DOUGLAS HENRIQUE PASSOS PÁDUA

8,0 (oito)

GERENCIAMENTO DE INFORMAÇÕES TÉCNICAS DE PROJETO

Trabalho de conclusão de curso apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Engenheiro Mecatrônico

Área de Concentração:
Informação do Produto

Orientador:
Marcos Ribeiro Pereira Barreto

São Paulo
1999

Pádua, Douglas Henrique Passos
Gerenciamento de Informações Técnicas de Projeto. São Paulo, 1999.
112 p.

Trabalho de conclusão de curso – Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecânica – Automação e Sistemas

1. Informação do produto 2. Gerenciamento de banco de dados utilizando Java e SQL. I. Universidade de São Paulo. Departamento de Engenharia Mecânica – Automação e Sistemas. I I. t

Aos meus pais, José Deusmir e Gleida, por sua incondicional paciência, dedicação e apoio ao longo dos últimos 22 anos – uma tarefa muito mais difícil que desenvolver um trabalho.

AGRADECIMENTOS

A meu orientador Prof. Dr. Marcos Ribeiro Pereira Barreto pelas diretrizes seguras e permanente incentivo.

A todo o pessoal das áreas de Administração e Sistemas da Engenharia de Produtos da General Motors do Brasil.

A todos que direta ou indiretamente colaboraram no desenvolvimento deste trabalho.

SUMÁRIO

LISTA DE FIGURAS

RESUMO

O objetivo deste trabalho é disponibilizar informações técnicas de projeto, que são parâmetros e características de um produto que são úteis para sua descrição, fabricação e análise de desempenho, para os engenheiros de projeto de um determinado produto (no caso utilizado – um automóvel) e demais interessados em uma forma barata e que estes possam fazer consultas rapidamente, com alta confiabilidade e sem o auxílio de intermediários, reduzindo as perdas de tempo com burocracia desnecessária, consultas a desenhos e arquivos em papel e dificuldade de acesso aos respectivos responsáveis pela informação.

A solução apresentada por este trabalho foi criar um site na intranet da empresa que acessa um banco de dados via um programa que gerencia todas as suas atividades. Todas as entradas e saídas para o usuário são páginas em HTML, o banco de dados utilizado foi o Access e tanto o servidor quanto o programa principal foram desenvolvidos em Java.

Este trabalho também levou em conta restrições de acesso, tais como se o usuário tem direito ou não a alterar o banco de dados e qual informação cada usuário pode ter acesso ou não, garantindo assim a integridade e a segurança das informações confidenciais e secretas e preservando a estratégia geral da empresa.

Outro recurso interessante deste projeto é a constante notificação ao usuário de alterações feitas em informações do banco de dados, para que o usuário fique sempre atualizado e evitando assim conflitos de projeto.

Por fim, cada informação apresentada na saída pode ter um link que "aponta" para um documento auxiliar, que serve para verificar todo o histórico de reuniões realizadas para definir esse dado e identificar as justificativas pelas quais ele foi adotado como a melhor solução, ou para verificar o status de desenvolvimento do mesmo

"ABSTRACT"

The objective of this work is to exhibit technical information of project, which are parameters and characteristics of a product which are useful for your description, fabrication and performance analysis, to the project engineers of a given product (in this case – an automobile) and other interested people in a economic and fast way, with high confidence and without the help of others, reducing the time loss with unnecessary bureaucracy, consultation to paper files and drawings and difficulty to access the responsible for the information.

The solution presented in this work was to create a site in the company's Intranet that accesses a database with a program that manages all its activities. All the inputs and outputs to the user are HTML pages, the database is Microsoft Access based and both the server and the main program were developed in Java.

This work supports access restrictions too, like if the user is allowed or not to modify the database and which information the user can access or not, to guarantee the integrity and security of the secret and confidential information, preserving the general strategy of the company.

Another interesting resource of this project is the constant notification to the user about modifications done in the database, so that the user has always stayed informed and avoiding project conflicts.

Each information presented at the output can be a link that point out to a auxiliary document, that is good to verify all the historic of the meetings realized to define this data and identify the justification for their adoption as the best solution, or to verify the status of its development.

1. INTRODUÇÃO

1.1. Descrição do problema

Em um ambiente globalizado como o que vivemos atualmente, torna-se cada vez mais evidente a necessidade das empresas apresentarem seus produtos com um tempo de desenvolvimento cada vez menor (time to market pequeno) e com seu respectivo custo o mais reduzido possível para continuar sendo competitivas.

Levando-se em conta este cenário, as informações técnicas de projeto, que são parâmetros e características de um produto que são úteis para sua descrição, fabricação e análise de desempenho, devem estar disponíveis para os engenheiros e demais interessados em uma forma barata e que estes possam fazer consultas rapidamente, com alta confiabilidade e sem o auxílio de intermediários, reduzindo as perdas de tempo com burocracia desnecessária, consultas a desenhos e arquivos em papel e dificuldade de acesso aos respectivos responsáveis pela informação.

Uma forma interessante de cumprir esta tarefa é criar um banco de dados eficiente, não só com as informações em si, mas também com informações complementares, tais como:

- documentos auxiliares (ex: minutas de reuniões) que definem um certo dado do banco, ou comprovam sua veracidade, ou indicam o status daquele dado;
- e-mail de todos os interessados em saber sobre alterações deste dado;

- outras informações que forem interessantes para os usuários.

1.2. Marcas

Várias marcas e nomes de produtos podem estar citados neste documento e que pertencem a terceiros. Dado que este documento pretende ter circulação apenas interna, não será feita uma discriminação caso a caso.

1.3. Escopo

Este documento visa descrever as tecnologias e os requisitos necessários ao Gerenciamento de Informações Técnicas de Projeto.

2. COMPLEMENTAÇÃO TÉCNICA

2.1. Metodologia do projeto de software – RUP (Rational Unified Process)

2.1.1. Introdução ao RUP

O RUP é uma metodologia de projeto de software que proporciona uma abordagem disciplinada a cada tipo de tarefa e responsabilidade de cada indivíduo na organização do projeto. Isto serve para assegurar a produção de softwares de alta qualidade que satisfazem as necessidades do usuário final, com duração e orçamento previsíveis.

O RUP aumenta a produtividade do grupo de desenvolvimento do software, providenciando a qualquer membro deste fácil acesso a base de dados com diretrizes, templates e tutoriais para todas as atividades críticas do processo. Tendo todos os membros do grupo acessando a mesma base de dados, não importa se ele trabalha com requisições, design, testes, ou administração do projeto, assegura-se que todos os membros do grupo de desenvolvimento do software dividem a mesma linguagem, metodologia e visão de como desenvolvê-lo.

Evitando a produção de grandes quantidades de documentos em papel, o RUP enfatiza o desenvolvimento e manutenção de modelos – que são representações mais ricas do software em desenvolvimento.

O RUP é um manual de como usar efetivamente o UML (Unified Modeling Language), que é a linguagem padrão da indústria que nos permite definir requerimentos, arquiteturas e estilos claramente. O UML foi originalmente

criado pela Rational Software, e é agora mantido pela organização de padrões OMG (Object Management Group).

O RUP é suportado por ferramentas que automatizam grande parte dos processos. Elas são utilizadas para criar e manter vários artefatos - modelos em particular – de metodologias de projeto de software, tais como: modelos visuais, programas, testes, etc. Eles não são úteis para suportar todo o arquivo associado com uma mudança de administração ou a configuração da administração que acompanha cada iteração.

O RUP é um processo configurável. Nenhum processo por si só é apropriado para todos os projetos de software. O RUP se ajusta aos pequenos grupos de projeto tão bem quanto a grandes organizações de projeto. O RUP é fundamentado numa metodologia de arquitetura simples e clara que proporciona homogeneidade ao longo de um família de processos. Além disso, ele pode ser variado para acomodar diferentes situações. Ele contém um kit de desenvolvimento, proporcionando suporte para configuração da metodologia para preencher os requisitos de uma dada organização.

2.1.2. As seis práticas mais apropriadas

O RUP reuni muitas das mais modernas práticas de desenvolvimento de software em uma forma que se ajusta a uma grande gama de projetos e organizações. Desenvolver essas práticas usando o RUP como guia oferece um grande número de vantagens para o grupo de projeto de software. Abaixo estão enumeradas as seis práticas fundamentais do RUP:

- *Iteratividade no desenvolvimento do software:* Dados os sofisticados sistemas de software de hoje em dia, não é possível seqüencialmente primeiro definir todo o problema, elaborar a solução inteira, construir o software e depois testar o produto até o final. Uma abordagem iterativa proporciona uma compreensão mais completa do problema por refinamentos sucessivos, e uma crescente eficiência da solução. O RUP suporta uma abordagem iterativa que identifica os itens de maior risco a cada estágio do ciclo, reduzindo bastante o risco do projeto. Como cada iteração termina com uma versão executável, o grupo de projeto fica focado na produção de resultados, com freqüente checagem de status, que ajuda a assegurar que o projeto está bem encaminhado. Isto também habilita o usuário final a ter um contínuo envolvimento com o projeto, o que proporciona um melhor *feedback* deste. Outra vantagem é que a abordagem iterativa proporciona uma fácil acomodação de mudanças táticas nos requisitos, aspectos e cronograma do projeto;
- *Requisitos de administração:* O RUP descreve como organizar, concluir, e documentar as obrigações e funcionalidades requeridas; documentar decisões; e facilmente capturar e comunicar requisitos de negócios. As noções de *use cases* e cenários proscritos no processo tem provado ser um excelente meio de capturar requisitos funcionais e assegurar que estes dirijam o design, implementação e teste do software, tornando-o mais parecido com o sistema final que atende as necessidades do usuário final;

- *Arquiteturas baseadas no uso do componente:* O processo enfoca em um adiantado desenvolvimento de uma arquitetura executável robusta, dando prioridade para os recursos de desenvolvimento em larga escala. Isso descreve como se faz uma arquitetura que é flexível, suporta mudanças, é intuitivamente compreensível, e promove a reciclagem mais efetiva do software. O RUP suporta o desenvolvimento de software baseado em componentes. Os componentes são módulos não triviais, subsistemas que desempenham uma função clara. O RUP proporciona uma abordagem sistemática para definir a arquitetura usando componentes novos e já existentes. Eles são montados em uma arquitetura melhor definida, ou em uma infraestrutura componente como a internet, CORBA, e COM, para os quais a indústria de componentes recicláveis está emergindo;
- *Visualização de modelos:* A metodologia mostra como visualizar modelos de software para analisar a estrutura e o funcionamento da arquitetura e dos componentes. Com isso, o usuário não precisa se preocupar com detalhes e escrever códigos de programas usando "blocos de construção gráfica". Visualizações abstratas ajudam o programador a se comunicar com diferentes aspectos do software; ver como os elementos do sistema trabalham juntos; ter certeza que os blocos de construção são consistentes com seu código; manter a coerência entre o design e sua implementação; e proporcionar comunicações não ambíguas. O UML (Unified Modeling Language) padrão da indústria é a base para a visualização de modelos;

- *Verificação da qualidade do software:* Aplicações com performances pobres e pouco confiáveis são fatores comuns que dramaticamente inibem a aceitação das aplicações dos softwares atuais. Consequentemente, a qualidade deve ser revisada com respeito aos requisitos baseados na confiabilidade, funcionalidade, performance da aplicação, e performance do sistema. O RUP dá assistência no planejamento, design, implementação, execução e avaliação desses tipos de testes. A qualidade é tratada na metodologia em todas as atividades, envolvendo todos os participantes, usando medidas e critérios objetivos. Ela não é tratada como uma atividade final ou separada, sendo executada por grupo separado;
- *Controle de mudanças do software:* A habilidade de administrar mudanças – tendo certeza se cada mudança é aceitável ou não e programando-as – é essencial em um cenário no qual a mudança é inevitável. A metodologia descreve como controlar, programar e monitorar mudanças para se obter um desenvolvimento iterativo com sucesso. Isso também serve de guia em como estabelecer um espaço de trabalho seguro para cada programador garantindo isolamento de mudanças feitas em outros espaços de trabalho e controlando mudanças de todos os artefatos do software (modelos, códigos, documentos, etc). Isso provoca um time coeso trabalhando como uma unidade única descrevendo como automatizar a integração e desenvolver a administração.

2.1.3. Descrição da metodologia

A metodologia pode ser descrita em duas dimensões, ou ao longo de dois eixos:

- A abcissa representa o tempo e mostra o aspecto dinâmico do processo e como é ordenado, e é expresso em termos de ciclos, fases, iterações e marcos;
- A ordenada representa o aspecto estático do processo: como é descrito em termos de atividades, artefatos, trabalhadores e workflows (fluxos de trabalho).

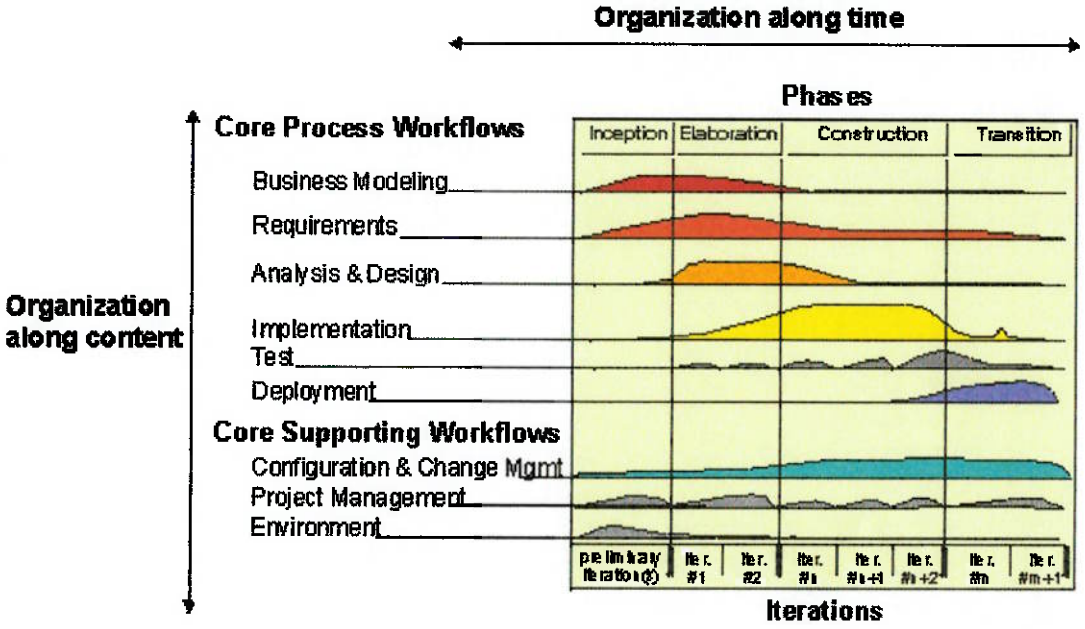


Figura 2.1 O modelo gráfico iterativo que mostra como o processo é estruturado ao longo das duas dimensões.

2.1.4. Fases e iterações

O ciclo de vida do software é quebrado em ciclos, cada qual trabalhando numa nova geração do produto. O RUP divide um ciclo do projeto em quatro fases:

- Fase Inception;
- Fase Elaboration;
- Fase Construction;
- Fase Transition.

Cada fase é concluída com um marco – um ponto no tempo no qual certas decisões devem ser tomadas.

2.1.4.1. Fase Inception

Durante a fase inception, deve ser estabelecido um *business case* para o sistema e delimitado o escopo do projeto. Para que isso seja executado, deve-se identificar todas as entidades externas com as quais o sistema vai interagir e definir a natureza dessa interação a um alto nível. Isso envolve identificar todos os *use cases* e descrever alguns mais significantes. O *business case* inclui o critério de sucesso, análise de risco, uma estimativa dos recursos necessários, e um plano de fase mostrando datas e principais marcos.

A saída da fase inception é:

- Um documento de visualização geral dos requisitos do projeto e características principais;
- Um modelo de *use case* inicial (10% ~ 20% completo);
- Um glossário do projeto inicial;
- Um *business case* inicial, o que inclui o contexto do negócio, critério de sucesso e previsão financeira;
- Uma análise de risco inicial;
- Um plano de projeto, mostrando as fases e as iterações;
- Um *business model,* se necessário;
- Um ou vários protótipos.

2.1.4.2. Fase Elaboration

A finalidade da fase elaboration é analisar o problema, estabelecer uma fundamentação da arquitetura, desenvolver um plano de projeto, e eliminar os elementos de maior risco de projeto. Para realizar esses objetivos, deve-se ter uma visão abrangente, mas pouco profunda do sistema. Decisões de arquitetura têm que ser tomadas com um entendimento de todo o sistema: seu escopo, função principal e requisitos de monofunção, como os requisitos de performance.

Na fase de elaboração, um protótipo executável é construído em uma ou mais iterações, dependendo do escopo, tamanho, risco e inovação do projeto. Este esforço deve ao menos enumerar os *use cases* críticos identificados na fase inception, a qual normalmente expõe os maiores riscos técnicos do

projeto. Enquanto um protótipo evolucionário de um componente de qualidade de produção é sempre o objetivo, isso não exclui o desenvolvimento de um ou mais modelos de apoio, dispensando protótipos de correr riscos específicos como requisitos de design, estudo de possibilidade de componente, ou demonstrações para investidores, requisitantes e usuários.

A saída da fase elaboration é:

- Um modelo de *use case* (ao menos 80% completo);
- Requisitos suplementares contendo os requisitos sem função ou qualquer requisito que não for associado com um *use case* específico;
- Uma descrição da arquitetura do software;
- Uma lista de riscos revisada e um *business case* revisado;
- Um plano de desenvolvimento para todo o projeto, incluindo o plano de projeto inicial, mostrando iterações e critérios de avaliação para cada iteração;
- Um estudo de desenvolvimento, especificando a metodologia a ser utilizada;
- Um manual do usuário preliminar (opcional).

2.1.4.3. Fase Construction

Durante a fase construction, todos os componentes remanescentes e características da aplicação são desenvolvidas e integradas no produto, e todas as características são testadas por completo. A fase construction é, em um sentido, um processo de manufatura onde a ênfase é colocada nos

recursos de administração e operações de controle para otimizar os custos, programas e qualidade. Neste sentido, o conjunto administrativo sofre uma transição do desenvolvimento intelectual durante a fase inception e elaboration, para um desenvolvimento de produtos físicos durante a fase construction e transition.

Muitos projetos são tão grandes que a construção paralela de incrementos pode ser utilizada. Essas atividades paralelas podem aumentar significantemente a rapidez com que uma versão física esteja disponível; elas podem também aumentar a complexidade de recursos de administração e a sincronização dos workflows. Uma arquitetura robusta e um plano claro são altamente correlacionados. Em outras palavras, Uma das qualidades principais da arquitetura é sua facilidade de construção. Essa é a razão pela qual o desenvolvimento balanceado da arquitetura e do plano é salientado na fase elaboration.

A saída da fase construction é um produto pronto para por nas mãos dos usuários finais. No mínimo, isso consiste em:

- O software integrado com as plataformas adequadas;
- O manual do usuário;
- Uma descrição da versão atual.

2.1.4.4. Fase Transition

A proposta da fase transition é a transição do software para a comunidade de usuários. Uma vez que o produto tiver sido apresentado ao usuário, a

experiência mostra que o projetista é requisitado a desenvolver novas versões, corrigir alguns problemas, ou terminar as características que foram previstas.

A fase transition é iniciada quando a base do produto está madura suficiente para ser desenvolvida no ambiente do usuário final. Isso geralmente exige que alguns subconjuntos utilizáveis do sistema estejam completos em um nível aceitável de qualidade e que a documentação do usuário esteja disponível para que a transição para o usuário forneça resultados positivos para todas as partes. Isso inclui:

- Testes "beta" para validar o novo sistema contra os erros esperados dos usuários;
- Operação paralela com um back-up de sistema que vai se substituindo;
- Conversões de databases operacionais;
- Treinamento de usuários e realizadores de manutenção;
- Repassar o produto para marketing, distribuição e times de vendas.

A fase transition enfoca as atividades requisitadas para colocar o software nas mãos do usuário final. Geralmente, essa fase inclui várias iterações, incluindo versões beta, versões de disponibilidade geral, tanto quanto versões de depuração de erros e mais completas. Um esforço considerável é esperado no desenvolvimento do documento de orientação do usuário, treinamento de usuários, suporte para os usuários em seus primeiros contatos com o software, e reagindo ao *feedback* do usuário final. Neste ponto do projeto, entretanto, o *feedback* do usuário deve ser confinado principalmente ao ajuste, configuração, instalação e uso do produto.

Os principais objetivos da fase transition incluem:

- Realizar o alto atendimento do usuário;
- Realizar uma análise das bases de desenvolvimento – se estão completas e coerentes com critério de avaliação;
- Obter uma base do produto final rapidamente e efetividade de custos na prática.

2.1.4.5. Iterações

Cada fase no RUP pode ser quebrada em iterações. Uma iteração é um looping de desenvolvimento completo resultando em uma versão (interna ou externa) de um produto executável, um subconjunto do produto final em desenvolvimento, o qual cresce de iteração em iteração até se tornar o produto final.

2.1.5. Estrutura Estática da Metodologia

2.1.5.1. Trabalhador

Um trabalhador define a conduta e as responsabilidades de um indivíduo, ou grupo de indivíduos trabalhando juntos como um time. Um trabalhador pode ser encarado como um chapéu que um indivíduo pode usar em um projeto. Um indivíduo deve usar muitos chapéus diferentes. Isso é um conceito importante porque é natural pensar em um trabalhador como um indivíduo ou time, mas no RUP o trabalhador é mais a própria definição de que tipo de tarefa um indivíduo

realiza no projeto. As responsabilidades que são conferidas ao trabalhador incluem tanto realizar um certo conjunto de atividades quanto ser o administrador de um conjunto de artefatos.

A figura 2.2 abaixo mostra como isso funciona.

