Desenvolvimento de sistemas em Linux

Anderson Pereira Ataides
Prof
Heitor
Augustus Xavier
Costa

Desenvolvimento de sistemas em Linux

por e Anderson Pereira Ataides
Prof
Heitor
Augustus Xavier
Costa

Índice

Lista de Tabelas

Lista de Figuras

Capítulo 1. Introdução

Nos dias de hoje, onde as empresas estão sob pressão constante, seja dos fornecedores que querem vender mais caro, seja dos clientes que querem comprar mais barato, qualquer economia que ela buscar pode ser fundamental para que ela se mantenha funcionando e prncipalmente, dando lucros. Atentos a estas exigências em busca de produtividade a baixo custo, muitos diretores buscaram na informática um ponto de apoio para auxiliá-los nesta empreitada.

Mesmo com esta pressão e mesmo com a necessidade de aumento de produtividade, muitas empresas ainda não conseguiram fazer este investimento. E os motivos para isto são vários:

Preço do equipamento

Com a rápida e constante evolução das máquinas, e conseqüente pressão para vender as obsoletas, vemos os equipamentos de informática cada vez mais baratos e com muitas facilidades para pagamento. Portanto esta justificativa já não tem muito fundamento, apesar de ainda ser aceitável.

Falta de mão de obra

Com os inúmeros cursos técnicos, bem como universidades que estão "despejando" profissionais no mercado, vemos uma situação diferente: o *excesso* de profissionais. É claro que existem os maus profissionais, mas sabendo escolher, podemos conseguir um bom profissional para auxiliar no processo de informatização.

Falta de tempo para treinamentos

Com tantas escolas de informática oferecendo cursos em vários horários diferenciados, não há como usar esta falta de tempo para não informatizar. E algumas destas escolas vão mais além: fecham turmas dentro da empresa, cobrando mais barado pelo pacote e dão o treinamento *in loco*.

Mas a maior dificuldade é realmente o investimento inicial para a informatização. A grande maioria das empresas brasileiras são pequenas e não têm caixa suficiente para bancar esta informatização. Mesmo vencida a barreira do custo dos equipamentos, exitem os custos da mão de obra, dos próprios treinamentos e um outro custo que quase sempre é deixado de lado, mas mas que representa a maior fatia dos investimentos: o do *software*. Na maioria dos casos apenas o de gerenciamento é considerado no levantamento de custos.

Os custos com os *softwares* restantes, como sistema operacional, suítes *office*, geralmente são considerados básicos e que é uma obrigação do fabricante do equipamento fornecê-los já instalados. E realmente os fabricantes já nos fornecem estes sistemas, mas com um pequeno detalhe: *piratas*. Se a empresa tentar fugir desta pirataria, terá que desembolsar alguns milhares de reais para ter sua máquina legalizada.

Felizmente, com o surgimento do Linux (http://www.linux.org), que popularizou a filosofia do software livre (http://www.fsf.org) ou ainda do software *open source* (http://www.opensource.org) ficou bem mais fácil legalizar estes softwares "básicos". A empresa náo precisa mais se preocupar com as "licenças de uso", ou com o medo de instalar um programa em várias máquinas sob o risco de violar os contratos de tais licenças.

Na filosofia do software livre, ao obtê-lo, seja gratuitamente através de um *download*[1], ou pela bondade de um amigo, ou mesmo pela compra, o software é seu e com ele você faz o que desejar, desde que as alteraçãoes mantenham o software livre. Assim diferentemente do software proprietário, onde é necessário

uma licença de uso para cada instalação, no software livre, você o adquire e tem todo o direito de instalá-lo onde desejar.

Quando se fala em software livre, não há como deixar de falar no Linux, pois é comum as pessoas ligarem o termo software livre a ele. Existem bons argumentos hoje para que o Linux seja adotado em máqinas *desktop*, ou estação de trabalho para o usuário final. Nas empresas porém, o Linux tem sido utilizado apenas nos servidores, enquanto as estações continuam com o sistema proprietário anterior, na maioria dos casos o Windows. Isto porque apesar de ter inúmeros aplicativos disponíveis, não tem o principal para a empresa: um software de gerenciamento.

Este projeto surgiu para incentivar os programadores a completar a gama de softwares livres para empresas comerciais. Já temos o sistema operacional, o banco de dados, as linguagens de programação as suítes *office*, todos livres, mas ainda ão temos ofertas de softwares de gerenciamento livres. Temos sim muitos softwares eficientes, mas nenhum distribuído sob a filosofia do software livre. A partir da descrição de um sistema de vendas simples, são apresentadas ferramentas para auxiliar no desenvolvimento, desde a modelagem até a implementação propriamente.

1.1. Por que um software de vendas?

Apenas um argumento justifica a escolha de um software deste tipo: falta de aplicativo comercial para Linux. Em se tratando de software livre o problema ainda é mais grave, pois não existe, pelo menos até agora, um software de dessa natureza que seja livre. Existem pelo menos dois modelos de negócio seguidos pelos fornecedores deste tipo de software: venda e aluguel. Na verdade, o que quase sempre acontece, é que adquirimos o software e depois o fornecedor nos vem oferecendo uma "manutenção" em troca de uma taxa mensal para atualizações do produto. Concluindo, acabamos por comprar e alugar o software ao mesmo tempo.

Outro grande problema no modelo proprietário, é que na maioria dos casos, os fornecedores cobram por licença de uso, ou seja, ele cobra a utilização em cada máquina da rede, o que onera mais ainda o custo de informatização. No caso de softwares alugados, a história não é muito diferente, pois geralmente a "taxa de manutenção" é cobrada de acordo com o número de máquinas onde o software vai funcionar.

Mas ainda tem mais. O modelo proprietário nos prende a um único fornecedor, ele tem o poder sobre o software. Nós temos apenas o direito de utilizar o sistema. Dependemos do fornecedor para instalar, dar treinamento e manutenção, e ainda efetuar eventuais alterações que forem necessárias para que o sistema atenda melhor nossas necessidades. Se a alteração não for interessante para outros clientes (do fornecedor do software), pior ainda, pois eles vão deixar nossa alteração em segundo plano sob a justificativa que eles têm outras mais prioritárias para desenvolver.

A filosofia do software livre muda o poder das mãos do fornecedor para as mãos do cliente principalmente pelos seguintes motivos:

O software livre tem código aberto

> Teoricamente, qualquer programador com conhecimento na linguagem na qual o sistema foi desenvolvido pode efetuar alterações no sistema, deixando-o da forma que desejamos.

O software livre pode ser instalado em outras máquinas

> A partir do momento que eu obtive o software, eu posso instalá-lo onde eu desejar, ou seja, posso instalar em uma ou em todas as máquinas de minha empresa sem ter que pagar mais por isto.

Mas a maior motivação para se elaborar este trabalho baseando em um sistema de vendas foi pela necessidade própria. Trabalhando em uma loja de auto peças (Casa das Carretas), um dia surgiu a necessidade de se fazer um upgrade em uma máquina. Na época, esta empresa tinha tudo legalizado, desde o SO, passando pelo software de rede até o software gerencial. Como a máquina veio com um sistema operacional mais evoluído, decidiu-se evoluir toda a infra-estrutura da empresa. Em pouco tempo, houve a necessidade de um novo upgrade e após um minucioso levantamento, decidiu-se pela utilização do software livre.

