Revista Eletrônica em Gestão e Tecnologia
PRINT ISSN 2316-4972
ONLINE 2447-0422
FAQI
FACULDADE QI BRASIL

SOFTWARE DE MÚLTIPLAS FUNCIONALIDADES VERSATILITY

Mateus Corrêa Vidal[1]

Eduardo Réus Souza[2]

RESUMO

Este projeto teve como foco o desenvolvimento de um *software* denominado Versatility, que tem como objetivo ser um programa que pode ser configurado para ser usado para diversas finalidades, permitindo que o usuário configure acessos de menu na tela de menu principal, e que para cada acesso possa ser configurado uma tela para algum objetivo específico, possibilitando criar campos de formulário pré-configurados e alterando suas propriedades de acordo com a necessidade, além disso é possível criar comandos de ações a serem executadas quando acionados eventos do componente de tela, como o clique de um botão por exemplo. Os comandos podem ser usados para executar várias funções como cálculos de equações, armazenamento de dados em arquivo e banco de dados, comunicação entre componentes e validações de dados. Tudo dentro de um sistema com configurações básicas de uso, como controle de acesso por usuários com configuração de permissões, e opções de compartilhamento de configurações via *WebService* e configurações de *backup*.

Palavras-chave: Versatility. *Software*. Customizável.

1 INTRODUÇÃO

Atualmente o mercado de *software* está sobre muita exigência. A cada dia o número de empresas aumenta, e a virtualização dos procedimentos das empresas e organizações também, o que torna a eficiência nos procedimentos entre a aquisição e uso de um sistema um fator a ser considerado para o ramo.

Atualmente existem dois modelos de negócios de *software* mais comumente usados, o *software* de prateleira, que seria um *software* construído seguindo bases

[1] Tecnólogo em Análise e Desenvolvimento de Sistemas pela Faculdade QI Brasil, Gravataí, 2019. E-mail: mateus.vidal95@gmail.com
[2] Professor do curso de Análise e Desenvolvimento de Sistemas da– Faculdade QI Brasil. Gravataí.

de negócios genéricas para abranger os procedimentos de uma gama de negócios maior possível (SOMMERVILLE, 2011), para que o mesmo *software* possa ser vendido para as diversas empresas, porém tornando necessário a manutenção e correções apenas no mesmo *software*, e o *software*s customizados, para o negócio e procedimentos específicos de uma empresa ou um grupo de empresas, sendo o mais adequado para empresa com um modelo de negócio e forma de trabalhar específico, o que acaba tornando a manutenção mais complicada para a empresa desenvolvedora, fazendo com que ela tenha que trabalhar na manutenção de cada *software* customizado individualmente, para seus cliente de forma específicada para cada um deles.

Versatility é um *software* que tem como finalidade oferecer ferramentas e recursos para ser usado como um *software* de qualquer finalidade, possibilitando a criação e configuração de menus e telas capazes de diversas funções como armazenamento de dados tanto em banco de dados relacional como em arquivos de texto, execução de cálculos matemáticos e implementações de processos de gestão.

1.1 TEMA

Desenvolvimento de sistema com menus, telas e procedimentos configuráveis via interface gráfica viabilizando alinhamentos de funcionalidades a analistas e usuários.

1.2 DELIMITAÇÃO DO TEMA

Projeto delimitado ao desenvolvimento de *Software* na linguagem Java com biblioteca *swing* e com SGBD (Sistema de Gerenciamento de Banco de Dados) Mysql e com o uso de *WebService* via servlet e uso de padrões de projetos.

1.3 PROBLEMA

Como poderia ser construído um *software* que possa ter seus procedimentos definidos pelo usuário?

1.4 OBJETIVOS

Nas subseções a seguir, serão apresentados os Objetivos Geral e Específicos.

1.4.1 Objetivo Geral

Desenvolver *Software* que ofereça ferramentas e recursos para ser usado como um *software* de qualquer finalidade, possibilitando a criação e configuração de menus e telas capazes de diversas funções como armazenamento de dados, tanto em banco de dados relacional como em arquivos de texto, execução de cálculos matemáticos e implementações de processos de gestão. Tendo suas configurações feitas através de sua interface de uso mais simples e prática do que via programação.

1.4.2 Objetivos Específicos

Abaixo são apresentados os elementos que visam atingir o Objetivo Geral através de sua execução.

a) Analisar interfaces de *software* para obter entendimento das melhores práticas no desenvolvimento da própria interface do *software* visando facilitar o uso devido natureza indireta e abrangente do tipo de *software* a ser desenvolvido;

b) Desenvolver parte inicial do *software* que tem a finalidade de manter cadastro de usuários e permitir logon dos mesmos;
c) Desenvolver Menu Principal editável e Telas Genéricas editáveis que permitam ao usuário a elaboração das interfaces de uso do sistema de acordo com sua finalidade específica para uso do sistema;
d) Desenvolver comandos internos no sistema para que seja possível criar ações e eventos diversos nas telas do sistema;
e) Desenvolver persistência dos dados do sistema em arquivo e banco de dados através dos comandos internos do sistema;
f) Desenvolver repositório de configurações de telas para integração de configurações de sistema entre máquinas diferentes com comunicação viabilizada através de *web service*;
g) Testes e correções finais;
h) Elaborar a documentação necessária para o sistema;

1.5 JUSTIFICATIVA

Criar um meio termo entre os modelos *software* customizado e o de prateleira, para que se tenha um novo modelo onde o sistema pode ser comercializado como um *software* de prateleira, porém tendo opções de customização de acordo com as especificidades dos procedimentos dos clientes. Mesmo tendendo a deixar o processo de implantação mais complexo, ele torna o desenvolvimento e a manutenção mais simples e eficientes, pois ele possibilita que os ajustes da suas funcionalidades sejam feitos pelos próprios analistas, tornando dispensável o problema ser analisado e passado para programação, e permitindo que a programação foque apenas nas funcionalidades padrões do sistema.

2 FUNDAMENTAÇÃO TEÓRICA (DESENVOLVIMENTO)

Embasamento teórico sobre os conhecimentos e tecnologias usadas na elaboração do projeto proposto neste trabalho.

2.1 PRODUTIVIDADE E DESENVOLVIMENTO

Um ponto vital para qualquer tipo de negócio é a Produtividade (Alexandre Shigunov Neto e Letícia Mirella Fischer Campos, 2016), ela pode ser atingida através de vários meios, e depende de vários fatores que variam de acordo com o cenário, porém um ponto de grande relevância é entender o seu significado e dar a ela o foco necessário para assim se obter o sucesso.

De acordo com Rattner (1967, p. 25),

> Considera-se a produtividade uma medida que avalia a eficiência e a racionalidade das atividades econômicas. Na prática, esta medida é definida como a relação entre o insumo (*input*) e a produção (*output*), no nível da empresa, da indústria ou da economia global. Elevar o nível da produtividade, de um ponto de vista estático, significaria aumentar a produção (*output*) com a mesma combinação dos fatores de produção (*input*), ou ainda, manter o nível de produção, realizando economias no insumo dos fatores. O próprio processo de medição é importante para determinar e averiguar o desempenho da empresa ou da economia, bem como para avaliar a exequibilidade dos planos e metas do desenvolvimento econômico.

Atingir um bom nível de produtividade é possível através de vários meios, ele pode ser obtido investindo em procedimentos já em prática, os executando em escalas maiores ou aprimorando seus componentes, ou a produtividade pode ser aumentada usando novos métodos inovadores ou já existentes, trabalhando em novos procedimentos, e alinhando a fórmula para melhor atender a execução de uma tarefa (COSTA JR, 2012). Para tais desafios é importante ter a compreensão

do contexto que envolve os procedimentos sendo trabalhados e de como as variáveis do procedimento pode impactar no resultado final.

2.1.1 Produtividade no Desenvolvimento de *Software*

Em questão a produtividade no desenvolvimento de *software*, se observado de um aspecto mais amplo. Ela não diferencia muito do molde já estabelecido em outras áreas. O *software* é produzido assim como a maioria dos produtos, através de pessoas com capacitação específica para o trabalho, com ferramentas específicas, demandando recursos específicos, levando um prazo para ficar pronto e tendo uma prévia análise e um planejamento do que será construído. Porém mesmo com essa similaridade, por ser uma ciência considerada jovem a mesma apresenta um número de problemas maior do que a média, e também por ter um grande nível de complexibilidade na sua aplicação, uma complexibilidade nova e distinta das que o ser humano se adaptou a enfrentar com o passar dos tempos.

Verificando os problemas que interferem na produtividade do *software*, ao analisar seus processos mais detalhadamente, se tem os seguintes problemas como causadores da instabilidade da área.

Estimativas de prazo (meses, anos) e custo imprecisos (PRESSMAN, 2009), pelo alto nível de especificidade de cada *software* e a variação da sua conformidade com seu objetivo se torna uma tarefa difícil estimar um prazo para sua conclusão. Produtividade abaixo da praticada pelo mercado (PRESSMAN, 2009),
a grande expansão da área afeta a produtividade. *Software* de baixa qualidade (erros e não conformidades com requisitos que tiram a confiança do cliente sobre o produto) (PRESSMAN, 2009), falhas no funcionamento costumam ser comuns devido a complexibilidade do desenvolvimento, além das dificuldades do caminho entre a obtenção das informações sobre as finalidades do *software* e seu entendimento e as reais necessidades do cliente.

Revista Eletrônica em Gestão e Tecnologia
PRINT ISSN 2316-4972
ONLINE 2447-0422
FAQI
FACULDADE QI BRASIL

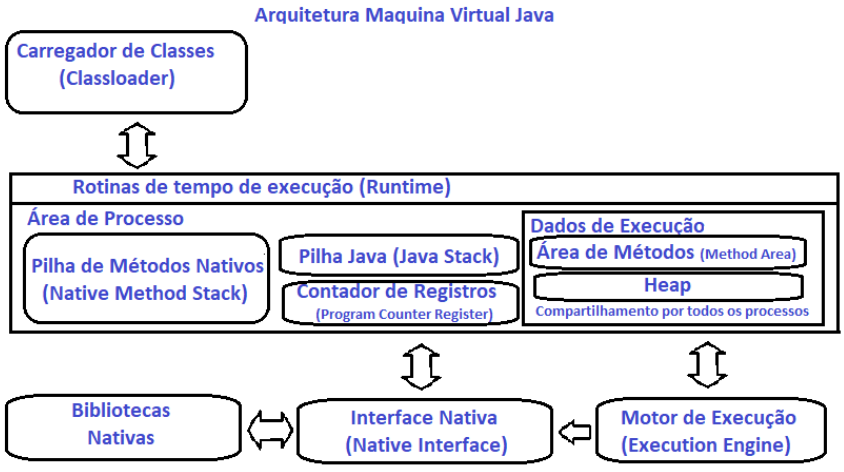
2.2 LINGUAGEM DE PROGRAMAÇÃO JAVA

De acordo com Horstmann e Cornell (2010, p. 13),

> No Fim de 1995, a linguagem de programação Java fez sua entrada fenomenal na cena da Internet e ganhou status de celebridade instantânea. A promessa da tecnologia Java era de que ele se tornaria a cola universal que conecta usuários a informações, quer essas informações venham de servidores Web, banco de dados, provedores de informações, quer venham de qualquer outra fonte imaginável. Na verdade, o Java está em uma posição única para cumprir essa promessa. Ele é uma linguagem extrema e solidamente projetada que ganhou aceitação por parte das principais empresas, exceto da Microsoft. Sua segurança embutida e os recursos de segurança dão mais tranquilidade tanto a programadores como a usuários dos programas Java. O Java tem até mesmo suporte embutido que simplifica tarefas de programação avançadas, como programação de rede, conectividade de banco de dados e *multithreading*.

A Linguagem de programação Java, foi desenvolvida com princípio de seguir o paradigma de programação OO (Orientado a Objeto), ela foi criada pela empresa Sun Microsystems, e teve como programador principal James Gosling, sua linguagem é compilada para *bytecode* é interpretada via máquina virtual (a JVM) diferente no funcionamento das linguagens comuns.

Imagem 1 – Representação gráfica da arquitetura da JVM



Fonte: Autor

A Sun Microsystems foi incorporada pela Oracle *Corporation* em 2008, a fim de obter o Java entre outros produtos.