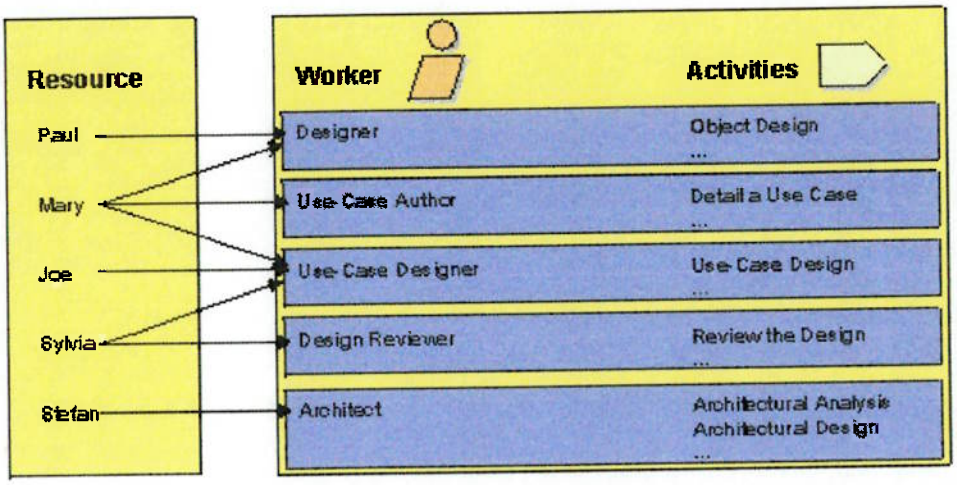


Figura 2.2 As pessoas e os trabalhadores

2.1.5.2. Atividade

Uma atividade de um trabalhador específico é uma unidade de trabalho que um indivíduo pode vir a realizar. A atividade tem uma proposta clara, geralmente expressa em termos de criação ou melhoramento de alguns artefatos, como um modelo ou um plano. Toda atividade é atribuída a um trabalhador específico. A duração de uma atividade varia em geral de algumas horas até alguns dias, geralmente envolvendo apenas um trabalhador e afeta um ou um pequeno número de artefatos. Uma atividade deve ser usada como um elemento de planejamento e progresso ; se isso é muito pequeno , será

negligenciado, e se for muito grande, o progresso deverá ser expresso em termos de partes da atividade.

Alguns exemplos de atividades são:

- Planejar uma iteração, para o trabalhador: administrador do projeto;
- Achar use cases e softwares de interface, para o trabalhador: analista de sistemas.

2.1.5.3. Artefatos

Um artefato é um pedaço de informação que é produzido, modificado, ou usado pelo processo. Artefatos são os produtos tangíveis do projeto, as coisas que o projeto produz ou usa enquanto trabalha para obter o produto final. Artefatos são usados como entradas para os trabalhadores para realizar as atividades, e são o resultado da saída de algumas atividades. Em termos de design orientado do objeto, as atividades são operações de um objeto ativo (o trabalhador), artefatos são os parâmetros dessas atividades.

Artefatos assumem várias formas:

- Um modelo, como um modelo de *use case* ou design;
- Um documento, como um *business case* ou documento de arquitetura de software.

2.1.5.4. Workflows

Uma mera enumeração de todos os trabalhadores, atividades e artefatos não é suficiente para constituir um processo. É necessário um jeito de descrever as seqüências de atividades que produzem algum resultado válido, mostrando as interações entre os trabalhadores.

Um workflow é uma seqüência de atividades que produzem um resultado de valor observável.

Em termos de UML, um workflow pode ser expresso como um diagrama seqüencial ou um diagrama de atividades.

O tipo mais essencial de workflow para o RUP é chamado core workflow. Existem nove destes no RUP, que representam uma partição de todos os trabalhadores e atividades em um grupo lógico.

Os core workflows estão divididos em seis core "engeneering" workflows:

- Workflow de modelamento de negócios;
- Workflow de requisitos;
- Workflow de análise e design;
- Workflow de implementação;
- Workflow de testes;
- Workflow de melhoramento.

E três core "supporting" workflows:

- Workflow de administração de projeto;
- Workflow de configuração e administração de mudanças;
- Workflow de cenários.

2.2. SQL / DBMS

2.2.1. Introdução ao SQL

As linguagens de programação enfrentam um dilema, que é conciliar o poder e o número dos seus recursos e a qualidade do seu desempenho com a facilidade de uso.

A SQL (Structured Query Language).surgiu exatamente dessa necessidade, na tentativa de oferecer uma linguagem que fosse ao mesmo tempo simples para o usuário final e versátil para o programador de aplicações.

Esta linguagem de programação é uma interface para quase todos os bancos de dados relacionais atuais. Bancos de dados para microcomputadores geralmente possuem uma interface gráfica com o usuário que permite aos usuários manipular os dados diretamente. Contudo, bancos de dados baseados em um servidor são acessados apenas através da SQL. A maioria dos bancos de dados em microcomputadores também possui uma interface SQL, mas geralmente não suportam a gama completa de recursos ANSI SQL 92, o padrão atual para SQL.

Dessa forma, a SQL existe em duas modalidades: como linguagem autônoma, interativa, permitindo que através de um terminal ou de um computador o usuário acesse diretamente os dados que procura, e como uma linguagem de programação (para definição, manipulação e controle de dados), acoplada a uma outra linguagem que se encarregue das estruturas básicas de programação, da formatação de telas, da impressão de relatórios, etc.

A adoção da SQL permitiria que houvesse um idioma comum entre o usuário final e os analistas – programadores, formada por um conjunto bastante compacto de comandos e funções (a SQL não chega a ter 30 comandos e funções).

2.2.2. DBMS (Database Management System)

Um DBMS deve fornecer um método para armazenamento e manipulação de informações sobre uma variedade de objetos e as relações entre esses objetos. A categoria mais importante de um DBMS é a relacional.

Sem entrarmos a fundo na matemática, pode-se dizer que ela possui um ramo denominado Álgebra Relacional tratando, basicamente, de conjuntos e tabelas. Toda a estrutura da SQL deriva da Álgebra Relacional, onde as tabelas são conhecidas como relações; esta é a origem do termo relacional.

Ou seja, em termos gerais, um sistema relacional é um sistema cujas informações ficam armazenadas em tabelas, sendo endereçadas através do nome da tabela, do nome ou número da coluna, e do número da linha.

No DBMS, o usuário pode:

- definir seu *schema* (estrutura lógica dos dados), usando uma linguagem apropriada (DDL, data definition language);
- realizar perguntas de acesso ("query") sem conhecimento da estrutura de armazenamento dos dados, usando uma linguagem apropriada (DML, data manipulation language ou query-language);
- acessar grandes volumes de dados de forma eficiente;
- acessar dados de forma independente uns dos outros sob controle de acesso.

2.2.3. Banco de dados

Diversos procedimentos executados em todos os tipos de instituições exigem o acúmulo de grande volume de dados. Pode ser uma indústria controlando seu estoque de matérias – primas, ou uma empresa cadastrando os funcionários inscritos em seus programas de treinamento, ou um hotel administrando as reservas e despesas dos hóspedes.

Quando todos esses dados são organizados segundo uma estrutura lógica que permita a sua definição, atualização, recuperação e controle, temos um banco de dados.

Um bando de dados é uma estrutura ampla formada pelos arquivos de dados – que recebem o nome de tabelas – e mais visões, índices e catálogos que armazenam informações sobre os próprios bancos de dados.

Uma tabela é um conjunto de dados organizados em linhas e colunas, de acordo com algumas regras simples:

- todas as colunas devem ter um nome;
- todas as colunas devem conter dados de um mesmo tipo;
- cada cruzamento de linha e coluna só pode conter uma única informação.

As linhas e colunas de uma tabela são equivalentes aos registros e campos de um arquivo de dados respectivamente.

As visões da SQL nada mais são do que tabelas lógicas – tabelas que existem apenas na memória do computador, não residentes fisicamente em disco.

Elas permitem que, durante uma mesma sessão de trabalho, se definam maneiras alternativas de visualizar uma tabela, sem que tenha de ocupar espaço em disco com um arquivo de uso eventual.

Usando visões é possível, entre outras coisas, restringir o acesso a dados confidenciais, trabalhar com versões reduzidas de tabelas de dados extensas, extrair informações específicas para emissão de relatórios.

Para tornar um banco de dados útil, é necessário ser capaz de adicionar dados nele, mudar os dados correntes, e deletar dados dele. É preciso também ser capaz de fazer perguntas de acesso (query), que são consultas ou pesquisas especiais que permitem recuperar dados de diversas tabelas ao mesmo tempo, com base em critérios determinados pelo usuário, e agrupar ou manipular esses dados da maneira que julgarmos conveniente.

Por exemplo, pode-se querer achar quais compradores têm um balanço que excede seu limite de crédito. Além disso, deseja-se produzir relatórios, como uma lista de informações sobre todas as partes de um produto.

Um DBMS tem várias maneiras de cumprir essa tarefa. A abordagem que está se tornando a mais utilizada é justamente a linguagem chamada SQL.

A SQL pode ser utilizada de dois modos: através de comandos diretos digitados pelo usuário, ou como complemento efetivo a outras linguagens de programação. No primeiro caso nós a chamamos de SQL Interativa e no segundo de SQL interna (ou "Embedded"), capaz de permitir a geração de um código mais compacto e ao mesmo tempo mais poderoso nas pesquisas a bancos de dados.

2.3. Java

2.3.1. Introdução ao Java

A família java é constituída do Java, ambiente de desenvolvimento e a linguagem da Sun Microsystems, e JavaScript, uma ferramenta de roteirização HTML, (HyperText Markup Language ou linguagem de estruturação de hipertexto) desenvolvida pela Netscape. Estas duas linguagens são responsáveis pelo desenvolvimento do conteúdo dinâmico para a World Wide Web.

Em resumo, a família java dá vida às suas páginas da Web. Seja através de um miniaplicativo Java ou uma função JavaScript, muitas páginas HTML estáticas, baseadas em roteiros CGI estão agora sendo substituídas por páginas que

oferecem conteúdo dinâmico e uma reação automática às ações do cliente sem recorrer à roteirização do lado do servidor.

2.3.2. Características da linguagem Java

A linguagem Java modifica a natureza pacífica da Internet e da World Wide Web ao permitir que códigos de arquitetura neurais sejam carregados dinamicamente e executados em uma rede heterogênea de máquinas como a Internet. Java proporciona essa funcionalidade ao incorporar as seguintes características à sua arquitetura. Estas características tornam o Java um promissor competidor para tornar-se o principal protocolo da Internet.

- *Portável:* Isto significa que pode ser executado em qualquer máquina que possua o interpretador Java portado a ela. Esta é uma característica importante para que uma linguagem seja usada na Internet onde qualquer plataforma poderia estar na extremidade empresarial de um cartão Ethernet;
- *Robusta: As características da linguagem e o ambiente de run-time (o instante da execução) asseguram que o código é bem comportado. Isto vem principalmente como resultado do esforço para a portabilidade e da necessidade de aplicativos sólidos que não vão derrubar um sistema quando um usuário tropeça em uma home page com um pequeno arquivo de animação;*

- *Segura:* Além de proteger o cliente contra ataques não intencionais, o ambiente Java deve protegê-lo contra também contra-ataques intencionais. A Internet é toda muito familiarizada com cavalos de Tróia, vírus e vermes que permitem que qualquer aplicativo seja carregado e executado;
- *Orientado a objetos:* A linguagem é orientada a objetos em seus fundamentos e permite a descendência e reutilização de código de maneira estática e dinâmica;
- *Alto desempenho:* A linguagem Java suporta diversas características de alto desempenho como multientrelaçamento, compilação just-in-time e uso de código nativo;
- *Fácil:* Como a própria linguagem pode ser considerada derivada do C e do C++, é familiar. Ao mesmo tempo, o ambiente assume pelo programador muitas das tarefas de erros tais como o gerenciamento de ponteiros e de memória.

A tarefa de oferecer um conteúdo dinâmico para a Internet é difícil, mas o protocolo que o conseguir se tornará tão universal como o correio eletrônico e a HTML o são hoje.

2.3.2.1. Java é Portável

A linguagem de programa Java proporciona portabilidade em diversas formas. Duas destas formas são:

- A linguagem Java é interpretada. Isto significa que qualquer computador em que se queira que seja executada precisa ter um programa para converter o código Java em código nativo de máquina;
- A linguagem Java não permite que a máquina em particular implemente diferentes tamanhos para tipos fundamentais como inteiros ou bytes.

Ao ser executado em um ambiente interpretado, o código Java não precisa obedecer a nenhuma plataforma de hardware em particular. O compilador Java que cria os programas executáveis a partir do código fonte compila para uma máquina que não existe – a Máquina Java Virtual, que é uma especificação de um processador hipotético que pode executar o código Java. O problema tradicional com os interpretadores tem sempre sido seu desempenho, ou melhor sua falta de desempenho. Java tenta superar isto compilando a um estágio intermediário, convertendo o código fonte a um código de bytes, que pode então ser eficientemente convertido em código nativo para um processador em particular.

Além de especificar um código de máquina virtual para garantir a portabilidade, a linguagem Java também se assegura de que os dados ocupem o mesmo espaço em todas as implementações. Os tipos de dados da linguagem C, por outro lado, mudam dependendo do hardware subjacente e do sistema operacional. Por exemplo, um inteiro que ocupava 16 bits sob Windows 3.1, agora ocupa 32 bits em Windows 95. O mesmo problema existe ao longo de plataformas de processadores, onde alguns computadores, como o DEC Alpha, são de 64 bits, enquanto outros, como o 486 da Intel, são de apenas 32 bits. Ao criar um único

padrão para o tamanho dos dados, o Java garante que os programas sejam independentes do hardware.

Estas são algumas das características que garantem que o Java é capaz de ser executado em qualquer máquina em que o interpretador está portado. Desta forma, uma vez que um único aplicativo tenha sido portado, o desenvolvedor e o usuário têm a vantagem de todos os programas escritos para Java.

2.3.2.2. Java é Robusto

O ambiente Java é robusto porque elimina os problemas tradicionais que os programadores têm ao criar um código sólido. Os inventores do Java procuraram estender o C++ para incluir a funcionalidade necessária para um programa distribuído, mas logo perceberam que isto seria por demais problemático. As principais dificuldades em tornar o C++ um programa portável são o seu uso de ponteiros para endereçar diretamente posições de memória e sua falta de gerenciamento automático da memória. Estas características permitem ao programador gerar um código que é sintática e semanticamente correto, e ainda provocar quedas no sistema por uma ou outra razão. Por outro lado, o Java garante um ambiente robusto ao eliminar ponteiros e oferecer gerenciamento automático da memória.

Como o ponto central dos programas Java é de automaticamente serem capazes de carregar e executar, seria inaceitável que houvesse uma possibilidade de que um destes aplicativos pudesse ter uma falha que faria derrubar o sistema

ao, por exemplo, escrever sobre o espaço de memória do sistema operacional. Por esta razão, o Java não faz uso de ponteiros. Os endereços de memória não podem ser desreferenciados e um programador não pode usar a aritmética de ponteiros para mover-se pela memória. Além disso, o Java oferece para matrizes a verificação de limites de modo que um programa não pode indexar um espaço não alocado à matriz.

O Java propicia gerenciamento automático da memória na forma de um coletor automático de refugos. Este coletor de refugos mantém o registro de todos os objetos e referências a estes objetos em um programa Java. Quando um objeto já não possui mais referências, o coletor de refugos é executado como um enlace de baixa prioridade em segundo plano, e retira o objeto, retornando sua memória de volta a área de uso quando o programa não estiver usando muitos ciclos de relógio, ou se houver uma necessidade imediata por mais memória. Ao executar como um enlace em separado, o coletor de refugos pode proporcionar a facilidade de uso e a robusteza de um gerenciamento automático de memória sem o trabalho adicional de um modelo de tempo integral de gerenciamento.

2.3.2.3. Java é seguro

As necessidades da computação distribuída demanda os mais altos níveis de segurança para os sistemas operacionais clientes. O Java proporciona segurança através de diversas características do seu ambiente de run-time.

- Um verificador do código de bytes;
- Esquema da memória em tempo de execução;
- Restrições ao acesso a arquivos.

Quando o código Java entra inicialmente no interpretador, antes mesmo de ter uma oportunidade de ser executado, é verificado quanto à adequação à linguagem. Mesmo se o compilador somente gerar códigos corretos, o interpretador o verifica novamente, apenas para ter certeza, porque o código pode ter sido intencional ou não intencionalmente alterado entre o momento da compilação e o da execução.

O interpretador Java em seguida determina o esquema de memória para as classes. A classe é a unidade básica de execução do Java – equivalente a um objeto na expressão da OOP (programação orientada a objetos).

Isto significa que os hackers não podem inferir nada a respeito de como poderia ser uma classe no próprio hardware e então usar esta informação para forjar um acesso. Além disso, o carregador de classes coloca cada classe carregada da rede em sua própria área de memória.

Mesmo assim, a segurança das verificações do interpretador Java continua ao garantir que as classes carregadas não acessem o sistema de arquivos exceto da forma específica em que são permitidos pelo cliente ou usuário. Embora administradores de localidades da Internet possam contorcer-se em suas cadeiras com a idéia de programas automaticamente se carregando e executando, a equipe do Java fez todos os esforços para garantir aos administradores que seus

piores receios, como um vírus especialmente eficiente ou cavalo de Tróia, nunca se tornem realidade.

2.3.2.4. Java é orientado a objetos

A característica mais importante do Java é que é uma linguagem verdadeiramente orientada a objetos. Os projetistas do Java decidiram romper com qualquer linguagem existente e criaram uma do princípio. Embora o Java tenha a aparência e a semelhança a C++, é de fato uma linguagem completamente independente, projetada para ser orientada a objetos desde o início. Isto proporciona diversos benefícios, incluindo os seguintes:

- Reutilização de código;
- Possibilidade de ser estendido;
- Aplicativos dinâmicos.

O Java oferece o elemento fundamental da programação orientada a objetos (OOP) – o objeto – através da classe. A classe é uma coleção de variáveis e métodos que são encapsulados funcionalmente em um objeto reusável e dinamicamente carregável. Isto significa que uma vez que a classe tenha sido criada, pode ser usada como modelo para a criação de classes adicionais que proporcionam maior funcionalidade. Por exemplo, um programador pode criar uma classe para apresentar retângulos na tela e em seguida decidir que seria

interessante ter um retângulo preenchido. Em lugar de escrever toda uma nova classe, o programador pode simplesmente ordenar ao Java para usar a classe antiga, com algumas características adicionais. De fato, o programador pode fazer isto sem nem mesmo possuir o código fonte original.

Depois que uma classe é criada, o ambiente de run-time do Java permite o carregamento dinâmico de classes. Isto significa que aplicativos existentes podem adicionar funcionalidade ao adicionarem novas classes que encapsulam os métodos necessários. Por exemplo, podemos estar surfando na Net e descobrir um arquivo para o qual não temos o aplicativo de tratamento. Tradicionalmente, ficaríamos bloqueados procurando um aplicativo que pudesse manusear o arquivo. Por outro lado, o navegador Java pode solicitar ao servidor um arquivo para uma classe que pode tratar o arquivo, carregá-lo dinamicamente juntamente com o arquivo e apresentá-lo sem nem mesmo suspirar.

2.3.2.5. Java tem alto desempenho

Tipicamente o custo desta portabilidade, segurança e robusteza é a perda de desempenho. Não é razoável acreditar que o código interpretado possa ser executado à mesma velocidade que o código nativo; entretanto, o Java possui alguns truques que reduzem significativamente a quantidade de trabalho adicional.

- Multientrelaçamento incluído;
- Códigos de byte eficientes;

- Compilação just-in-time;
- Capacidade de unir métodos nativos em C.

Uma forma pela qual o Java supera os problemas de desempenho de interpretadores tradicionais é incluir a capacidade de multientrelaçamento. É raro que um programa esteja constantemente utilizando ciclos da CPU.

Em lugar disso, os programas precisam aguardar entradas, arquivos ou acessos à rede. Estas ações deixam o processador inativo em aplicativos de enlace único. Em lugar disso, Java utiliza este tempo ocioso para realizar a necessária limpeza de refugos e manutenção geral do sistema que é a causa de os interpretadores deixarem lentos muitos aplicativos.

Além disso, os códigos de byte compilados do Java são muito próximos do código de máquina, de modo que interpretá-los em qualquer plataforma específica pode ser bem eficiente. Em casos em que o interpretador não estiver sendo suficiente, o programador tem duas opções: compilar o código no momento da execução para código nativo ou unir código C nativo. Unir código C nativo é a mais rápida das duas, mas impõe uma sobrecarga ao programador e reduz a portabilidade. Compilar no momento da execução significa que o código permanece portável, mas há um atraso inicial enquanto o código é compilado.

2.3.2.6. Java é fácil de usar

Finalmente, a linguagem Java é fácil. É simples e eficiente devido à seu projeto e implementação bem concebidos. Os três elementos mais importantes que tornam a linguagem fácil são:

- É familiar porque é modelado segundo o C++;
- Elimina elementos problemáticos da linguagem;
- Oferece poderosas bibliotecas de classes.

O Java é conscientemente modelado segundo a linguagem C++, oferecendo uma aparência e uma impressão com que a maioria dos programadores se sente confortável. Ao mesmo tempo, o Java elimina os elementos problemáticos do C++ como ponteiros e gerenciamento de memória. Isto significa que os programadores empregam menos tempo preocupando-se com se o código vai executar e mais tempo desenvolvendo funcionalidade. Java também possui um poderoso conjunto de bibliotecas de classes que oferecem praticamente toda a funcionalidade básica necessária para se desenvolver um aplicativo rapida e eficientemente.

2.4. HTML (HyperText Markup Language)

2.4.1. Introdução

Em poucas palavras, HTML é a linguagem utilizada para criar uma página Web. Ela é constituída de um conjunto relativamente pequeno de símbolos (Tags) que determina a aparência e a impressão de uma página Web.

Como introdução, vamos definir o que significa cada uma das palavras que compõem o nome desta linguagem:

HyperText Um link de hipertexto é uma palavra ou frase especial de uma página Web que "aponta" para outra página Web. Quando o usuário dá um clique em um destes links, o seu browser o transporta imediatamente para a outra página Web. Como o link de hipertexto é o recurso real que diferencia a World Wide Web, as páginas Web freqüentemente são conhecidas como documentos de hipertexto. Portanto, a HTML possui a palavra "HyperText" porque o usuário a usa para criar estes documentos de hipertexto (ela também poderia ser chamada WPML – Web Page Markup Language).

Markup Define-se a palavra "markup" como "instruções estilísticas detalhadas escritas num manuscrito que será colocado em processo de composição". Para atender ao contexto da HTML, pode-se rescrever esta definição assim: "instruções estilísticas detalhadas digitadas dentro de um documento de texto que será publicado na World Wide Web." Em resumo, isto é HTML. Ela possui alguns códigos simples para detalhes como: transformar o texto em negrito ou itálico, criar listas com bullets, inserir imagens e, é claro, definir os

links de hipertexto. Basta digitar estes códigos nos lugares apropriados de um documento de texto comum e o browser apresentará automaticamente a página do jeito que o programador quiser.

Language A HTML é uma linguagem de programação, mas apenas no sentido de ter uma pequena coleção de combinações de duas ou três letras e palavras que são usadas para especificar estilos como negrito ou itálico ou indicar a presença de uma imagem ou link de hipertexto.

2.4.2. Recursos do HTML

Utilizando os recursos oferecidos pela HTML, o usuário pode apresentar uma página Web de várias maneiras. Por exemplo, no que diz respeito à formatação do texto, pode-se:

- utilizar seis tamanhos de fontes que podem ser usados como níveis de títulos;
- apresentar o texto da página Web em negrito;
- enfatizar outros itens com itálico;
- fazer com que o texto fique parecido com texto escrito à máquina;
- usar tamanhos e cores diferentes de fonte para os caracteres.