Ao partir para uma plataforma livre, chegamos a outro dilema: o software gerencial era escrito para o outro SO. Tentou-se fazê-lo funcionar sob um emulador, mas diferenças no tratamento do sistema de arquivos levaram a corrupção dos arquivos e conseqüente perda de dados. Como o fornecedor do software não mostrou muita disposição em elaborar um software que funcionasse na plataforma adotada pela empresa, decidiu-se então partir para o desenvolvimento do software.

A partir desta decisão, procuramos encontrar ferramentas que facilitassem o processo de desenvolvimento em Linux, de forma que fosse tão rápido como no Windows. No sistema da Microsoft, já existem ferramentas consagradas de desenvolvimento e devido à esta disponibilidade não há como negar que é relativamente fácil desenvolver sistemas para Windows.

Depois de muita leitura e pesquisa em sites na Internet, leitura de *how-to's*[2] e de alguns livros, conseguimos "aprender" a utilizar algumas destas ferramentas e então as adotamos no desenvolvimento de nossos sistemas. Para mostrar aos programadores que desenvolver aplicativos em Linux pode ser tão fácil como no Windows, foi desenvolvido este trabalho.

1.2. Organização deste trabalho

Este trabalho descreve com o máximo de detalhes possível, o processo de desenvolvimento de um sistema em Linux. Ele contempla desde a sua especificação, obtida em conversas com o usuário, passando pela modelagem até a implementaço do sistema propriamente. Além da documentação do próprio sistema, também procuramos descrever de uma forma resumida e objetiva as ferramentas utilizadas em cada fase do projeto.

O capítulo 2 apresenta as ferramentas escolhidas e os motivos que levaram à sua escolha. O capítulo 3 apresenta o ponto de partida para o desenvolvimento do sistema: sua especificação. Ele detalha o processo de vendas de de uma forma bem objetiva supondo em primeira análise o funcionamento perfeito, sem exceções a serem corrigidas.

Descrito o problema e escolhidas as ferramentas, partimos para o desenvolvimento do sistema. Os capítulos 4 e 5 respectivamente apresentam os modelos de dados e os diagramas UML resultado da modelagem do sistema a partir de sua descrição. Além de apresentar o resultado (modelos) também é feita uma breve explicação de como obter o resultado com a ferramenta utilizada.

O capítulo 6 aborda a implementação do sistema. Para este trabalho, supõe-se que o leitor tenha um mínimo de conhecimento da linguagem C/C++. O objetivo aqui é mostrar que, apesar da má fama (em termos de facilidade) o C/C++ não é nenhum "bicho-papão" e que é possível trabalhar com ele com relativa facilidade quando se usa as ferramentas adequadas.

O capítulo 7 apresenta ao leitor como desenvolver a documentação do sistema, tanto para o desenvolvedor como para o usuário. São mostrados aqui de forma bem resumida e objetiva como usar o kdoc e o DocBook para gerar manuais e páginas de ajuda para documentar um sistema.

Finalmente no capítulo 8, são apresentadas as considerações finais sobre o trabalho, como superar alguns obstáculos no desenvolvimento de sistemas em Linux e algumas sugestões de implementações adicionais para melhorar o sistema descrito neste trabalho.

Esperamos que a leitura deste trabalho seja no mínimo interessante e que ajude-o a derrubar alguns mitos sobre o desenvolvimento de software em Linux. Se este material icentivar alguém a desenvolver um software, mesmo que não seja livre, em Linux ele já terá cumprido seu objetivo - o de facilitar a adoção do Linux também nas estações de trabalho das empresas.

Notas

1. Download é o processo de baixar, ou copiar, arquivos de um computador remoto, no caso a *Internet*
2. *Howto* é um guia que explica como fazer alguma coisa. Existem inúmeros *howto's* disponíveis na Internet que ensinam desde a configuração básica de um computador até a construção de clusters. A maioria deles pode ser vista em http://www.lpi.org/

Capítulo 2. A escolha das ferramentas

Este capítulo mostra como foi feita a escolha das ferramentas utilizadas no decorrer do desenvolvimento do sistema descrito neste trabalho. É apresentado aqui os critérios de escolha, uma breve apresentação da ferramenta e uma justificativa pela sua escolha.

2.1. Critérios de escolha

Até chegar ao ponto de "colocar a mão na massa" e começar o desenvolvimento, temos que passasr por um processo chato e que sempre gera polêmicas: a escolha das ferramentas. Chato no sentido de que estaremos vendo várias ferramentas que fazem basicamente as mesmas coisas e polêmica porque não há como dizer que uma ferramenta é melhor que a outra, mas que uma ferramenta *para mim* foi mais adequada. No mundo Windows é relativamente fácil fazer esta avaliação, pois já existem ferramentas consagradas que facilitam muito o processo de desenvolvimento. Já no domínio do *software livre* a escolha já é mais trabalhosa pois existem ferramentas ótimas mas incompletas, ou outras completas mas difíceis de usar.

A escolha das ferramentas aqui se deu considerando as seguintes variáveis:

Mercado

: Quando falamos de mercado, significa que estamos analisando se a ferramenta é disponibilizada pelas distribuições Linux, se ela funciona nas distribuições e ambientes mais populares e se tem respaldo da comunidade *open source* para continuidade de seu desenvolvimento.

Facilidade de instalação e uso

: Variável importante, já que não adianta ser eficiente se não oferecer facilidade de uso e instalação. Muitas pessoas desistem de usar sistemas livres (principalmente Linux) justamente por achar que eles são difíceis de usar.

Popularidade

: O quesito popularidade é importante porque quanto mais pessoas utilizam a ferramenta, mais fonte de informação você terá para solucionar alguma dúvida.

Opinião pessoal

: É importante deixar claro que esta variável foi considerada, pois assim o leitor entenderá mais facilmente a adoção de uma ferramenta em detrimento de outra que também atenda às outras variáveis. Só para exemplificar, ao mesmo tempo existem pessoas que amam e outras que odeiam o Windows, ou seja, a opinião pessoal tem um peso muito grande na escolha de um ou outro sistema.

Para o completo desenvolvimento do sistema, foram adotadas cinco ferramentas além do ambiente nativo do sistema e do banco de dados utilizado. Para todas elas foram feitos apenas alguns testes e analisadas as variáveis descritas acima. A escolha foi feita tentando o máximo de imparcialidade e deixando a opinião pessoal apenas como "critério de desempate".

2.2. O ambiente

Diferentemente do que acontece no Windows, o Linux por si só não oferece um ambiente de trabalho, um gerenciador de janelas único. Existe um servidor gráfico que oferece as rotinas básicas de manipulação da tela, um cliente que faz o acesso a este servidor e diversos gerenciadores de janela que oferecem ao usuário um ambiente de trabalho. Entre estes gerenciadores de janela estão o WindowMaker, Blanes, XFCE e os populares Gnome e KDE. Na verdade, desde que se tenha as bibliotecas requeridas instaladas, um aplicativo pode funcionar em qualquer um dos gerenciadores disponíveis, e dessa forma um aplicativo feito especificamente para o Gnome roda no Blanes desde que as bibliotecas do Gnome estejam instaladas no sistema.

A escolha do ambiente gráfico é polêmica pois a opinião pessoal tem um peso muito forte na decisão. Talvez este é o ponto onde esta variável tem peso mais forte. Como o sistema funciona independente do ambiente, a polêmica pode ser amenizada, mas o simples fato de ter que instalar as bibliotecas do outro ambiente já pode gerar um ponto de discórdia entre os usuários. Indo direto ao ponto, o ambiente escolhido para o sistema aqui descrito foi o KDE e passamos agora a justificar a escolha.

