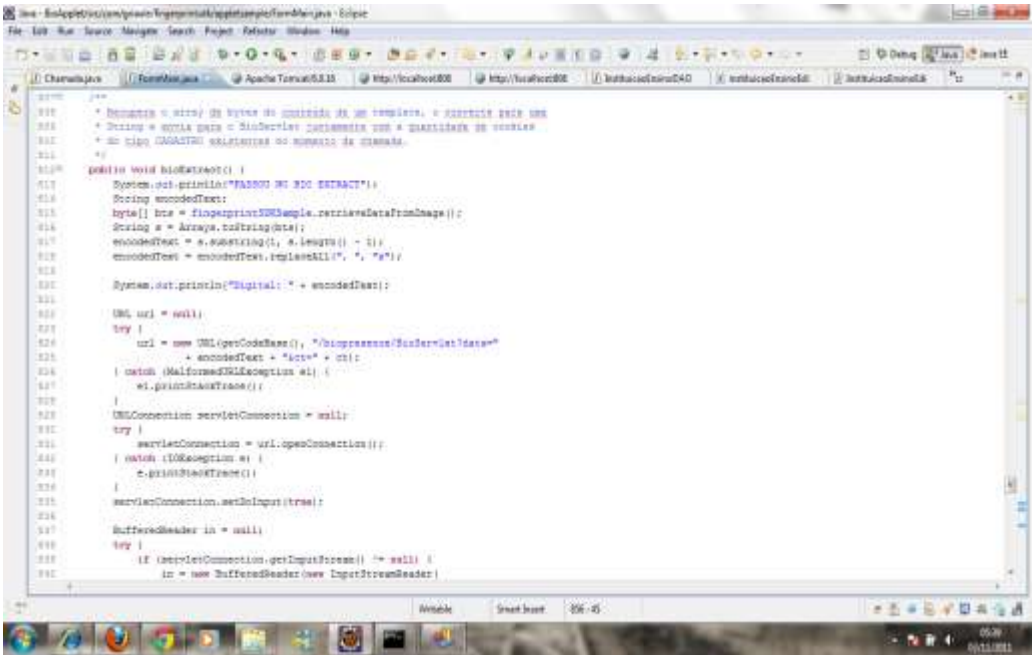


Figura 5.6: Classe manipuladora de uma página do sistema.



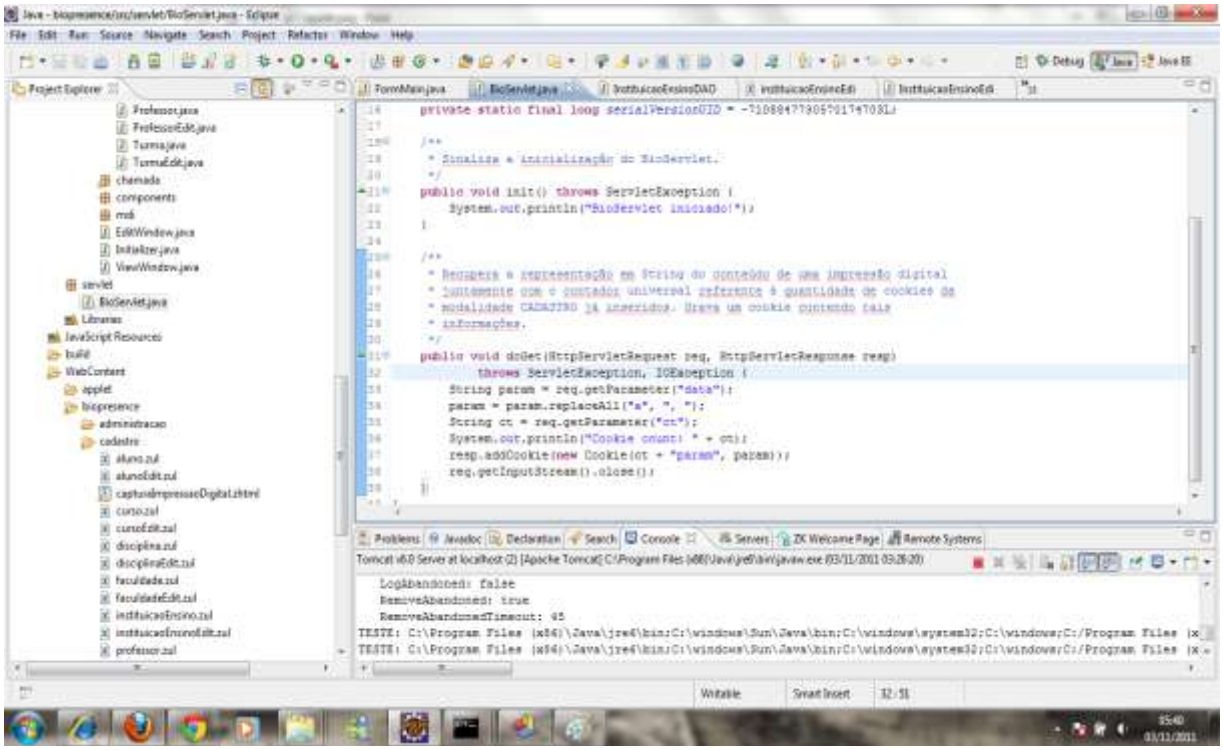


Figura 5.8: Servlet escrita para intermediar a comunicação entre applet e o sistema.