Se o usuário estiver apresentando informações na página Web, será útil se ele puder mostrar os seus dados de uma forma que faça sentido e que seja fácil de

ler. Em alguns casos, isto significa organizar os dados em listas, tal como uma lista de bullets ou numerada.

Outro recurso que a HTML suporta é a possibilidade de disponibilizar links de hipertexto que "apontam" para outras páginas Web. O designer da página pode definir quatro tipos básicos de links:

- Para outra página das suas páginas Web;
- Para um local diferente da mesma página Web. Isto é bom para páginas especializadas; pode-se, por exemplo, colocar um "sumário" na parte superior da página que possua links para as várias seções do documento;
- Para qualquer página, em qualquer lugar da Web;
- Para um software de e-mail (Outlook Expess, por exemplo) com o campo "to" preenchido com um ou mais endereços de e-mail.

Com todos estes efeitos decorativos de texto, listas e muitos links o designer pode desenvolver uma página Web com sucesso. Mas, para que uma página seja realmente agradável, é necessário adicionar uma ou duas imagens. Contanto que se tenha a imagem em um arquivo gráfico, pode-se utilizar a HTML para posicioná-la apropriadamente na página. Uma inovação poderosa neste campo é a imagem de fundo – uma figura, modelo ou cor que ocupa um lugar do fundo cinza que geralmente é encontrado nas páginas Web padrão.

Mas se a página Web precisa apresentar dados formatados em linhas e colunas, o designer não tem a possibilidade de utilizar tabulações ou espaços para separar textos ou imagens, pois a HTML reduz espaços múltiplos para um único

espaço, e ela também ignora completamente as tabulações. Para realizar esta tarefa é necessário utilizar um recurso especial da HTML, que é a criação de tabelas, podendo assim encaixar os dados em linhas e colunas.

Por fim, a maioria dos programas modernos apresenta uma caixa de diálogo na tela do usuário quando precisam extrair alguma informação deste último. Por exemplo, selecionar o comando "Print" de um programa provavelmente fará surgir algum tipo de caixa de diálogo "Print". A finalidade desta caixa de diálogo será pedir informações tais como o número de cópias desejado, as páginas que o usuário deseja imprimir e assim por diante.

A HTML disponibiliza um recurso para este fim, o formulário, que é simplesmente o equivalente da página Web a uma caixa de diálogo. É uma página repleta de caixas de texto, listas suspensas e botões de comando para obter informações do usuário.

3. REVISÃO BIBLIOGRÁFICA BÁSICA

3.1. PDM (Product Data Management)

O objetivo do PDM é manter o controle dos dados do produto e distribuí-los automaticamente para os interessados – quando eles precisam. Uma das características do sistema PDM que garante vários benefícios é o fato de que os dados mestres são arquivados apenas uma vez em um "receptáculo" seguro onde pode ser garantida sua integridade e todas as mudanças são monitoradas, controladas e arquivadas.

Por outro lado, cópias de referência dos dados mestres podem ser distribuídos livremente, para usuários de vários departamentos para design, análise e aprovação. Os novos dados são então arquivados de volta no "receptáculo". Quando ocorre uma mudança no dado, o que geralmente ocorre é que uma cópia do dado, assinado e datado, é arquivada no "receptáculo" depois do valor antigo, o qual permanece em sua forma original como arquivo permanente.

Este é o princípio simplificado por trás dos mais avançados sistemas de PDM. Para um melhor entendimento, é interessante olhar separadamente como esses sistemas controlam os dados do produto (administração dos dados e administração do processo).

3.1.1. Administração dos dados

Empresas do ramo manufatureiro geralmente são boas em reunir componentes e conjuntos de desenhos sistematicamente, mas muitas vezes não mantêm conjuntos de informações como tamanho, peso, função, etc. Como resultado, os engenheiros têm problemas em acessar a informação que precisam. Isso causa um deficiência na sua capacidade de administrar os dados de seus produtos efetivamente. Sistemas de administração de dados devem ser capazes de administrar tanto os atributos e documentos dos dados do produto quanto as relações entre eles por meio de um sistema de banco de dados relacional.

Com tantos dados sendo gerados, deve ser estabelecida uma técnica de classificar essas informações rapidamente e facilmente.

A classificação deve ser uma habilidade fundamental de um sistema de PDM. Informações de tipos similares devem ser capazes de serem agrupados em classes. Uma classificação mais detalhada seria possível utilizando os atributos para descrever as características essenciais de cada componente de uma classe.

3.1.1.1. Classificação de componentes

Os componentes serão a entrada do banco de dados sobre uma variedade de classes as quais satisfazem suas necessidades de *business*. As classes podem ser agrupadas em um título conveniente e amplo. Isso permite que todas as estocagens de componentes da empresa sejam organizadas em uma estrutura de

rede hierárquica e facilmente localizável. Cada parte pode ter seu próprio conjunto de atributos. Além disso, outros sistemas permitem que se registre que certos componentes sejam avaliáveis com atributos específicos opcionais. Isso pode ser inviável no controle dos *Bills of materials (BOMs)* por razões que serão tratadas neste trabalho mais adiante.

3.1.1.2. Classificação de documentos

Documentos relativos aos componentes e junções podem ser classificados similarmente. Por exemplo, classes devem ter desenhos, modelos 3D, publicações técnicas, etc. Cada documento pode ter seu próprio conjunto de atributos – partes números, autor, dados colocados. Ao mesmo tempo, as relações entre os documentos e seus respectivos componentes podem ser mantidas. Então, por exemplo, um dossiê para um "mancal específico" pode ser obtido, contendo desenhos 2D, modelos sólidos e arquivos FEA.

Os sistema PDM variam muito em suas capacidades de classificação. Alguns não têm nenhuma. Outros têm a capacidade de definir a classificação apenas quando o banco de dados estiver implementado. Os sistemas de PDM mais recentes têm fornecido a capacidade de poder definir e modificar de acordo com a mudança da demanda da organização.

3.1.1.3. Estrutura do produto

A terceira maneira que os dados do produto podem ser acessados é por estrutura do produto. Para qualquer produto selecionado, a relação entre a montagem dos componentes e as partes que compõem essas montagens deve ser mantida. Isso significa que é possível abrir um BOM completo, incluindo documentos e partes para o produto inteiro ou montagens selecionadas. Uma vantagem distinta é a possibilidade de trabalhar não apenas com as relações físicas entre as partes de uma montagem, mas também com outros tipos de estrutura; por exemplo, de manufatura, financeira, manutenção ou documentos relacionados. Então, é possível para especialistas membros do grupo visualizar a estrutura do produto pelos seus pontos de vista.

3.1.1.4. Pesquisando o dado (query)

É necessário estar capacitado para chegar aos dados dos componentes e montagens por várias rotas. Pode-se mover para cima e para baixo na árvore de classificação; escolher um jeito pela estrutura do produto; simplesmente chamar o dado que se deseja procurando por seu nome ou número código; ou procurando por grupos de dados especificando um atributo ou combinação de atributos. Por exemplo, pode-se pedir para ver todos os rebites de aço com cabeça de menos de 3 mm de raio.

3.1.2. Administração do processo

Até agora apenas foi tratada a organização dos dados e como é fácil seu acesso e referência, basicamente procedimentos passivos. A administração do processo, por outro lado, é sobre o controle de como os dados são criados e modificados – procedimentos ativos.

Isso pode soar como um novo nome para "administração de projeto", mas não é. Administração de projeto se preocupa apenas com a delegação de tarefas, enquanto que a administração do processo lida com o impacto das tarefas nos dados.

Sistemas de administração de processo geralmente tem três funções principais:

- administrar o que ocorre com os dados quando alguém trabalha com eles (Administração do trabalho);
- administrar o fluxo de dados entre pessoas (Administração de workflow);
- conservar os registros de todos os eventos e movimentos que ocorrem nas funções 1 e 2 durante a história do projeto (Administração do histórico de trabalho).

Os sistemas de PDM variam profundamente em como eles desempenham essas funções.

3.1.2.1. Administração do trabalho

Os engenheiros criam e mudam os dados continuamente. O ato de projetar algo é exatamente isso. Um modelo sólido, por exemplo, deve sofrer centenas de mudanças durante o curso do desenvolvimento, cada um envolvendo grandes modificações dos dados básicos de engenharia. Muitas vezes o engenheiro desejará simplesmente explorar uma abordagem particular, abandonando-a mais tarde em favor da versão anterior.

Um sistema de PDM oferece uma solução agindo no ambiente de trabalho do engenheiro, meticulosamente capturando todo dado novo e mudado quando ele é gerado, mantendo o registro de qual versão ele pertence, chamando-o de novo de acordo com a demanda e efetivamente guardando o registro de cada movimento do engenheiro.

É óbvio que, quando um engenheiro é solicitado a fazer uma modificação no projeto, ele ou ela normalmente requererá mais do que apenas o projeto original e uma ECO (Engineering Change Order). Muitos documentos, arquivos e formulários precisam ser referenciados e outros membros do grupo de projeto envolvidos também. Em um ambiente de projeto tradicional, pode-se compilar uma pasta ou dossiê de projeto ao qual o grupo de projeto pode consultar quando necessário.

Os sistemas de PDM atuais satisfazem esse requisito em vários níveis de sucesso. Uma abordagem é a que emula processos em papel usando-os como "pacotes do usuário".

O pacote permite ao engenheiro administrar e modificar vários documentos mestres diferentes simultaneamente tão bem quanto fornecendo vários documentos de suporte como referência. Essa abordagem também suporta o princípio de engenharia corrente. Por exemplo, embora apenas um usuário pode estar trabalhando no design mestre, os colegas trabalhando no mesmo projeto podem estar instantaneamente sendo notificados que existe um design mestre atualizado, e cópias de referência disso estarão disponíveis para eles em seus próprios pacotes. Um dado pacote pode ser enviado apenas para o usuário que estiver fora do sistema, mas este conteúdo pode ser copiado por todos que tiverem permissão de acesso.

3.1.2.2. Administração de workflow

Pacotes têm a vantagem de tornar a divisão de grupos de documentos fácil para os membros da equipe, mas eles são úteis por outro motivo também. Eles tornam possível o movimento de trabalho de departamento para departamento, ou de indivíduo para indivíduo em blocos organizados logicamente.

Durante o desenvolvimento de um produto, muitos milhares de partes precisam ser projetadas. Para cada parte, arquivos precisam ser criados, modificados, visualizados, checados e aprovados por muitas pessoas diferentes, muitas vezes em quantidade excessiva. Além disso, cada parte necessitará de técnicas de desenvolvimento diferentes e tipos de dados diferentes – modelos sólidos para alguns, diagramas de circuitos para outros, dados FEA para outros.

Um trabalho em qualquer desses arquivos mestres terá um impacto potencial em outros arquivos relacionados. Então é necessária uma checagem cruzada contínua, modificação, ressubmissão e rechecagem. Com todas essas mudanças excessivas, fica muito mais fácil de um engenheiro investir tempo e esforço consideráveis em continuar um projeto que já foi invalidado por um trabalho feito por outra pessoa em outra parte do projeto. Organizar este workflow altamente complexo é o que os sistemas de PDM fazem melhor. Particularmente, eles guardam os registros de milhares de decisões individuais que determinam quem faz o que depois.

A maioria dos sistemas PDM permitem que o líder do projeto controle o progresso via "status" usando "triggers" pré-determinados e uma lista de rotinas que deve variar de acordo com o tipo de organização e desenvolvimento do projeto que está envolvido. O diferencial do sistema está em quanta flexibilidade eles permitem dentro da disciplina da estrutura. Os sistemas mais rígidos são baseados em procedimentos. Todo indivíduo ou grupo de indivíduos são feitos para representar um estado do procedimento – "Iniciado", "Submisso", "Checado", "Aprovado", "Revisado"; um arquivo ou registro não pode se mover de um indivíduo ou grupo para o próximo sem mudar de estado. Alguns sistemas tornam possível dar a tarefa uma identidade própria, separada das pessoas trabalhando nela.

Por exemplo, suponha que um engenheiro trabalhando em um projeto queira conferir com os colegas a melhor maneira de abordá-lo. Como o modelo mestre e todos os arquivos de referência associados estão contidos e controlados por um

pacote, é simples passar o trabalho inteiro por qualquer número de pessoas diferentes sem encadear uma mudança de estado. O procedimento formal de workflow é descompromissado com este procedimento informal porque a autoridade de mudar o estado do arquivo não se move com o pacote. Ele se mantém com o indivíduo designado.

A comunicação dentro do time de projeto também é aumentada. Quando pacotes de dados e arquivos são passados, eles podem ser acompanhados por instruções, notas e comentários. Alguns sistemas têm capacidade de "redlining"; outros até têm provisões para arquivos de anotações informais com o equivalente eletrônico de notas "post-it".

Em outras palavras, os sistemas de administração de processo podem ser vistos como um jeito de aumentar a liberdade do seu ambiente de trabalho, ao mesmo tempo que o normaliza. O desafio é o quanto permitir de trabalho de grupo informal e ainda manter um controle da administração geral de custos e linhas finais de projeto.

A maioria dos sistemas permitem uma revisão de status da tarefa toda, com todos os dados de suporte, para ser registrada e visualizada por indivíduos autorizados a qualquer hora.

Um pacote representa uma tarefa em um projeto de desenvolvimento do produto de milhares. Cada pacote segue seu próprio caminho pelo sistema mas as relações entre pacotes também precisam ser controladas.

Para coordenar um workflow tão complexo efetivamente é necessário ser capaz de definir a interdependência das tarefas para se obter o jeito que o projeto

é estruturado. Nem todos os sistemas são fáceis de customizar desse jeito. Aqueles que têm a habilidade de criar uma relação hierárquica entre os arquivos. Por exemplo, o sistema poderia ser instruído a prevenir um engenheiro de disponibilizar uma montagem para liberação antes que todas as partes tenham sido individualmente liberadas.

3.1.2.3. Administração do histórico de trabalho

Como tem-se visto, os sistemas de PDM não devem apenas manter registros de base de dados do estado atual do projeto, eles devem registrar também os estados anteriores do projeto. Isso significa que eles são potencialmente uma origem valiosa de dados testados. A habilidade de realizar auditorias de processos regulares é um requisito fundamental para a conformidade com padrões internacionais de administração de qualidade, como o ISO9000, EN29000 e BS5750. Mas a administração do histórico de trabalho é também importante para permitir ao usuário voltar a um ponto específico do desenvolvimento do projeto onde o problema surgiu, ou no qual se deseja começar uma nova linha de desenvolvimento.

Para que os marcos específicos de desenvolvimento que o sistema registra são importantes? Alguns sistemas registram apenas as mudanças nas propriedades dos documentos. Então as propriedades podem ser traçadas em um ponto específico no tempo, mas não a própria modificação. Outros têm a habilidade de registrar mudanças mas devem fazer isso como uma série de "snap

– shots " obtidos apenas quando um arquivo muda de estado. Isso pode deixar grandes buracos no histórico do workflow como um usuário que pode fazer modificações no projeto por várias semanas sem qualquer mudança no estado. Alguns sistemas fornecem um registro de histórico que é como uma "figura móvel", permitindo ao usuário registrar mudanças para qualquer sistema – escolhe-se um nível definido – por exemplo, toda vez que um arquivo modificado é salvo.

Este nível de registro histórico, tão bom quanto disponibilizar auditorias, também permite o monitoramento ativo da performance individual – inviável durante o tempo crítico de projeto.

3.1.3. Benefícios

Alguns dos benefícios que o PDM traz são:

- Time-to-market reduzido;
- Maior produtividade de projeto;
- Maior acurácia de projeto e manufatura;
- Melhor uso da criatividade do grupo;
- Confortável de usar;
- Integridade dos dados assegurada;
- Melhor controle do projeto;
- Melhor administração das mudanças de engenharia;
- Grande passo rumo a qualidade total.

3.2. Workflow

Toda jornada longa exige um mapa. Antes de realizar reengenharia em uma corporação (ou em uma pequena parte dela), você e seus companheiros de trabalho precisarão de um template para visualizar o potencial das tecnologias de workflow de transformar seu negócio.

Fundamentalmente, você precisará concordar com uma definição de processo de negócio (também conhecido como workflow). Um bom ponto inicial é a definição encontrada em *The Reengineering Handbook: A Step-by-Step Guide to Business Transportation* por Raymond L. Manganelli e Mark M. Klein.

"Um processo ... é uma série inter-relacionada de atividades que converte as entradas de negócio em saídas de negócio."

Manganelli e Klein definem três tipos de atividades das quais os processos de negócio são compostos:

- *Agregação de valor:* Essas atividades de negócio trazem forma, substância e coerência para o produto final e são aquelas que mais importam para o consumidor. Dentro do ambiente de workflow, elas correspondem às aplicações de produtividade (por exemplo: leitores eletrônicos e design computacional) e que são automaticamente chamados pelo sistema para facilitar cada passo no processamento do item de trabalho.
- *Transferências:* Essas atividades de negócios transportam o trabalho pela função, departamento, ou limite organizacional. Dentro do ambiente do

workflow, elas correspondem às utilidades básicas ou trabalhos de rede – geralmente um sistema de distribuição de arquivos ou suporte principal de E-mail – usados para passar itens de trabalho de usuário para usuário.

- *Controle:* Essas atividades controlam os limites de transferências. Dentro do ambiente do workflow, eles correspondem às várias técnicas usadas para projetar e executar procedimentos automatizados e registrar a localização e status atuais dos itens de trabalho.

Workflow pode ser definido como o fluxo de informação e controle em um processo de negócio. Workflows devem ser seqüenciais no projeto, envolvendo transferências de um indivíduo A para um indivíduo B, e depois do indivíduo B para um indivíduo C. Eles devem ser paralelos no caso do indivíduo A transferir cópias de um mesmo item para os indivíduos B e C ao mesmo tempo. Um processo concorrente é um processo paralelo no qual os indivíduos B e C devem transferir o item para outro indivíduo ao mesmo tempo. Além disso, existem workflows condicionais, nos quais o indivíduo A transfere para os indivíduos B e C dependendo de algumas condições – por exemplo, direcionando todas as ordens de compra maiores de $10,000 para o indivíduo B.

Um programador de software pode intuitivamente utilizar este conceito de workflow como uma transferência de informação e controle, porque seu trabalho é definir um tipo de fluxo análogo – um que trabalha dentro do limite de um computador particular ou artifício computadorizado.

Um exemplo de workflow é mostrado na figura 3.1.

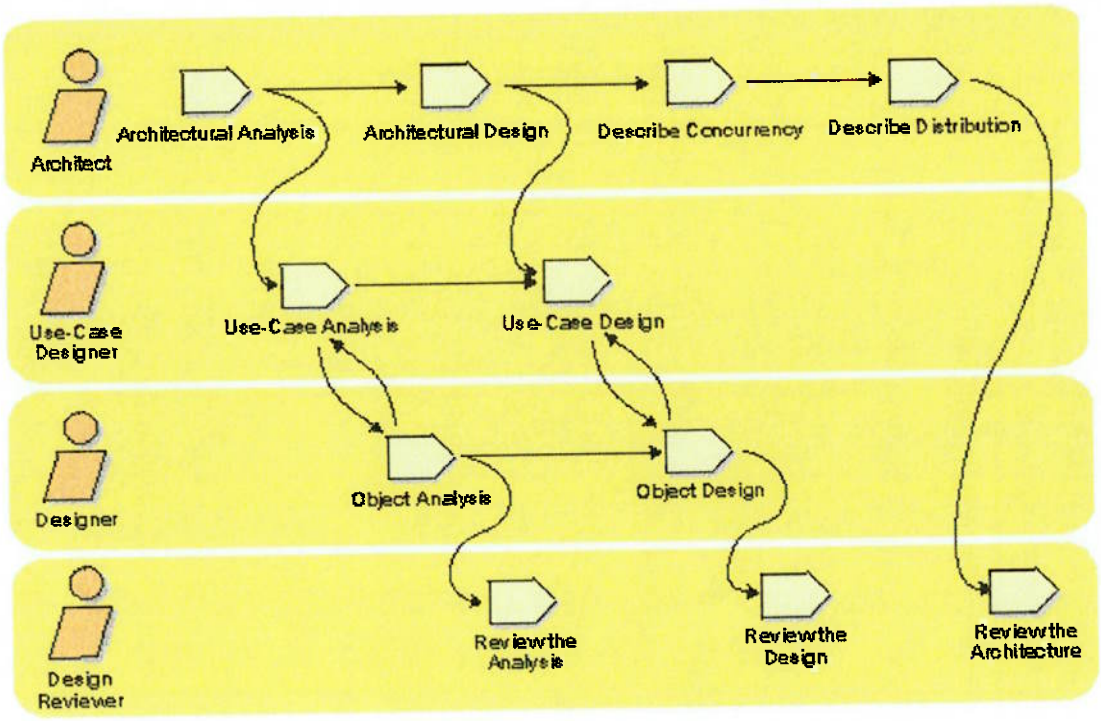


Figura 3.1 Exemplo de workflow

Note que não é sempre possível ou prático representar todas as dependências entre as atividades. Muitas vezes duas atividades são mais fortemente interligadas do que mostrado, especialmente quando envolvem o mesmo indivíduo ou trabalhador (vide RUP). Pessoas não são máquinas, e os workflows não podem ser interpretados literalmente como um programa para pessoas, para ser seguido exatamente ou mecanicamente.

As aplicações de workflow podem ser moldadas nos contornos precisos dos seus processos corporativos, sistemas e cultura. Para definir a solução de workflow que melhor se ajustar a certos requisitos, deve-se olhar para sua organização com novos olhos, sem estar influenciado pela estrutura e processos atuais, mas consciente das dificuldades políticas associadas a abandonar totalmente o processo atual e começar do zero.

4. ESPECIFICAÇÃO DE REQUISITOS

O objetivo deste trabalho é disponibilizar informações técnicas de projeto para os engenheiros e demais interessados na forma de um banco de dados acessível via rede contendo também informações complementares para tornar as consultas mais completas e eficientes.

Atualmente estas informações são obtidas por meio de consultas realizadas diretamente com os engenheiros responsáveis pelo dado ou com um profissional que centraliza estas informações e as distribui para os interessados via telefone, malote, fax ou e-mail. Os dados se encontram em arquivos de papel, ou em planilhas do Excel ou do Access.

Este método de consulta não é eficiente, pois apresenta um tempo de execução muito grande devido a atrasos com burocracia desnecessária, consultas a desenhos e arquivos em papel e dificuldade de acesso aos respectivos responsáveis pela informação, o que não acontece no método proposto por este trabalho.

Esta solução também deve prever se o usuário tem direito ou não a alterar o banco de dados e qual informação cada usuário pode ter acesso ou não, garantindo assim a integridade e a segurança das informações confidenciais e secretas e preservando a estratégia geral da empresa.