Sem dúvida alguma os dois maiores, mais populares e mais ativos projetos de gerenciadores de desktop disponíveis no mundo *open source* são o KDE (http://www.kde.org) e o GNOME (http://www.gnome.org/). Pensando nos critérios utilizados para a escolha estas são as duas melhores alternativas, pois são populares, têm o respaldo da comunidade, são fornecidas em todas as distribuições LInux do mercado e são ambientes padrão na maioria das distribuições.

Dentre os dois selecionados (GNOME e KDE), a escolha é ainda mais difícil e mais polêmica, pois ambas tem um conjunto de ferramentas e recursos semelhantes, são ambas muito agradáveis de utilizar e ambas derivam de bibliotecas robustas e completas. A escolha caiu sobre o KDE porque este ambiente tem pelo menos três diferenciais em relação ao GNOME: é o ambiente padrão na maioria das distribuições Linux, a comunidade de desenvolvimento é maior e a biblioteca na qual ele se baseia é 100% orientado a objeto.

Dos três diferenciais apresentados talvez o que teve maior peso na escolha foi a orientação a objetos. Apesar da biblioteca GTK (http://www.gtk.org/) que é a base do GNOME ter uma extensão orientada a objetos, ela não é nativa, ou seja, a biblioteca padrão não é e isto sugere que exista mais uma camada de abstração, o que não acontece com a biblioteca QT (http://www.trolltech.com). Esta biblioteca foi concebida originalmente usando orientação a objeto. Além disto, a biblioteca QT tem uma documentação mais farta e completa, o que facilita bastante o seu aprendizado.

Assim escolhemos o KDE, lembrando que para utilizar o sistema não significa que o seu ambiente gráfico tenha que ser o KDE. Basta que as bibliotecas dele estejam instaladas que ele vai funcionar independente de qual gerenciador de janelas você estiver usando.

2.3. Banco de dados

Aqui outra decisão polêmica. No universo do software livre, os dois banco de dados mais conhecidos e largamente utilizados são o MySQL (http://www.mysql.com/) e o PostgreSQL (http://www.postgresql.org/). Em termos de facilidade de uso e recursos disponíveis para administração a escolha cai sobre o MySQL. Porém se considerarmos os recursos de banco de dados e padronização, escolheríamos o PostgreSQL.

A primeira proposta foi de utilizar o PostgreSQL e a justificativa tem muito fundamento. O PostgreSQL implementa recursos avançados como integridade referencial e trnsações. Ainda implementa as chamadas *VIEWS* o que aumenta a segurança, pois podemos implementar uma política onde o usuário tem acesso

apenas às *VIEWS* e não diretamente às tabelas. Outro recurso interessante e que pode ser utilizado são as sub-consultas.

Mas com o surgimento da versão 4 do MySQL, dois recursos importantes foram adicionados: transações e integridade referencial. As versões atuais do MySQL incorporam as tabelas do tipo InnoDB (http://www.innodb.com/) que adicionaram estas funcionalidades ao MySQL. Com a implementação destes recursos, o MySQL passa a ser tão competente quanto o PostgreSQL e um excelente candidato a ser adotado no projeto.

Pela facilidade de uso e por ter uma API com uma documentação mais direta e de fácil implementação, a escolha foi o MySQL. Devido à forma como o sistema foi projetado está fácil de alterar o banco de dados. Basta alterar o código de algumas classes para fazer a conexáo com outro banco de dados e pronto. Não é necessário alterar as classes de mais alto nível que tratam das interfaces com o usuário ou alguma outra que *usa* o banco de dados.

2.4. Ferramentas de análise

Para desenvolver um bom sistema, não basta ter uma boa idéia, sentar à frente do computador e começar a implementar o sistema partindo diretamente para sua codificação. Para sistemas simples esta estratégia pode funcionar, mas para sistemas maiores uma boa documentação deve ser elaborada para orientar a implementação além de possibilitar a validação com o usuário de uma forma mais clara e consistente. O desenvolvimento de um sistema de computador deve funcionar como na construção de uma casa, onde existem os projetos de arquitetura que são aprovados pelos donos e os projetos de estrutura, fundação entre outros que direcionam a construção.

Atualmente dois modelos são adotados pelos desenvolvedores de sistema: a análise estruturada e a análise orientada a objetos. Ambas possuem suas vantagens e desvantagens, mas no caso do sistema descrito aqui, foi adotada a análise orientada a objetos. Existem pelo menos dois bons motivos para esta escolha a biblioteca do ambiente escolhido e a disponibilidade de ferramentas. Com relação à biblioteca, o KDE é fundamentado no QT que é 100% orientado a objetos e assim usando a análise também orientada a objetos, ela fica mais consistente com a implementação do sistema. Já no caso da disponibilidade de ferramentas, todas as ferramentas analisadas ou oferecem recursos para ambos modelos ou apenas para o modelo orientado a objetos.

Dentro da análise orientada a objetos, a modelagem do sistema é feita usando-se a UML - Unified Modeling Language (http://www.uml.org) e portanto a ferramenta escolhida deve oferecer os recursos necessários para esta modelagem. Depois de muita procura, chegamos a quatro ferramentas que foram fáceis de instalar e utilizar. Já existe no Linux, uma versão do Rational Rose (http://www.rational.com), ferramenta muito utilizada no Windows, mas para usar apenas ferramentas livres, descartamos esta excelente ferramenta. As quatro ferramentas analisadas foram o dia (http://www.lysator.liu.se/~alla/dia), o TCM - Toolkit for Conceptual Modeling (http://www.cs.utwente.nl/~tcm/), o ArgoUML (http://www.argouml.org) e o Umbrello (http://www.umbrello.org). Destes, usamos o TCM para a modelagem de dados e o Umbrello para desenhar os diagramas UML. O TCM foi escolhido principalmente por permitir documentar cada elemento do diagrama (mesmo que de forma primária) e gerar diagramas em vários formatos entre eles o PostScript.

A escolha do Umbrello se deu basicamente por dois motivos. O primeiro deles é que das ferramentas analisadas só o Umbrello implementa o diagrama de seqüência. O outro motivo é que o Umbrello é parte

do projeto KDE a partir da versão 3.2, e conseqüentemente é disponibilizado junto com o ele. Assim o Umbrello se torna a ferramenta mais conveniente para utilização.

2.5. Linguagem de programação

Para decidir, entre tantas opções disponiíveis, vários fatores precisam ser considerados e ainda assim a escolha pode não ser a ideal. Se pensarmos apenas em termos de mercado (brasileiro, pelo menos), a linguagem escolhida deveria ficar entre o MS Visual Basic (http://msdn.microsoft.com/vbasic/) ou o Borland Delphi (http://www.borland.com/delphi/index.html).

Mas apesar de muito competentes e fáceis de utilizar, ambas fugiriam de um dos propósitos seguidos neste trabalho: de utilizar apenas ferramentas livres. Neste quesito, o Kylix (http://www.borland.com/kylix/index.html) que tem uma versão gratuita poderia ser cogitada para utilização no projeto, porém apesar de gratuito[1], o Kylix não é software livre e portanto, foi descartado.

Dentre as linguagens realmente livres, poderímos decidir entre Free Pascal (http://www.freepascal.org/), GNU Pascal (http://www.gnu-pascal.org/), GNU C/C++ (http://www.gnu.org/) entre várias outras. Pensando em implementação em ambiente web, já teríamos o PHP (http://www.php.net/) ou ainda o PERL (http://www.perl.org/).