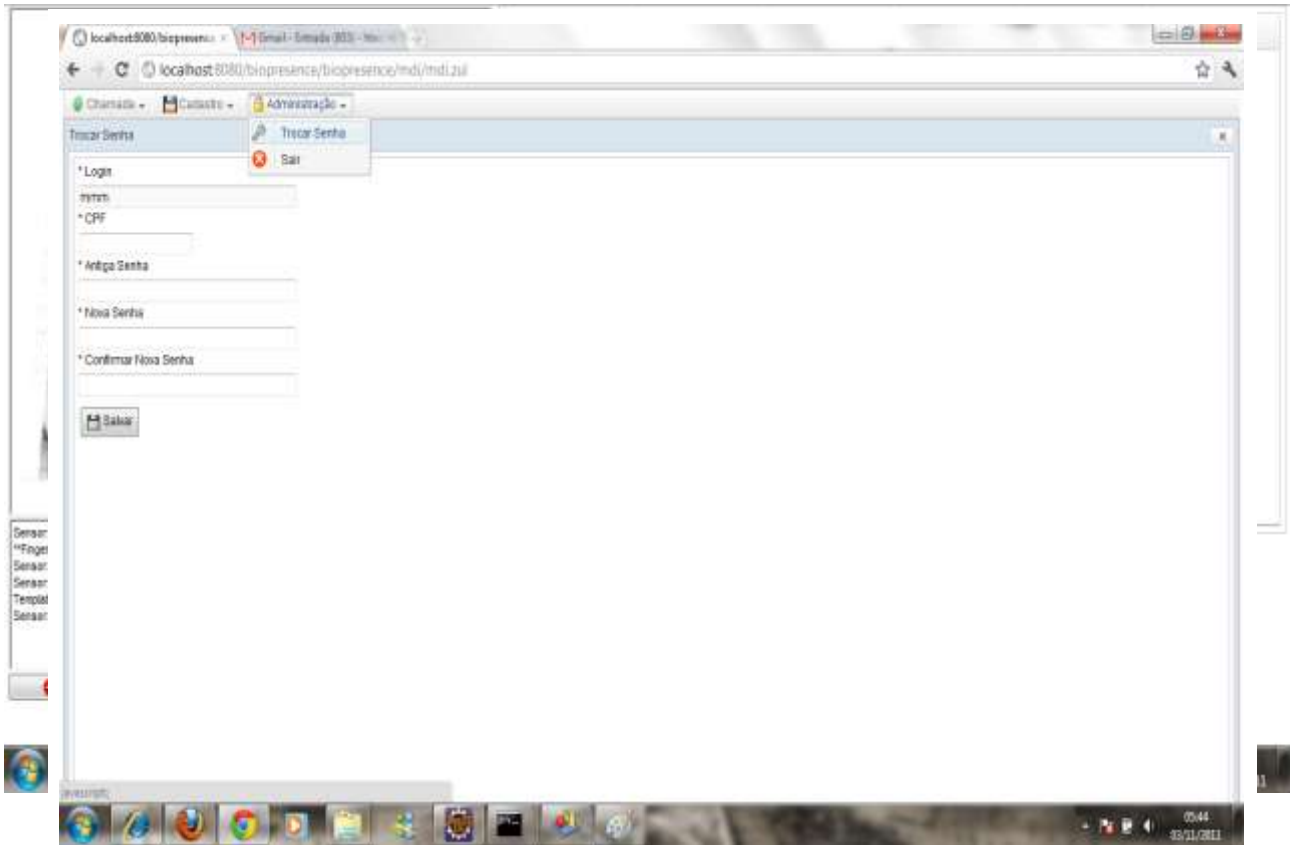


Figura 5.9: Visão do menu de Administração e da tela de troca de senha

5.2.1.3 - Realização da chamada

O processo de realização de chamada se inicia com o acesso, por parte do professor, a tal funcionalidade existente no sistema. Após selecionar a turma em que será feita a chamada, o sistema ativa o(s) leitor(es) de impressão digital para que os mesmos possam receber as imagens das impressões digitais dos alunos. A partir dessas imagens, o sistema vai comparando as *strings* binárias das mesmas com as salvas no banco de dados no momento de cadastro de cada aluno. A leitura é feita através do *applet* de leitura de impressão digital e a troca de valores para a leitura e resultados de comparação é feita através do *servlet* codificado. A partir dos resultados destas comparações, a presença dos alunos vai sendo registrada no banco de dados. Quando o professor perceber que nenhum novo registro será recuperado, visto que as impressões digitais de todos os alunos presentes já foram identificadas pelo sistema, o mesmo finaliza o processo de realização de chamada. A figura 5.10 mostra a tela de realização de chamada.

5.2.2 - Descrição da Implementação

5.2.2.1 - Ambiente do Cliente

O processo de configuração do ambiente nas máquinas cliente é composto, basicamente, de duas etapas. A primeira delas, que já foi enumerada anteriormente, consiste na instalação do ambiente Java (JRE). Este processo é realizado através do *download* e posterior execução de um *setup* disponível no próprio *site* da *Oracle* para tal finalidade. Vale ressaltar que até a corrente versão, a *applet* de leitura de impressão digital utilizada só possui compatibilidade com JRE's de 32 *bits*. A segunda e última etapa consiste na instalação do *driver* para o respectivo leitor de impressão digital utilizado. No caso deste o projeto, o leitor utilizado foi o *Microsoft Fingerprint Reader* e seu *driver* é consiste no arquivo *Fingerprint_SDK_Java_2009_Installer.jar*. Tal arquivo é disponibilizado no *site* da própria empresa fornecedora da API de leitura de impressão digital, *Griaule Biometrics*. Após instalada a JRE, deve-se abrir tal arquivo utilizando-se a JVM. Feito isso, basta proceder com a instalação do mesmo. Algumas versões de sistema operacional possuem restrição quanto a execução de *drivers* de plataforma 32 *bits*. Para desativar tal restrição, deve-se forçar o desligamento da mesma no momento de inicialização do sistema operacional. A figura 5.11 mostra a tela inicial de instalação da JRE 32 *bits*. A figura 5.12 mostra a interface para execução do instalador do *driver* do leitor de impressão digital. A figura 5.13 mostra a instalação do *driver* do leitor de impressão digital.

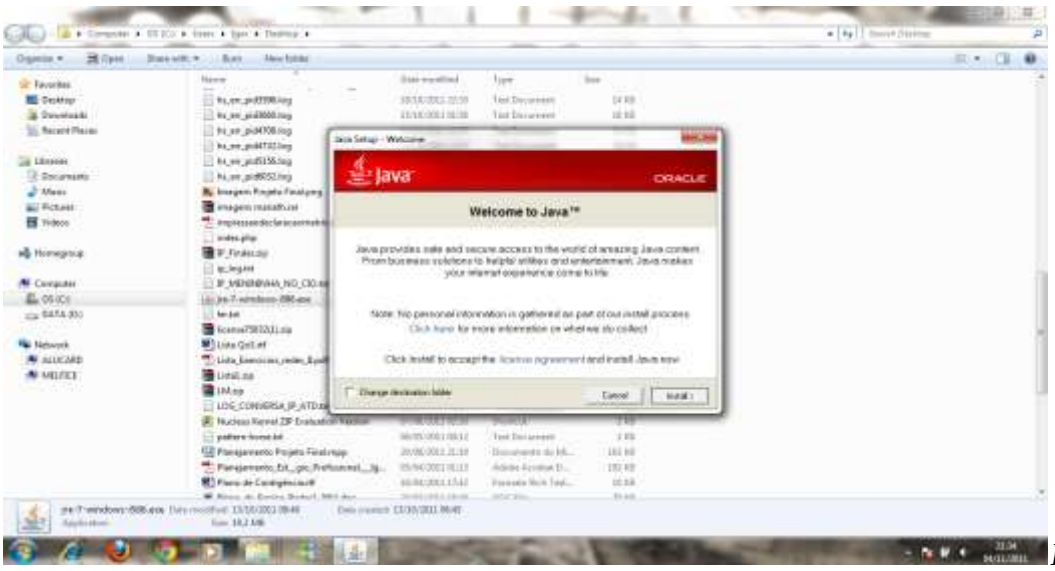


Figura 5.11: Instalação da JRE 32 bits.