Outro requisito interessante deste projeto é a constante notificação ao usuário de alterações feitas em informações do banco de dados, para que o usuário fique sempre atualizado e evitando assim conflitos de projeto.

4.1. Informações técnicas de projeto

Estas informações são parâmetros e características de um produto que são úteis para sua descrição, fabricação e análise de desempenho. No caso de um carro, alguns tipos de dados e seus respectivos exemplos são:

- Identificação do veículo (versão);
- motor e transmissão (potência do motor e relações de marcha);
- pesos (peso bruto total);
- pneus (raio dinâmico);
- consumo de combustível (consumo na cidade);
- performance (velocidade máxima);
- capacidades de enchimento de reservatórios (circuito do freio);
- dimensões da carroceria (capacidade de porta-malas).

4.2. Outras informações disponíveis

Além da informação técnica de projeto em si e sua referência de plataforma, modelo, ano, carroceria, motor, e outros, serão fornecidos algumas informações extras para auxiliar o usuário a realizar uma consulta mais eficiente. Tais informações são:

- *Dados complementares:* pode-se acessar este campo para verificar todo o histórico de reuniões realizadas para definir esse dado e identificar as

justificativas pelas quais ele foi adotado como a melhor solução, ou para verificar o *status* de desenvolvimento do mesmo;

- *Dados correlacionados:* existem mudanças que não podem ser feitas isoladamente de outras, pois podem gerar inconsistências no projeto e ocasionar problemas fatais que poderiam ser prevenidos se houvesse uma maior comunicação entre as áreas da empresa ou mesmo entre engenheiros da mesma área. Um exemplo deste tipo de dado é um certo aspecto geométrico do motor de um veículo. Se o grupo de engenharia de powertrain desenvolver um motor mais potente mudando uma peça que altere o volume do motor, este pode não caber no espaço que o grupo de carroceria projetou para ele, ou se o peso do motor ficar maior e o grupo de chassis não for avisado que tem que projetar uma suspensão mais resistente, ou se o motor de arranque necessário for mais potente e o grupo de elétrica (EEHVAC) não for avisado que tem que usar um bateria mais potente. Este exemplo mostra uma alteração que mexe com todas as áreas, ou seja, pode-se ter a impressão que é óbvio que todos serão avisados sobre uma mudança deste tamanho, mas e as pequenas mudanças? Elas são mais difíceis de perceber e apresentam um risco muito grande para o projeto, pois podem tornar inviável um trabalho de meses. A empresa não pode correr riscos como esse, pois isso pode aumentar muito o *time to market* de seu produto final. Por isso, é necessário haver amarrações internas do software entre estes dados e os respectivos engenheiros, para que estes possam

acessar estas informações e estar na lista de e-mail destes para serem avisados sobre atualizações;

- *Lista de e-mail de interessados:* cada dado tem um grupo de pessoas que tem direito a acessá-lo. Essas pessoas devem ser avisadas via e-mail que seu valor foi alterado, pois estas podem guardar o valor antigo na memória e não realizar uma nova consulta quando precisar deste dado de novo devido ao hábito de manuseá-los constantemente, gerando inconsistências;
- *Outros:* informações adicionais podem ser incorporadas ao banco de dados à medida que se mostrarem interessantes do ponto de vista do usuário para facilitar sua utilização ou aumentar seus recursos disponíveis.

4.3. Consultas

O usuário poderá realizar a consulta dos dados de um a quatro veículos, permitindo comparações entre veículos da própria empresa ou com os da concorrência. Para facilitar a visualização dos dados, a apresentação destes será feita na forma de uma tabela, na qual cada coluna mostrará as informações de um veículo, deixando os dados de mesma natureza na mesma linha, facilitando assim a comparação entre eles.

Mas para obter esta tabela, o usuário deverá preencher um formulário com especificações que definem um veículo em particular. Este formulário deverá

ter interface amigável, facilitando a consulta e assim diminuindo o tempo da mesma e evitando a necessidade de treinamento.

Para facilitar a definição destas especificações, um formulário com menos itens e com especificações mais imediatas para o usuário deve ser disponibilizado, retornando o conjunto de especificações definidoras de todos os veículos que satisfazem a seleção determinada pelo usuário.

4.4. Perspectiva Histórica

Inicialmente, tomou-se contato com as tecnologias utilizadas para o desenvolvimento deste projeto, tais como e as linguagens que serviram para a construção dos programas do ambiente (HTML, SQL e JAVA); o software de banco de dados utilizado (access); a metodologia de projeto de software (RUP); um dos softwares que podem realizar interface futuramente com este sistema (PDM); e conceitos auxiliares (Workflow).

Depois disso, tomou-se conhecimento de como as informações técnicas de projeto são obtidas. Foi constatado que o administrador das informações obtém estes dados com o responsável pela plataforma (departamento de programas do produto) ou diretamente com o engenheiro responsável e os organiza segundo o conjunto a que pertence, mantendo sempre uma continuidade de tipos de dados.

Em um primeiro momento, esta metodologia será mantida, sendo o administrador das informações responsável pela atualização do banco de dados, mas com uma maior fiscalização do engenheiro responsável pelo dado pelo fato deste poder acessar o banco para confirmações.

Outro ponto importante é o fato do banco ter que ser atualizado a cada mudança do projeto do produto, o que é facilmente realizável utilizando os recursos de rede, sendo portanto tarefa do próprio administrador das informações.

O próximo passo foi consultar o administrador das informações atual para constatar qual a melhor maneira de se organizar a informação do banco de dados, quais os campos adicionais que devem constar nele e que tipo de consulta deve ser oferecida ao usuário para que este possa utilizar o banco da maneira mais eficiente. A resposta a essas perguntas se encontram nos itens 4.2 e 4.3 deste trabalho.

4.5. Perspectiva de Sistemas

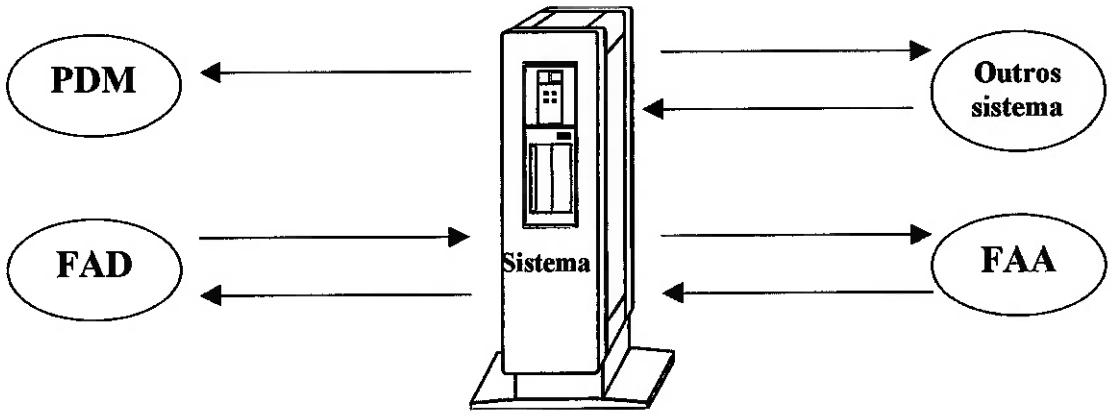


Figura 4.1: Possível integração com outros sistemas

- FAD (Ferramenta de Alteração de Dados) sistema que, utilizando um workflow padrão envolvendo diversas áreas, facilitará a aprovação de alterações de um dado qualquer do banco e, de acordo com o

resultado obtido, o modificará ou não. Note que este sistema não faz parte do escopo deste projeto. Ele é apenas uma sugestão de aperfeiçoamento do processo.

- FAA (Ferramenta de Atualização Automática) sistema que, quando um novo veículo tiver que ser inserido, auxiliará na atualização do banco de dados automaticamente de acordo com o responsável por definir o valor do dado. A mesma observação feita para o FAD cabe aqui.

O banco de dados também poderá fornecer ou receber dados de outros sistemas como o PDM, que é um sistema de informação geral para a fabricação do produto, utilizando um filtro adequado. Uma definição mais aprofundada do PDM é encontrada neste relatório no item 3.1.

4.6. Restrições Gerais

4.6.1. Desempenho

O tempo de resposta do banco deve ser muito menor do que o tempo de resposta que os procedimentos atuais utilizam para realizar uma consulta.

4.6.2. Ambiente de Execução e Contingência

O ambiente deve ser escalável, crescendo sob demanda. A capacidade pode ser aumentada com a adição de software e hardware duplicados.

A configuração básica para o servidor é uma workstation Sun Ultra 1 configurada com:

- CPU 167 Mhz
- 128MB RAM
- 2GB Disk

A configuração básica para um cliente PC é:

- 133 Mhz Pentium
- 32MB RAM
- 2GB Disk

A configuração básica para um cliente Sun é:

- SPARCstation IPC (SunOS 5.x) ou similar
- 32MB RAM
- 2GB Disk

4.6.3. Segurança

O banco de dados deve prever que tipo de dados cada usuário pode ter acesso, pois ele conterá várias informações confidenciais e secretas que podem prejudicar a estratégia da empresa se caírem em mãos erradas.

O sistema deve fornecer quatro níveis de restrição de acesso. A lista a seguir apresenta os quatro níveis de segurança necessários:

- *Segurança no nível de servidor:* Antes de poder acessar um banco de dados em um servidor, o usuário deve ter acesso a ele. O administrador do servidor controla seu acesso através da utilização de certificados anexados ao arquivo da sua identificação e acesso ás listas do servidor. Essa é a primeira e mais genérica camada de segurança.
- *Segurança no nível de banco de dados:* A segurança de banco de dados em um servidor é tratada pela lista de controle de acesso (LCA) ao banco de dados. A LCA relaciona usuários e servidores e lhes atribui direitos de acesso a esse banco de dados particular. Os níveis de acesso variam de Manager (Gerente) – que tem acesso total ao banco de dados – a No Acess (Sem acesso).
- *Segurança em nível de poder de alteração:* Segurança de poder de alteração consiste no controle de quais usuários podem alterar os dados do banco por meio da verificação do nível de acesso que o usuário tem na LCA.
- *Segurança em nível de campo:* A LCA também contém um campo para cada dado específico, que informa se o usuário pode ter acesso a ele ou não.

4.6.4. Ambiente Operacional

O ambiente operacional utilizado será:

- browser Internet Explorer 4.0;
- front end Java;

- servidor desenvolvido com linguagem Java
- base de dados access;

4.7. Perspectiva de Usuários, Produtos e Funções

O banco de dados de gerenciamento de informações técnicas de projeto é um produto único que cumpre todas as funções descritas.

Futuramente, novos produtos poderão ser definidos, a saber:

- *Ferramenta de ajuda para o usuário:* utilizada como auxílio ao atendimento de dúvidas do participante quanto ao procedimento de consulta e atualização do banco de dados;
- *Ferramenta de verificação de integridade da base de dados:* para identificação de dados repetidos, contraditórios, expirados, etc;
- *Filtro de conversão para PDM:* para levantar os dados disponíveis no banco de dados e convertê-los para os padrões do PDM.
- *Ferramenta de alteração de dados:* para realizar um workflow padrão envolvendo diversas áreas para a aprovação de alterações de um dado qualquer;
- *Ferramenta de atualização automática:* quando um novo veículo tiver que ser inserido, auxilia na atualização do banco de dados automaticamente de acordo com o responsável por definir o valor do dado.

4.7.1. Requisitos Funcionais

A seguir serão listados os requisitos funcionais identificados até o momento para o banco de dados para gerenciamento de informações técnicas de projeto:

- disponibilização de informações técnicas de projeto para consulta pelo usuário da rede da empresa (intranet) que puder ter acesso a esses dados;
- fácil manutenção da LCA (lista de controle de acesso) do banco devido a admissão, demissão ou mudança de cargo de cada usuário;
- disponibilidade de recursos para realizar comparações entre diferentes veículos com características similares;
- procedimento de manutenção e atualização dos dados rápido e fácil, com possibilidade de criação de novos formulários para novos produtos;
- fácil integração do banco de dados com o usuário (interface *"easy to use"*);
- fácil alteração de cada formulário de dados (inclusão, exclusão ou alteração de campos em um documento);
- notificação automática via e-mail sobre alterações de dados do banco aos usuários que têm direito de acesso a estes dados;
- disponibilização de dados extras para facilitar as consultas realizadas pelo usuário.

4.7.2. Restrições Funcionais

Há vários tipos de informações que poderiam ser disponibilizadas pelo banco de dados de gerenciamento de informações técnicas de projeto, devido à característica bem flexível de sua solução. Isto implicaria em um maior custo de manutenção, já que o número de dados cresceria consideravelmente. Deste modo, estaria desviando-se de um dos objetivos principais do banco, que é prover um meio fácil e rápido de manutenção dos dados. Assim, decidiu-se deixar algumas informações fora do banco de dados. Além disso, inicialmente serão disponibilizados apenas os dados dos produtos de quatro empresas, deixando os produtos das outras para um segundo momento deste projeto.

4.8. Perspectiva Gerencial

4.8.1. Evolução

Ciclo 1: desenvolvimento dos formulários, visões e Interface para usuários.

Horizonte: 03 Mai 99 – 30 Nov 99

Ciclo 2: desenvolvimento dos Procedimentos de Atualização e Manutenção.

Horizonte: 30 Nov 99 – 10 Dez 99

Ciclo 3: desenvolvimento das outras ferramentas de apoio.(fora do TF)

Horizonte: 03 Jan 00 – 31 Jun 00

4.8.2. Detalhamento do Ciclo 1

Fase Inception, Iteração 1

Produto: RGS básico com o use-case de cadastro/consulta de Technical Data

Duração: 03 Mai 99-07 Jun 99

Fase Elaboration, Iteração 1

Produtos: Architecture Views

Apresentação do plano de envolvimento

Duração: 07 Jun 99- 01 Ago 99

Fase Construction, Iteração 1

Produtos: Protótipo para cadastro/consulta do Technical Data

Duração: 07 Jun99- 15 Ago 99

Fase Transition, Iteração 1 - (fora do TF)

Produtos: Implantação do Technical Data

Duração: 15 Ago 99 - 30 Ago 99

Fase Inception, Iteração 2

Produto: RGS completo

Duração: 15 Ago 99 – 30 Ago 99

Fase Elaboration, iteração 2

Produtos: Design

Plano de desenvolvimento

Duração: 01 Set 99 – 30 Nov 99

Fase Construction, iteração 2

Produtos: código

Duração: 01 Set 99 – 30 Nov 99

Fase Transition, iteração 2 - (fora do escopo do TF)

Produtos: implantação na GM

Duração: 30 Nov 99 – 31 Jun 00

5. O AMBIENTE

5.1. Descrição geral

A solução desenvolvida para suprir as necessidades enunciadas na especificação de requisitos foi um site na intranet da empresa que acessa um banco de dados via um programa que gerencia todas as suas atividades. O ambiente desenvolvido pode ser descrito pela figura 5.1 abaixo.

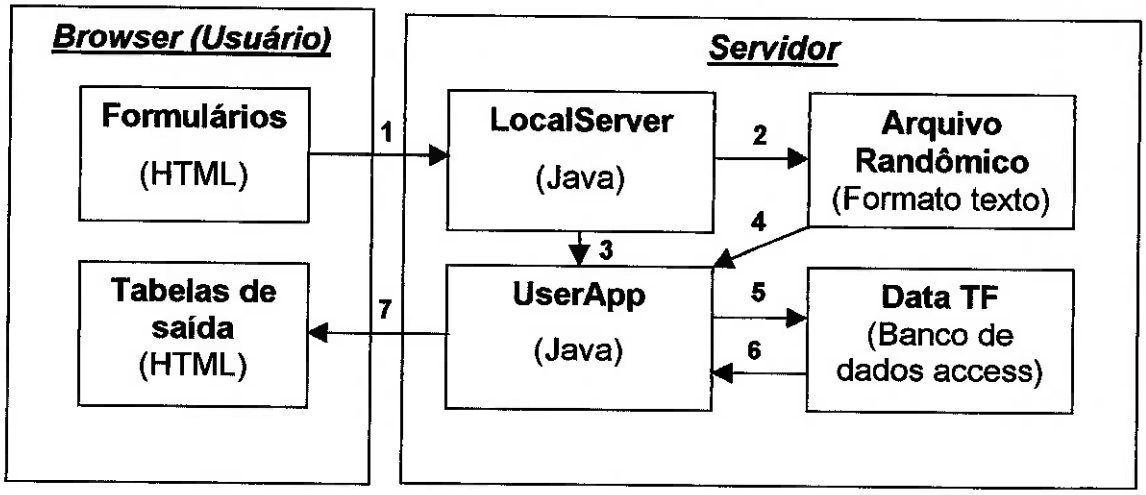


Figura 5.1 Descrição do ambiente

Para melhor explicar como funciona o ambiente, é melhor começar pelos seus componentes, que são:

- **Formulários:** por meio de menus que serão mostrados mais adiante neste trabalho, o usuário chega a um formulário de consulta (visualização de dados), inserção de um novo veículo, de deletar um veículo já existente, alteração dos dados de um veículo, ou de verificação das possíveis configurações (especificações). Estes

formulários estão no formato HTML e servem para coletar as opções de alteração ou consulta do usuário.

- **LocalServer:** é um programa em Java que faz o papel do servidor do ambiente.
- **Arquivo Randômico:** consiste em um arquivo gerado pelo servidor com nome aleatório e que contém todas as seleções feitas pelo usuário no formulário.
- **UserApp:** é o programa principal do ambiente. Ele recebe as entradas feitas pelo usuário no formulário, faz as consultas ou alterações necessárias no banco de dados e então apresenta os resultados em formato HTML no browser para o usuário na forma de tabela de dados, no caso de uma consulta, ou na forma de mensagens, no caso de uma alteração no banco de dados ou algum erro de processamento.
- **Data TF:** consiste no arquivo de banco de dados na plataforma access que contém todos os dados dos veículo, a LCA (lista de controle de acesso) e a lista de documentos auxiliares.
- **Tabelas de saída:** são as saídas do programa UserApp e o resultado final da consulta especificada pelo usuário. Se a ação selecionada for relacionada a alterações do banco de dados, a saída é uma mensagem que informa se a operação foi bem sucedida ou não e, dependendo da ação selecionada, um link para mandar um e-mail para os usuários que têm acesso aos dados alterados.

Vamos agora entender melhor as relações entre os entes descritos acima.

Inicialmente, o usuário faz suas seleções no formulário correspondente a operação que deseja realizar e dá um clique no botão de ação do formulário. Instantaneamente, as seleções realizadas vão para um porte, que é um canal de comunicação com a rede, e são resgatadas pelo programa servidor (LocalServer), correspondendo a relação 1 da figura 5.1.

Já com as seleções feitas pelo usuário, o programa servidor cria um arquivo randômico (com nome aleatório) e escreve as seleções nele (relação 2), e imediatamente inicia a execução do programa principal (UserApp) passando como parâmetro o nome do arquivo randômico (relação 3).

Então, o programa UserApp é executado e, por meio do parâmetro passado na sua chamada, abre o arquivo randômico e lê o seu conteúdo (relação 4), adquirindo as informações necessárias para executar os comandos de acesso ao banco de dados em SQL (relação 5), obtendo assim os resultados requeridos do banco de dados (relação 6) e imprimindo na tela do browser as saídas resultantes (relação 7).

A conexão entre o programa UserApp e o banco de dados é feita por uma ponte ODBC de 32 bits, que pode ser configurada no painel de controle do computador realizando-se os seguintes procedimentos:

- Clique no botão "Iniciar" da barra de tarefas de seu computador;
- Selecione "configurações" e em seguida clique em "painel de controle";
- Dê um duplo clique no ícone "ODBC de 32 bits";
- Em "NFD do Usuário", apenas selecione "Banco de dados MS Access 97 " e clique no botão "adicionar";
- Dê um double click em "Driver para o Microsoft Access (*.mdb)";

- Clique no botão "selecionar" e selecione o banco de dados em questão, no caso o arquivo DATATF.mdb;
- Digite "UserApp" no campo "Nome da fonte de dados;
- Clique no botão "OK";
- Se apareceu um item com nome UserApp em "NFD do Usuário", o procedimento está completo;
- Feche o painel de controle.

A figura 5.2 mostra como deve estar preenchida a sua caixa de diálogo.

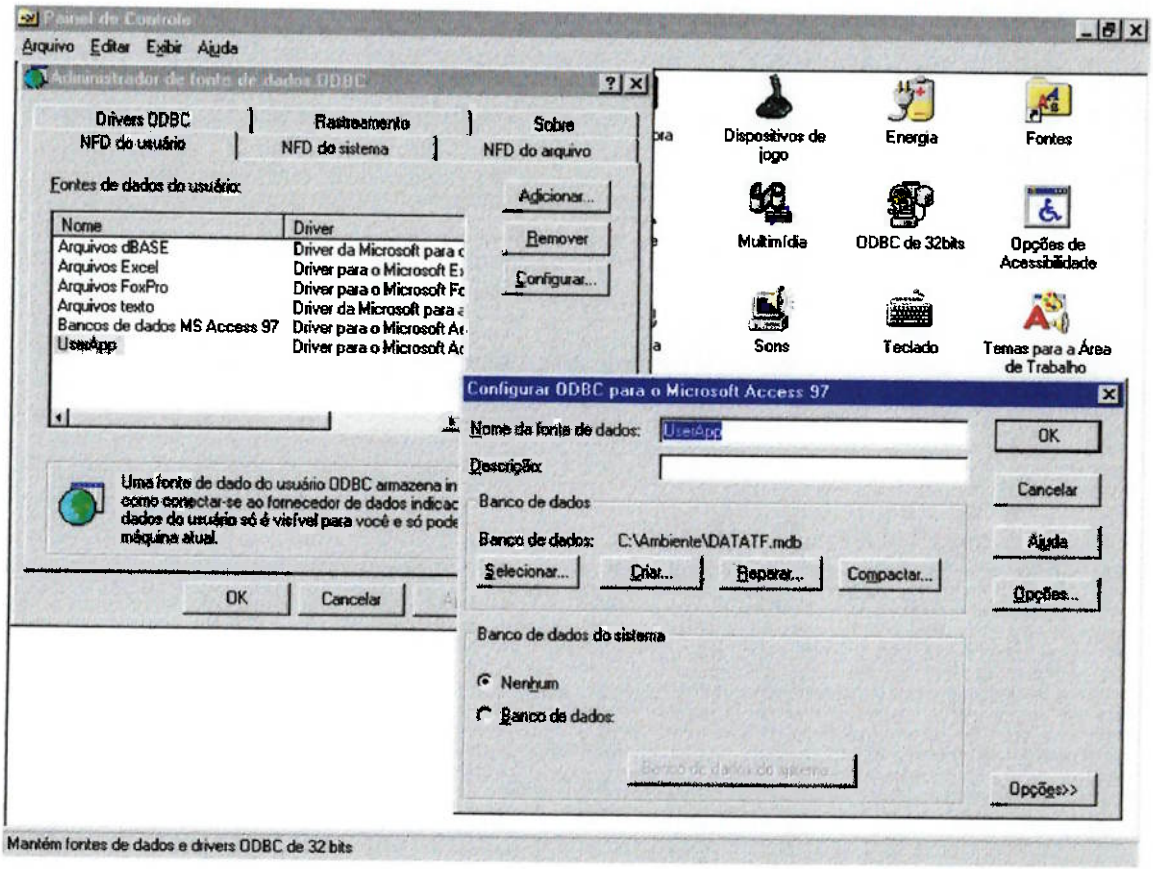


Figura 5.2 Caixa de diálogo para configurar a ODBC de 32 bits

Para um melhor entendimento do ambiente, segue uma explicação um pouco mais aprofundada de cada ente do sistema.