2.5.1. Ambiente Cliente/Servidor e o C++

Pela robustez, pela possibilidade de aproveitar melhor os recursos de rede e da própria estação, o ambiente cliente servidor é o ideal para a implementação do sistema. Como o processamento é feito na estação, a carga de processamento no servidor é minimizada - apenas o banco de dados roda no servidor. Outro ponto que pesa a favor deste ambiente é que apenas os dados solicitados ou enviados pela estação trafegam pela rede.

Decidido pelo ambiente cliente/servidor falta agora definir a linguagem de programação. Nenhuma das linguagens citadas proporcionam um ambiente de desenvolvimento rápido como temos no VisualBasic ou no Kylix e inicialmente a escolha parece ser impossível. Imagine ter que se preocupar, além do layout das telas, em escrever todas as rotinas de manipulação de telas e dados. Seria uma perda de tempo muito grande. Estava então difícil fugir do Kylix.

Felizmente, os desenvolvedores das interfaces gráficas (principalmente Gnome (http://www.gnome.org/) e KDE (http://www.kde.org/)) disponibilizaram as bibliotecas para que os programadores de aplicativos pudessem utilizar em seus sistemas. O problema agora é aprender a utilizar estas bibliotecas. No caso do Gnome, a biblioteca básica é a GTK (http://www.gtk.org/) e no KDE, é utilizado o Qt (http://www.trolltech.com/). A biblioteca escolhida foi o Qt e conseqüentemente a linguagem adotada foi o C++. Vamos agora justificar a escolha.

Por mais que o Gnome seja eficiente, enxuto e agradável, ele não consegue bater o KDE em popularidade. A maioria das distribuições Linux adotaram o KDE como ambiente gráfico padrão e portanto sua utilização aumenta a compatibilidade do sistema entre várias distribuições disponíveis. Outra grande facilidade do Qt é que a biblioteca é 100% orientada a objetos e reimplementar uma classe é mais intuitivo que trabalhar com ponteiros de funções, o que ocorre no GTK. Existe sim uma extensão do GTK orientado a objetos, mas isto não é nativo, diferentemente do Qt que já foi concebido usando POO.

Uma outra (grande) vantagem na adoção do Qt/KDE é que existem ferramentas que tornam o desenvolvimento C++ rápido e intuitivo. O Qt traz consigo o excelente Designer (http://www.trolltech.com/products/qt/designer.html) que torna o desenho das interfaces uma tarefa fácil e envolvente. Apesar de também gerar código C++ personalizado, os fontes gerados são estritamente Qt, que fogem ao estilo GNU de desenvolver sistemas (configure -> make -> make install).

Para trabalhar no código C++, a escolha foi o ambiente KDevelop (http://www.kdevelop.org/) que é bem integrado ao KDE, Designer e ainda gera distribuições no estilo GNU e distribuições rpm. Além disto, no KDevelop, a classe que representa a janela, é herdada e as alterações efetuadas nesta classe não interferem na interface desenhada pelo Designer. Some-se a isto o fato de que o KDevelop é mais competente para lidar com os códigos-fonte e com a manipulação de classes.

Para conectar ao banco de dados, também não há problemas, pois o MySQL (http://www.mysql.com/) já fornece uma API nativa em C. Para utilizá-la, basta acrescentar o header do MySQL e depois ao compilar o programa, fazer a ligação com a biblioteca.

Finalmente, a escolha da linguagem C++ também sofreu influências do próprio Linux e seus aplicativos, quase 100% deles escritos em C. Já que a grande maioria dos programadores Linux usam C, por que procurar outra linguagem? Além do mais o C já vem acompanhando qualquer distribuição Linux do mercado.

2.6. Documentação do sistema

Nenhum sistema é realmente bem desenvolvido se não tiver uma documentação que facilite, tanto ao desenvolvedor quanto ao usuário a compreensão e utilização de todas as partes que formam o sistema. Além disto, a forma na qual esta documentação é disponibilizada, também pode ser de fundamental importância na "popularização" do sistema.

As ferramentas de modelagem e de implementação utilizadas neste trabalho, facilitam bastante o trabalho "braçal", mas não são tão eficientes para disponibilizar a documentação. Para se tornar um software popular e para atrair o interesse de programadores da comunidade "open source", todo o sistema deve ser documentado de modo que o desenvolvedor consiga entender o funcionamento e implementação mesmo antes de ver o código fonte.

Para documentar as classes, os arquivos header (.h) foram todos documentados de acordo com as especificações do kdoc[2], que lé estes comentários e gera páginas html contendo a descrição das classes do sistema. Logicamente os comentários na codificação do sistema não foram esquecidas, mas a documentação estilo kdoc foi a que mereceu mais atenção, pois as classes ficam documentadas facilitando o seu entendimento sem nem mesmo ver o código fonte.

Para elaborar a documentação do sistema, manual do usuário e até mesmo este trabalho, poderia ter sido utilizado qualquer processador de texto. Porém, seguindo as especificações da TLDP - The Linux Documentation Project (http://www.tldp.org/), foi utilizado o DocBook (http://www.docbook.org/) que posteriormente gera a documentação em HTML, PDF, LaTeX, PostScript entre outros formatos a partir de um mesmo fonte xml ou sgml.

Notas

1. Conforme a Free Software Foundation (http://www.fsf.org/), o fato de o software ser gratuito não implica que ele seja livre. Para ser livre, o software precisa no mínimo ter o código fonte disponível, o que não acontece com o Kylix.
2. O KDoc geralmente acompanha o KDE nas distribuições do Linux.

Capítulo 3. Descrição do sistema

Neste capítulo, vamos descrever o sistema a ser desenvolvido conforme as informações obtidas junto aos funcionários que serão os usuários do sistema. O processo de vendas, que é onde o sistema deve atuar, pode ser dividido em três fases: negociação, acerto no caixa (pagamento) e expedição. Todas elas são descritas aqui e serão modeladas no decorrer deste trabalho.

3.1. A negociação

Durante o expediente, o vendedor atende os clientes via telefone ou diretamente no balcão. O vendedor, de acordo com as informações passadas pelo cliente, consulta a lista de mercadorias no computador, verifica a quantidade disponível e o preço da mercadoria. Para realizar esta consulta o vendedor pode pesquisar pela descrição da mercadoria, por grupo, pelo próprio código ou ainda pela referência (código do fabricante).

Os preços deverão ser dependentes da forma de pagamento. Existe um preço de venda sugerido (que considera um prazo de 30 dias) e dependendo da forma de pagamento pode ter mais ou menos descontos. Para não prender o vendedor a preços e descontos pré-fixados, o sistema deve permitir que ele conceda descontos até o limite máximo estabelecido para vendas a vista e para vendas a prazo. Para conceder este desconto, o vendedor pode digitar o preço da mercadoria diretamente ou ainda conceder desconto ao final da venda (desconto no rodapé da nota).

O cadastramento dos dados do cliente também está condicionado à forma de pagamento. Para vendas em dinheiro o cadastramento é opcional e o vendedor tem orientação para oferecer o cadastro para posteriormente o cliente poder comprar faturado. Para vendas a prazo ou em cheque (a vista ou pré-datado) é feito um cadastro criterioso do cliente. Para este cadastro, é exigido além dos dados completos (nome, endereço, telefone, etc) 3 referências comerciais onde o cliente tem cadastro para compras a prazo. O vendedor tem autorização para fazer o cadastro completo do cliente, mas não tem autorização para liberar o crédito.

Dependendo da situação do cliente (inadimplência), o departamento financeiro pode bloquear o seu cadastro de forma a não permitir vendas a prazo. Em casos extremos pode ser desejável bloquear por completo a venda para um determinado cliente - o depto financeiro usa este artifício para obrigar o cliente a no mínimo conversar com o gerente financeiro antes de efetuar qualquer compra.