Algumas características do Java além da Orientação a objetos, e da independência de plataformas graças a sua JVM já mencionada, a linguagem também dispõem de uma vasta biblioteca de recursos que garante uso de protocolos como TCP/IP e HTTP entre outros de forma mais simplificada.

A linguagem edições específicas para determinados tipos de programação, iremos explicar algumas delas nas próximas subseções.

2.2.1 Java SE

A edição do Java que é usada para trabalhar com Programação de Aplicativos *desktop* (execução em máquina(s) sem comunicação com rede externa (*internet*)) é o J2SE (*Java to Standard Edition*), todos seus recursos e configurações de desenvolvimento são focados na construção de aplicações dos mais variados tipos sendo executáveis de procedimentos automatizados, ou com interface moldada para ter comunicação com os usuários. (DEITEL, 2016)

A criação dessas interfaces gráficas mencionadas acima são geralmente feitas através de vários componentes, algumas das APIs mais importantes, são a AWT que é a mais antiga porém uma opção funcional para o criação de telas, a *Swing* mais completa porém com alguma dependencia da AWT, mais recentemente surgiu o JavaFX que promete substituir o Java *Swing*.

2.2.2 Java EE

O J2EE é a edição do Java que é usada para trabalhar com Programação Web (páginas de *websites*), com sua execução dependente de Navegadores de internet ou comunicação externa. Seu foco é em elementos web como Servlets e JSP (Java Server Pages), e tem recursos de integração com servidores de aplicação

como Glassfish, Weblogic, JBoss. Ele tem as mesmas funcionalidades básicas do J2SE porém estão encapsuladas em um contexto web. (DEITEL, 2016)

O Recurso Servlet, serve como um *script* de comunicação que recebe uma requisição de informação, processa os dados recebidos, e retorna uma resposta fórmulada através desses dados, ele funciona em conjunto com servidores de aplicação como os citados acima.

2.3 BANCO DE DADOS

De acordo com Vicci (2015, p. 6),

> Uma coleção de dados relacionados, com um significado implícito, é um banco de dados. Assim, o livro que você está lendo é um banco de dados, pois as palavras, os números e as figuras que constam nas páginas são dados relacionados, Mas essa definição é muito genérica. Em geral, um banco de dados deve ter as seguintes propriedades:
> -representar algum aspecto do mundo real, o qual é chamado de minimundo (ou universo de discurso);
> -ser uma coleção logicamente coerente de dados com algum significado inerente, e não uma variedade aleatória de dados;
> -ter uma finalidade específica;
> -possuir um grupo definido de usuários;
> -possuir aplicações previamente concebidas, nas quais os usuários estejam interessados.
> Em outras palavras, um banco de dados deve ter alguma fonte que oferece os dados, certo grau de interação com eventos do mundo real e um público interessado em seu conteúdo. Por exemplo, os dados relativos a uma partida de futebol por torcedores e organizações desportivas interessadas. Para que um banco de dados seja preciso e confiável, ele precisa ser um reflexo verdadeiro do minimundo que representa; por tanto, as mudanças precisam ser inseridas no banco de dados o mais breve possível. Se você quer ver em um site, por exemplo, os filmes que vão estrear no dia em que você estiver fazendo a pesquisa, é preciso que essa informação já esteja lá; do contrário, o site perderá credibilidade.

A função do banco de dados em sua aplicação em sistemas é o armazenamento dados, fisicamente ele pode consistir em um ou mais arquivos, para ser criado um banco de dados existem sistemas próprios para a função chamados SGBDs (Sistema de Gerenciamento de Banco de Dados). Existem vários SGBDs no

mercado, alguns exemplos dos mais conhecidos são: MySQL, SQL Server, Postgre e etc.

2.3.1 SGBD Mysql

O Mysql usa a linguagem SQL (que será melhor explicada na próxima seção) para criar e manipular suas bases de dados relacionais, ele é uma dos SGBDs mais populares do mercado, ele foi criado na década de 80 na suécia por suecos e finlandeses Allan Larsson, David Axmark e Michael Widenius.

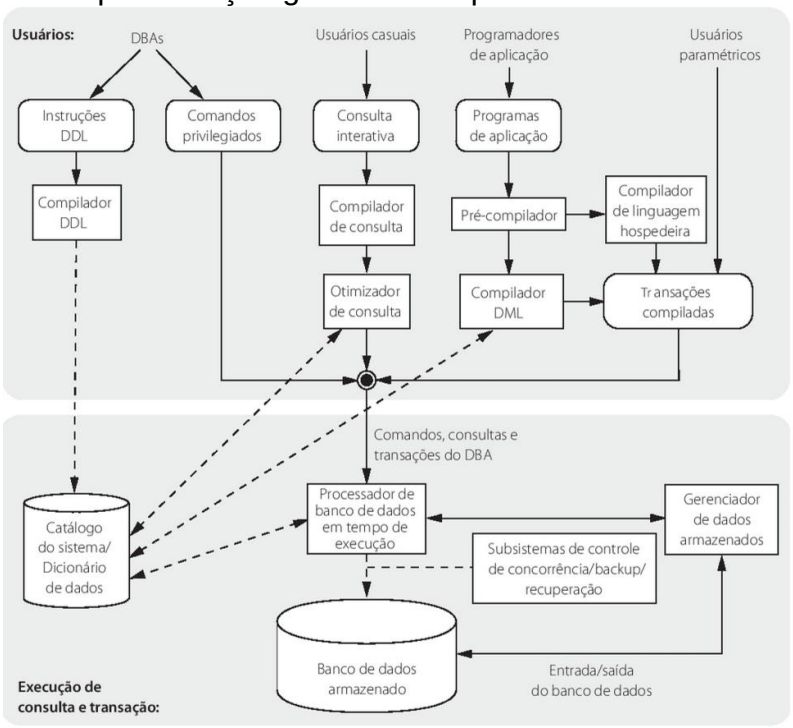
Características:
-Compatibilidade (tem drivers para todas as linguagens de programação populares)
-Portabilidade (suportado por todas plataformas atuais populares);
-Em comparação ao contexto do mercado tem um excelente desempenho;
-Não apresenta complicações quanto a novos recursos e novos *hardware* (boa integração com diferentes dispositivos);
-Usabilidade simplificada e opção de interface gráfica;
-Suporta controle transacional (método de comunicação com banco por controle de transações), *triggers* (criação de funções de gatilho), cursores (funções de retorno otimizadas para criar um fluxo de retorno mais leve), *procedures* (criação de procedimentos pré- configuradas no banco) e *functions* (criação de novas funções pré-configuradas no banco)
-Tem opções de replicação (Clonagem de banco);
-*Software Free* (Gratuito) com exceções caso o programa que for usá-lo não seja *Free* também;

2.3.2 SQL (*Structured Query Language*)

O SQL é um conjunto de linguagens utilizadas pelos SGBDs de banco de dados relacionais para fazer o controle do banco de dados através de comandos

pré-definidos com infinitas variações de combinações, através deles é possível fazer qualquer configuração, estruturação ou manutenção de registros na instância do banco de dados, abaixo uma breve explicação sobre as linguagens incluídas no sql.

Imagem 2 – Representação gráfica da arquitetura da do Banco de Dados



Fonte: Elmasri e Navathe (2011, p.27)

Sobre a linguagem de definição de dados, responsável pela estrutura do banco.

De acordo com Vicci (2015, p. 27-28),

> A. Linguagem de definição de dados (DDL - data definition *language*) - é usada em muitos SGBDs em que não é mantida nenhuma separação estrutura de níveis para definir os dois esquemas. O SGBD terá um compilador da DDL cuja função é processar instruções da DDL, a fim de identificar as descrições dos construtores de esquemas e armazenar a descrição de esquema no catálogo do SGBD. Nos SGBDs que mantêm uma separação clara entre os níveis conceitual e interno, a DDL é usada para específicar apenas o esquema conceitual.

Sobre a Linguagem de definição de armazenamento, responsável pelas configurações internas do banco.

De acordo com Vicci (2015, p. 27-28),

> B. Linguagem de definição de armazenamento (SDL - *Storage definition language*) - é utilizada para específicar o esquema interno. Os mapeamentos entre os dois esquemas podem ser específicados nessa linguagem ou na DDL. Na maioria dos SGBDs relacionais não existe uma linguagem específica que realize o paral de SDL. Em vez disso, o esquema interno é específicado por uma combinação de funções, parâmetros e específicações relacionadas ao armazenamento, que permitem aos DBAs controlar opções de indexação e mapeamentos dos dados que serão armazenados.

Sobre a Linguagem de definição de Visão, responsável pela apresentação externa dos dados.

De acordo com Vicci (2015, p. 27-28),

> C. Linguagem de definição de visão (VDL - *view definition language*) - é utilizada em uma arquitetura de três esquemas para específicar as visões do usuário e seus mapeamento ao esquema conceitual. Porém, na maioria dos SGBDs, a DDL tem a função de definir tanto o esquema conceitual como o externo. Nos SGBDs relacionais, a SQL é usada pela VDL para definir as visões do usuário ou da aplicação como resultados de consultas predefinidas.

Sobre a Linguagem de manipulação de dados, responsável por viabilizar o uso prático do banco de dados.

De acordo com Vicci (2015, p. 27-28),

> D. Linguagem de manipulação de dados (DML - *data manipulation language*) - é o conjunto de operações que permite que os usuários manipulem o banco de dados (recuperação, inserção, exclusão e modificação dos dados).
> Existem dois tipo de DML:
> DML de alto nível ou não procedural - pode ser utilizada para específicar operações de banco de dados complexas de forma concisa. Muitos SGBDs permitem que instruções de DML de alto nível sejam inseridas interativamente em um monitor ou terminal, ou sejam embutidas em uma linguagem de programação de uso geral. Nesse último caso, as instruções DML precisam ser identificadas dentro do programa, de modo que possam ser extraídas por um pré-compilador e processadas por um SGBD, As

> DMLs de alto nível, como a SQL, podem específicar e recuperar muitos registros em uma única instrução DML; portanto, elas são chamadas de DMLs de um conjunto de cada vez, ou orientadas a conjunto. Uma consulta em uma DML de alto nível normalmente específica quais dados recuperar, em vez de como recuperá-los; por isso, são também chamadas de declarativas.
> DML de baixo nível ou procedural - deve ser embutidas em uma linguagem de programação de uso geral. Esse tipo de DML recupera registros individuais ou objetos do banco de dados e processa cada um deles separadamente. Portanto, precisa de construções de linguagem de programação, como o *looping*, para recuperar e processar cada registro de um conjunto de registros.

2.4 ENGENHARIA DE *SOFTWARE*

De acordo com Sommerville (2007, p. 2),

> O conceito de engenharia de *software* foi inicialmente proposto em 1968, em uma conferência organizada para discutir o que foi então chamado de 'crise de *software*'. A crise de *software* resultava diretamente da introdução de novo hardware de computador baseado em circuitos integrados. Seu poder fez das aplicações de computador, consideradas até então não realizáveis, propostas viáveis. O *software* resultante era ordens de grandeza maior e mais complexo que sistemas anteriores de *software*.

O objetivo da engenharia de *software* é facilitar gerar maior qualidade na criação, desenvolvimento e manutenção de *software*, para tal objetivo ela visa organizar a equipe de profissionais envolvidos, incentivar padronizações dos procedimento envolvidos e fornecer ferramentas para ajudar planejar, executar e gerenciar os projetos de *software*.

Principais conceitos da Engenharia de *Software*:

-Modularidade: Há dois aspectos na arquitetura de um sistema que influenciam na modularidade, a Coesão, que trata sobre a responsabilidade de cada parte do sistema, como classes ou métodos, terem seu escopo bem específicadas e não fugirem dele, o segundo aspecto é o acoplamento, ele trata da dependência entre as partes do sistema, para se atingir a modularidade se tem como ideal evita o acoplamento.

-Rigor e Formalidade: Para que se tenha maior controle e capacidade de gerenciamento em um projeto de *software* como em qualquer outra área é

importante se ter regras e procedimentos de controle como documentação e prazos de entrega. Através desse conceito é possível se obter maior confiabilidade e profissionalismo.

-Antecipação a Mudanças: Trata-se de tentar prever mudanças futuras no contexto onde o *software* se encaixar para evitar problemas de compatibilidade e desempenho.

-Abstração: Tem como objetivo trazer para um sistema apenas suas características realmente relevantes, e ignorar (abstrair) o que não tem necessidade e que só iria aumentar a complexibilidade.