5.2. O servidor do ambiente (LocalServer)

O servidor do ambiente é responsável por:

- Apresentar no browser a tela de início do sistema quando o usuário digitar uma palavra chave específica (URL do site) na barra de endereços;
- Fazer com que o programa principal do sistema (UserApp) seja executado quando for solicitada uma operação qualquer e servir de ponte de dados entre o formulário e o programa principal.

Para cumprir estas funções, o servidor deve estar sempre em execução (no ar), para atender aos repentinos comandos do usuário.

Com relação à primeira função, dois arquivos auxiliares são necessários. O primeiro é o arquivo "hosts", que deve ser armazenado no diretório windows da máquina e não deve ter nenhuma extensão de arquivo. Ele contém o endereço da máquina utilizada (no exemplo abaixo é 127.0.0.1) e a palavra chave que deve ser digitada na barra de endereços do browser (localhost). A única linha contida neste arquivo é a que se encontra logo abaixo:

127.0.0.1 localhost

Note que o URL do site (localhost) pode ser trocado apenas mudando a palavra chave que se encontra neste arquivo "hosts".

O segundo arquivo auxiliar necessário é o arquivo "server.properties", que deve se encontrar no mesmo diretório que o programa LocalServer. Ele contém

o caminho de diretórios até a página inicial do site, o número do porte que o servidor utiliza e o nome do arquivo em HTML que é a página inicial do software. Um exemplo do seu conteúdo se encontra logo abaixo:

```
# Webster configuration file
#
root=c:/ambiente
portnumber=80
defaultfile=inicio.html
```

Além destes dois arquivos, LocalServer também necessita do arquivo HttpServer para funcionar. Isto porque, no jargão da linguagem Java, o objeto LocalServer estende o objeto HttpServer (em uma linguagem simplificada, isso quer dizer que o objeto LocalServer tem todas as funções que HttpServer e mais algumas novas, sendo que algumas destas funções só são definidas em HttpServer, tornando este arquivo indispensável).

Basicamente, o funcionamento do servidor se resume aos seguintes passos:

1. Se o servidor não estiver "no ar", o administrador do servidor deve iniciar sua execução;
2. O usuário digita o URL do site na barra de endereços do browser;
3. Por meio do arquivo hosts, o servidor é acionado e procura qual é o caminho e nome do arquivo HTML da página inicial via arquivo server;

4. Se as configurações estiverem certas, o browser exibe a página inicial do site.

Se a função exigida for a de executar uma função, os passos são os seguintes:

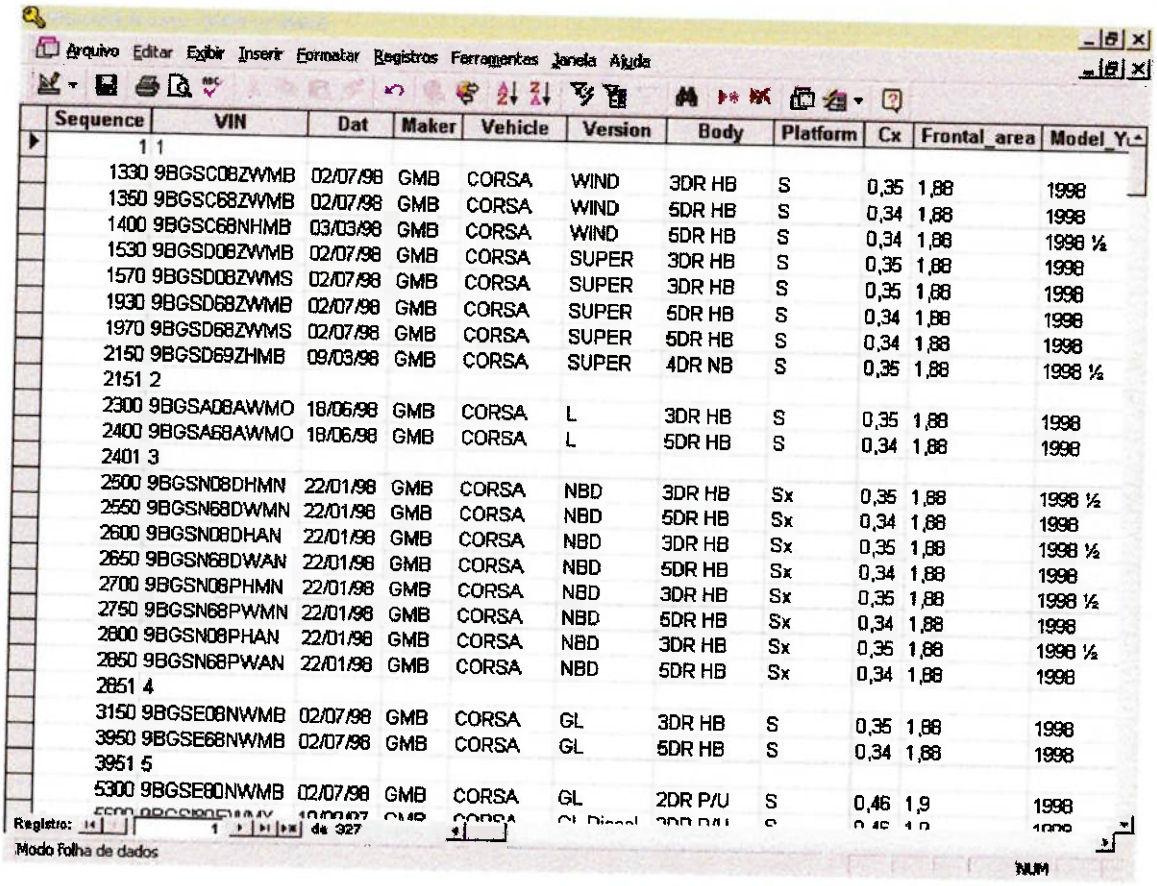
1. Considerando que o servidor está "no ar", o usuário dá um clique em um botão de execução qualquer do formulário e as seleções definidas pelo usuário seguem no porte especificado pelo arquivo server;
2. O servidor resgata as seleções no porte especificado e os armazena em um arquivo randômico gerado por ele;
3. Por fim, o servidor executa o programa principal (UserApp), passando como parâmetro o nome do arquivo randômico.

5.3. Arquivo de banco de dados

O arquivo "DATATF.mdb", que é um arquivo de banco de dados da plataforma access, contém três tabelas, que são:

- GMB;
- User;
- Referencia.

A primeira delas consiste em um banco de dados de todas as informações do veículo. Um exemplo desta tabela se encontra na figura 5.3.



Arquivo Editar Exibir Inserir Formatar Registros Ferramentas Janela Ajuda

Sequence	VIN	Dat	Maker	Vehicle	Version	Body	Platform	Cx	Frontal_area	Model_Y
1	1									
1330	9BGSC08ZWMB	02/07/98	GMB	CORSA	WIND	3DR HB	S	0,35	1,88	1998
1350	9BGSC68ZWMB	02/07/98	GMB	CORSA	WIND	5DR HB	S	0,34	1,88	1998
1400	9BGSC68NHMB	03/03/98	GMB	CORSA	WIND	5DR HB	S	0,34	1,88	1998 ½
1530	9BGSD08ZWMB	02/07/98	GMB	CORSA	SUPER	3DR HB	S	0,35	1,88	1998
1570	9BGSD08ZWMS	02/07/98	GMB	CORSA	SUPER	3DR HB	S	0,35	1,88	1998
1930	9BGSD68ZWMB	02/07/98	GMB	CORSA	SUPER	5DR HB	S	0,34	1,88	1998
1970	9BGSD68ZWMS	02/07/98	GMB	CORSA	SUPER	5DR HB	S	0,34	1,88	1998
2150	9BGSD69ZHMB	09/03/98	GMB	CORSA	SUPER	4DR NB	S	0,35	1,88	1998 ½
2151	2									
2300	9BGSA08AWMO	18/06/98	GMB	CORSA	L	3DR HB	S	0,35	1,88	1998
2400	9BGSA68AWMO	18/06/98	GMB	CORSA	L	5DR HB	S	0,34	1,88	1998
2401	3									
2500	9BGSN08DHMN	22/01/98	GMB	CORSA	NBD	3DR HB	Sx	0,35	1,88	1998 ½
2550	9BGSN68DWMN	22/01/98	GMB	CORSA	NBD	5DR HB	Sx	0,34	1,88	1998
2600	9BGSN08DHAN	22/01/98	GMB	CORSA	NBD	3DR HB	Sx	0,35	1,88	1998 ½
2650	9BGSN68DWAN	22/01/98	GMB	CORSA	NBD	5DR HB	Sx	0,34	1,88	1998
2700	9BGSN08PHMN	22/01/98	GMB	CORSA	NBD	3DR HB	Sx	0,35	1,88	1998 ½
2750	9BGSN68PWMN	22/01/98	GMB	CORSA	NBD	5DR HB	Sx	0,34	1,88	1998
2800	9BGSN08PHAN	22/01/98	GMB	CORSA	NBD	3DR HB	Sx	0,35	1,88	1998 ½
2850	9BGSN68PWAN	22/01/98	GMB	CORSA	NBD	5DR HB	Sx	0,34	1,88	1998
2851	4									
3150	9BGSE08NWMB	02/07/98	GMB	CORSA	GL	3DR HB	S	0,35	1,88	1998
3950	9BGSE68NWMB	02/07/98	GMB	CORSA	GL	5DR HB	S	0,34	1,88	1998
3951	5									
5300	9BGSE80NWMB	02/07/98	GMB	CORSA	GL	2DR P/U	S	0,46	1,9	1998

Registro: 1 de 327

Modo folha de dados

NUM

Figura 5.3 Uma parte da tabela GMB

O campo mais importante desta tabela é o "VIN", pois ele é a referência na tabela para o comando SQL definir qual o veículo (linha da tabela) que o usuário deseja realizar uma consulta. Este código de controle é formado por oito partes, sendo que cada uma corresponde a uma especificação do veículo, que é uma característica que classifica um veículo quanto a um critério. Cada veículo tem o seu VIN, que é único. A estrutura deste código é baseada na estrutura do número VIN *(Vehicle Identification Number)*, que é um número padronizado pela indústria automobilística de identificação de um veículo, porém com dígitos a mais para definir características que não eram previstas no VIN real. Alguns dos números que constam nesta tabela, que pode ser encontrada no disquete em anexo, são números inventados mesmo na parte

que imita o VIN, pelo fato do código original não ter sido achado. Uma descrição mais detalhada da estrutura do campo "VIN" será enunciada mais tarde neste trabalho.

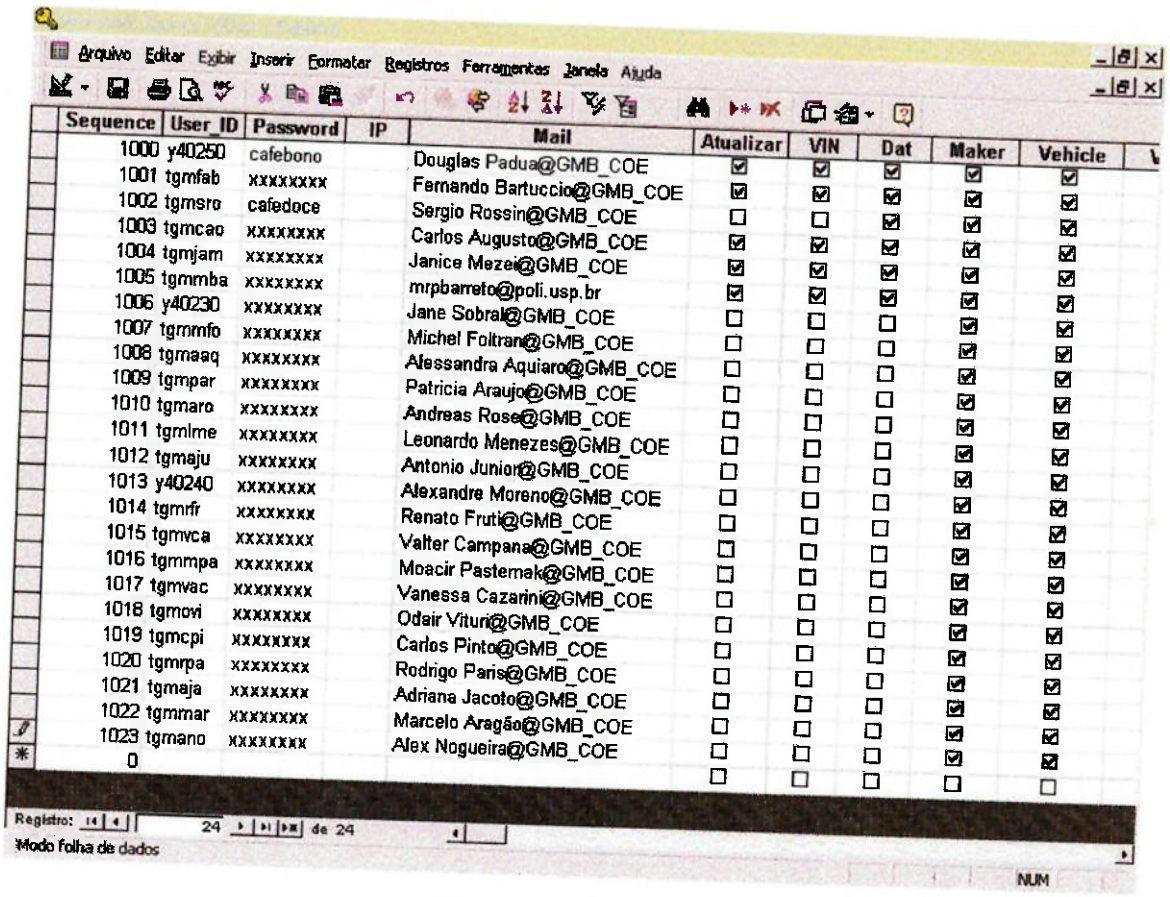


Figura 5.4 Tabela User

A segunda tabela do banco de dados, que é a tabela User mostrada na figura 5.4, contém informações diretamente ligadas a cada usuário. São elas:

- *User_ID*, que é a identificação do usuário que deve ser digitada na página inicial do site;
- *Password*, que consiste na senha para aquele User_ID na página inicial;

- *IP,* que é a identificação da máquina que o usuário está utilizando para acessar a página, e é necessário para possibilitar dois usuários acessarem o site ao mesmo tempo;
- *Mail,* que é o endereço eletrônico de e-mail do usuário;
- *Atualizar,* que consiste em um campo que diz se o usuário tem direito a atualizar o banco de dados (administrador do banco) ou não;
- *Outros,* que são campos específicos para cada dado e informam se o usuário tem direito ou não de visualizar o respectivo dado.

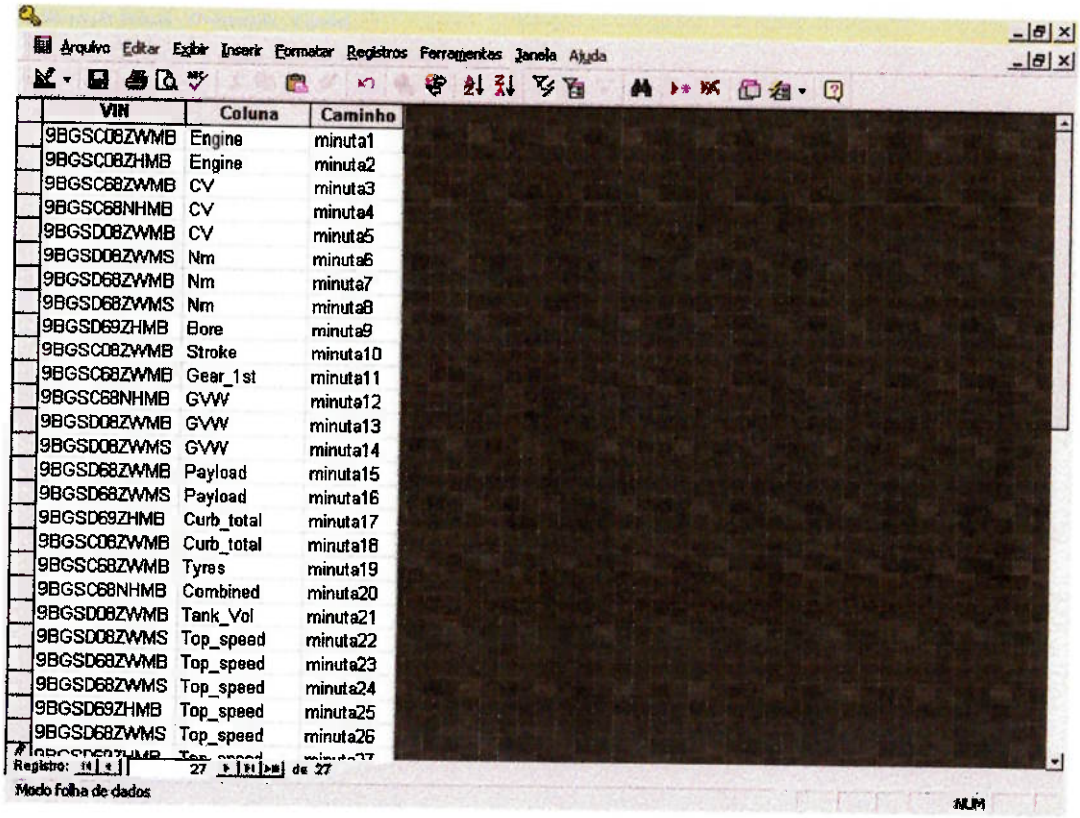


Figura 5.5 Tabela Referencia

Finalmente, a tabela Referencia diz respeito aos documentos auxiliares. Cada registro contém o nome do arquivo (Caminho), a qual veículo ele pertence (VIN) e a que dado ele deve estar vinculado (Coluna).

5.4. Arquivo Randômico

O arquivo Randômico é gerado a partir do servidor, que o associa a um nome que um número gerado a partir de uma função randômica. Ele armazena as seleções feitas pelo usuário no formulário, sendo que cada seleção é uma variável. O formato básico desta estrutura de armazenamento é:

Nome da variável + = + *Valor da variável* + & + *Nome da variável* + = ...

Um exemplo de um pedaço deste arquivo é:

fabricantet=NADA&versaos=NADA&fabricantes=NADA&versaoq=NADA ...

O programa UserApp deve ler este arquivo e converter este formato em variáveis do programa que armazenem os respectivos valores em cada uma delas.

6. MENUS, FORMULÁRIOS, ENTRADAS E SAÍDAS

O usuário pode navegar pelo site por meio de menus que auxiliam na escolha da operação desejada. O usuário pode acessar outra página por meio de um clique em um botão ou link. As relações entre as telas de menus e formulários estão na figura abaixo:

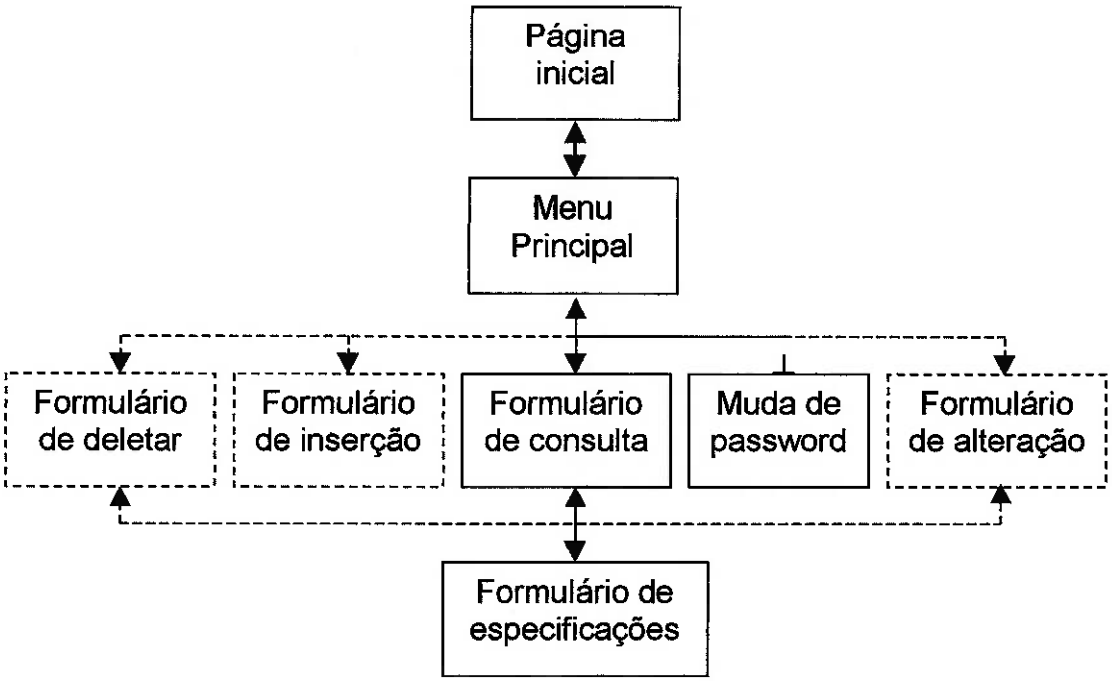


Figura 6.1 Relações entre os menus e formulários

Observe que os três blocos tracejados (formulários de inserção, de deletar e alteração) valem apenas para os usuários que têm direito a alterar os dados do banco (administrador).

Com a digitação da URL do site na barra de endereços do browser, o usuário visualizará a seguinte página inicial:

Figura 6.2 Página inicial

Se o usuário entrar com o password correto, então a seguinte tela aparecerá no browser, dependendo se o usuário puder alterar o banco de dados ou não:

(a) (b)

Figura 6.3 Menu principal (a) Usuário administrador e (b) Usuário comum.

Esta diferenciação é possível pelo fato das páginas não serem estáticas. Elas são impressas na tela a partir do programa UserApp. Isto será melhor explicado mais a frente neste trabalho.

No menu principal existem botões que possibilitam o acesso do usuário a outras páginas, conforme a figura 6.1, que mostra as relações entre os menus e formulários.