Concluídos todos os passos (negociação e cadastramento), o vendedor faz o registro da venda, gerando uma ordem de venda que é impressa no estoque e fica aguardando o pagamento no caixa. O cliente então é encaminhado ao caixa para efetuar o pagamento e posteriormente à expedição para retirada da mercadoria. Nas vendas por telefone em que será feita uma entrega, o entregador deve antes se dirigir ao caixa para entregar uma via da ordem de venda que fica como pendência para acertar na sua volta.

3.2. Acerto no caixa

Depois de concluída a negociação por parte do vendedor, o cliente é encaimnhado ao caixa para fazer o acerto da venda. No caixa, o sistema deverá exibir uma lista das vendas que foram negociadas e que ainda não foram acertadas. O cliente fornece o nome e a venda (que deverá estar relacionada na lista) será baixada e liberada para expedição. Feito o acerto, uma nota (ou cupom) fiscal é emitido para a venda.

Caso o cliente deseje mudar a forma de pagamento, ele deve ser encaminhado de volta ao vendedor para ver se é possível a alteração (lembrando que os preços dependem da forma de pagamento escolhida).

Para o caso de entregas, o entregador deverá informar ao caixa quais as vendas que ele está levando e elas são marcadas como parcialmente acertadas. Quando o entregador retornar, ele entrega ao caixa o recebimento propriamente dito. De posse deste recebimento, a venda deverá ser acertada e então é eliminada da lista de pendências no caixa.

3.3. Expedição

Quando a venda é negociada pelo vendedor, uma ordem de venda é emitida no estoque para que os estoquistas possam separar as peças para expedição. Se a venda for para ser entregue, o estoquista convoca um entregador, informa as vendas que ele vai entregar e o encaminha ao caixa para fazer o acerto preliminar das vendas. Quando a mercadoria for para despachar, o estoquista deve informar o número de volumes que a venda gerou (a nota fiscal contém esta informação). Como alternativa, o próprio estoquista pode emitir a nota fiscal que será impressa no caixa.

Na expedição, o sistema deve exibir uma lista das vendas que já foram acertadas no caixa. O estoquista somente poderá expedir a mercadoria se a venda estiver na relação apresentada pelo sistema, ou seja, vendas já acertadas. No caso de entregas, o acerto parcial já é suficiente para o estoquista fazer a expedição, pois o acerto real será feito no retorno do entregador. Depois de expedida a mercadoria, o estoquista então faz a entrega da venda removendo-a da lista de pendências.

3.4. Relatórios de acompanhamento

Para acompanhamento e totalizações, o sistema deverá ser capaz de emitir pelo menos os seguintes relatórios:

- Lista de preços (por grupo, por descrição).
- Relatório de vendas por grupo, exibindo para cada grupo o total de vendas
- Relatório de vendas por grupo e produtos.
- Relatório de vendas por cliente.
- Relatório de vendas por forma de pagamento.
- Totalização de vendas com comissão por vendedor.

Capítulo 4. O modelo de dados

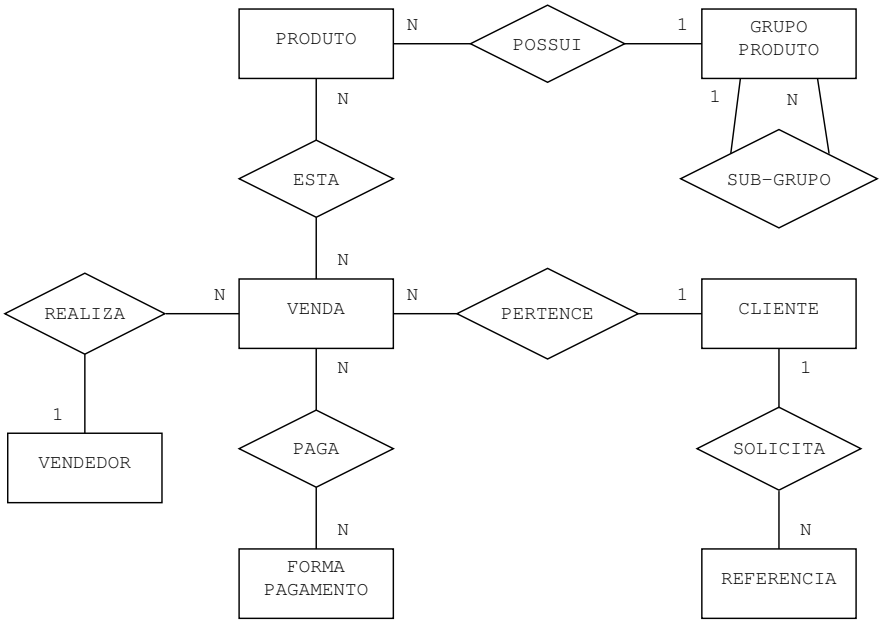
Neste capítulo, vamos realizar a modelagem de dados do sistema. Aqui definiremos o que deve ser armazenado no banco de dados, quais informaçç;ões deverão ser processadas pelo sistema. Primeiramente é apresentado o DER (Diagrama de Entidade Relacionamento) obtido segundo a especificação do sistema. Depois damos algumas dicas para obter o diagrama utilizando o TCM - Toolkit for Conceptual Modeling.

Depois de apresentado o DER, documentamos a estrutura das tabelas que deverão ser implementadas no banco de dados. Também mostramos como podemos criar as tabelas no banco de dados de forma rápida usando um script SQL.

4.1. O Diagrama Entidade-Relacionamento

A partir da especificação do sistema, podemos considerar como válido o DER mostrado logo abaixo. É claro que o modelo apresentado não representa uma solução única, mas foi a que melhor se encaixou nos requisitos apresentados.

Figura 4-1. O diagrama entidade relacionamento



É realmente um diagrama bem simples, assim como deve ser simples um sistema para vendas. Note que no modelo não há nenhuma referência aos documentos fiscais exigidos (nota e cupom). Isto porque cada empresa trabalha com um formulário personalizado, e além disto os produtos podem ser regidos por

legislações diferenciadas que alteram a forma de calcular impostos. Por esta complexidade e para deixar o sistema mais "genérico", decidiu-se por retirar esta funcionalidade do escopo deste trabalho.

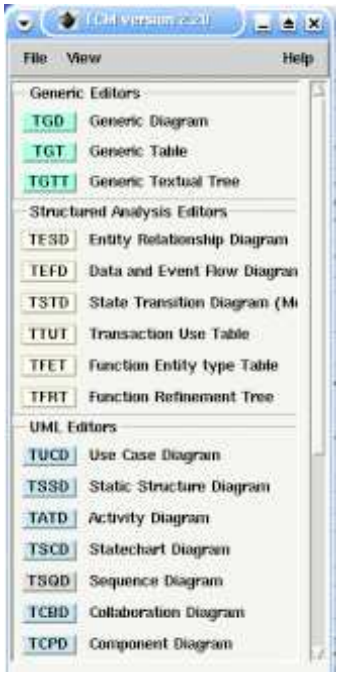
Como dito na escolha das ferramentas, usamos o TCM para elaborar o diagrama mostrado e agora passamos a explicar sucintamente como obtê-lo usando o TCM.

4.1.1. Gerando o DER com o TCM

O TCM - Toolkit for Conceptual Modeling, é uma ferramenta que fornece diversos editores gráficos para a modelagem de diferentes diagramas usados na especificação de sistemas. Estes diagramas representam a estrutura conceitual do sistema (daí o nome da ferramenta).