-Separação de Interesses: Quando se tem um objetivo grande costuma ser mais conveniente separá-lo em tarefas e organizar quem, quando e como irá irá resolvê-las, para se ter mais eficiência é importante que essas separações e designações sejam feitas de forma adequada.

-Incrementação: se refere capacidade do *software* para receber incrementações (ser adicionado mais funcionalidades), deixa o *software* suscetível a evolução.

-Generalidade: É um princípio específico para solução de problemas que procura identificar se a solução encontrada pode ser usada em problemas similares ou de mesma origem. (BAUER, 1997).

2.4.1 Programação Orientada a objetos (POO)

A Orientação a Objetos é um paradigma de programação que tem o intuito de facilitar a organização e a compreensão humana do código fonte dos sistemas, moldando a estrutura do sistema de acordo com aspectos do mundo real, convertendo o "algo" ou " alguém " e suas interações em "objetos".

Como por exemplo caso viesse a ser criado um sistema para cadastrar pessoas, ele viria a ter um objeto pessoa (no caso do java seria criada através de uma classe, a implementação pode variar de acordo com a linguagem de programação), nesse objeto seriam criados variáveis para representar atributos (se

seria os dados de uma pessoa) como nome, endereço, altura, filhos e etc. e também seriam criados métodos (funções, funções procedimentos... também pode variar com a linguagem) que seriam as atividades que uma pessoa poderia fazer dentro do sistema, registrar, cadastrar, vincular filhos e etc.

Um exemplo de como os objetos se relacionam em um sistema seria se fosse usado o método vincular filha para adicionar um objeto filho ou até uma outra instância do objeto pessoa dentro do atributo filhos, uma outra característica comum na orientação a objetos seria a herança onde um objeto "herda" atributos e métodos de outro para evitar repetição de código, um exemplo de herança seria um objeto filho que teria os mesmo dados de um objeto pessoa, além desses conceitos existem vários outros na no paradigma OO, como encapsulamento, retenção de estado, identidade de objeto, mensagens, a já mencionada herança, polimorfismo e generalização. (PAGE-JONES, 2000)

2.4.2 Padrões de projeto (*Design Patterns*)

Padrões de projetos, como o próprio nome diz são padrões de formas de programar ou desenhar (se refere a diagramação e documentação envolvida) em projetos de *softwares*, com o propósito de definir um padrão geral a se seguir tornando mais fácil o reconhecimento do trabalho para de mais envolvidos, além de passar as melhores formas de se resolver um determinado problema, gerando qualidade para o produto.

Um dos livros mais importantes sobre o assunto é o "Padrões de Projeto: soluções reutilizáveis de *software* orientado a objetos", ele traz os padrões Gof a sigla se refere a Gangue dos Quatro "*Gang Of Four*" (Devido a contribuição de quatro especialistas na área), o livros apresenta diversos tipos de soluções em forma de padrões de projetos, elas podem divididas em três classificações diferentes. Padrões de Criação: eles tem como objetivo "automatizar" e dar mais flexibilidade ao processo de instânciamento (criação) de objetos, tornando o sistema independente desses procedimentos, através de classes com determinadas a

"construção" de objetos. Padrões estruturais: eles visão lidar com a composição de objetos, para atribuir novas funcionalidades a eles e dar mais flexibilidade na sua reestruturação. Padrões comportamentais: eles servem para definir de forma dinâmica os comportamentos dos objetos, e na forma como eles se comunicam. Algo em comum nesses padrões é o uso de heranças e encapsulamento.

2.4.2.1 Padrão DAO (Data Access Object ou objeto de acesso a dado)

O Padrão tem o intuito de separar a lógica interna do sistema da lógica de persistência dos dados, abstraindo toda a lógica de comunicação com a sgbd em objetos específicos dedicados somente a essa comunicação. Através dessa forma de organização de código a manutenção do mesmo se torna muito mais simples e direta.

2.4.2.2 Padrão DTO (Objeto de Transferência de Dados ou *Data transfer object*)

O DTO sugere a criação de um objeto próprio para passar os dados da Visão para o Modelo ao invés de usar o objeto de negócio da Modelo, pois dessa forma violaria a proposta da arquitetura MVC, a diferença entre um objeto DTO para outros objetos de negócio séria que a única função do DTO seria transportar dados de uma parte para outra. Quando o padrão é aplicado ao Java a classe DTO carrega a nomenclatura POJO no nome que significa "*Plain Old Java Object*".

2.4.2.3 GATEWAY

A proposta do padrão é abstrair toda a lógica de uma fonte de comunicação externa específica a uma objeto, exemplos dessas fontes seria arquivos de texto e sgbds, a vantagem de ter um objeto específico para essas comunicações é que toda a configuração da conexão entre o sistema e o ponto externo seria concentrada

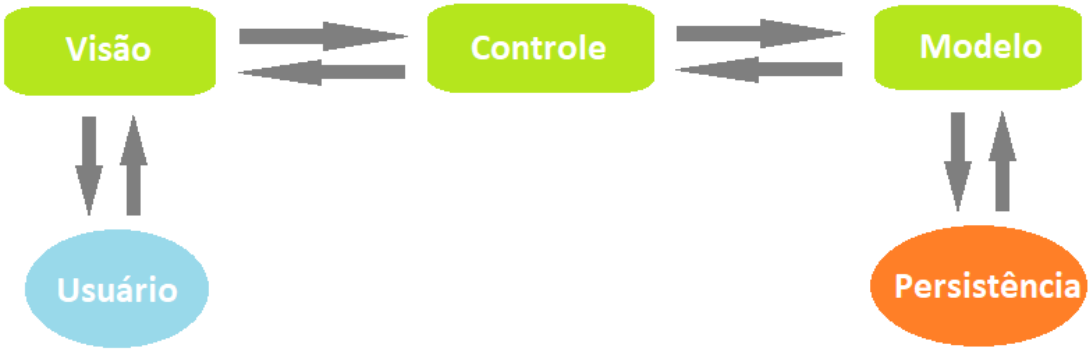
apenas em um objeto, evitando repetição de código e tornando a manutenção mais fácil.

2.4.3 Arquitetura MVC (*Model-View-Controller* ou Modelo-Visão-Controle)

De acordo com Reenskaug (1996),

> O MVC foi concebido como uma solução geral para o problema dos usuários que controlam um conjunto grande e complexo de dados. A parte mais difícil foi encontrar bons nomes para os diferentes componentes da arquitetura. *Model-View-Editor* ou Modelo-Visão-Editor foi a primeira escolha. Após longas discussões, particularmente com Adele Goldberg, acabamos nos termos *Model-View-Controller* ou Modelo-Visão-Controle.

Imagem 3 – Esboço conceitual da Arquitetura MVC



Fonte: Autor

O MVC divide a estrutura do sistema em 3 níveis, o "Modelo", que seria a camada mais interna com a lógica do sistema, que seria a parte responsável por executar as funções, a "Visão", que seria camada mais externa que seria a interface por onde o usuário usará para solicitar as funções, e por fim o "Controle", que seria a camada intermediária do sistema que tem a finalidade de fazer a comunicação entre a Visão e o Modelo. Essa arquitetura além de facilitar o desenvolvimento separando a construção das três partes que tem desenvolvimento específicos, tem como maior

ponto reutilização de código, pois tendo essas três partes separados é possível reaproveitar a "Modelo" quando criada uma nova interface de acesso ao sistema, ou reaproveitar a "Visão" caso seja criada uma nova forma de executar as mesmas requisições em um outro contexto.

2.4.4 UML (*Unified Modeling Language*)

De acordo com Medeiros (2004),

> A finalidade da UML é proporcionar um padrão para a preparação de planos de arquitetura de projetos de sistemas, incluindo aspectos conceituais, como processos de negócios e funções de sistema, além de itens concretos, como as classes escritas em determinada linguagem de programação, esquemas de bancos de dados e componentes de *software* reutilizáveis (Booch, Rumbaugh e Jacobson).
> A específicação da UML pode ser encontrada para download em *http://www.omg.org/tecnology/documents/formal/uml*, onde é possível encontrar a última versão desse documento em PDF, Z1P ou *PostScript*. Percebemos, logo à primeira vista, que a UML não nos indica como devemos fazer um *software*. Ela indica apenas as formas que podem ser utilizadas para representar um *software* em diversos estágios de desenvolvimento.
> Utilizando a UML, conseguimos ‘pensar’ um *software* em um local e codificá-lo em outro. É evidente que alguma outra comunicação adicional se fará necessária, porém deve ser minimalista. Se muitas perguntas estão surgindo em função de determinado diagrama, isso é um sério sinal de que esse diagrama deve ser revisado.

Através dessa definição é possível compreender a relevância da UML para o planejamento e documentação dos projetos de *software*, nas subseções seguintes será mostrado alguns dos recursos da UML.

2.4.4.1 Diagrama de caso de uso (UML)

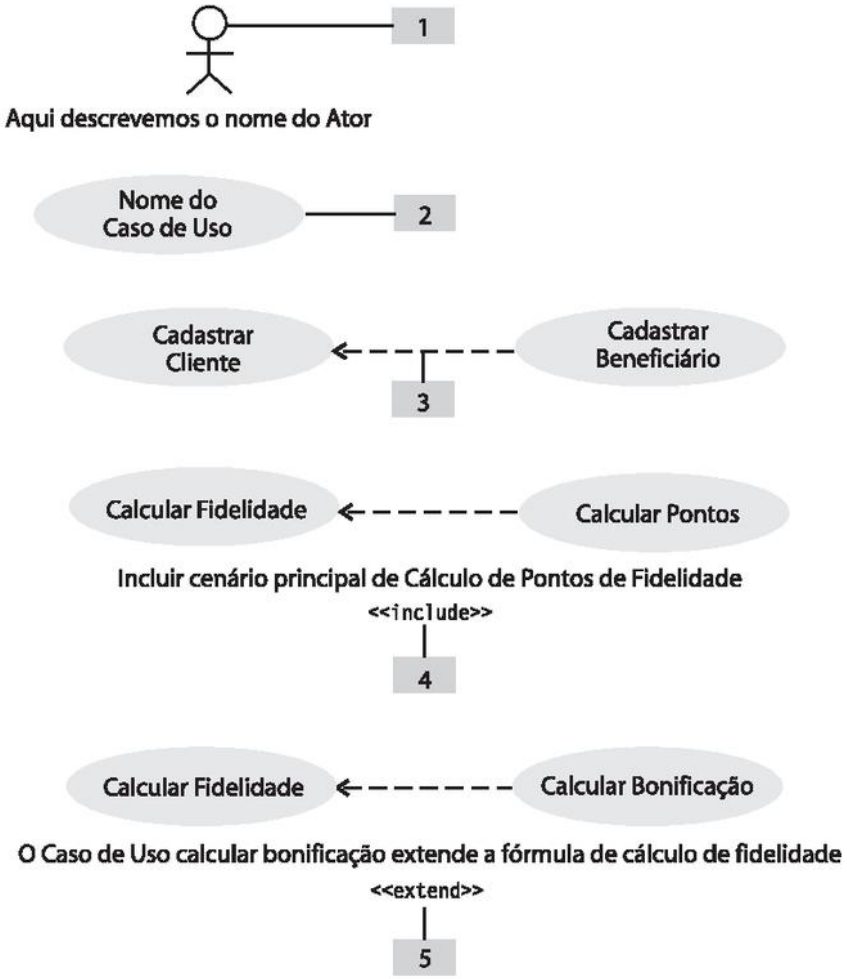
Umas das ferramentas mais conhecidas da UML são os seus diagramas que tem como estratégia ilustrar aspectos dos projetos. Um dos diagramas mais comumente usados é o de caso de uso, seu foco é ilustrar a interação dos usuários (tidos como atores) com as funcionalidades do sistema (tidas como um caso de uso).