O primeiro botão é o de visualização de dados, que abre um formulário como o da figura 6.4 logo abaixo.

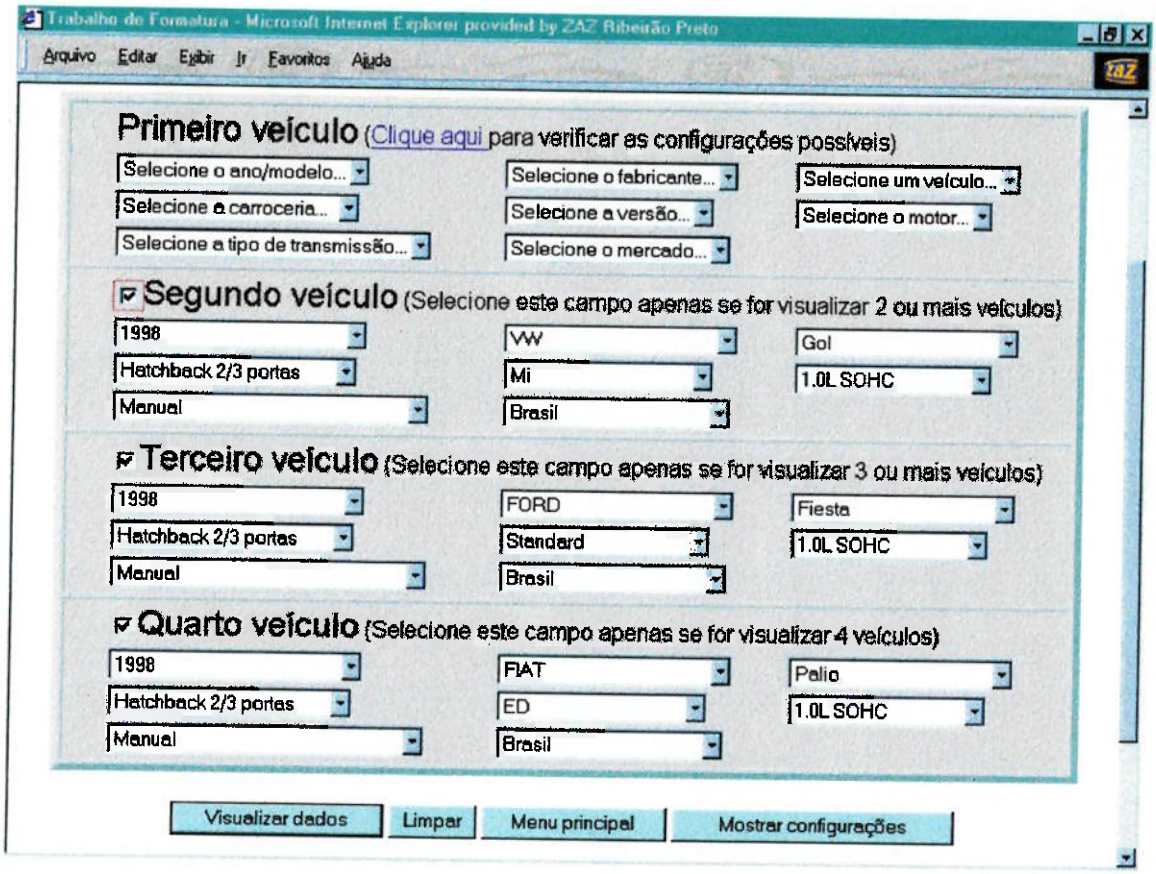


Figura 6.4 Formulário de consulta

Se o usuário estiver interessado em consultar apenas um carro, então ele apenas terá que selecionar os campos do primeiro veículo. Note que,

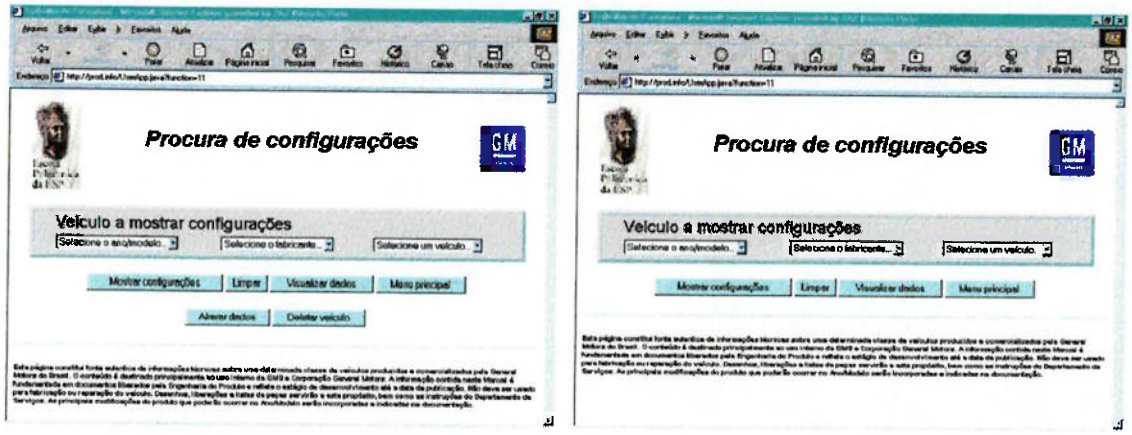
originalmente, todos os campos estarão inicializados como os campos do primeiro veículo (Selecione ...). A figura 6.4 apresenta os campos dos três últimos veículos selecionados apenas para servir de exemplo de seleção.

Se o usuário tiver interesse em visualizar os dados de dois ou mais veículos, então ele deve selecionar o campo do veículo em questão e selecionar corretamente as especificações deste último.

As especificações necessárias para definir um veículo e seus respectivos exemplos são:

- Ano/modelo (1998);
- Fabricante (GMB);
- Veículo (corsa);
- Carroceria (Hatchback 2/3 portas);
- Versão (Wind);
- Motor (1.0L SOHC);
- Tipo de transmissão (manual);
- Mercado (Brasil).

Mas suponhamos que o usuário não saiba ao certo se a opção que ele deseja realmente exista ou quais as opções que ele tem de possíveis configurações para um veículo. Então ele tem a opção de dar um clique no link que aparece logo ao lado do primeiro veículo ou no botão "Mostrar configurações" que aparece no final da página para acessar o formulário de especificações (ou configurações), mostrado na figura 6.5.



(a) (b)

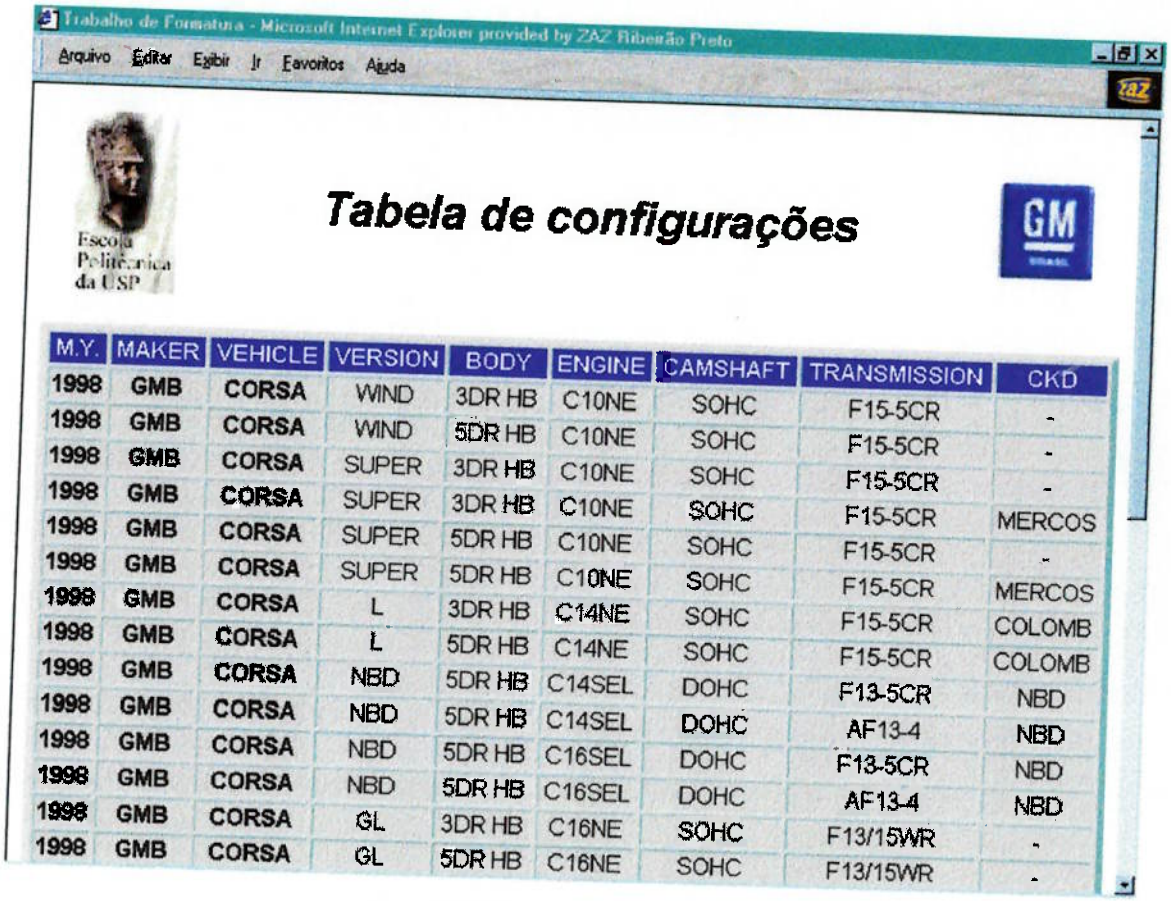
Figura 6.5 Formulário de especificações (a) Usuário administrador e (b) Usuário comum

Note que os formulários de alteração e de deletar também acessam esta página. Por isso, ela apresenta os botões de volta para estas páginas se o usuário tiver acesso de administrador do banco de dados.

Neste formulário, o usuário deve selecionar as seguintes especificações:

- Ano/modelo (1998);
- Fabricante (GMB);
- Veículo (Corsa).

Depois de selecionar os campos acima, o usuário deve dar um clique no botão "Mostrar configurações" para obter todas as possíveis especificações de veículo existentes de serem selecionadas mo formulário de visualização de veículos, como mostrado na figura 6.6 abaixo.



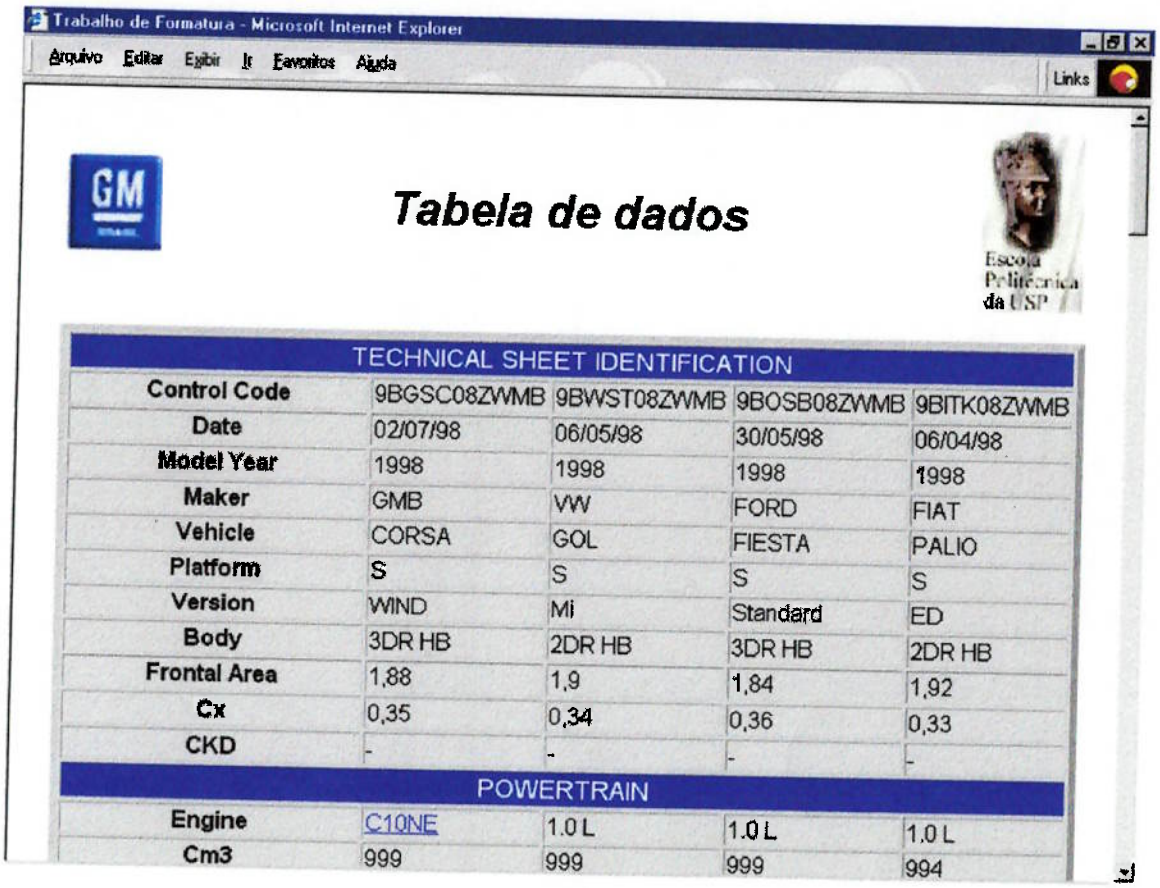
M.Y.	MAKER	VEHICLE	VERSION	BODY	ENGINE	CAMSHAFT	TRANSMISSION	CKD
1998	GMB	CORSA	WIND	3DR HB	C10NE	SOHC	F15-5CR	-
1998	GMB	CORSA	WIND	5DR HB	C10NE	SOHC	F15-5CR	-
1998	GMB	CORSA	SUPER	3DR HB	C10NE	SOHC	F15-5CR	-
1998	GMB	CORSA	SUPER	3DR HB	C10NE	SOHC	F15-5CR	MERCOS
1998	GMB	CORSA	SUPER	5DR HB	C10NE	SOHC	F15-5CR	-
1998	GMB	CORSA	SUPER	5DR HB	C10NE	SOHC	F15-5CR	MERCOS
1998	GMB	CORSA	L	3DR HB	C14NE	SOHC	F15-5CR	COLOMB
1998	GMB	CORSA	L	5DR HB	C14NE	SOHC	F15-5CR	COLOMB
1998	GMB	CORSA	NBD	5DR HB	C14SEL	DOHC	F13-5CR	NBD
1998	GMB	CORSA	NBD	5DR HB	C14SEL	DOHC	AF13-4	NBD
1998	GMB	CORSA	NBD	5DR HB	C16SEL	DOHC	F13-5CR	NBD
1998	GMB	CORSA	NBD	5DR HB	C16SEL	DOHC	AF13-4	NBD
1998	GMB	CORSA	GL	3DR HB	C16NE	SOHC	F13/15WR	-
1998	GMB	CORSA	GL	5DR HB	C16NE	SOHC	F13/15WR	-

Figura 6.6 Tabela de configurações

Agora, com as especificações do veículo desejado definidas, pode-se preencher o formulário de visualização (consulta) dos dados (figura 6.4) e, com um clique no botão "Visualizar dados", obtém-se a tabela da figura 6.7 para a seleção da figura 6.6 com o campo do primeiro veículo preenchido com as configurações da primeira linha da tabela da figura 6.6:

É importante lembrar que nesta figura só foi mostrada uma pequena parte da tabela, que contém dados sobre o motor, transmissão, peso, pneus, consumo de combustível, performance, capacidades de enchimento de reservatórios e dimensões da carroceria do veículo.

Para visualizar a tabela inteira, é só seguir os passos descritos até aqui com os arquivos disponibilizados no disquete em anexo deste trabalho.



TECHNICAL SHEET IDENTIFICATION				
Control Code	9BGSC08ZWMB	9BWST08ZWMB	9BOSB08ZWMB	9BITK08ZWMB
Date	02/07/98	06/05/98	30/05/98	06/04/98
Model Year	1998	1998	1998	1998
Maker	GMB	VW	FORD	FIAT
Vehicle	CORSA	GOL	FIESTA	PALIO
Platform	S	S	S	S
Version	WIND	Mi	Standard	ED
Body	3DR HB	2DR HB	3DR HB	2DR HB
Frontal Area	1,88	1,9	1,84	1,92
Cx	0,35	0,34	0,36	0,33
CKD	-	-	-	-
POWERTRAIN				
Engine	C10NE	1.0 L	1.0 L	1.0 L
Cm3	999	999	999	994

Figura 6.7 Tabela de dados

Visualizando com um pouco mais de cuidado a tabela de dados da figura 6.7 acima, nota-se que existe um link no item do motor do corsa. Ele "aponta" para um documento auxiliar, que serve para verificar todo o histórico de reuniões realizadas para definir esse dado e identificar as justificativas pelas quais ele foi adotado como a melhor solução, ou para verificar o *status* de desenvolvimento do mesmo. Um exemplo é mostrada na figura 6.8.

Note que o documento em questão se encontrava originalmente no formato do Lotus Notes (software de e-mail) e foi transformado em HTML. Os outros tipos de formato podem seguir o mesmo princípio de conversão para serem exibidos como documentos auxiliares.

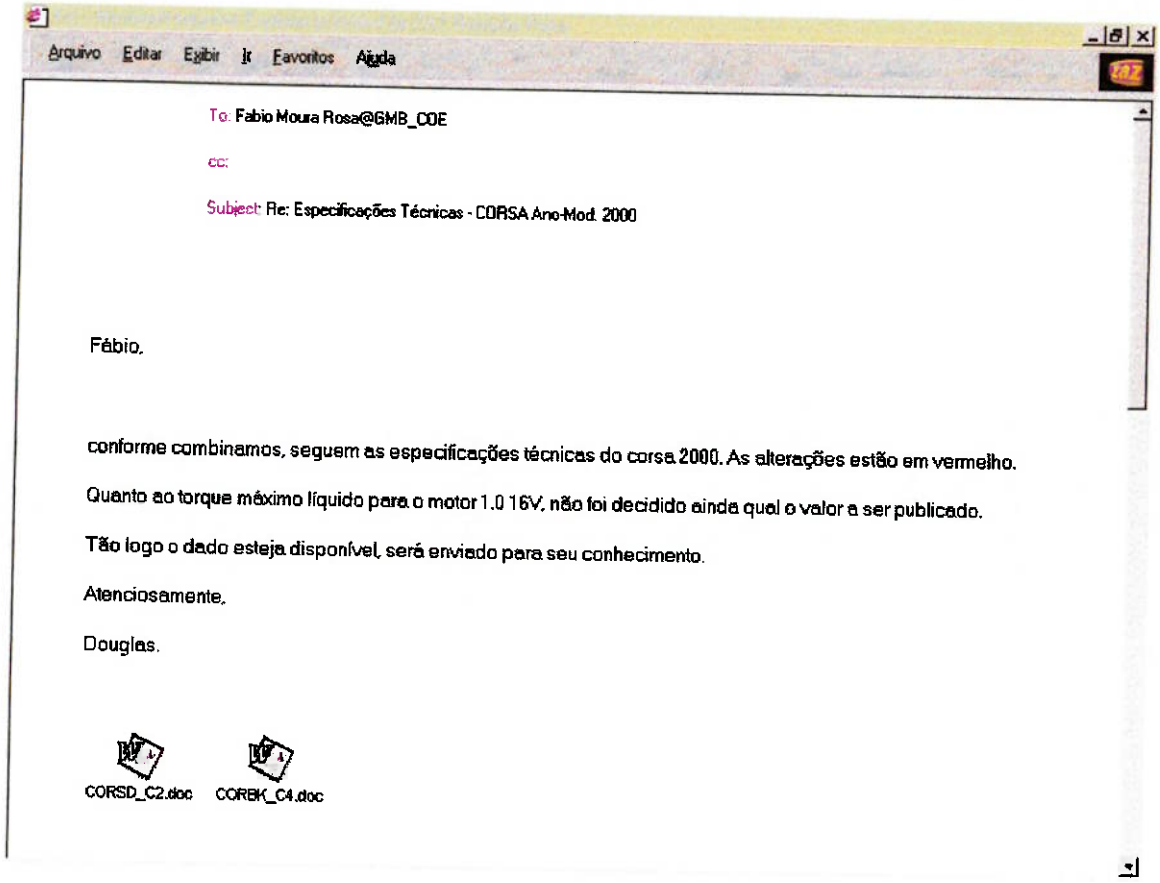


Figura 6.8 Documento auxiliar

Analisado agora o segundo botão do menu principal, mais uma vez lembrando que este botão só estará acessível se o usuário tiver direitos de administrador do banco de dados (figura 6.3), o botão "Inserir veículo" acessa o formulário de inserção, que se encontra na figura 6.9.

O administrador do banco de dados deve preencher o formulário com os dados do novo veículo que tem disponíveis e dar m clique no botão "Inserir registro" para que os dados sejam armazenados no banco de dados. Será retornada uma mensagem informando, se a operação foi bem sucedida ou não.