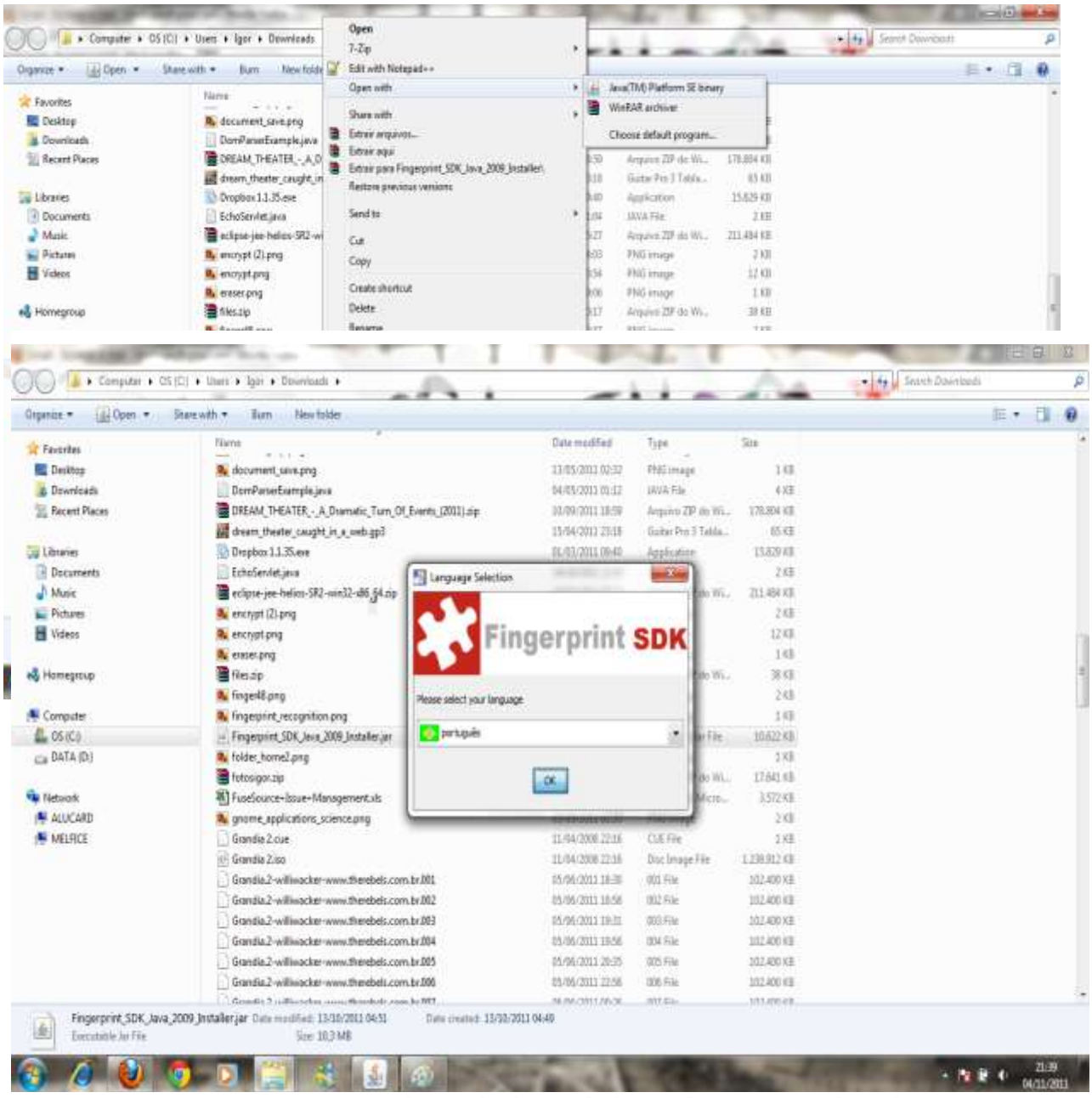


Figura 5.13: Instalação do driver do leitor de impressão digital

5.2.2.2 - Ambiente do Servidor

O processo de configuração do ambiente da máquina servidor é constituído basicamente da instalação do contâiner *web*. No caso deste projeto, o contâiner utilizado foi o *Apache Tomcat 6*. Para instalar o *Apache Tomcat 6,* é necessário apenas fazer o *download* do instalador no próprio site da *Apache* e executá-lo. No decorrer da instalação, algumas configurações poderão ser ajustadas, como a porta de acesso do serviço do *Apache Tomcat*, por exemplo. Concluída a instalação, resta apenas copiar o arquivo da aplicação para a pasta *webapps* localizada no diretório de instalação do *Tomcat.* Lembrando que para acessar a aplicação via URL, o serviço do *Tomcat* deve estar inicializado. Isto pode ser feito facilmente, executando-se o aplicativo *tomcat6.exe* localizado na pasta *bin* do diretório de instalação do *Tomcat*. A figura 5.14 mostra o *Apache Tomcat 6* sendo inicializado.