Para cada diagrama, o TCM oferece um editor específico. O TCM consegue gerar diagramas para a análise estruturada, análise orientada a objetos (diagramas UML) e também tem série de diagramas adicionais (genéricos). A tela principal do TCM é dividida em seções contendo os diagramas específicos para cada abordagem de desenvolvimento. Para elaborar o diagrama de Entidade Relacionamento, usamos o editor *TESD (Entity Relationship Diagram).*

Figura 4-2. A tela principal do TCM

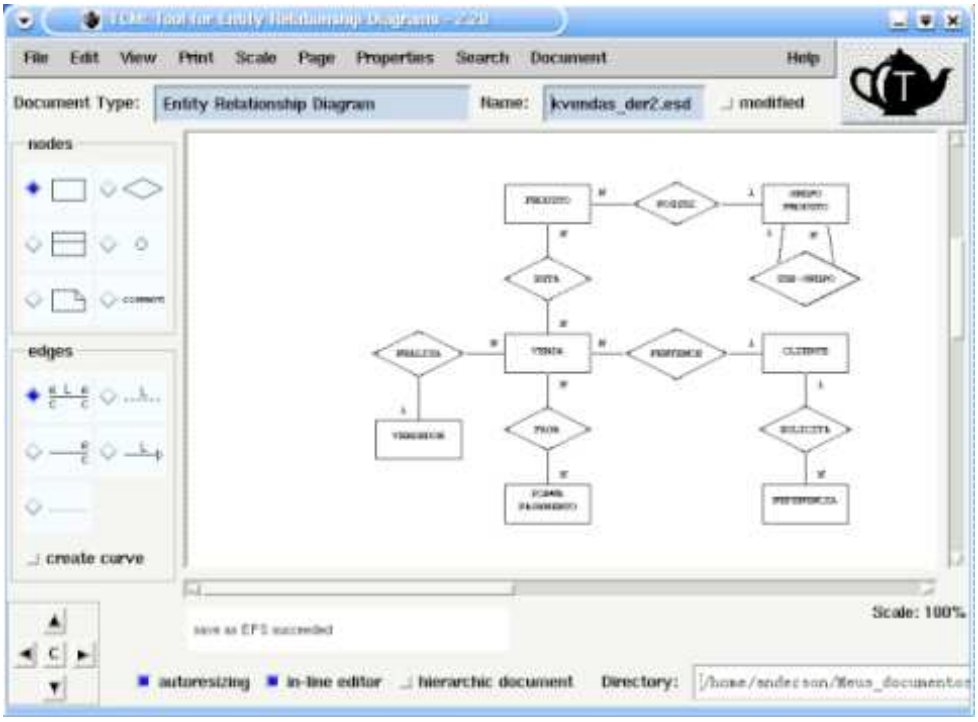


Selecionada o editor, uma nova janela se abre para podermos desenhar o diagrama. A tela do TESD apresenta os elementos do diagrama e as ligações possíveis à esquerda, algumas informações sobre o documento na parte de baixo e à direita, ocupando a maior parte da tela está a área de desenho.

Para desenhar um elemento no diagrama, basta clicar sobre o elemento desejadoe clicar na área de desenho que ele será criado. Para ligar doiselementos selecione o tipo de ligação desejada e com obotão central

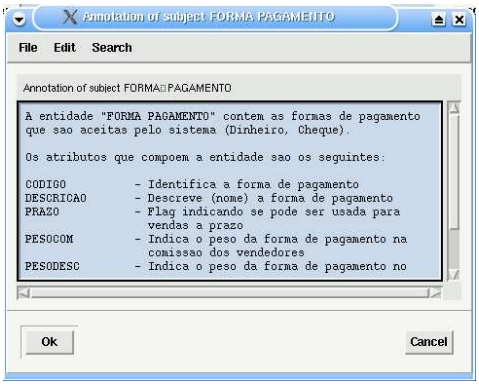
do mouse ligue os dois elementos. O TCM faz algumas validações nas ligações, evitando ligar elementos incompatíveis com o tipo de ligação selecionado.

Figura 4-3. O Editor TESD - Entity Relationship Diagram



É possível documentar cada elemento do diagrama, bastando para isto selecionar o elemento e no menu *Properties* escolher a opção *Node/Edge Annotation*. Uma tela se abre para você fazer a documentação do elemento. Em nosso projeto, utilizamos esta facilidade para colocar a estrutura de cada entidade.

Figura 4-4. Documentação da entidade FORMA PAGAMENTO



Depois de desenhado o diagrama, podemos imprimi-lo, mas geralmente este diagrama fará parte de um documento contendo a especificação do sistema (como neste trabalho). Devemos então gerar o diagrama em uma forma que a imagem possa ser incorporada a este documento.

O TCM consegue exportar o diagrama para vários formatos, entre eles o png, *PostScript* (ps) e *Encapsulated PostScript* (eps). Para a realização deste trabalho[1], foi usado o formato eps. Para converter de eps para outro formato (gif por exemplo) basta usar um editor gráfico como o Gimp (http://www.gimp.org/).

4.2. Estutura do banco de dados

Depois de elaborado o DER, devemos criar as tabelas no banco de dados. A grosso modo, basta criar uma tabela para cada entidade do diagrama, mas a estrutura final do banco de dados depende de algumas análise sobre o DER. Os relacionamentos "um-para-muitos", por exemplo, são mapeados exportando-se a chave primária da entidade que está no lado "um " para a entidade do lado "muitos". Nos relacionamentos "muitos-para-muitos" são criadas tabelas adicionais com as chaves de ambas as entidades e algum atributo adicional requerido pelo relacionamento.

Dessa forma conseguimos a estrutura do banco de dados que será mostrada a seguir. Cada tabela apresenta uma breve descrição e para cada atributo "não óbvio" também conté uma breve explicação de sua funcionalidade no sistema.

4.2.1. A estrutura das tabelas

Tabela 4-1. GRUPOPRODUTO - Agrupa produtos semelhantes

Campo	Tipo de dados	Descrição
CODIGO	Numérico	Identificação do grupo de produtos
DESCRICAO	Alfa-numérico	Descrição do grupo
COMISSAO	Porcentagem	Comissão paga nas vendas de peças do grupo
GRUPOPAI	Numérico	Código do grupo ao qual o grupo pertence (se for um subgrupo). Este atributo mapeia o relacionamento SUBGRUPO.