De acordo com Medeiros (2004, p. 26),

> Para iniciarmos a confecção de qualquer diagrama da UML, é necessário conhecermos a sua notação, ou seja, a forma como devemos representá-lo e suas semânticas.
> O diagrama de Caso de Uso tem uma notação bem simples. Vamos entendê-la.
> Acompanhe a numeração na Figura 2.1 com os nossos comentários.
> 1. Este é um *Stick Man* ou Ator. O Ator pode ser uma pessoa, um sistema ou mesmo o que chamávamos, na análise estruturada, de Entidade Externa; por exemplo: um banco - instituição financeira. O ator pode ser até algo como um roteador. Ele realiza uma atividade. O ator sempre atua sobre um Caso de Uso.
> 2. Aqui nós temos a representação de um Caso de Uso, ou Use Case, em inglês. Ambas as denominações são utilizadas por autores nacionais. A Elipse é a notação de um Caso de Uso. O Caso de Uso é uma atividade. É a macroação que o ator realiza.
> 3. Isto é uma relação de dependência. Significa que Cadastrar Beneficiário depende diretamente da conclusão do Caso de Uso Cadastrar Cliente. A notação é uma seta tracejada, sem identificação aparente. Nada impede que haja uma identificação. A seta de dependência sempre parte do Caso de Uso que depende de outro em algum momento, apontando para o Caso de Uso que fornece a necessidade desejada. Esse comportamento vale para Inclusão e Extensão.
> 4. Aqui, Notação pressupõe uma inclusão de um Caso de Uso ou de parte dele em outro Caso de Uso. Neste exemplo, o Caso de Uso Calcular Pontos utilizará integralmente a forma de Cálculo de Pontos de Fidelidade que se encontra documentada em Calcular Fidelidade. Isso serve para não termos que digitar duas vezes específicações que se repetem em diferentes Casos de Uso. Evitamos escrever demais ou de menos a cada vez que escrevemos. É uma boa forma de centralizarmos a descrição dos Casos de Uso.
> 5. O formato anterior prevê que existe um cálculo em Calcular Bonificação, e que esse cálculo irá se estender, ampliando o significado de uma fórmula já existente no Caso de Uso Calcular Fidelidade. São válidas aqui, também, as observações feitas no tópico 4.

Revista Eletrônica em Gestão e Tecnologia
PRINT ISSN 2316-4972
ONLINE 2447-0422
FAQI
FACULDADE QI BRASIL

Imagem 4 – Ilustração explicativa dos itens usados em um Diagrama de caso de Uso (UML)



Notação do diagrama de Caso de Uso na UML.

Fonte: Ernani Medeiros (2004).

Para exemplificar o uso do diagrama de caso de uso, segue abaixo uma análise e elaboração de um caso simples.

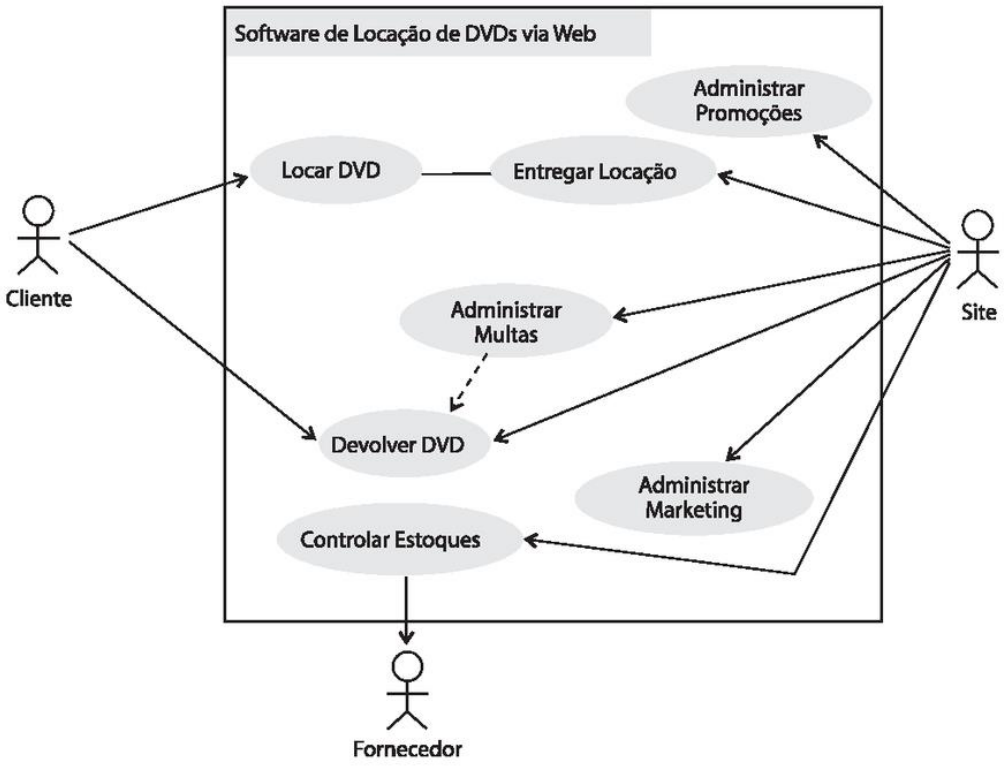
De acordo com Medeiros (2004, p. 26),

> Agora sabemos o que é um diagrama dc Caso de Uso e como utilizá-lo, então vamos criar o diagrama de Caso de Uso para o documento Visão (Figura 2.2). No próximo capítulo, faremos os diagramas de Casos de Uso para os níveis 1.
> Fazer esse diagrama é um exercício pessoal; não poderei olhar o seu diagrama e dizer que está errado. Um diagrama desse tipo, na verdade,

Revista Eletrônica em Gestão e Tecnologia
PRINT ISSN 2316-4972
ONLINE 2447-0422
FAQI
FACULDADE QI BRASIL

apenas informa visualmente os principais tópicos que podem ser explorados ao longo do seu estudo. Pode acontecer que uma elipse não tenha elementos suficientes para ser explorada em outro diagrama, o dc nível 1. Isso não fere em nada o seu *software* ou planejamento.
Alguns autores não colocam a figura do quadrado, para designar os limites - *boundary* - do *software*. Não há regra que se sustente aqui. A única regra é você saber que a notação UML deve ser mantida, os Casos de Uso devem representar macroatividades a serem realizadas; os atores, aqueles que executam essas atividades. A única notação não comentada é a da seta direcional, a qual apenas liga um Ator a um Caso de Uso, ou um Caso de Uso a outro Caso de Uso.

Imagem 5 – Exemplo de Diagrama de caso de Uso (UML)



Fonte: Ernani Medeiros (2004).

2.5 JSON (*JAVASCRIPT OBJECT NOTATION*)

Json é padrão de escrita de dados, criado com a finalidade de uso na transmissão e recepção de dados entre sistemas, embora o nome sugira que seu uso é explosivo da linguagem de programação *JavaScript*, ele pode ser usado por qualquer sistema de qualquer origem, pois o Json é apenas um modelo de

estruturação de dados, abaixo segue um exemplo da sintaxe do json sendo aplicada.

Modo de transmissão:

{"Pessoa": {"Nome": "João Silva","Filhos": {"NumeroFilhos": "3","ListaFilhos": ["João Silva Jr","maria Silva","Daniel Silva"]}}}

Modo identado para mais fácil compreensão visual:

{
"Pessoa": {
"Nome": "João Silva",
"Filhos": {
"NumeroFilhos": "3",
"ListaFilhos": ["João Silva Jr","Maria Silva","Daniel Silva"]
}
}
}

No exemplo é criado um objeto "Pessoa", que possui os atributos nome que está preenchido com o valor "João Silva", também possui o atributo filhos que também é um objeto, e possui os atributos "NumeroFilhos" preenchido com valor numérico "3", e o atributo "ListaFilhos" que está preenchido com um array (uma sequência composta de dados), tem que os seguintes valores "João Silva Jr", "Maria Silva" e "Daniel Silva".

3 MÉTODOLOGIA (DESENVOLVIMENTO)

Para o desenvolvimento desse projeto foram aplicadas as seguintes ferramentas e conhecimentos, que serão devidamente explicados nas subseções desta seção.

3.1 LINGUAGEM E IDE

A linguagem de programação escolhida para o desenvolvimento dos sistemas proposto por esse projeto é a linguagem Java. Foi usada J2SE para o desenvolvimento do sistema principal que é responsável por todas as funcionalidades principais propostas do sistema, desde o login a elaboração de menu, telas e suas configurações, Para a funcionalidade específica de atualização de arquivos de configuração do sistema foi criado um sistema com característica de *Web Service* baseado em J2EE. Para o desenvolvimento de ambos a ferramenta escolhida para o desenvolvimento do seu código fonte foi a IDE (Ambiente de Desenvolvimento Integrado) Netbeans, uma ferramenta muito popular para o desenvolvimento Java, ela também proporciona recursos para desenvolvimento em outras linguagens.

3.2 ARQUITETURA DE *SOFTWARE*

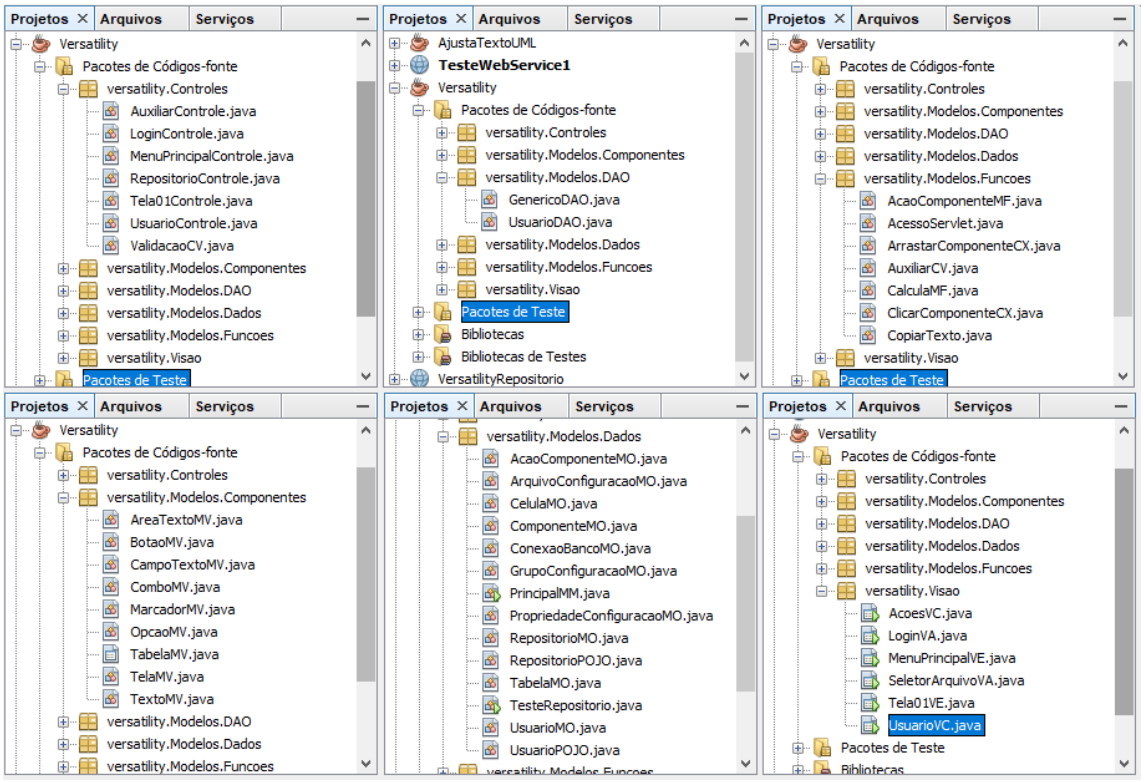
A arquitetura adotada no desenvolvimento da estrutura do *software* foi a MVC, a forma convencionalmente usada para aplicar essa arquitetura em projetos Java seria separando as classes das três naturezas em três pacotes raiz: Modelo, Visão e Controle.

Além dois três pacotes também seria uma boa prática criar subpacotes caso o número de classes de um pacote aumente muito, e hajam grupos de classes com maiores especificidades para se ter mais organização, como foi o ocorrido no pacote Modelos, nele ainda foi criado os pacotes Componentes, onde ficam as classes responsáveis pelas configurações dos componentes de tela que podem ser criados no sistema (exemplo rótulos, campos de texto, botões), o pacote DAO, onde ficam as classes que seguem o padrão de projeto DAO que viabilizam as requisições e retornos do banco de dados, o pacote Dados, onde ficam as classes usadas para a instanciação de objeto de dados, objetos de comunicação com banco (padrão *Gateway*) e de movimentação de dados (padrão DTO), e por fim o pacote Funções,

Revista Eletrônica em Gestão e Tecnologia
PRINT ISSN 2316-4972
ONLINE 2447-0422
FAQI
FACULDADE QI BRASIL

responsável pela execução de ações do sistema, cálculos, ação de componentes e etc.