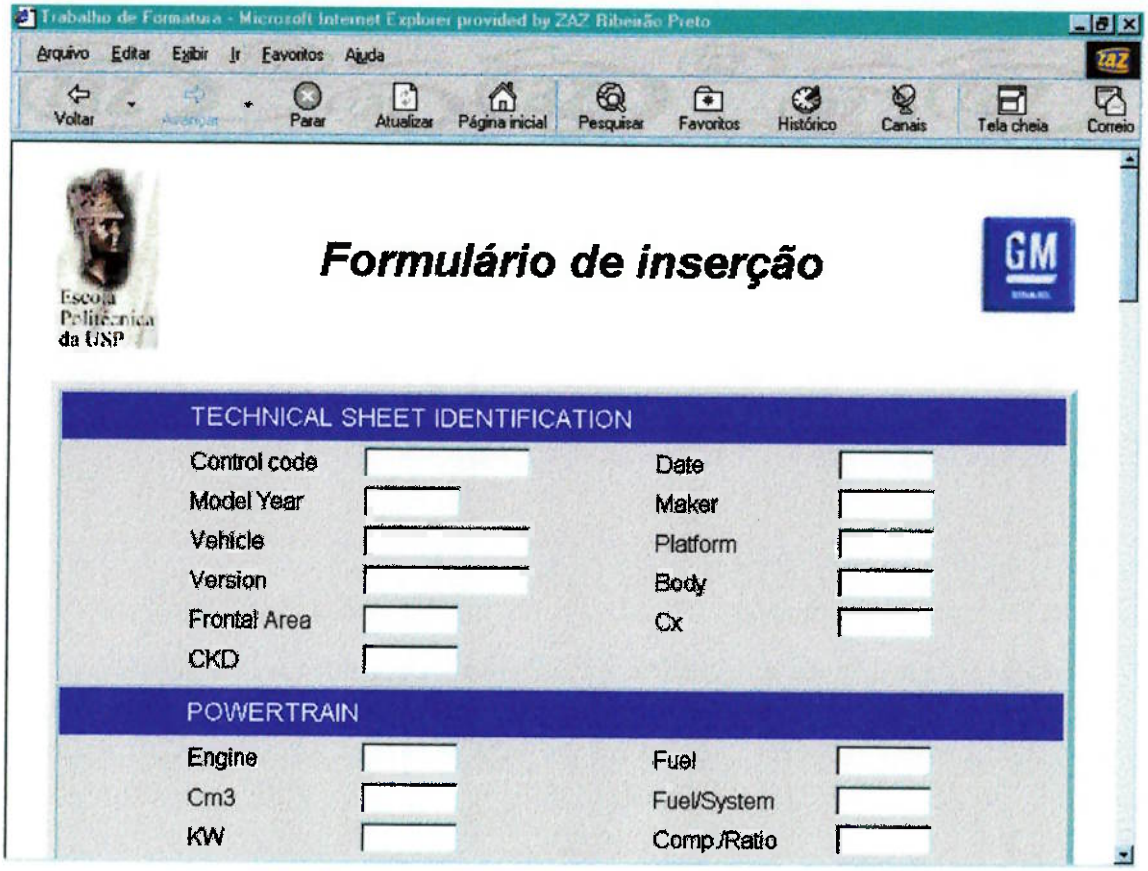


Figura 6.9 Formulário de inserção de veículos

Com relação ao botão "Alterar dados" do menu principal, que também só aparece se o usuário for um administrador do banco de dados, pode-se dizer que tem quase o mesmo formato que o de inserção de veículos, com as seguintes diferenças: o administrador do banco deve selecionar um veículo já existente no banco de dados por meio de um formulário semelhante ao de consulta de veículos, mas com opção de selecionar apenas um veículo, podendo este inclusive acessar a página de visualização de configurações para facilitar no preenchimento das especificações; e que o administrador do banco deve selecionar os campos que deseja mudar no banco. Um exemplo de uma parte do formulário de alteração é mostrado na figura 6.10.

Figura 6.10 Formulário de alteração de dados

Se o procedimento for seguido corretamente, a saída do formulário deve ser a seguinte:

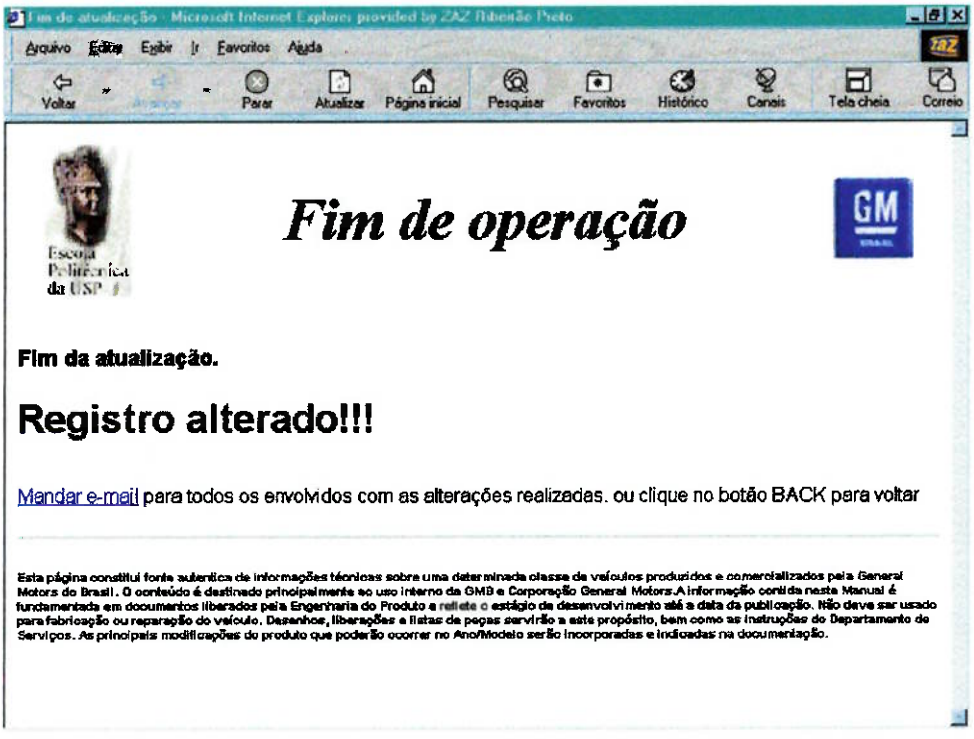


Figura 6.11 Saída da operação de alteração de dados

Visualizando com um pouco mais de cuidado a figura 6.11, nota-se que existe um link na tela que "aponta" para a tela "compose" do software de e-mail padrão selecionado no browser com o campo "to" preenchido com os endereços de todos os usuários que acessam aqueles dados do banco que foram alterados menos o usuário atual, como mostra a figura 6.12.

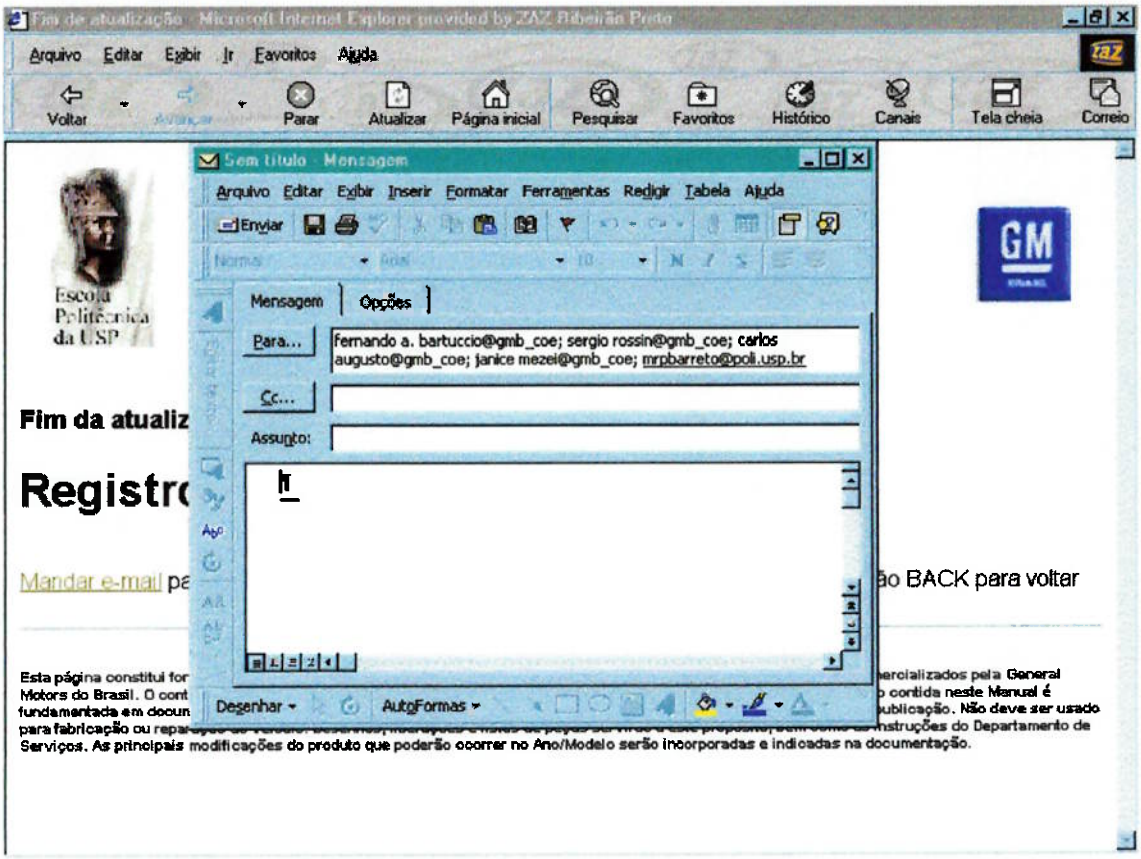


Figura 6.12 Mandando e-mail para os usuários

Este procedimento garante que todos os usuários fiquem sempre atualizados sobre alterações do banco de dados, evitando conflitos de projeto.

O último botão de manutenção do banco de dados é o "Deletar veículo". Ele acessa o formulário de deletar, que é mostrado na figura 6.13.

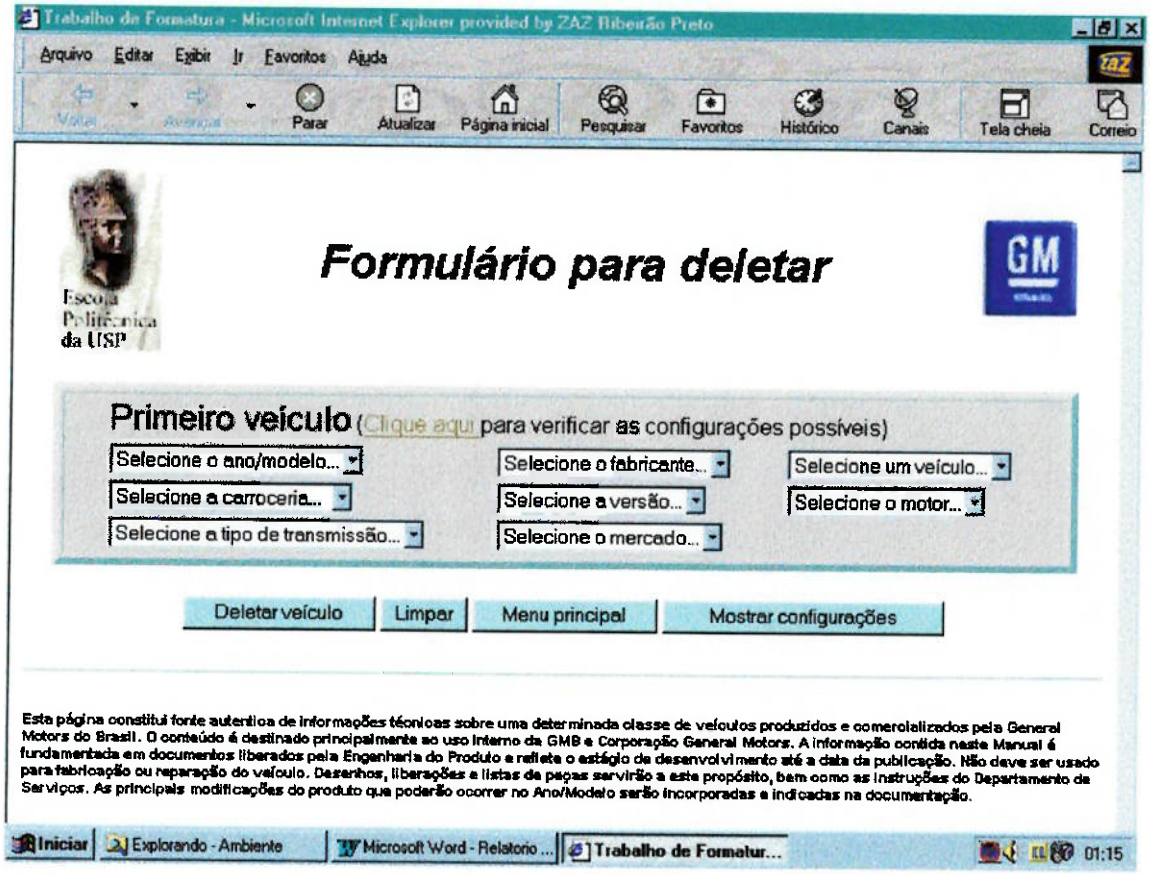


Figura 6.13 Formulário para deletar um veículo

O método de seleção e a resposta deste formulário é muito parecido com o de alteração de dados, apenas com diferença na operação final e no fato de não precisar selecionar e colocar o valor de um dado específico, pois serão todos deletados.

Com relação ao botão de mudança de password, que aparece para todos os usuários, pode-se dizer que é um botão que serve apenas como complemento, pois ele acessa um formulário que tem a função apenas de mudar a senha do próprio usuário. Este último tem apenas que digitar de novo a sua senha e em seguida digitar duas vezes a senha nova (apenas para confirmação). Este formulário se encontra na figura 6.14.

Figura 6.14 Formulário de mudança de password

E por fim, o botão "Logout", que serve para sair do ambiente do usuário atual (voltar para a página inicial).

7. O PROGRAMA USERAPP

7.1. Descrição geral

Pode-se dizer que o programa UserApp é o coração do ambiente. Ele recebe as entradas feitas pelo usuário no formulário através da leitura do arquivo randômico, faz as consultas ou alterações necessárias no banco de dados e então apresenta os resultados em formato HTML no browser para o usuário na forma de tabela de dados, no caso de uma consulta, ou na forma de mensagens, no caso de uma alteração no banco de dados ou algum erro de processamento.

Este programa está desenvolvido em Java, que é uma das linguagens mais utilizadas em aplicações para a internet atualmente, utilizando comandos em linguagem SQL via JDBC, que é a conexão entre o programa e o banco de dados, para recuperar informações do banco por meio de uma *query*.

Vale salientar que este programa não está o tempo todo em execução enquanto o usuário está acessando o site. Ele só entra em execução enquanto uma ação está sendo feita, como acessar outra página, realizar uma consulta ou alterar informações do banco de dados, ou seja, no intervalo de tempo entre um clique de botão ou link e a impressão da saída na tela.

Como todo programa Java, o programa UserApp utiliza o conceito de programação orientada a objeto.

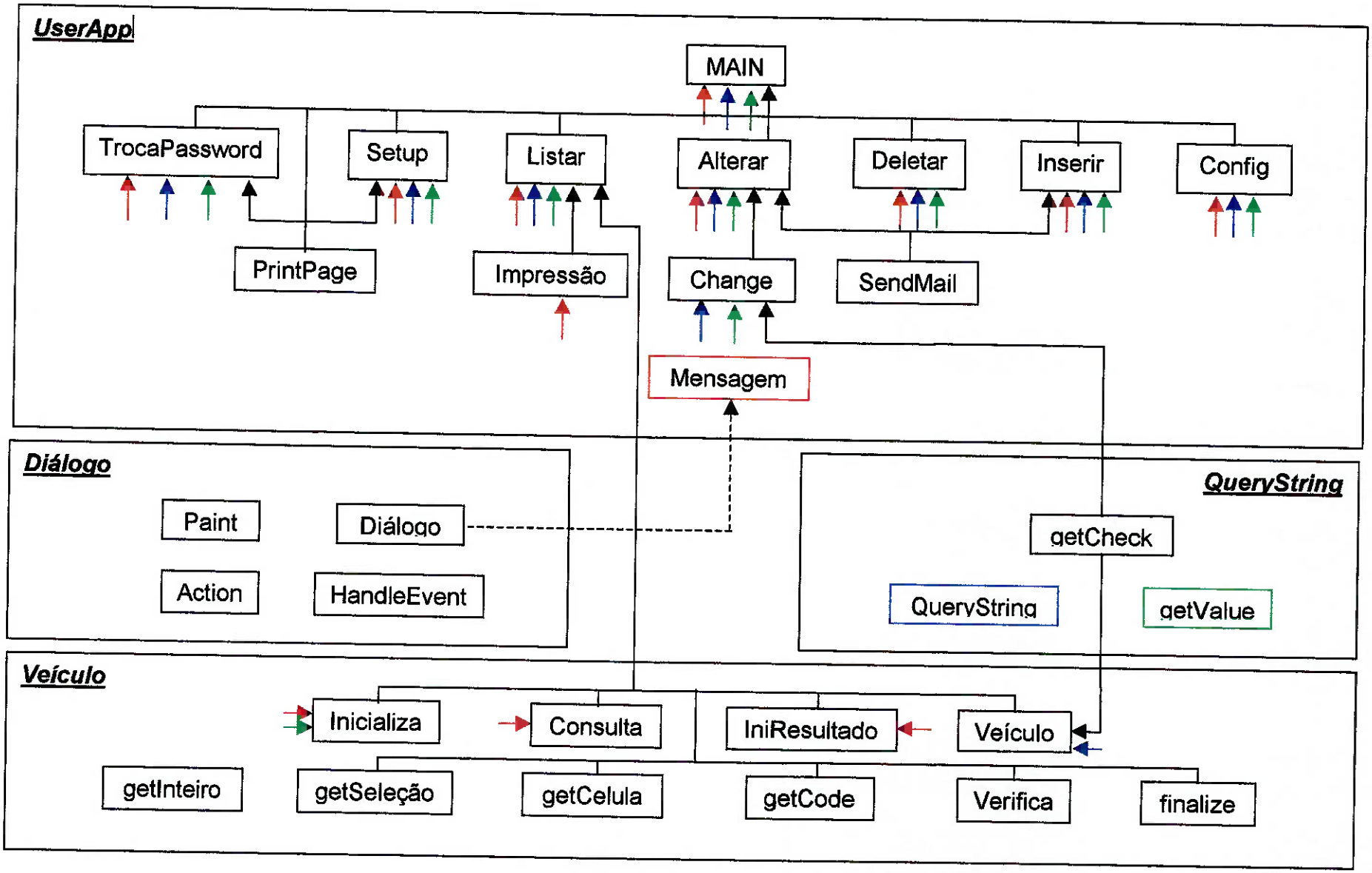
Existem quatro objetos neste programa, que seriam:

- ***UserApp:*** é o objeto principal e que é responsável pelas funções básicas do programa e que utiliza os outros objetos;
- ***QueryString:*** consiste em uma estrutura que utiliza uma *Hashtable* para armazenar os parâmetros de seleção definidos pelo usuário e que foram lidos no arquivo randômico;
- ***Veículo:*** representa o segundo, terceiro ou quarto veículo definido pelo usuário quando a ação que está sendo executada é a de visualizar os dados de dois ou mais veículos na tabela de dados;
- ***Dialogo:*** é uma estrutura de caixa de diálogo *(Frame)* que seria utilizada para exibir uma mensagem de erro utilizando o pacote AWT, mas como notou-se que esta estrutura não funciona em ambiente de rede, optou-se por imprimir todas as mensagens em formato HTML e não utilizar este objeto no programa. Porém, como o objeto já estava pronto, preferiu-se deixá-lo disponível para uma eventual utilização em algum caso particular.

A figura 7.1 mostra as relações entre todas as funções dos objetos do programa UserApp.

Note que o bloco no início da flecha é a função chamada e no fim, a função que chama.

A flecha tracejada na chamada da função diálogo se justifica devido ao fato da relação não existir na configuração do programa atual, mas se um programador quiser utilizar a caixa de diálogo do pacote AWT, é fácil a mudança, pois o objeto já existe.


UserApp
MAIN
TrocaPassword
Setup
Listar
Alterar
Deletar
Inserir
Config
PrintPage
Impressão
Change
SendMail
Mensagem
Diálogo
Paint
Diálogo
Action
HandleEvent
QueryString
getCheck
QueryString
getValue
Veículo
Inicializa
Consulta
IniResultado
Veículo
getInteiro
getSeleção
getCelula
getCode
Verifica
finalize

Quanto às flechas coloridas que só apresentam blocos de chegada (que chama), a função de saída (chamada) é a que tiver a borda da mesma cor. Esta referência foi utilizada para deixar a figura mais organizada.

7.2. Descrição dos objetos e suas funções

7.2.1. Objeto Diálogo

Tipo: -

Objetos a que estende: Frame;

Parâmetros: t1: texto de tipo de erro;

t2: texto de descrição do erro;

Funções: Diálogo, Paint, Action, handleEvent.

7.2.1.1. Função Diálogo

Tipo: Função pública;

Exceções: -

Funções que chama: -

Funções que a chamam: Se utilizada, mensagem;

Parâmetros de entrada: texto1: texto de tipo de erro;

texto2: texto de descrição do erro;

Parâmetro de saída: -

Descrição: Inicializa o objeto com os textos a serem exibidos, define o botão "OK" e o título da caixa de diálogo e organiza o seu lay-out.

7.2.1.2. Função Paint

Tipo: Função pública;

Exceções: -

Funções que chama: -

Funções que a chamam: Funções internas do objeto Frame;

Parâmetros de entrada: g: objeto gráfico;

Parâmetro de saída: -

Descrição: Define a fonte dos textos e imprime os textos na caixa de diálogo de uma forma que, mesmo que o usuário redimensione a caixa de diálogo, os textos ficam sempre centralizados.

7.2.1.3. Função Action

Tipo: Função pública;

Exceções: -

Funções que chama: -

Funções que a chamam: Funções internas do objeto Frame;

Parâmetros de entrada: evt: evento a ser tratado;

arg: argumento de botão;

Parâmetro de saída: flag de funções do objeto Frame;

Descrição: Fecha a caixa de diálogo se o usuário clicar no botão "OK".

7.2.1.4. Função handleEvent

Tipo: Função pública;

Exceções: -

Funções que chama: -

Funções que a chamam: Funções internas do objeto Frame;

Parâmetros de entrada: evt: evento a ser tratado;

Parâmetro de saída: flag de funções do objeto Frame;

Descrição: Além dos comandos da função de mesmo nome do objeto Frame, fecha a caixa de diálogo quando o usuário clica no botão de "fechar" (no canto superior direito da caixa de diálogo).

7.2.2. Objeto QueryString

Tipo: -

Objetos a que estende: -

Parâmetros: qvars: *Hashtable* que armazena os parâmetros de seleção definidos pelo usuário e que foram lidos no arquivo randômico;

Funções: QueryString, .getCheck, getValue.

7.2.2.1. Função QueryString

Tipo: Função pública;

Exceções: -

Funções que chama: -

Funções que a chamam: UserApp: Main, TrocaPassword, Setup, Listar, Alterar, Deletar, Inserir Config, Change; *Veículo:* Veículo;

Parâmetros de entrada: query: conteúdo do arquivo randômico;

Parâmetro de saída: -

Descrição: armazena em uma *Hashtable* os parâmetros de seleção definidos pelo usuário e que foram lidos no arquivo randômico.

7.2.2.2. Função getCheck

Tipo: -

Exceções: -

Funções que chama: -

Funções que a chamam: Veículo: Veículo; *UserApp:* Change;

Parâmetros de entrada: name: variável da *Hashtable*.

Parâmetro de saída: flag que indica se a variável *name* está definida;

Descrição: Informa se uma variável da *Hashtable* está definida.

7.2.2.3. Função getValue

Tipo: -

Exceções: -

Funções que chama: -

Funções que a chamam: UserApp: Main, TrocaPassword, Setup, Listar, Alterar, Deletar, Inserir Config, Change; *Veículo:* Inicializa;

Parâmetros de entrada: key: variável da *Hashtable;*

Parâmetro de saída: Valor da variável key;

Descrição: Retorna o valor da variável pedida ou "Key not found" se a variável não existir.