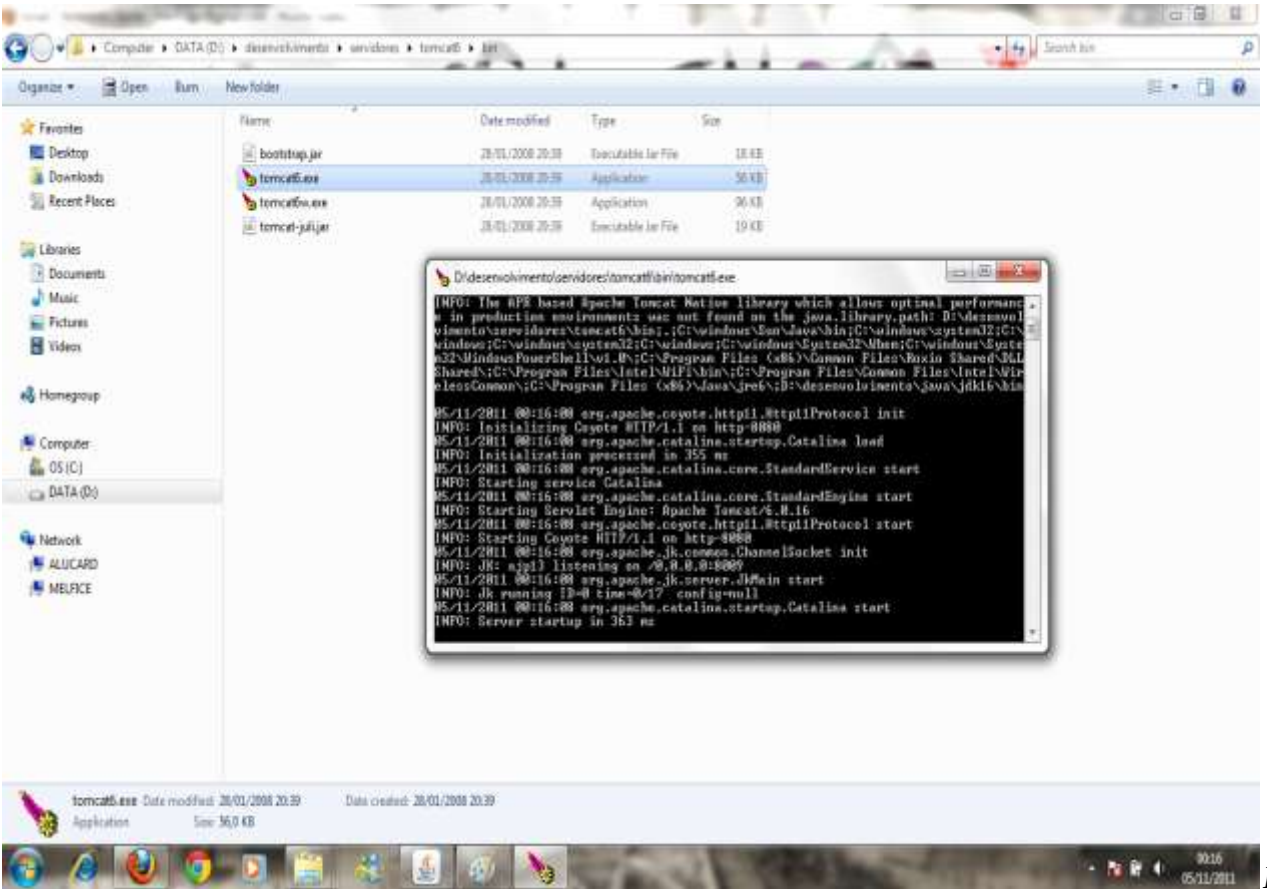


Figura 5.14: Inicializando o Apache Tomcat 6 no Windows 7 Home Premium

5.2.2.3 - DAO

Um *Data Access Object* (DAO) é um objeto responsável por realizar e controlar as operações transacionais com o banco de dados. É o DAO que se comunica com uma ou mais tabelas do banco, recuperando/persistindo os dados relevantes para o devido armazenamento e posterior consulta/disponibilização dos mesmos em formas de objeto. Essa comunicação entre DAO e banco e dados é feita através de APIs do Java. Neste projeto foi adotado a metodologia JDBC para comunicação com banco de dados, através de uma arquitetura própria e autônoma., incluindo a definição de um *pool* de conexões. Desta forma, foram codificados vários DAOs no decorrer do desenvolvimento para orquestrar a interação entre aplicação e banco de dados. A figura 5.15 mostra o código de uma classe responsável por interagir com uma determinada tabela do banco de dados.

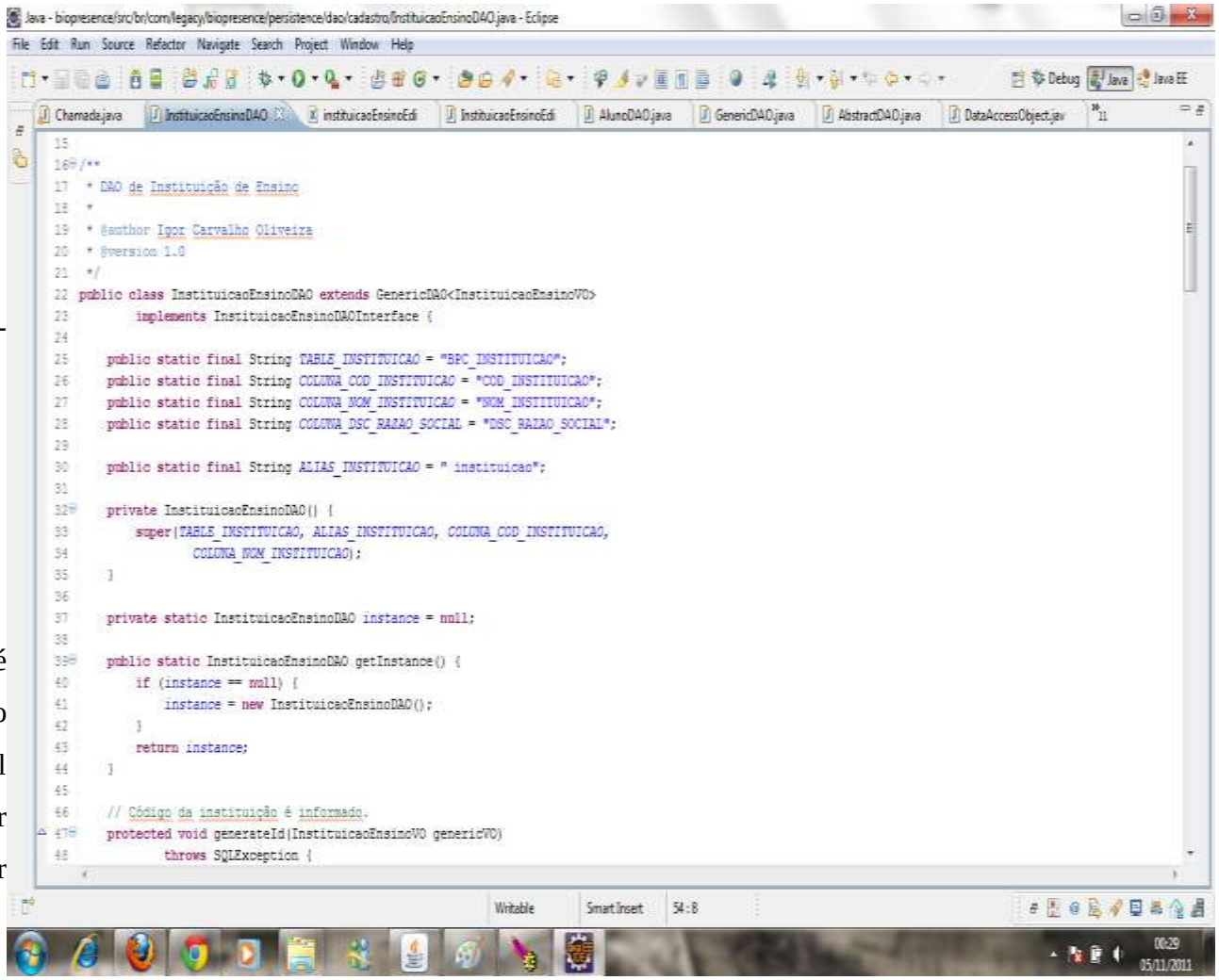


Figura 5.15: DAO de Instituição de Ensino.

5.2.2.4 - *Factory*

Dentro do contexto de desenvolvimento, uma *factory* é literalmente uma fábrica de objetos. Tal padrão de desenvolvimento foi adotado na camada de modelo com a finalidade de se centralizar a criação de DAOs, com a finalidade de se ter um código mais organizado e também para proporcionar o reaproveitamento de DAOs já criados. Desta forma, quando se desejar realizar

qualquer operação com o banco de dados, ao invés de instanciar um DAO por operação, a *factory* verifica se já existe um DAO criado do tipo desejado. Caso exista, tal DAO é disponibilizado para a camada de controle, a mesma que realiza a operação com o banco de dados em questão. A figura 5.16 mostra o código de uma *factory* do sistema.