Tabela 4-2. PRODUTO - Cadastro de produtos

Campo	Tipo de dados	Descrição
CODIGO	Numérico	Identifica o produto no sistema
GRUPO	Numérico	Grupo ao qual o produto pertence (chave de GRUPOPRODUTO)
DESCRICAO	Alfa-numérico	Descrição completa do produto

Campo	Tipo de dados	Descrição
APELIDO	Alfa-numérico	Descrição resumida do produto
UNIDADE	Alfa-numérico	Identifica a unidade em que é vendida o produto
FRACAO	Sim/Não	Indica se pode vender quantidades fracionadas
DESCONTO	Porcentagem	Desconto máximo permitito ao produto
COMISSAO	Porcentagem	Comissão sobre o valor de venda
SITUACAO	Alfa-numérico	Indica se um produto está disponível para negociação (A=Ativo, I=Inativo)

Tabela 4-3. CLIENTE - Esta tabela contém os dados de cadastro do cliente

Campo	Tipo de dados	Descrição
CODIGO	Numérico	Identifica o cliente no sistema
NOME	Alfa-numérico	Nome do cliente (para jurídica representa a razão social)
APELIDO	Alfa-numérico	Apelido do cliente (para jurídica representa o nome fantasia)
ENDERECO	Alfa-numérico	Endereço completo da pessoa (rua, número, complemento)
BAIRRO	Alfa-numérico	Bairro
CIDADE	Alfa-numérico	Cidade
ESTADO	Alfa-numérico	Sigla do estado
CEP	Numérico	Número do CEP
TEL1	Numérico	Telefone completo (incluindo DDD)
TEL2	Numérico	Telefone completo (incluindo DDD)
FAX	Numérico	Fax (incluindo DDD)
EMAIL	Alfa-numérico	Caixa postal na Internet
CPF	Numérico	Número do CPF da pessoa (pessoa física)
IDENTIDADE	Numérico	Número da identidade
CNPJ	Numérico	Número CNPJ (pessoa jurídica)
INSCEST	Numérico	Número da inscrição estadual
DATACADASTRO	Data	Data de cadastro da pessoa
TIPO	Alfa-numérico	Tipo de cliente (Oficina, Revenda, Consumidor, Transportadora)

Campo	Tipo de dados	Descrição
CADASTRO	Numérico	Código do funcionário que cadastrou o cliente
VENDEDOR	Numérico	Código do vendedor que solicitou o cadastro
SITUACAO	Alfa-numérico	Situação de cadastro do cliente (Vista, Prazo, Bloqueado)

Tabela 4-4. REEFERENCIA - Referências do cliente

Campo	Tipo de dados	Descrição
CODIGO	Numérico	Código do cliente (chave na tabela CLIENTE)
NUMERO	Numérico	Identifica a referência do cliente
NOME	Alfa-numérico	Nome da referência
TELEFONE	Alfa-numérico	Telefone da referência
CONTATO	Alfa-numérico	Pessoa que forneceu informação
DTULTIMACOMPRA	Data	Data da última compra
VLULTIMACOMPRA	Moeda	Valor da última compra
DTMAIORCOMPRA	Data	Data da maior compra
VLMAIORCOMPRA	Moeda	Valor da última compra
ANOCADASTRO	Data	Ano de cadastro do cliente
CONCEITO	Alfa-numérico	Conceito dado ao cliente pela referência
OBSERVACAO	Alfa-numérico	Observações adicionais sobre o cliente

Tabela 4-5. USUARIO - Dados dos usuários do sistema

Campo	Tipo de dados	Descrição
CODIGO	Numérico	Identificação do usuário
NOME	Alfa-numérico	Nome do usuário
VENDEDOR	Sim/Não	Indica se o usuário é um vendedor para permitir ou não vendas registradas em seu nome.
TELEFONE	Alfa-numérico	Telefone do usuário (incluindo DDD)
SENHA	Alfa-numérico	Senha de acesso ao sistema
SITUACAO	Alfa-numérico	Situação do usuário (Ativo,Inativo)

Tabela 4-6. FORMAPAGTO - Formas de pagamento aceitas na empresa

Campo	Tipo de dados	Descrição
CODIGO	Numérico	Identifica a forma de pagamento
DESCRICAO	Alfa-numérico	Descrição da forma de pagamento
PRAZO	Sim/Não	Indica se pode usar esta forma em vendas à prazo
PESOCOM	Porcentagem	Indica o redutor aplicado sobre a comissão
PESODESC	Porcentagem	Indica o redutor aplicado sobre o desconto máximo
SITUACAO	Alfa-numérico	Situação da forma de pagamento(Ativo,Inativo)

Tabela 4-7. VENDA - Registro das vendas (Orçamento, Ordem de venda e Venda)

Campo	Tipo de dados	Descrição
NUMERO	Numérico	Número da venda
DATA	Data	Data da venda
VENDEDOR	Numérico	Código do vendedor que realizou a venda
CLIENTE	Numérico	Código do cliente para o qual a venda foi realizada
TIPOPAGTO	Alfa-numérico	Tipo do pagamento (Vista, Prazo)
TIPODOCUMENTO	Alfa-numérico	Indica o tipo de documento gerado (Orçamento, Venda)
INFORMACOES	Alfa-numérico	Informações complementares sobre a venda
VENDAFINAL	Numérico	Número da venda que agrupa esta e outras vendas para o mesmo cliente
USREGISTRO	Numérico	Usuário que registrou a venda
DTESTORNO	Data	Data de estorno da venda
USESTORNO	Numérico	Código do funcionário que fez o estorno davenda
MOTIVOESTORNO	Alfa-numérico	Motivo do estorno da venda

Tabela 4-8. ITEMVENDA - Relação de peças vendidas

Campo	Tipo de dados	Descrição

Campo	Tipo de dados	Descrição
VENDA	Numérico	Número da venda (chave na tabela VENDA)
SEQUENCIA	Numérico	Seqüencial da peça na venda
PRODUTO	Numérico	Código do produto vendido
QUANTIDADE	Numérico	Quantidade vendida do produto
PRECOVENDA	Moeda	Preço negociado na venda
PRECOLISTA	Moeda	Preço de venda na lista

Tabela 4-9. PAGAMENTO - Desdobramento do pagamento da venda

Campo	Tipo de dados	Descrição
VENDA	Número	Número da venda à qual este pagamento se refere
SEQUENCIA	Número	Seqüencial da forma de pagamento para a venda
FORMAPAGTO	Número	Forma de pagamento utilizada
VENCIMENTO	Data	Data de vencimento da parcela
VALOR	Moeda	Valor da parcela

4.2.2. Equivalência DER x Banco de dados

Nesta seção mostramos o que representa cada tabela do banco de dados em relação ao DER modelado para o sistema.

Tabela 4-10. Tabela de equivalência DER x BD

Tabela	Elemento do DER
GRUPOPRODUTO	Entidade GRUPOPRODUTO
PRODUTO	Entidade PRODUTO
CLIENTE	Entidade CLIENTE
REFERENCIA	Entidade REFERENCIA
USUARIO	Entidade VENDEDOR
FORMAPAGTO	Entidade FORMA PAGAMENTO
VENDA	Entidade VENDA
ITEMVENDA	Relacionamento ESTA entre as entidades VENDA e PRODUTO
PAGAMENTO	Relacionamento PAGA entre VENDA e FORMA PAGAMENTO

4.3. Gerando as tabelas no banco de dados

Depois de especificado o banco de dados, devemos partir para a criação das tabelas no banco de dados. É possível fazer isto com qualquer ferramenta de administração de banco de dados, mas se mudarmos o gerenciador de banco de dados, teremos que recriar uma a uma as tabelas no novo banco de dados.

Existem ferramentas que criam as tabelas no banco de dados baseado no DER diretamente, mas este infelizmente não é o nosso caso. Para amenizar a situação podemos aproveitar o fato de que os bancos de dados conhecem a linguagem SQL e criar um script para que a criação das tabelas seja feita de uma forma menos traumática.

Cada banco de dados pode ter alguns comandos que "extendem" as funcoinalidades do SQL, mas para deixar o script mais "portável" devemos nos ater aos comandos do SQL padrão ANSI. Ao gerar o script, podemos inclusive documentá-lo já que o SQL permite este recurso (tudo que está após a seqüência "--" é considerado como comentário e não é processado pelo banco de dados.