Imagem 6 – Pacotes e Classes do Versatility



Fonte: Autor

3.3 PADRÕES DE PROJETO

Para a persistência dos dados será utilizado o padrão DAO para construção de comandos SQL, e para efetuar os comandos das classes DAO no banco de dados será usado uma classe no padrão *Gateway*, imbuída de toda a responsabilidade de comunicação com o banco de dados, além desses padrões, também serão usados os padrões DTO para transporte de dados.

3.4 BANCO DE DADOS

A SGBD escolhida para a criação da instância do banco de dados foi o MySQL, na inicialização do sistema caso não tenha um caminho de banco de dados salvo será questionado ao usuário se ele deseja selecionar o caminho de uma banco de dados já existente os se ele prefere que seja criado um novo, após sistema criará um usuário no banco de dados próprio para uso interno do sistema, e as tabelas de usuário e permissões para os usuário do sistema, esses usuários serão indiferentes aos usuários do banco de dados por questões de segurança.

4 ANÁLISE DOS RESULTADOS (DESENVOLVIMENTO)

Será apresentado os resultados dos esforços empregados a esse projeto nas subseções seguintes.

4.1 MODELAGEM DO SISTEMA

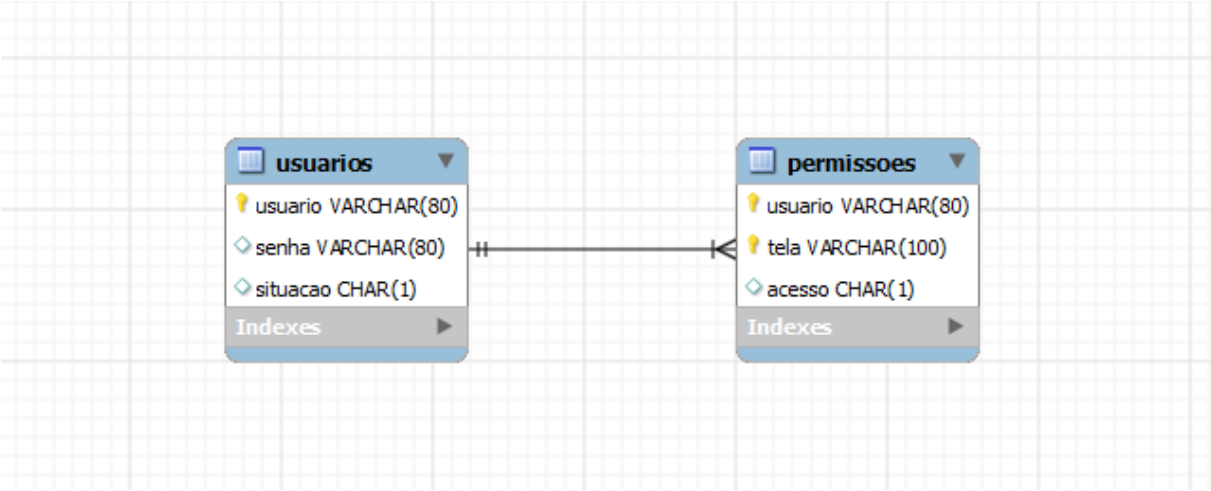
Nesta subseção será exibido os diagramas modelados seguindo os moldes da UML.

4.1.1 Modelo Lógico do Banco de Dados

A imagem abaixo mostra o diagrama de tabelas padrões do sistema no banco de dados, as duas tabelas apresentadas tem o objetivo de armazenarem dados referentes ao logon de usuários no sistema, porém o sistema permite a criação, manutenção e uso de novas tabelas para armazenar quaisquer dados que o usuário precisar.

Revista Eletrônica em Gestão e Tecnologia
PRINT ISSN 2316-4972
ONLINE 2447-0422
FAQI
FACULDADE QI BRASIL

Imagem 7 – Diagrama do modelo lógico padrão do banco de dados



Fonte: Autor

O diagrama mostrado na imagem 7 esboça duas tabelas do banco de dados do Versatility, que serão padrão em todas aplicações do sistema, a tabelas usuários será usada para armazenar registros de logon de usuários no sistema, nela é encontrado o campo usuário que guardará o nome de acesso do usuário, essa coluna tem uma chave na frente porque será a coluna de identificação dos registros da tabela, em banco de dados ela é chamado de chave primária, a regra da chave primária é que a informação nesta coluna dela não pode repetir em outros registros, dessa forma o registro se torna único e pode ser encontrado através da chave, e nessa situação essa regra impede o cadastro de usuários com mesmo nome de acesso, assim a regra de integridade do banco de dados também garante a integridade de uso do sistema.

Além do usuário há a de senha, onde será armazenado a senha do usuário, e a coluna situação que dirá se o usuário está ativo ou inativo para acessar o sistema, quanto a tabela de permissões podemos ver que há uma linha a ligando a tabela de usuários, isso se deve as permissões terem um vínculo com o usuário, os símbolos nas extremidades das linhas que ligam as tabelas reflete a cardinalidade entre elas, esse termo significa como é a ligação entre as tabelas, na extremidade conectada a

tabela de usuários há dois traços paralelos, eles sinalizam que um registro na tabela de permissões se refere a apenas um usuário não mais, a contrario o símbolo encontrado na extremidade da tabela de permissões se sinaliza que o registro da tabela usuário pode ser referenciado por mais de um registro.

Quanto às colunas da tabela de permissões é apresentado uma coluna usuário, ela não só compõem a chave primária junto com a coluna tela, como também é uma chave estrangeira, a chave estrangeira é a ou as colunas referentes a chave primária de uma tabela ligada a ela, os sejá um registro coluna usuário da tabela de permissões terá o mesmo dados de um registro da coluna usuário da tabelas usuários, assim é possível saber a qual usuário a permissão se refere, quanta a coluna tela, ela irá guardar o nome da tela qual o registro de permissão se refere, ela compõe a chave primária junto a coluna usuário, pois haverão mais de um registro de permissões para um usuário como mencionado antes, com a coluna tela como chave poderá haver um registro de permissão para cada usuário com cada tela do sistema.

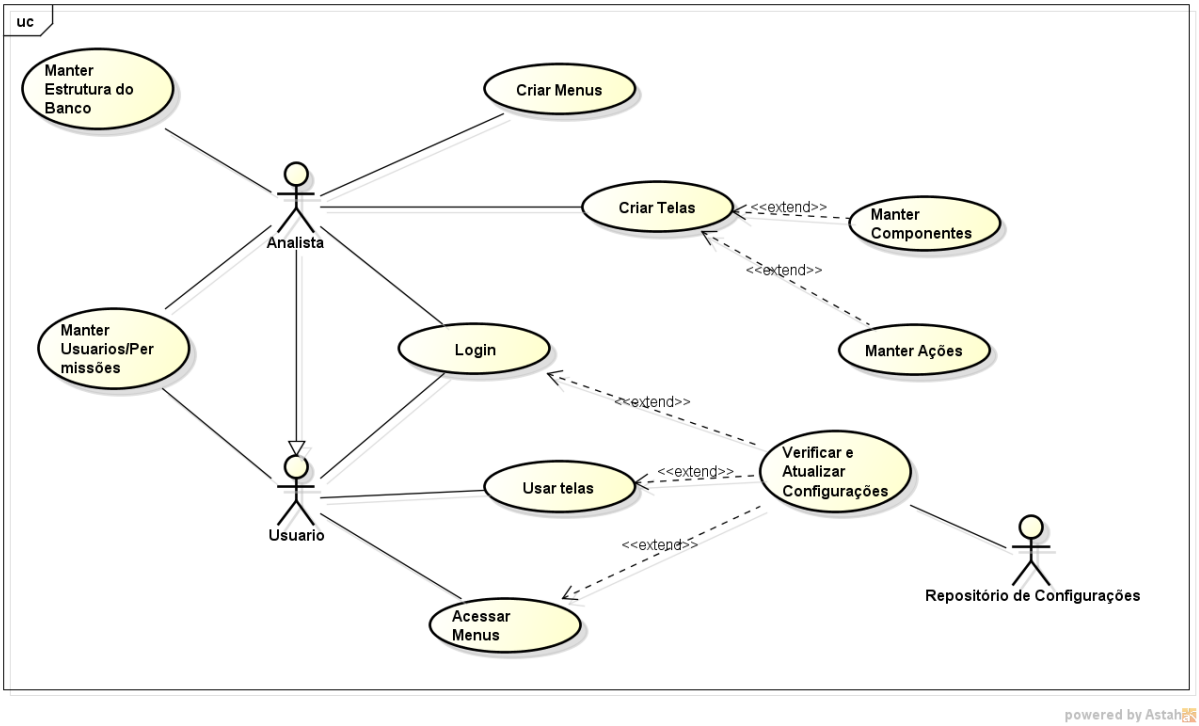
E por último a coluna acesso, ela dirá se o usuário tem permissão de acesso a tela ou não. Ao lado no nome de cada coluna há o tipo de dado da coluna, nas colunas apresentadas podemos ver dois tipo, VARCHAR e CHAR, VARCHAR é um tipo de dados usado para colunas que irão guardar caracteres de texto (letras, pontuação e caracteres especiais) de um tamanho variável no limite do número ao lado do tipo entre parênteses, a tipo CHAR também é para caracteres de texto, mas que serão usados para um tamanho específico, como na colunas situação e acesso que guardarão apenas a letra S ou N indicando a permissão de acesso ou atividade do usuário, além dos tipo usados nessas tabelas há vários outros em banco de dados para tipos diferentes como números, imagens etc.

4.1.2 Diagrama de Caso de Uso

A imagem 8 mostra o diagrama de caso de uso do sistema principal do projeto, e logo após uma explicação dos seus componentes.

Revista Eletrônica em Gestão e Tecnologia
PRINT ISSN 2316-4972
ONLINE 2447-0422
FAQI
FACULDADE QI BRASIL

Imagem 8 – Diagrama de Caso de Uso do Versatility



Fonte: Autor

O diagrama apresentado na imagem 8 mostra a interação dos usuários com as funcionalidades do sistema, nele podemos ver três atores.

O ator mais acima do diagrama é o Analista, ele representa o analista de sistemas que irá configurar o sistema para os usuários, através da implantação ou da manutenção, ele poderá se autenticar no sistema pela tela de login que liberará todas as funções de todos os casos de uso ligados ao seu ator no diagrama, além de todas as funções dos casos de uso ligados ao ator usuário, isso é sinalizado pela seta que sai do ator analista e aponta para o ator usuário. O analista pode, manter estrutura do banco de dados, o que significa que ele pode ver, criar, alterar ou excluir tabelas, colunas ou qualquer outra função relacionada ao banco de dados, além pode manter usuários e permissões de usuários, significa que ele pode ver, criar, alterar ou excluir usuários e permissões de acesso de usuários. O analista também tem a opção de criar menus e opções de menus na tela principal que darão

acesso às telas personalizadas que ele pode criar, podendo livremente criar, alterar ou excluir componentes de tela como campos de texto, botões e etc. e ele também tem ferramentas para criar ações no sistema para buscar ou cadastrar informações no banco de dados, fazer cálculos entre outras opções.