5.2.2.5 - *Create, Recover, Update* e *Delete* (CRUD)

O modelo CRUD foi adotado em todas as telas de cadastro da aplicação. Tal modelo é utilizado em sistemas que visem manter, em parte ou na sua totalidade, registros de entidades físicas, como pessoas, instituições e empresas. O modelo consiste na manutenção dos registros de uma determinada entidade no sentido que de os mesmos possam ser criados, recuperados, atualizados e excluídos através de operações feitas no sistema. Essas operações são realizadas sob o banco de dados, que é responsável por manter as informações de tais registros.

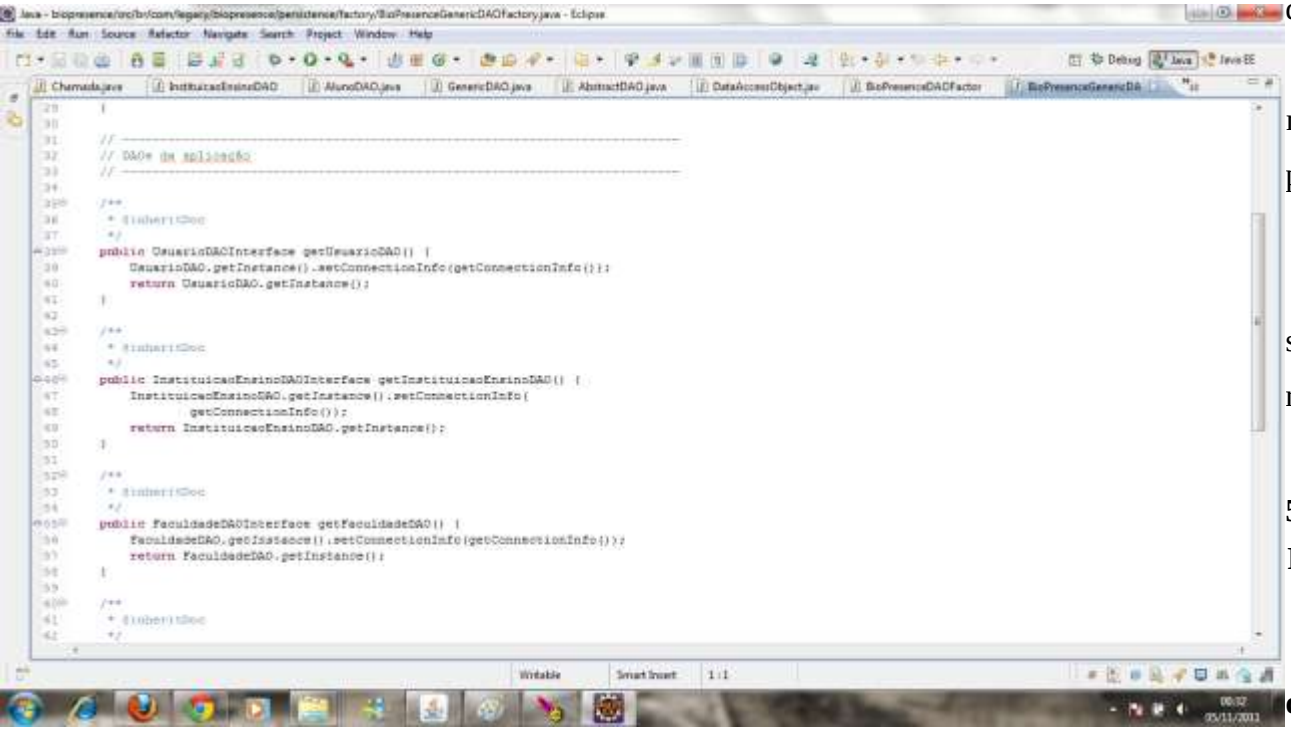


Figura 5.16: Factory do sistema.

5.3 - Resultados da Aplicação do Modelo

Na aplicação do modelo, foi realizado o cadastro das impressões digitais de 20 pessoas e depois leitura/comparação, simulando o atendimento à chamada. As comparações das impressões digitais são coerentes com a realidade, possuindo uma taxa de acerto de, em média, 95%, conforme mostra a tabela 5.1. Também foi realizado um teste de negatividade com estas mesmas 20 pessoas, visando analisar a possibilidade de alguém burlar o procedimento de comparação de impressões

digitais, se passando por outra pessoa. Tal teste de negatividade não obteve nenhuma ocorrência de validação incoerente de impressões digitais. Todavia, uma questão que foi constatada foi a da definição de um tamanho limite de pacote trafegado sob o protocolo AJP no *Tomcat*, onde, dependendo do tamanho da digital capturada, pode haver o estouro deste valor. Desta forma, é importante elevar tal valor limite nas configurações do *Tomcat* para que não haja nenhum imprevisto.

Leitura/comparação bem sucedida	Leitura/comparação mal sucedida
19	1
Total de pessoas: 20	
Taxa: 95%	

Tabela 5.1: Resultados de testes de leitura e comparação de impressão digital

As imprecisões e incoerências das leituras de impressão digital possuem sua causa no mal posicionamento do dedo no leitor e em possíveis sujeiras existentes no dedo da pessoa, não abrangendo o algoritmo estatístico utilizado pela *applet*. A figura 5.17 mostra o sistema sendo acesso através de um *notebook*, atuando tanto como servidor como cliente.