7.2.3. Objeto Veículo

Tipo: -

Objetos a que estende: -

Parâmetros: result: resultado de consulta;

u: objeto UserApp para utilizar a função *Mensagem*;

d: objeto *QueryString* para acessar os dados de seleção;

b: flag que controla a efetuação de tarefas;

v: define se é o segundo, terceiro ou quarto veículo;

numvin: código de controle de definição de um veículo;

a: parte do código de controle para ano/modelo;

fab: parte do código de controle para fabricante;

vei: parte do código de controle para veículo;

c: parte do código de controle para carroceria;

ver: parte do código de controle para versão;

mot: parte do código de controle para motor;

t: parte do código de controle para transmissão;

mer: parte do código de controle para mercado;

stat: statement da query;

con: conexão JDBC ao banco de dados;

Funções: Veículo, Inicializa, IniResultado, Consulta, getInteiro, Verifica, finalize, getSeleção, getCelula, getCode.

7.2.3.1. Função Veículo

Tipo: Pública;

Exceções: -

Funções que chama: QueryString: QueryString; UserApp: getCheck

Funções que a chamam: UserApp: Listar;

Parâmetros de entrada: veic: informa se é o segundo, terceiro ou quarto veículo;

data: objeto *QueryString* para acessar os dados de seleção

Parâmetro de saída: -

Descrição: Inicializa as variáveis v, d, b.

7.2.3.2. Função Inicializa

Tipo: Pública;

Exceções: -

Funções que chama: QueryString: getValue; *UserApp:* Mensagem;

Funções que a chamam: UserApp: Listar;

Parâmetros de entrada: -

Parâmetro de saída: -

Descrição: Inicializa as partes do código de controle.

7.2.3.3. Função Consulta

Tipo: Pública;

Exceções: -

Funções que chama: UserApp: Mensagem;

Funções que a chamam: UserApp: Listar;

Parâmetros de entrada: -

Parâmetro de saída: -

Descrição: Abre conexão com o banco de dados, define o código de controle e realiza uma *query* (consulta) ao banco de dados, obtendo o veículo especificado.

7.2.3.4. Função IniResultado

Tipo: Pública;

Exceções: SQLException;

Funções que chama: UserApp: Mensagem;

Funções que a chamam: UserApp: Listar;

Parâmetros de entrada: -

Parâmetro de saída: -

Descrição: Inicializa o resultado da *query.*

7.2.3.5. Função finalize

Tipo: Pública;

Exceções: SQLException

Funções que chama: -

Funções que a chamam: UserApp: Listar;

Parâmetros de entrada: -

Parâmetro de saída: -

Descrição: Fecha o *statement* e a conexão com o banco de dados.

7.2.3.6. Função getSeleção

Tipo: Pública;

Exceções: -

Funções que chama: -

Funções que a chamam: UserApp: Listar;

Parâmetros de entrada: -

Parâmetro de saída: b: flag que controla a efetuação de tarefas;

Descrição: Retorna o flag *b.*

7.2.3.7. Função getCelula

Tipo: Pública;

Exceções: SQLException;

Funções que chama: -

Funções que a chamam: UserApp: Listar;

Parâmetros de entrada: nome: dado o qual se deseja obter;

Parâmetro de saída: Valor do dado desejado;

Descrição: Retorna o valor do dado desejado.

7.2.3.8. Função getInteiro

Tipo: Pública;

Exceções: SQLException

Funções que chama: -

Funções que a chamam: Se fosse utilizada, *UserApp:* Listar

Parâmetros de entrada: name: dado o qual se deseja obter;

Parâmetro de saída: Valor do dado desejado;

Descrição: Retorna o valor do dado desejado.

7.2.3.9. *Função Verifica*

Tipo: Pública;

Exceções: SQLException;

Funções que chama: -

Funções que a chamam: UserApp: Listar.

Parâmetros de entrada: -

Parâmetro de saída: Próximo resultado da *query*;

Descrição: Retorna o próximo resultado da *query* .

7.2.3.10. *Função getCode*

Tipo: Pública;

Exceções: -

Funções que chama: -

Funções que a chamam: UserApp: Listar;

Parâmetros de entrada: -

Parâmetro de saída: numvin: Código de controle de definição de um veículo;

Descrição: Retorna o código de controle de definição do veículo.

7.2.4. Objeto UserApp

Tipo: Público;

Objetos a que estende: -

Parâmetros: primeiro: flag que indica se é o primeiro dado a ser alterado ou não;

command: comando de consulta (*query*) à tabela User para obter o e-mail dos usuários que acessam um dado alterado.

Funções: Mensagem, SendMail, Impressão, Listar, Inserir, Change, Alterar, Config, Setup, PrintPage, TrocaPassword, Main.

7.2.4.1. Função Mensagem

Tipo: Pública e estática.

Exceções: -

Funções que chama: Se utilizasse o pacote AWT, *Diálogo:* Diálogo;

Funções que a chamam: UserApp: Main, TrocaPassword, Setup, Listar, Alterar, Deletar, Inserir, Config, Impressão; *Veículo:* Inicializa, Consulta e IniResultado;

Parâmetros de entrada: s1: texto de tipo de erro;

s2: texto de descrição do erro;

x: se utilizasse AWT, a largura da caixa de diálogo;

y: se utilizasse AWT, a altura da caixa de diálogo;

Parâmetro de saída: -

Descrição: Imprime na tela uma mensagem de erro contendo os textos s1 e s2.

7.2.4.2. Função SendMail

Tipo: Pública e estática;

Exceções: SQLException;

Funções que chama: -

Funções que a chamam: Inserir e Alterar.

Parâmetros de entrada: fun: flag que identifica a função que a chamou;

comando: comando de consulta (*query*) à tabela User para obter o e-mail dos usuários que acessam um dado alterado.

con: conexão JDBC ao banco de dados;

Parâmetro de saída: -

Descrição: Imprime na tela uma mensagem de sucesso de operação, inclusive com um link para mandar um e-mail para os usuários que têm acesso àquele dado.

7.2.4.3. Função Impressão

Tipo: Pública e estática

Exceções: SQLException;

Funções que chama: Mensagem;

Funções que a chamam: Listar;

Parâmetros de entrada: coluna: nome do dado na tabela GMB;

code: código de controle para definir um veículo;

dado: valor do dado;

con: conexão JDBC ao banco de dados;

Parâmetro de saída: item da tabela de dados;

Descrição: Faz uma consulta à tabela Referencia e, se existir um documento auxiliar para aquele dado daquele veículo específico, retorna o valor do dado com um link para o documento auxiliar. Senão, retorna o próprio dado.

7.2.4.4. Função Listar

Tipo: Pública e estática;

Exceções: SQLException;

Funções que chama: Impressão, Mensagem; *QueryString*: QueryString, getValue; *Veículo:* Veículo, Inicializa, IniResultado, Consulta, Verifica, finalize, getSeleção, getCelula e getCode;

Funções que a chamam: Main;

Parâmetros de entrada: dados: objeto *QueryString* para acessar os dados de seleção;

con: conexão JDBC ao banco de dados

Parâmetro de saída: -

Descrição: Define os nomes dos dados da tabela GMB e dos títulos da tabela de dados em dois vetores. Inicializa o código de controle de definição de um veículo e faz uma consulta à tabela GMB para obter os dados do veículo selecionado pelo usuário no primeiro campo de seleção do formulário de visualização de dados. Cria três objetos "Veículo" e, se for necessário, faz o mesmo para as seleções dos outros campos. Dependendo das seleções feitas em relação a exibir um veiculo ou não, imprime na tela do browser a tabela de dados correspondente às seleções definidas no formulário de visualização de dados.

7.2.4.5. Função Inserir

Tipo: Pública e estática;

Exceções: SQLException;

Funções que chama: Mensagem, SendMail; *QueryString:* QueryString, getValue.

Funções que a chamam: Main;

Parâmetros de entrada: dados: objeto *QueryString* para acessar os dados de seleção;

con: conexão JDBC ao banco de dados;

Parâmetro de saída: -

Descrição: Insere um veículo na tabela GMB de acordo com as seleções definidas no formulário de inserção e chama *SendMail* para imprimir na tela do browser a mensagem do resultado da operação.

7.2.4.6. Função Deletar

Tipo: Pública e estática.

Exceções: SQLException;

Funções que chama: Mensagem, *QueryString:* QueryString, getValue.

Funções que a chamam: Main;

Parâmetros de entrada: dados: objeto *QueryString* para acessar os dados de seleção;

con: conexão JDBC ao banco de dados;

Parâmetro de saída: -

Descrição: Deleta um veículo da tabela GMB de acordo com as seleções definidas no formulário de deletar.

7.2.4.7. Função Change

Tipo: Pública e estática.

Exceções: SQLException;

Funções que chama: QueryString: QueryString, getValue e getCheck;

Funções que a chamam: Alterar;

Parâmetros de entrada: par1: indica se o usuário selecionou aquele dado para ser alterado ou não;

par2: é nome do dado na tabela GMB;

par3: é o nome da variável que armazena o dado na *Hashtable;*

code: código de controle para definir um veículo;

dados: objeto *QueryString* para acessar os dados de seleção;

con: conexão JDBC ao banco de dados;

Parâmetro de saída: número de registros alterados ou "1" para dados que não foram selecionados para serem alterados;

Descrição: Altera um dado de um veículo da tabela GMB de acordo com as seleções definidas no formulário de alteração.

7.2.4.8. Função Alterar

Tipo: Pública e estática;

Exceções: SQLException;

Funções que chama: Mensagem, SendMail, Change; *QueryString:* QueryString, getValue

Funções que a chamam: Main;

Parâmetros de entrada: dados: objeto *QueryString* para acessar os dados de seleção;

con: conexão JDBC ao banco de dados;

Parâmetro de saída: -

Descrição: Define o código de controle de definição de um veículo e chama a função *Change para* alterar cada dado do veículo da tabela GMB de acordo com as seleções definidas no formulário de alteração e chama *SendMail* para imprimir na tela do browser a mensagem do resultado da operação.

7.2.4.9. Função Config

Tipo: Pública e estática;

Exceções: SQLException;

Funções que chama: Mensagem, *QueryString:* QueryString, getValue;

Funções que a chamam: Main;

Parâmetros de entrada: dados: objeto *QueryString* para acessar os dados de seleção;

con: conexão JDBC ao banco de dados;

Parâmetro de saída: -

Descrição: Retorna a tabela de configurações a partir das seleções feitas no formulário de configurações.

7.2.4.10. Função Setup

Tipo: Pública e estática;

Exceções: SQLException, IOException;

Funções que chama: Mensagem, PrintPage; *QueryString:* QueryString, getValue;

Funções que a chamam: Main;

Parâmetros de entrada: dados: objeto *QueryString* para acessar os dados de seleção;

con: conexão JDBC ao banco de dados;

Parâmetro de saída: -

Descrição: Confere se existe o UserID digitado na página inicial e se a senha está correta e, se estiverem, coloca o IP da máquina no seu respectivo campo na tabela "User". Chama a função PrintPage para imprimir na tela do browser o menu principal.

7.2.4.11. Função PrintPage

Tipo: Pública e estática;

Exceções: SQLException, IOException;

Funções que chama: -

Funções que a chamam: Setup, TrocaPassword e Main;

Parâmetros de entrada: arq: nome do arquivo que contém o código HTML a ser impresso no browser;

con: conexão JDBC ao banco de dados;

Parâmetro de saída: -

Descrição: Consulta a tabela User para saber se o usuário tem *status* de administrador do banco de dados. Lê o conteúdo do arquivo *arq* e o imprime na tela do browser da seguinte maneira:

Enquanto não encontrar um asterisco

Lê e imprime os comando HTML

Enquanto não encontrar outro asterisco

Se o usuário puder alterar o banco de dados

Lê e imprime os comando HTML

Senão

Apenas lê os comandos HTML

Lê e imprime os comando HTML até o final do arquivo

7.2.4.12. Função TrocaPassword

Tipo: Pública e estática;

Exceções: SQLException, IOException;

Funções que chama: Mensagem, PrintPage; *QueryString:* QueryString, getValue;

Funções que a chamam: Main;

Parâmetros de entrada: dados: objeto *QueryString* para acessar os dados de seleção;

con: conexão JDBC ao banco de dados;

Parâmetro de saída: -

Descrição: Se a senha que o usuário digitou no formulário de mudança de password estiver correta e se as duas senhas novas forem iguais, então muda o password da tabela User do usuário para a nova senha e imprime na tela o menu principal.

7.2.4.13. Função Main

Tipo: Pública e estática;

Exceções: -

Funções que chama: Mensagem, TrocaPassword, PrintPage, Setup, Listar, Alterar, Deletar, Inserir, Config; *QueryString:* QueryString, getValue;

Funções que a chamam: -

Parâmetros de entrada: args[0]: nome do arquivo randômico;

Parâmetro de saída: -

Descrição: Função principal do objeto e do programa. Estabelece uma conexão JDBC com o banco de dados. Lê o arquivo randômico e armazena o seu conteúdo em um objeto QueryString. De acordo com o parâmetro *function* escondido em cada formulário e passado para o programa via o arquivo randômico, chama uma função específica para cumprir a tarefa definida. Deleta o arquivo randômico.

8. COMENTÁRIOS FINAIS

8.1. Cronograma

Tomando como base o cronograma inicialmente proposto no início do semestre, foram cumpridos todas as previsões estabelecidas naquele documento, com exceção dos itens da Iteração I: Protótipo, que foram realizados com um mês de atraso em relação ao cronograma inicial. Visto que este cronograma é apenas uma estimativa, considerou-se o resultado satisfatório para este projeto.

Além disso, o item de complementação técnica – Visual Basic ou Delphi – foi trocado por Java devido a maior integração com os recursos de rede e a natureza do problema deste projeto.

8.2. Conclusões

O site na intranet proposto aqui trás inúmeros benefícios para seus usuários e para a empresa, tais como: tempo de resposta menor, devido a retirada de fatores como burocracia desnecessária, consultas a desenhos e arquivos em papel e dificuldade de acesso aos respectivos responsáveis pela informação; alta confiabilidade; adição de recursos inexistentes, como a comparação mais eficiente entre veículos e notificação automática de alterações de dados; e redução de custos, devido ao excessivo gasto com papel e carga de trabalho, o que contribui para uma empresa mais competitiva.

Com a finalidade de maximizar os recursos disponíveis aos usuários, pode-se disponibilizar os dados para os ano modelos futuros e antigos da mesma forma que já está implementada para os ano modelos atuais, facilitando o fluxo de informações para um projeto mais seguro, barato, eficiente e com um *time to market* reduzido.

Analisando as tecnologias estudadas e a especificação de requisitos, nota-se que o projeto é viável e está bem encaminhado.

Alguns módulos e funções podem ser adicionados a este projeto para melhorar sua manutenção e performance, como os já sugeridos na especificação de requisitos e outros como consultas personalizadas para cada usuário e funções que realizem a manutenção automática dos formulários quando novas opções forem inseridas no banco de dados.

Outra melhoria seria também alterar a estrutura do banco de dados para poder pintar de vermelho o melhor dado de cada linha da tabela de dados, facilitando a comparação dos dados dos veículos.

REFERÊNCIAS BIBLIOGRÁFICAS

1. Rational Software Corporation, ***Rational Unified Process: Best Practices for Software Development Teams,*** 1995 – 1999.
2. Ferreira, Flávio Pontes, ***Como usar a SQL (structured query language) do dBase IV,*** **McGraw-Hill**, 1989.
3. Pratt, Philip J., ***A guide to SQL,*** **boyd & fraser publishing company**, 1995.
4. McFedries, Paul, ***Guia Incrível para Criação de Páginas Web com HTML,*** *1997.*
5. Cornell, Gary; Horstmann, Cay S.; ***Core Java.***
6. Ritchey, Tim, ***Programando Java & JavaScript para Netscape 2.0,*** Quark, 1996.
7. PDM Information Company, ***Understanding Product Data Management.***
8. Kobielus, James G., ***Workflow Strategies,*** IDG Books Wordwide, 1997.

APÊNDICE - COMO INSTALAR O AMBIENTE CONTIDO NO DISQUETE

Neste apêndice será mostrado como instalar o ambiente contido no disquete anexo a este trabalho. As instruções serão direcionadas para que o ambiente seja rodado de seu computador. Ele poderia ser rodado do disquete fazendo-se algumas alterações, mas é recomendado que seja instalado manualmente no computador por questões de performance.

Inicialmente, copie o arquivo *hosts* para o diretório windows de sua máquina. Uma explicação do conteúdo deste arquivo se encontra na página 69 deste trabalho.

Feito isso, crie um diretório novo na raiz do seu winchester chamado "ambiente" e copie todo o conteúdo do disquete para este diretório. Se quiser utilizar um outro caminho de diretório que não este, consulte as páginas 69 e 70 deste trabalho para uma explicação mais detalhada da parte do ambiente que deve ser alterada.

A próxima tarefa a ser executada é criar uma ponte ODBC de 32 bits, que pode ser configurada no painel de controle do computador realizando-se os seguintes procedimentos, que já foram mostrados mas serão repetidos logo abaixo:

- Clique no botão "Iniciar" da barra de tarefas de seu computador;
- Selecione "configurações" e em seguida clique em "painel de controle";
- Dê um duplo clique no ícone "ODBC de 32 bits";

- Em "NFD do Usuário", apenas selecione "Banco de dados MS Access 97 " e clique no botão "adicionar";
- Dê um double click em "Driver para o Microsoft Access (*.mdb)";
- Clique no botão "selecionar" e selecione o banco de dados em questão, no caso o arquivo DATATF.mdb;
- Digite "UserApp" no campo "Nome da fonte de dados;
- Clique no botão "OK";
- Se apareceu um item com nome UserApp em "NFD do Usuário", o procedimento está completo;
- Feche o painel de controle.

A figura 5.2 mostra como deve estar preenchida a sua caixa de diálogo.

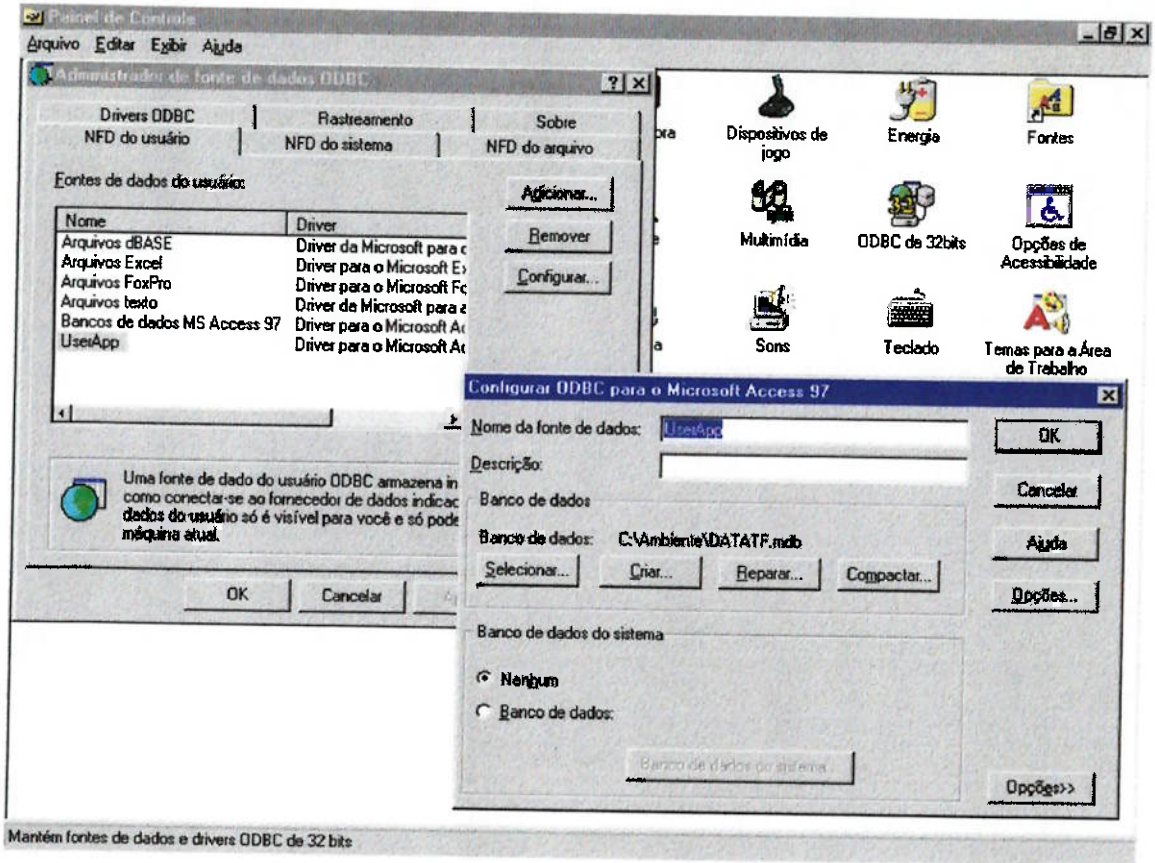


Figura 1 Caixa de diálogo para configurar a ODBC de 32 bits

Agora, com tudo praticamente pronto para que o ambiente funcione, é só rodar o servidor. Isso é feito digitando o comando abaixo no diretório que contém o ambiente a partir do prompt do DOS.

java LocalServer

A resposta do servidor será:

HttpServer listening on port 80

e indica que o servidor está no ar. É importante lembrar que para os programas em Java serem executados é necessária uma JVM *(Java Virtual Machine),* como qualquer outro programa Java.

Agora, clicando no botão que abre outra janela do windows ou ao mesmo tempo nas teclas "Alt" e "Tab", inicialize o seu browser. Tente digitar localhost na barra de endereços do browser e, se for aberta a página inicial do site, então a instalação está completa. Consulte o capítulo 6 deste trabalho para aprender como navegar no site e os capítulos 5 e 7 se desejar alguma explicação sobre o ambiente e seus componentes.

Mas, se o browser não conseguir achar a página e retornar uma mensagem pedindo para se conectar à Internet, então é necessário fazer uma alteração na configuração de seu browser. Se você estiver usando o Internet Explorer, uma possível saída para este problema é:

- Clique na barra de ferramentas em "Exibir";

- Selecione "Opções da Internet";
- Na caixa de diálogo de Opções da Internet clique em "Conexão";
- No primeiro campo, selecione "Conectar-se a Internet usando a rede local".

Pronto, agora deve dar certo. No Netscape Navigator existe uma outra opção, que é configurar "proxies" manualmente, como segue abaixo:

- Clique na barra de ferramentas em "Edit";
- Selecione o campo "Preferences...";
- Clique em "advanced / Proxies";
- Selecione "Manual proxy configuration" e clique no botão "View";
- Digite *127.0.0.1* nos campos "Address of proxy server to use" e 80 nos campos "Port".