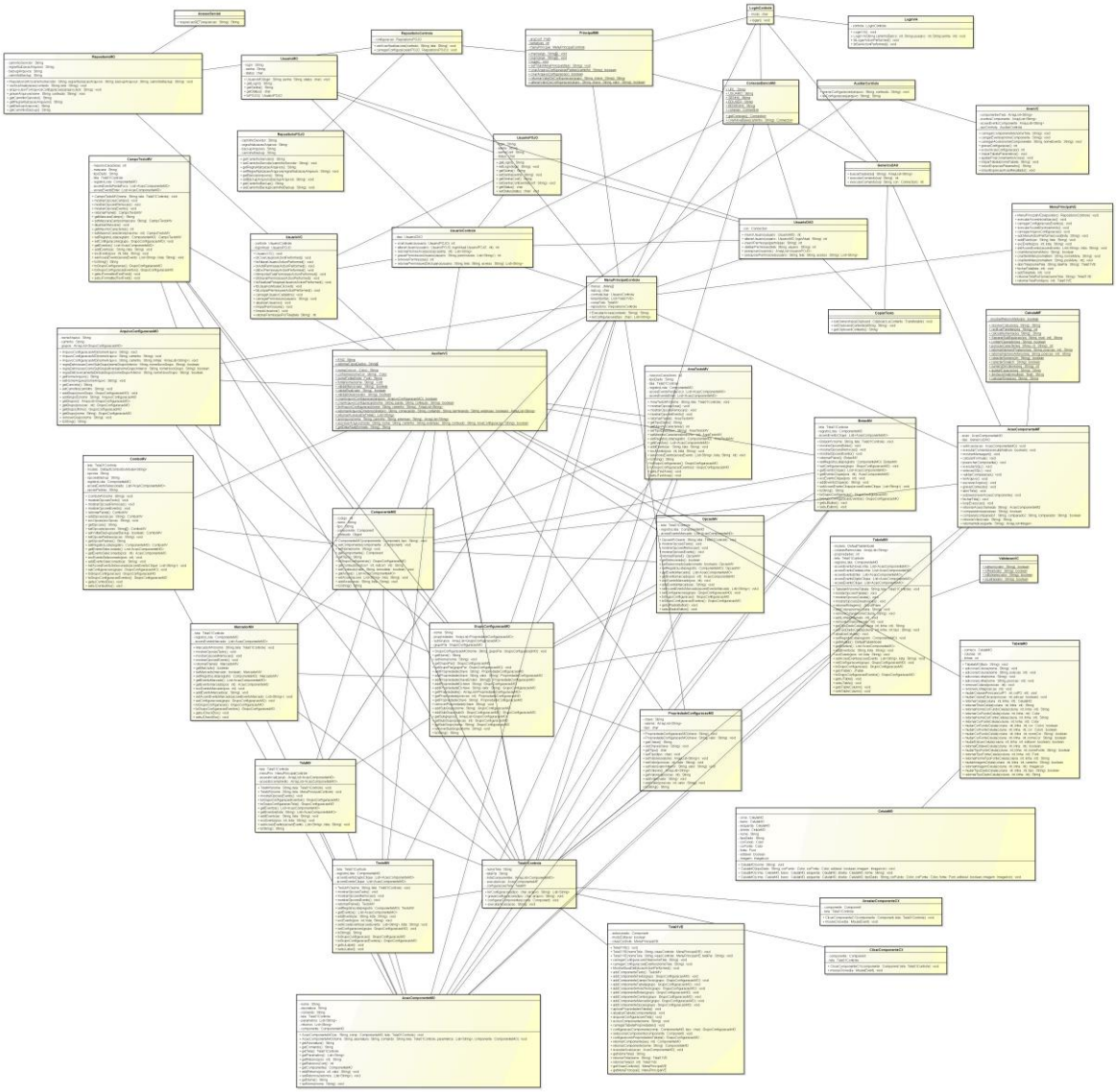
Quanto ao usuário comum visto no diagrama como o ator usuário, ele consegue fazer o login no sistema, mudar sua senha, inativar se próprio usuário, acessar os menus configurados e através deles ter acesso às telas personalizadas pelo analista, lembrando que o analista consegue fazer as mesmas ações que o usuário comum.

E por fim temos um último ator no diagrama, o Repositório de Configurações, esse ator não representa uma pessoa, mas sim o segundo sistema do projeto o Versatility repositório, ele é acessado elas pelas personalizadas do sistema para que elas copiem as configurações de tela no servidor, e atualizem suas configurações para as estabelecidas no servidor, assim quando for feita alguma alteração em alguma configuração no sistema, ela pode ser feita apenas no servidor e as outras estações se atualizarão automaticamente atravès do Repositório.

4.1.3 Diagrama de Classes

Revista Eletrônica em Gestão e Tecnologia
PRINT ISSN 2316-4972
ONLINE 2447-0422
FAQI
FACULDADE QI BRASIL

Imagem 9 – Diagrama de Classes do Versatility



Fonte: Autor

No diagrama de Classes do Versatility podemos ver todas as classes do sistema e como elas estão conectadas entre si, quando executado a primeira classe a ser chamada é a PrincipalMM, ela é responsável por iniciar e manter as configurações de inicialização do sistema e por chamar as classes de login LoginControle que criação a tela de autenticação instanciando a classe LoginVA,

elas verificam a autenticação montando um comando SQL pela GenericoDAO e executando o comando pela ConexaoBancoMO.

Após a autenticação, caso configurado, a principalMM verifica as atualizações de configurações pelas classes RepositorioControle, RepositorioMO, RepositorioPOJO, que são usadas para chamar a AcessoServlet. Na próxima etapa a PrincipalMM solicita a Criação da Tela de Menu Principal (MenuPrincipalVE) pela classe MenuPrincipalControle, Pelo Menu Principal o sistema possibilita o acesso e manutenção a telas customizáveis (Tela01VE) que é acessada pela classe Tela01Controle.

A tela criada pela classe Tela01VE permite a criação e manutenção de objetos de tela, essas ações são viabilizadas pelas classes ComponenteMO, BotaoMV, TextoMV, CampoTextoMV, AreaTextoMV, ComboMV, MarcadorMV e TabelaMV (tem os dados controlados por uma coleção de dados interna construída com as classes TabelaMO e CelulaMO), essas classes servem como modelo de componentes e estendem os componentes de objetos de tela padrão do Java *Swing*, e possibilita o reposicionamento dos objetos em tela pelo arrastar do mouse via as classes ArrastarComponenteCX e ClicarComponenteCX.

Nesses componentes, além de serem configuradas características visuais, também permitem a configurações de comandos de ações que são moldadas pela classe AcaoComponenteMO, executadas pela AcaoComponenteMF, utilizando outras classes dependendo a ação como as de componentes de tela citadas anteriormente para alterações visuais ou de conteúdo, a classe AuxiliarControle e AuxiliarCV para escrita e leitura em arquivos, a classe calculaMF para solução de cálculos e aplicação de fórmulas matemáticas, e a GenericoDAO e ConexaoBancoMO para uso de comandos em banco de dados.

Todos os dados de configuração são moldados, armazenados e arquivados, por uma coleção de dados própria do sistema criada através das classes ArquivoConfiguracaoMO, GrupoConfiguracaoMO, ProriedadeConfiguracaoMO e auxiliarCV, A Manutenção de Usuários é feita através da tela UsuarioVC, que recebe e envia dados para a UsuarioControle ela classe UsuarioPOJO, a classe

UsuarioControle molda os dados de usuário e os grava no banco de dados pela classe UsuarioMO, UsuarioDAO e ConexaoBancoMO.

4.2 INTERFACES DO SISTEMA

Nesta subseção serão apresentadas as telas do sistema e como usá-las.

4.2.1 Requisitos Para Uso do Versatility

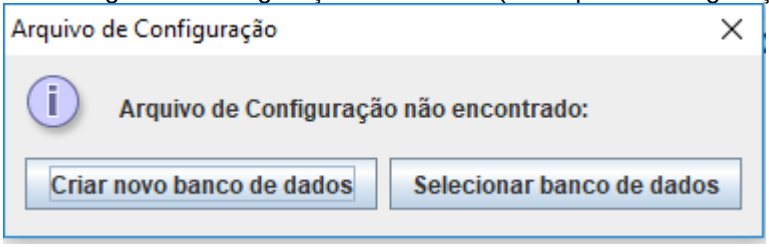
O sistema é executado na plataforma Windows, para seu uso é necessário ter o pacote JDK e o Mysql instalados com as permissões necessárias.

4.2.2 Configuração Inicial

É aconselhável criar um pasta própria para o sistema, pois ele usará seu diretório para armazenamento de diversos arquivo de configuração, o usuário do windows deve ter totais permissões na pasta, além de permissão para acessar o banco de dados, ele estando local ou outra máquina na rede interna.

Ao executar o sistema pela primeira vez são apresentadas duas opções, é possível selecionar um banco de dados já criado para sistema, ou solicitar a criação de um na máquina. Caso seja escolhida a segunda opção, virá como padrão apenas um usuário cadastrado, o usuário Administrador com a senha "administrador", é aconselhável mudar a senha logo após por questões de segurança.

Imagem 10 – Mensagem de configuração do sistema (1ª etapa da configuração)



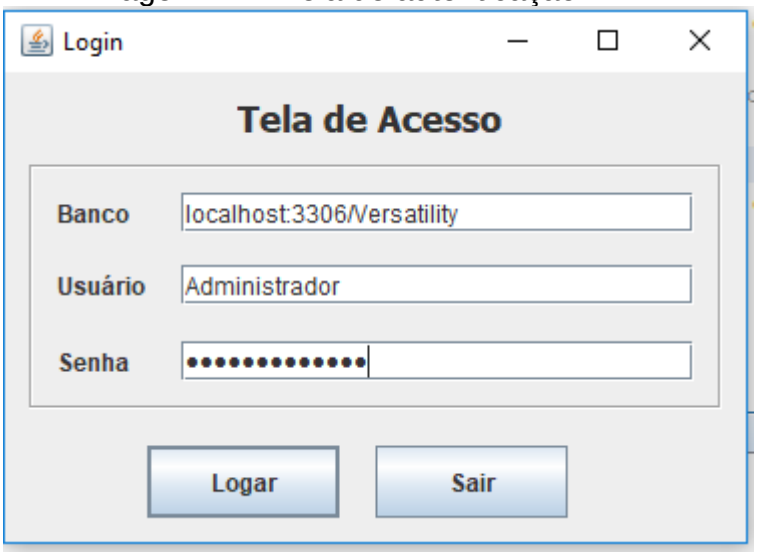
Fonte: Autor

4.2.3 Login, usuários e Permissões

Após a Configuração inicial do sistema ao executá-lo será exibida uma tela de login, nela terá os campos de usuário, senha e dependendo da configuração escolhida também o caminho do banco de dados (para mostrar ou ocultar campo basta alterar parâmetro ocultar caminho banco com sim ou não no arquivo configArq.ini)

Após preencher dados e clicar em Logar o menu principal irá abrir caso banco seja válido e tenha o usuário com a senha correta.

Imagem 11 – Tela de autenticação

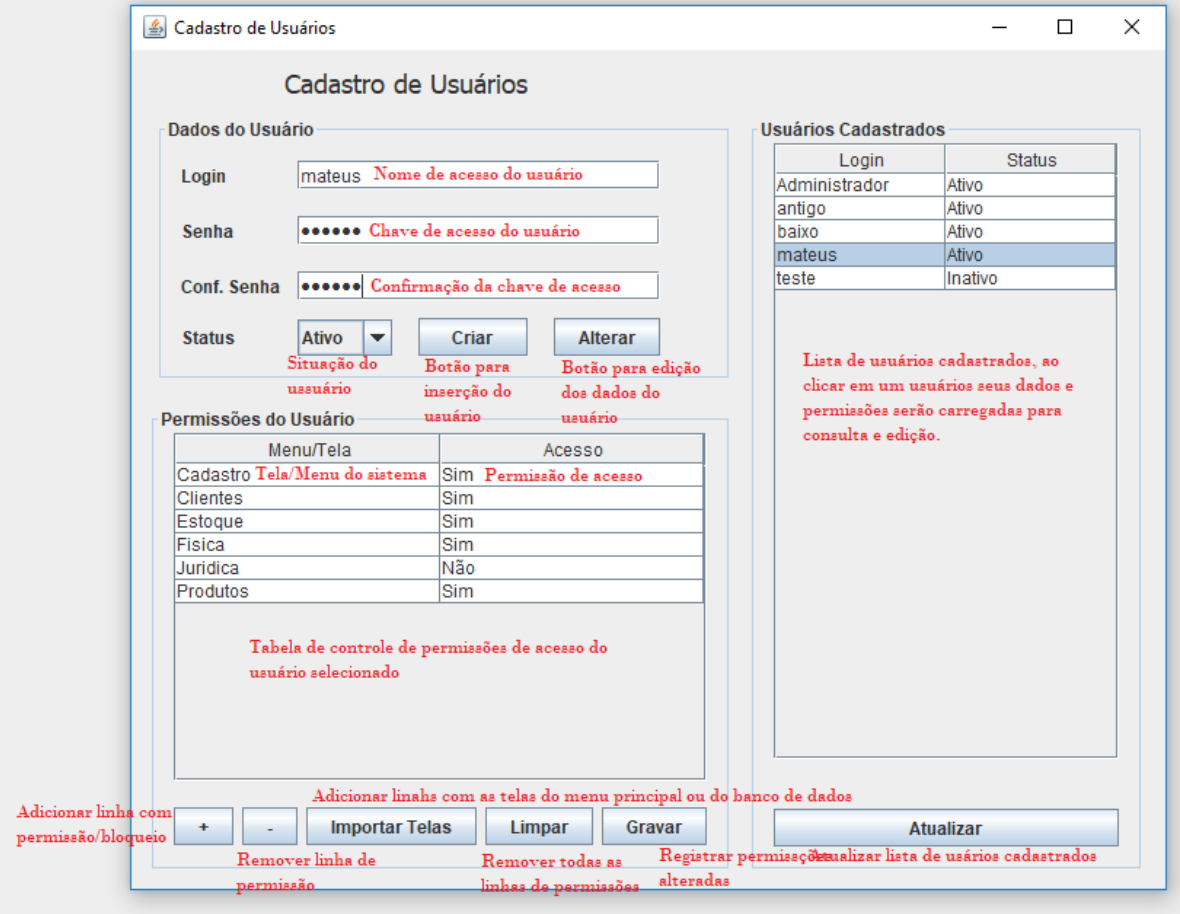


Fonte: Autor

No menu principal, no canto inferior esquerdo da tela terá o nome do usuário logado, ao clicar nele o sistema abrirá o cadastro de usuários, um usuário comum consegue apenas alterar seus dados (login, senha, status) porém não será possível alterar os outros usuários ou suas permissões de acesso, ou criar usuários, para essas ações será necessário logar com o usuário Administrador.