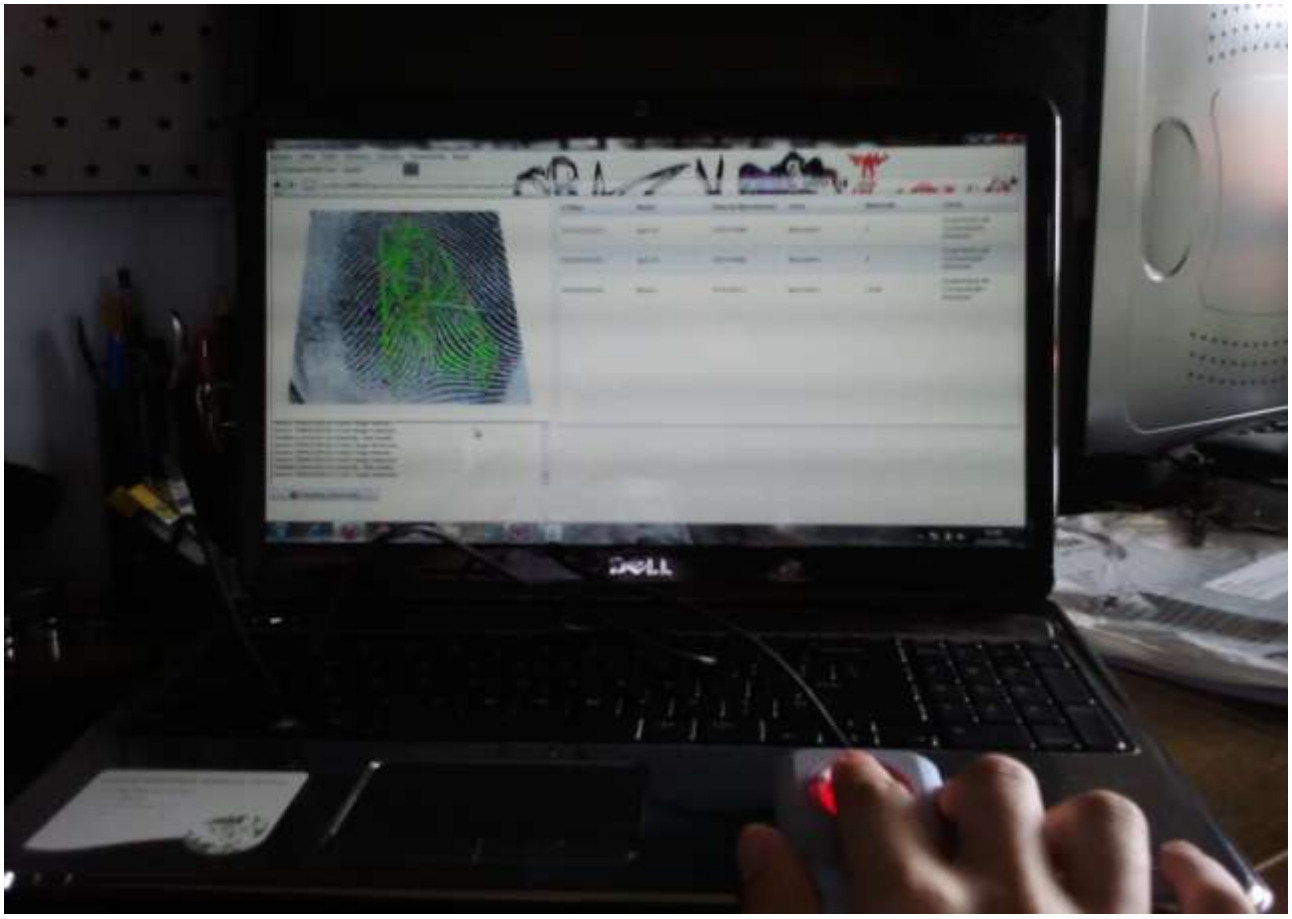


Figura 5.17: Sistema em execução

5.4 - Custos do modelo proposto

Os custos demandados por este projeto se constituem da necessidade de se comprar leitor(es) de impressão digital e licença(s) de utilização da *applet* da empresa *Griaule Biometrics*. O leitor de impressão digital *Microsoft Fingerprint Reader*, utilizado neste projeto, custa em torno de 90,00 reais. Uma licença de utilização da *applet* custa em torno de 95 dólares, o que representa, aproximadamente, 165 reais.

Assim, o custo base total do projeto é de 255 reais, podendo somente aumentar, devido à quantidade de leitores que serão utilizados na instituição e à necessidade de múltiplas licenças ou de uma licença mais cara, visto que a mesma, dentro do contexto *web*, tem seu valor pautado na quantidade média de acessos.

Em relação aos leitores de impressão digital, pode-se reduzir custos proporcionais comprando-se em grande escala. Desta forma, o valor unitário de cada leitor diminui. Em relação à

licença de utilização da *applet*, tal questão deve ser acordada com a empresa *Griaule Biometrics*. Como não existe nenhum outro projeto e implantado nesse sentido, não é possível a comparação com outros preços.

5.5 - Avaliação Global do Modelo

O modelo apresentado é aplicável, simples e flexível. Aplicável, pois o sistema é bastante consistente. Simples, pois é composto, além da parte de *software*, apenas de leitor(es) de impressão digital e licença(s) de utilização da *applet*. E flexível, pois a própria instituição que venha a implantar este projeto pode escolher quantos leitores deseja utilizar.

A implantação do trabalho realizado apresenta ganhos no que diz respeito à velocidade e coerência. A realização do processo de chamada através de um sistema de leitura biométrica se mostra menos demorada do que processo normal, utilizado atualmente, visto que todos os alunos poderão responder a tal chamada simultaneamente, resultando já na posterior computação das mesmas. Em relação ao aspecto da coerência, todas as situações problemáticas enumeradas no Capítulo 2 passam a ser solucionadas com este processo sendo realizado através de leitura de impressão digital.

CAPÍTULO 6 - CONCLUSÃO

6.1 - Conclusões

Após implementação da solução proposta por este projeto, constatou-se que tanto os objetivos gerais quanto os específicos foram atendidos totalmente com sucesso. A leitura, gravação e comparação das impressões digitais foram realizadas pelo sistema de maneira coerente e válida. Além disso, o sistema proporciona de fato um ambiente para manutenção de registros de entidades integrantes do contexto acadêmico, tornando possível sua implantação e posterior aplicação em casos reais.

Podem-se evidenciar limites na solução desenvolvida na questão do acesso concorrente aos recursos do banco de dados. Como a comparação de impressões digitais feita no momento da realização da chamada possui um processo separado que executa repetidas leituras no banco de

dados, na medida em que a quantidade de acessos a tal recurso cresce, mais pesada fica essa tarefa. Contudo, vale ressaltar que é muito difícil e improvável que, em um caso real de aplicação do sistema, tal limite seja atingido a ponto de comprometer o correto funcionamento da aplicação.

6.2 - Sugestões para Trabalhos Futuros

No que tange a trabalhos futuros que podem ser realizados baseando-se neste projeto, podem ser citados: substituição dos leitores de impressão digital USB por leitores *Bluetooth*, implementação de camada de integração entre o sistema e os demais sistemas pertinentes da instituição de ensino e o desenvolvimento de um módulo emissor de relatórios (pauta).

Com a finalidade de se evitar uma sala com vários cabos, pode-se tratar como projeto a substituição dos leitores de impressão digital USB por leitores *Bluetooth*. Tal mudança no modelo do projeto acarretaria, basicamente, no desenvolvimento de uma *applet* que funcione com tal tipo de leitores.

Para que o sistema possa realizar posteriores consultas e prover informações a serem consultadas por outros sistemas pertinentes da instituição, pode-se criar uma camada de integração orientada a serviços, que trafegaria informações relevantes entre vários sistemas via *Web Services* ou barramentos, por exemplo.

Por fim, pode-se tratar como projeto futuro o desenvolvimento de um módulo emissor de relatórios, que fornecerá ao professor a materialização das informações das presenças e ausências dos alunos gravadas no banco de dados. Através da definição de um *template* que será preenchido dinamicamente de acordo com os filtros informados e com as informações salvas no banco de dados, o professor pode imprimir sua própria pauta em diversos formatos e assiná-la, validando o documento.

REFERÊNCIAS

DEITEL, H. M. Java: Como Programar. 8ª ed. Pearson, 2010. 1176 p.

LUCKOW, D. C.; MELO, A. A. Programação Java para a Web. NOVATEC, 2010. 640 p.

QIAN, K..; ALLEN, R.; GAN, M.; BROWN, R. Desenvolvimento Web Java. LTC, 2010. 452 p.

SOULDERS S. Alta Performance em Sites Web. Altabooks, 2007. 152 p.

VIGLIAZZI D. Biometria Medidas de Segurança. 2ª ed. Visual Books, 2006. 82 p.

http://zkbrasil.blogspot.com/ (disponível em 10/11/2011);
http://javafree.uol.com.br/artigo/871498/ (disponível em 27/10/2011);
http://www.wthreex.com/rup/process/workflow/ana_desi/co_warchpatt.htm (disponível em 26/10/2011);
http://devjr.wordpress.com/tecnologia-java/ (disponível em 27/10/2011);

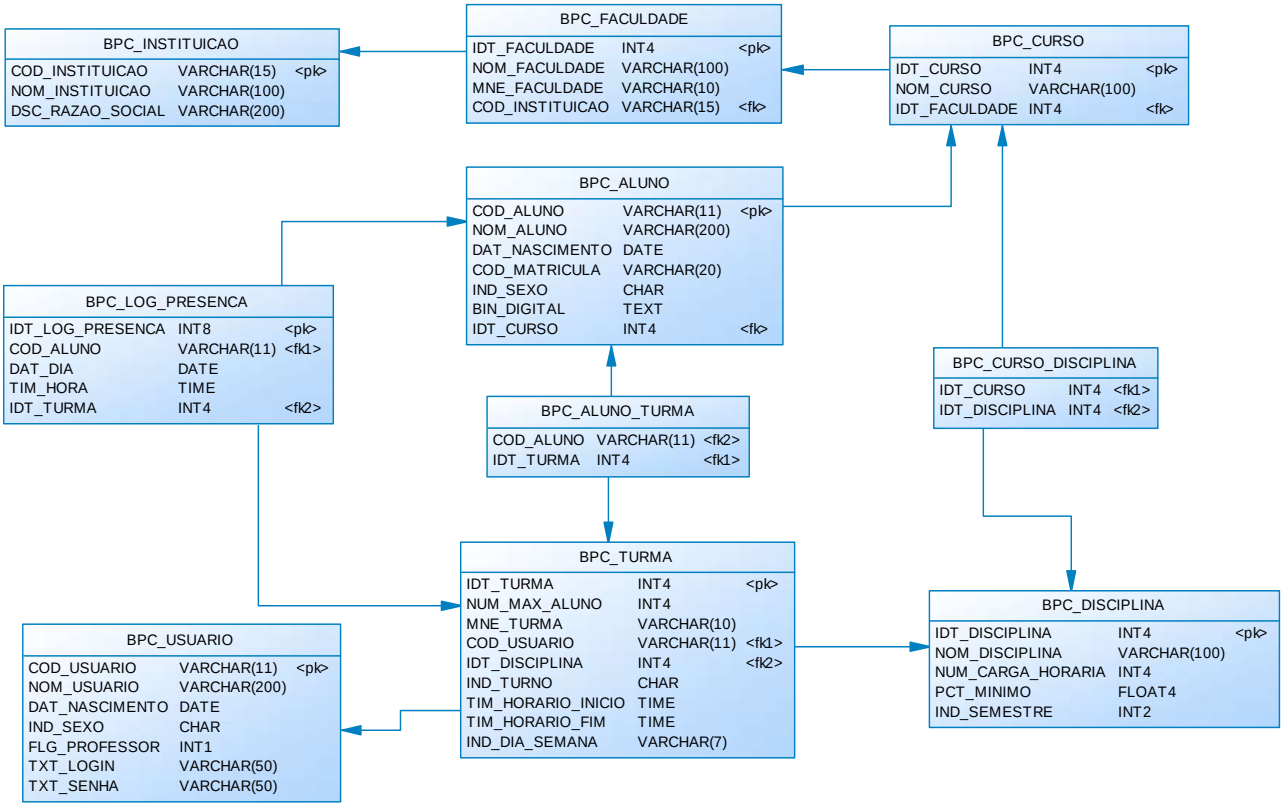
http://cavalcante.us/Programacao/ (disponível em 27/10/2011);
http://www.portalsaofrancisco.com.br/alfa/biometria/biometria-digital.php/ (disponível em 28/10/2011).

APÊNDICE A – RELATÓRIO FÍSICO DO MODELO DE DADOS

PDM Diagrama

I.1 Diagramas de nível de modelo

I.1.1 BioPresence – Diagrama Físico



II Lista de objetos de nível de modelo

II.1 Objetos comuns

II.1.1 Lista de diagramas

Nome	Código
BioPresence - Physical Diagram	BIOPRESENCE___PHYSICAL_DIAGRAM

II.2 Objetos do diagrama físico

II.2.1 Lista de colunas de tabela

Nome	Código

COD_INSTITUICAO	COD_INSTITUICAO
NOM_INSTITUICAO	NOM_INSTITUICAO
DSC_RAZAO_SOCIAL	DSC_RAZAO_SOCIAL
IDT_FACULDADE	IDT_FACULDADE
NOM_FACULDADE	NOM_FACULDADE
MNE_FACULDADE	MNE_FACULDADE
COD_INSTITUICAO	COD_INSTITUICAO
IDT_CURSO	IDT_CURSO
NOM_CURSO	NOM_CURSO
IDT_FACULDADE	IDT_FACULDADE
IDT_DISCIPLINA	IDT_DISCIPLINA
NOM_DISCIPLINA	NOM_DISCIPLINA
NUM_CARGA_HORARIA	NUM_CARGA_HORARIA
PCT_MINIMO	PCT_MINIMO
IND_SEMESTRE	IND_SEMESTRE
IDT_CURSO	IDT_CURSO
IDT_DISCIPLINA	IDT_DISCIPLINA
IDT_TURMA	IDT_TURMA
NUM_MAX_ALUNO	NUM_MAX_ALUNO
MNE_TURMA	MNE_TURMA
COD_USUARIO	COD_USUARIO
IDT_DISCIPLINA	IDT_DISCIPLINA
IND_TURNO	IND_TURNO
TIM_HORARIO_INICIO	TIM_HORARIO_INICIO
TIM_HORARIO_FIM	TIM_HORARIO_FIM
IND_DIA_SEMANA	IND_DIA_SEMANA
COD_ALUNO	COD_ALUNO
NOM_ALUNO	NOM_ALUNO
DAT_NASCIMENTO	DAT_NASCIMENTO
COD_MATRICULA	COD_MATRICULA
IND_SEXO	IND_SEXO
BIN_DIGITAL	BIN_DIGITAL
IDT_CURSO	IDT_CURSO
COD_ALUNO	COD_ALUNO
IDT_TURMA	IDT_TURMA
COD_USUARIO	COD_USUARIO
NOM_USUARIO	NOM_USUARIO
DAT_NASCIMENTO	DAT_NASCIMENTO
IND_SEXO	IND_SEXO
FLG_PROFESSOR	FLG_PROFESSOR
TXT_LOGIN	TXT_LOGIN
TXT_SENHA	TXT_SENHA
IDT_LOG_PRESENCA	IDT_LOG_PRESENCA
COD_ALUNO	COD_ALUNO
DAT_DIA	DAT_DIA
TIM_HORA	TIM_HORA
IDT_TURMA	IDT_TURMA