Revista Eletrônica em Gestão e Tecnologia
PRINT ISSN 2316-4972
ONLINE 2447-0422
FAQI
FACULDADE QI BRASIL

Imagem 12 – Tela Manutenção de usuários



Fonte: Autor

A tela de Cadastro de Usuários, permite a criação e alteração de usuários e de suas permissões de acesso a menus e telas e acesso a todos os usuários para o usuário Administrador, para os de mais usuários ela permite apenas a alteração de senha e inativação do próprio cadastro.

4.2.4 Criação de Menus

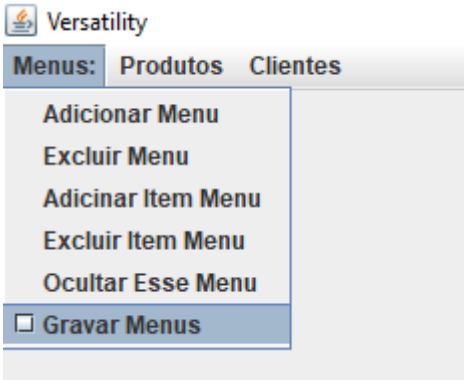
Após logar no sistema, ele exibirá o menu principal, ele serve para dar acesso as outras telas do sistema, sendo as padrões como a tela de manutenção de usuários e permissões, como também os formulários customizados, para criá-los

primeiro é necessário ser criado um item de menu para acessá-lo, na barra de menu na parte superior há o menu "Menus" nele são encontradas as opções de edição da barra de menu, é possível adicionar ou remover menus ou itens de menu, porém para essas configurações serem salvas, antes de fechar o sistema é preciso ter a opção gravar menus habilitada, ela também se encontra na barra menus mas só pode seu usada pelo usuário administrador, outra opção encontrada na barra de menu é "ocultar essa barra" optando por ela a barra de menus não aparecerá mais, e para reabilitá-la será necessário apagar a linha de configuração [NaoEditar] no arquivo ltPrincipal.ini.

Após a criação de um menu e um item de menu para dar acesso a uma nova tela, basta clicar no item de menu e será criada um novo formulário editável, com o nome do item de menu (obs. itens de menu com o mesmo nome abrirão o mesmo formulário, mesmo que esteja em menus diferentes).

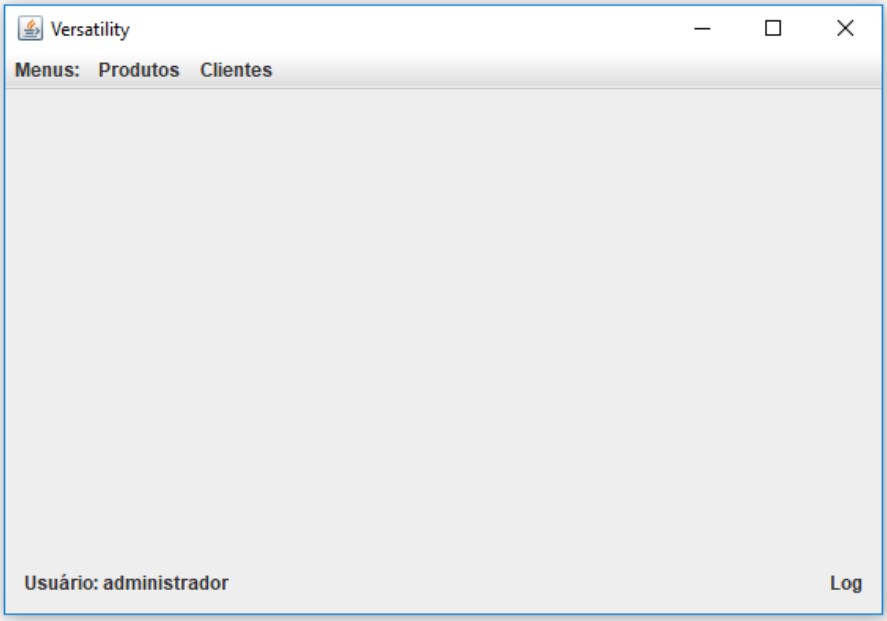
No menu principal também é possível definir ações padrões para serem executadas automaticamente ao entrar no sistema ou sair dele, para fazer essas configurações basta dar duplo-clique na tela principal, explicações sobre como criar comandos de ações na subseção Criação de comandos de ações em tela.

Imagem 13 – Menu da tela principal



Fonte: Autor

Imagem 14 – Tela Principal



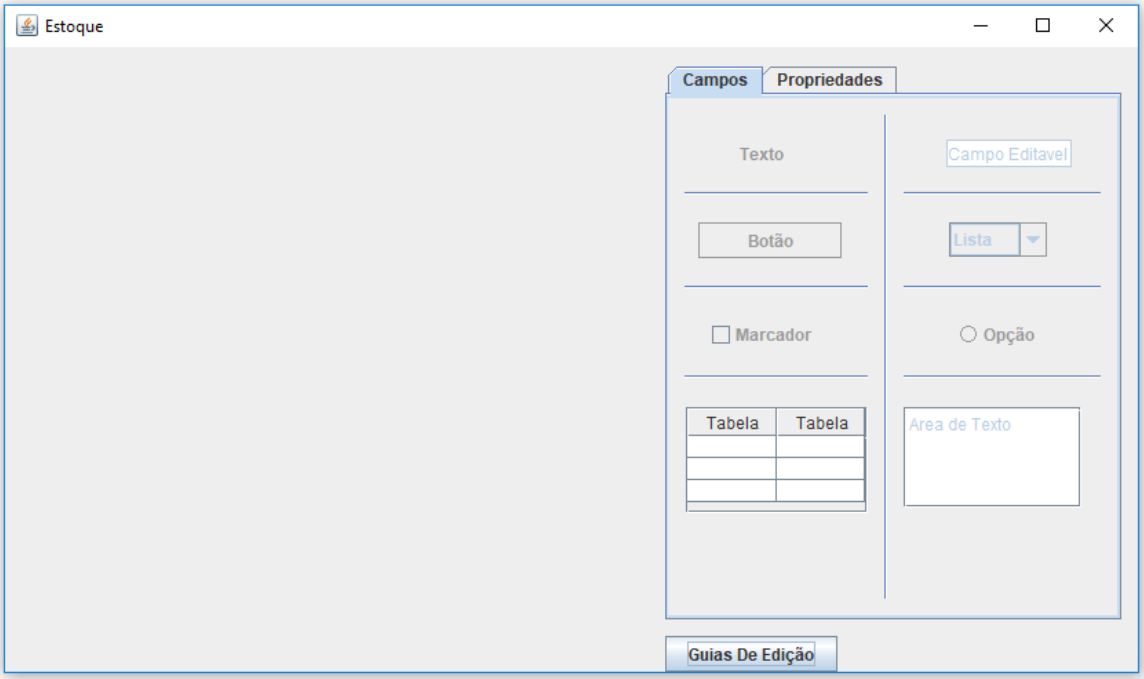
Fonte: Autor

4.2.5 Criação de Componentes de Tela

Ao acessar os formulário editáveis através do menu principal eles virão vazios, sem nenhum componente em tela, porém se o usuário logado for o Administrador, no canto superior direito da tela terá o botão Guias de edição, clicando nele será exibido um painel lateral com abas que permitem a criação e configuração de componentes de tela, a primeira aba (Campos), contêm os componentes que podem ser adicionados a tela (Rótulos de texto, Campos de Edição de Texto, Combos de Opções, Botões, Tabelas, Marcadores, Opções de Seleção, Área de Edição de Texto), para adicioná-los a tela basta clicar, e o componente será adicionado na primeira posição do formulário, após basta arrastar o componente adicionado com o mouse para uma posição a ser definida dentro do formulário. Cada componente tem suas propriedades ex. Posição, tamanho, Conteúdo...

Revista Eletrônica em Gestão e Tecnologia
PRINT ISSN 2316-4972
ONLINE 2447-0422
FAQI
FACULDADE QI BRASIL

Imagem 15 – Formulário Editável Novo



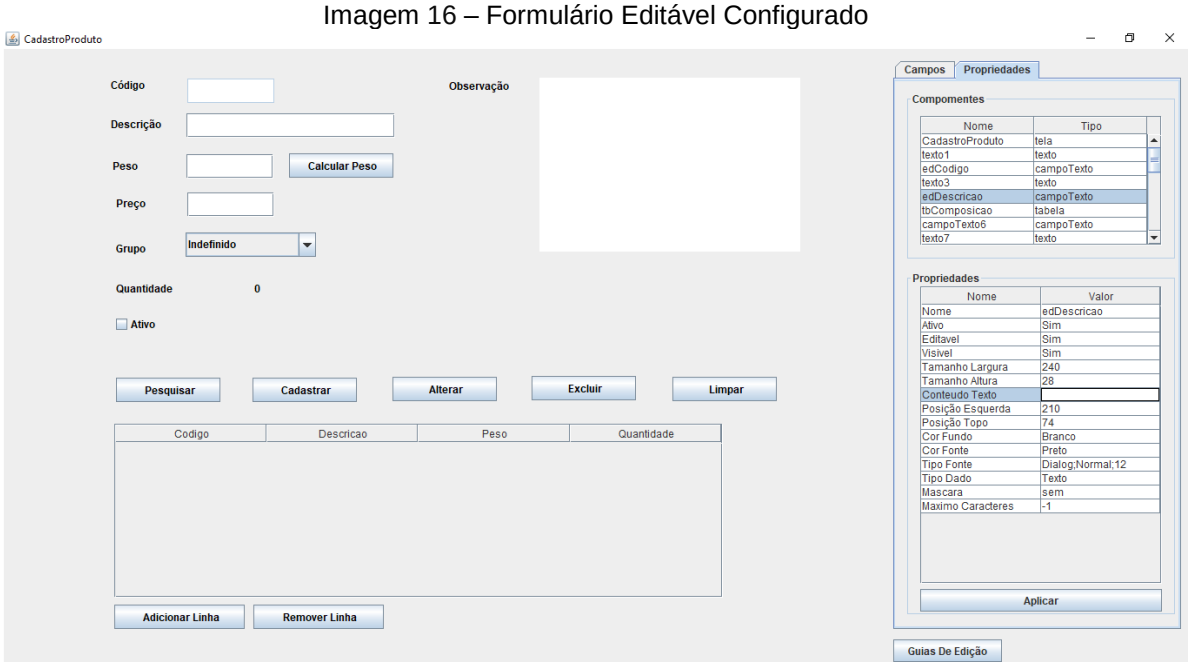
Fonte: Autor

Ao criar o componente ele vem com valores padrão nessas propriedades, para alterá-las há algumas formas, Na segunda aba (Propriedades) na guia de edição, é possível selecionar o componente na lista de componentes de tela, ou apenas clicar nele no formulário para carregar suas propriedades na segunda tabela da aba, ela mostra as propriedades do componente e seu valor atual, e permite que você edite os valores e aplicá-los nos componentes, uma propriedade interessante de se alterar é o nome do componente, pois isso pode facilitar a criação dos comandos que usarão esses componentes, outras formas de alterar as propriedades além da guia de Edição é pelo clique direito do mouse nos componentes, e pelas Teclas F1, F2, F3, F4,

Após a criação do formulário essas configurações serão salvas em arquivos de configuração na pasta do sistema, nos arquivos tlNomeDaTela.ini, para que essas configurações sejam gravadas a guia de edição tem que estar aberta na tela, caso contrário as alterações não serão arquivadas.