II.2.2 Lista de índices de tabela

Nome	Código	Unique	Cluster	Primary	Foreign Key	Alternate Key	Tabela
IDX_PK_COD_INSTITUICAO	IDX_PK_COD_INSTITUICAO	X		X			BPC_INSTITUICAO
IDX_PK_IDT_FACULDADE	IDX_PK_IDT_FACULDADE	X		X			BPC_FACULDADE
IDX_FK_FAC_INST	IDX_FK_FAC_INST				X		BPC_FACULDADE
IDX_PK_IDT_CURSO	IDX_PK_IDT_CURSO	X		X			BPC_CURSO
IDT_FK_CUR_FAC	IDT_FK_CUR_FAC				X		BPC_CURSO
IDX_PK_IDT_DISCIPLINA	IDX_PK_IDT_DISCIPLINA	X		X			BPC_DISCIPLINA
IDX_PK_IDT_CURSO_DISCIPLINA	IDX_PK_IDT_CURSO_DISCIPLINA	X		X			BPC_CURSO_DISCIPLINA
IDX_PK_IDT_TURMA	IDX_PK_IDT_TURMA	X		X			BPC_TURMA
IDX_FK_TURM_USU	IDX_FK_TURM_USU				X		BPC_TURMA
IDX_FK_TURM_DISC	IDX_FK_TURM_DISC				X		BPC_TURMA
IDX_PK_IDT_ALUNO	IDX_PK_IDT_ALUNO	X		X			BPC_ALUNO
IDX_FK_ALU_CUR	IDX_FK_ALU_CUR				X		BPC_ALUNO
IDX_PK_IDT_ALUNO_TURMA	IDX_PK_IDT_ALUNO_TURMA	X		X			BPC_ALUNO_TURMA
IDX_FK_AT_TURM	IDX_FK_AT_TURM				X		BPC_ALUNO_TURMA
IDX_FK_AT_ALU	IDX_FK_AT_ALU				X		BPC_ALUNO_TURMA
IDX_PK_COD_USUARIO	IDX_PK_COD_USUARIO	X		X			BPC_USUARIO
IDX_TXT_LOGIN	IDX_TXT_LOGIN	X					BPC_USUARIO
IDX_PK_IDT_LOG_PRESENCA	IDX_PK_IDT_LOG_PRESENCA	X		X			BPC_LOG_PRESENCA
IDX_FK_LOG_ALU	IDX_FK_LOG_ALU				X		BPC_LOG_PRESENCA
IDX_FK_LOG_TURM	IDX_FK_LOG_TURM				X		BPC_LOG_PRESENCA

II.2.3 Lista de chaves de tabela

Nome	Código	Tabela
PK_BPC_INSTITUICAO	PK_BPC_INSTITUICAO	BPC_INSTITUICAO
PK_BPC_FACULDADE	PK_BPC_FACULDADE	BPC_FACULDADE
PK_BPC_CURSO	PK_BPC_CURSO	BPC_CURSO
PK_BPC_DISCIPLINA	PK_BPC_DISCIPLINA	BPC_DISCIPLINA
PK_BPC_CURSO_DISCIPLINA	PK_BPC_CURSO_DISCIPLINA	BPC_CURSO_DISCIPLINA
PK_BPC_TURMA	PK_BPC_TURMA	BPC_TURMA
PK_BPC_ALUNO	PK_BPC_ALUNO	BPC_ALUNO
PK_BPC_ALUNO_TURMA	PK_BPC_ALUNO_TURMA	BPC_ALUNO_TURMA
PK_BPC_USUARIO	PK_BPC_USUARIO	BPC_USUARIO
PK_BPC_LOG_PRESENCA	PK_BPC_LOG_PRESENCA	BPC_LOG_PRESENCA

II.2.4 Lista de referências

Nome	Código	Tabela Pai	Tabela filha
FK_ALUNO_CURSO	FK_ALUNO_CURSO	BPC_CURSO	BPC_ALUNO
FK_ALUNO_TURMA_ALUNO	FK_ALUNO_TURMA_ALUNO	BPC_ALUNO	BPC_ALUNO_TURMA
FK_ALUNO_TURMA_TURMA	FK_ALUNO_TURMA_TURMA	BPC_TURMA	BPC_ALUNO_TURMA
FK_CURSO_DISCIPLINA_CURSO	FK_CURSO_DISCIPLINA_CURSO	BPC_CURSO	BPC_CURSO_DISCIPLINA
FK_CURSO_DISCIPLINA_DISCPLINA	FK_CURSO_DISCIPLINA_DISCPLINA	BPC_DISCIPLINA	BPC_CURSO_DISCIPLINA
FK_CURSO_FACULDADE	FK_CURSO_FACULDADE	BPC_FACULDADE	BPC_CURSO
FK_FACULDADE_INSTITUICAO	FK_FACULDADE_INSTITUICAO	BPC_INSTITUICAO	BPC_FACULDADE
FK_LOG_PRESENCA_ALUNO	FK_LOG_PRESENCA_ALUNO	BPC_ALUNO	BPC_LOG_PRESENCA
FK_LOG_PRESENCA_TURMA	FK_LOG_PRESENCA_TURMA	BPC_TURMA	BPC_LOG_PRESENCA
FK_TURMA_DISCIPLINA	FK_TURMA_DISCIPLINA	BPC_DISCIPLINA	BPC_TURMA
FK_TURMA_USUARIO	FK_TURMA_USUARIO	BPC_USUARIO	BPC_TURMA

II.2.5 Lista de tabelas

Nome	Código
BPC_ALUNO	BPC_ALUNO
BPC_ALUNO_TURMA	BPC_ALUNO_TURMA
BPC_CURSO	BPC_CURSO
BPC_CURSO_DISCIPLINA	BPC_CURSO_DISCIPLINA
BPC_DISCIPLINA	BPC_DISCIPLINA
BPC_FACULDADE	BPC_FACULDADE

BPC_INSTITUICAO	BPC_INSTITUICAO
BPC_LOG_PRESENCA	BPC_LOG_PRESENCA
BPC_TURMA	BPC_TURMA
BPC_USUARIO	BPC_USUARIO