Revista Eletrônica em Gestão e Tecnologia
PRINT ISSN 2316-4972
ONLINE 2447-0422
FAQI
FACULDADE QI BRASIL

Imagem 16 – Formulário Editável Configurado



Fonte: Autor

4.2.6 Criação de Comandos de Ações em Tela

Os comandos são uma parte essencial do sistema, porém também a mais complicada, eles são responsáveis pelas ações executadas pelo sistema, podem ser vinculados em eventos de telas e componentes, para criar comandos na tela principal, basta dar um duplo-clique em qualquer área vazia dele e aparecerá as opções de eventos onde você pode criar comandos, no caso são no evento de abertura da tela, ou seja após ser efetuado o login no sistema, ou no evento de encerramento, quando o sistema estiver sendo fechado. Para acessar os eventos de uma tela de formulário editável, basta clicar com o botão direito em qualquer parte vazia do formulário e escolher a opção de configurações de eventos, os eventos da tela de formulário igualmente a tela de menu principal são inicialização e encerramento. Para acessar as ações dos componentes, você pode clicar com o botão direito em cima do componente ou pressionar F6 após selecionar o componente, cada tipo de componente tem eventos diferente exemplos dele são os

eventos de clique no componente, duplo-clique, digitação em campos de edição de texto, e etc...

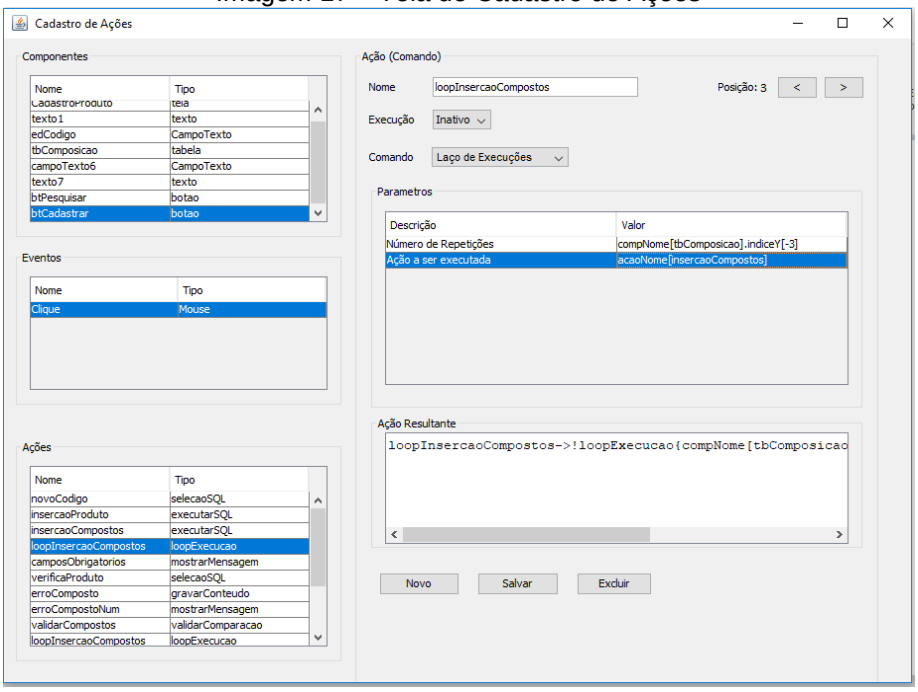
Após escolher o evento em tela ou componente, é exibida uma lista com os comandos já criados no evento até o momento, e abaixo tem opções para adicionar remover ou alterar esses comandos.

Os comandos tem a seguinte sintaxe e são compostos de:

nome->ação{parâmetro1;parâmetro2;parâmetro3...};

O nome é opcional, é possível criar comandos com apenas ação e parâmetros (no caso sem ->), porém é interessante colocar nomes nos comandos para eles poderem ser acessados mais facilmente por outros comandos. A ação é a parte que defino o que o comando irá fazer, segue abaixo as opções de comandos disponíveis atualmente e a explicação de seu funcionamento, outro ponto importante sobre é que quando o comando for criado para ser executado apenas por outros comandos e não automaticamente, deve ser posto um "!" na frente da ação, isso deixar o comando inativo para não ser executado diretamente quando o evento da tela ou componente for acionado.

Imagem 17 – Tela de Cadastro de Ações



Fonte: Autor

Lista de Ações

Modelo da lista:
Nome da Ação na tela de cadastro (Nome da ação no arquivo)
Descrição da ação
Lista de Parâmetros da ação
Lista de Retornos da ação

Mostrar Mensagem (mostrarMensagem):
Chama Caixa de Mensagem
parâmetro0: texto de mensagem

Calcular Fórmula (calcularFórmula):
Calcula equação matemática e armazena resultado
parâmetro0: fórmula a ser calculada
retorno0: armazena resultado

Preencher Componente (preencherComponente):
Preenche componente com um determinado conteúdo
parâmetro0: componente (onde componente está localizado)
parâmetro1: conteúdo a ser inserido
parâmetro2: remodelar componente (add ou exc linhas table) S-sim, N-não

Executar Comando SQL (executarSQL):
Executa comando sql no banco de dados como ex. *create,insert,delete*
parâmetro0: expressão sql
parâmetro1,3,5...: parâmetro interno do comando sql
parâmetro2,4,6...: valor correspondente ao parâmetro anterior
retorno0: numero linhas afetadas com o comando

Executar seleção SQL (selecaoSQL):
Executa seleção sql comando select no banco de dados
parâmetro0: comando sql
parâmetro1,3,5...: parâmetro interno do comando sql
parâmetro2,4,6...: valor correspondente ao parâmetro anterior

Validar Comparação (validarComparacao):
Analisa se expressão de comparação(ões) é verdadeira ou falsa e executa outras ações de acordo com o resultado
parâmetro0: expressão de comparação
parâmetro1: ações a serem executadas caso resultado seja verdade
parâmetro2: ações a serem executadas caso resultado seja falso

Ler Arquivo (lerArquivo):
Acessa dados de um arquivo externo e os armazena
parâmetro0: nome do arquivo
parâmetro1: caminho do arquivo
parâmetro2: extensão do arquivo
parâmetro3: caso informado é usado para filtrar linhas do arquivo
retorno0,1,2...: em cada retorno contém uma das linhas lidas

Escrever em Arquivo (escreverArquivo):
Exporta conteúdo para um arquivo de texto
parâmetro0: modos: "Escrever","Reescrever","Adicionar","Excluir"
parâmetro1: nome do arquivo
parâmetro2: caminho do arquivo
parâmetro3: extensão do arquivo
parâmetro4: conteúdo do arquivo

Gravar Conteúdo (gravarConteudo):

Armazena conteúdo no comando
parâmetro0,1,2...:conteúdo que será guardado no retorno0,1,2
retorno0,1,2...: guarda conteúdos dos parâmetros0,1,2

Laço de Execuções (*loop*Execucao):
Executa uma determinada ação um determinado número de vezes
parâmetro0: número de vezes que a ação será executada
parâmetro1: Ação que será executada

Abrir Tela (AbrirTela):
Abre nova tela do sistema
parâmetro0: nome da tela a ser aberta

Sobrescrever Ações (sobrescreverAcoes):
Sobrescrever lista de ações de um componente
parâmetro0: componente a ter ações alteradas (sua localização)
parâmetro1: Nome de evento da lista de ações a ser alterada
parâmetro2: ações (nome ou id de ações registradas ao componente)

Fechar Tela (fecharTela):
Fechar uma tela do sistema
parâmetro0: tela a ser fechada (localização da tela)

Como pode ser observado, o número de parâmetros e o tipo de informação que eles deve conter em cada posição depende ca sintaxe da ação.

Os parâmetros dos comandos vezes obrigam vezes possibilitam o uso de outros componentes, ou o conteúdo de outros componente e até as ações e retorno de outros componentes, para passar essas informações existem funções próprias para uso interno na execução dos comandos, são elas:

telaId["número"]

-retorna o formulário na posição informada, procurando na lista formulários abertos.

telaNome["nomeDoFórmulario"]

-retorna o formulário com o nome informado, procurando na lista formulários abertos.

compId["número"]

-retorna o componente (objeto de formulário) na posição informada, procurando na lista de componentes do formulário.

compNome["nomeDoComponente"]

-retorna o componente (objeto de formulário) com o nome informado, procurando na lista de componentes do formulário.

acaoId["número"]

-retorna a ação (comando) da posição informada, procurando na lista de ações do componente.

acaoNome["nomeDaAcão"]

-retorna a ação (comando) com nome informado, procurando na lista de ações do componente.

retornoId["número"]

-retorna o valor de retorno em uma ação na posição informada.

indiceX["número"]

-retorna a parte de um valor de acordo com a posição informada (coluna), as partes são separadas por ";"

indiceY["número"]

-retorna a parte de um valor de acordo com a posição informada (linha), usado quando valor for uma matriz.

As funções puxam por padrão a tela de formulário aberto, o componente acionado, e ação executada, porém quando é necessário acessar informação de outra tela ou outro componente, é possível usar funções em sequência separados por "." exemplo: acaoNome[contaCompostosErros].retornoId[0], nessa cadeia de funções o retorno da posição zero será retornado da ação indicada ao invés da ação

em execução, o mesmo pode ser feito com as demais funções desde que esteja na ordem correta.

As funções podem ser concatenadas ("encaixadas") com valores e outras funções usando o "&" ex.: erroCompostoNum->!mostrarMensagem{Composto linha &acaoNome[erroComposto].retornoId[0]&, produto inexistente ou inativo, ou quantidade inválida!}; essa função mostrará uma mensagem com o seguinte texto: "Composto linha 1, produto inexistente ou inativo, ou quantidade inválida!" onde 1 supostamente é o retorno da ação de nome erroComposto.

5 CONSIDERAÇÕES FINAIS

Este trabalho apresentou uma métodologia que possibilita ser seguida uma nova abordagem em projetos de *software*, através de um sistema que pode ser configurado para atender necessidades de *software* diversificadas, pelas suas ferramentas próprias de personalização, tornando o alinhamento de procedimento dos usuários configurável apenas por analistas, sem a necessidade de envolvimento da programação.

Como sugestão para trabalhos futuros, poderia ser criado uma versão mais aprimorada do Versatility, com mais opções de ações no sistema, mais opções de componentes, como também uma versão web do sistema.

REFERÊNCIAS

AMADEU, Claudia Vicci. **Banco de Dados**. São Paulo: Pearson, 2015.

COSTA JUNIOR, Eudes Luiz. **Gestão e, Processos produtivos**. Curitiba: Ibpex, 2008.

DEITEL, Paul e Harvey. **Java Como Programar**. 10. ed. Rio de Janeiro: Pearson, 2016.

GAMMA, Erich et al. **Padrões de Projeto**: soluções reutilizáveis de *software* orientado a objetos. Boston: Addison-Wesley, 2000.

HORSTMANN, Cay S. **Core Java**. 8. ed. São Paulo: Pearson, 2009.

MEDEIROS, Ernani. **Desenvolvendo *Software* com UML Definitivo 2.0**. Rio de Janeiro: Pearson, 2004.

PAGE-JONES, Meilir. **Fundamentos do Desenho Orientado a Objeto com UML**. São Paulo: Makron Books, 2001.

PRESSMAN, Roger S. **Engenharia de *Software***. 6. ed. New York: Mcgraw-Hill, 2005.
REENSKAUG, Trygve. ***Working with objects***. Oslo: Manning Greenwich, 1996.

SHIGUNOV NETO, Alexandre CAMPOS. **Introdução à gestão da qualidade e produtividade**: conceitos, história e ferramentas. Curitiba: InterSaberes, 2016.

SOMMERVILLE, Ian. **Engenharia de *Software***. 9. ed. São Paulo: Pearson Addison, 2005