Assim para gerar a tabela FORMAPAGTO no banco de dados, podemos gerar o seguinte script:

```
-- ************************************************************************
-- Tabela FORMAPAGTO
-- Tabela contendo as formas de pagamento disponíveis para negociação.
-- Restrições como número máximo de parcelas e prazos máximos são
-- restrições de programa não sendo implementadas no banco de dados.
-- ************************************************************************
CREATE TABLE FORMAPAGTO (
 CODIGO  SMALLINT NOT NULL PRIMARY KEY,
 DESCRICAO VARCHAR(30) NOT NULL,
 PRAZO  VARCHAR(1) NOT NULL DEFAULT 'S',
 PESOCOM  DECIMAL(2,1) NOT NULL DEFAULT 1.0,
 PESODESC DECIMAL(2,1) NOT NULL DEFAULT 1.0,
 SITUACAO VARCHAR(1) NOT NULL DEFAULT 'A' -- Ativo/Inativo
);
```

Juntando as especificações de cada uma das tabelas a ser gerada, montamos um script que pode ser enviado ao banco de dados, criando todas as tabelas de uma só vez. Supondo que estamos usando o Linux com o MySQL e que o banco de dados `kvendas` já tenha sido criado e ainda que exista o script SQL `kvendas_db.sql` podemos usar o seguinte comando:

```
$ mysql kvendas < kvendas_db.sql
```

Existe um comando similar que gera as tabelas no PostgreSQL:

```
$ psql kvendas < kvendas_db.sql
```

Notas

1. No caso específico deste trabalho, onde foi usado o DocBook, usou-se o formato eps e gif. Como o DocBook gera documentos em vários formatos diferentes (HTML, PDF) o formato eps é usado para gerar documentos impressos (PDF, PS) e o formato gif é usado para formatos online (HTML, XML)

Capítulo 5. Modelagem do sistema

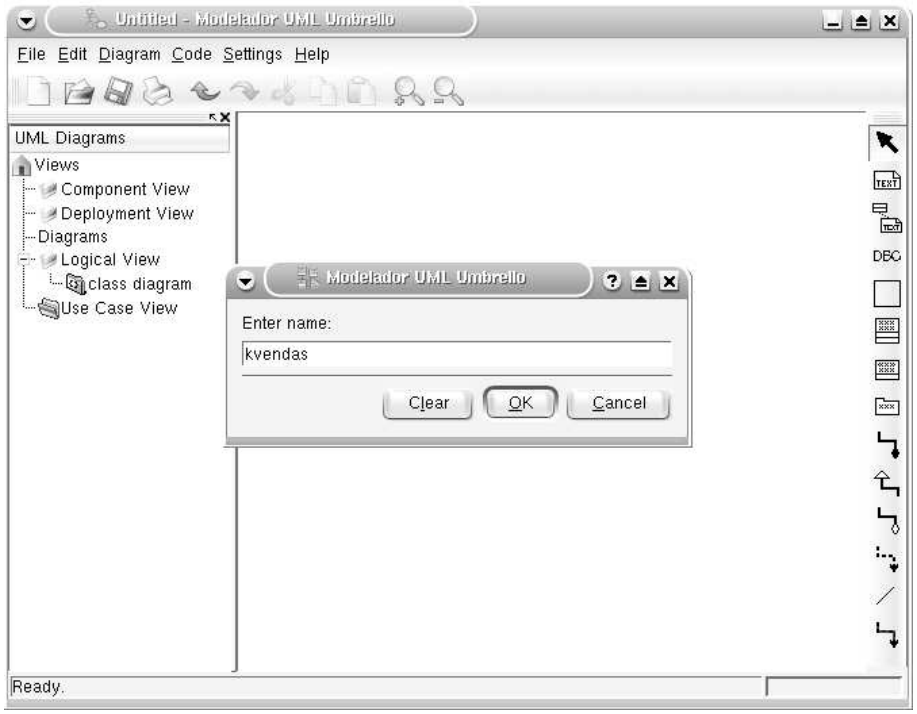
Neste capítulo, apresentamos a modelagem das funcionalidades do sistema de vendas. Aqui apresentaremos alguns dos diagramas elaborados para o sistema e como obtê-los usando a ferramenta escolhida, o Umbrello. Para ver toda a modelagem, você pode baixar o documento kvendas.xmi, gerado pelo Umbrello que está disponível em http://andersonataides.tripod. com.br/arl (http://andersonataides.tripod.com.br/arl/).

5.1. Diagramas de casos de uso

A primeira tarefa ao estudar as especificações do sistema é identificar os casos de uso, ou seja, como os usuários vão interagir com o sistema. Portanto os diagramas de caso de uso vão modelar os cenários ajudando a compreender as exigências do sistema. Neste momento não nos interessa como o sistema vai responder a um determinado evento, mas apenas que ele responde a este estímulo externo.

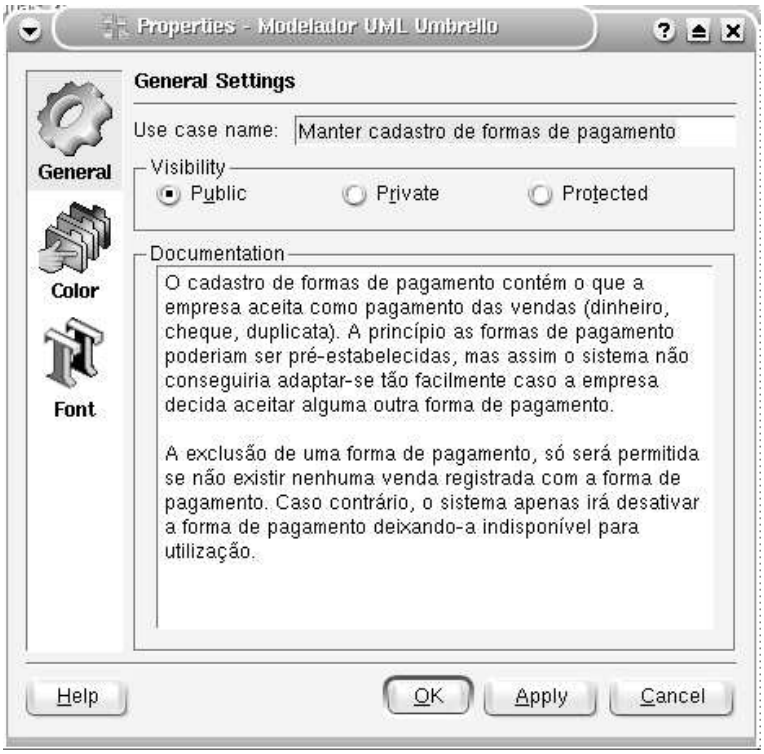
Vamos mostrar agora como criamos nosso diagrama de casos de uso com o Umbrello. Ao abrir o Umbrello, ele já está pronto para desenhar um diagrama de classes. O nosso objetivo agrora é o diagrama de casos de uso e para isto devemos criar um novo diagrama. Para isto basta ir ao menu, `Diagram-New-Use Case Diagram`[1]. Uma caixa de diálogo será exibida para você digitar o nome do diagrama de uso. Vamos dar o nome de kvendas para o nosso diagrama.

Figura 5-1. Umbrello - criando um diagrama de caso de uso



A janela do Umbrello está agora pronta para elaborarmos nosso diagrama. Para criar um ator, basta clicar no ator na barra de ferramentas à direita e clicar na área do diagrama. Para criar um caso de uso, o procedimento é o mesmo. Para associar os elementos, selecione o tipo de ligação na barra de ferramentas e clicar nos elementos a ser conectados. Depois de criados os elemntos do diagrama,podemos passar a documentá-los. O procedimento é simples, basta dar um duplo clique sobre o elemento e uma janela surge para entrarmos a documentação daquele elemento.

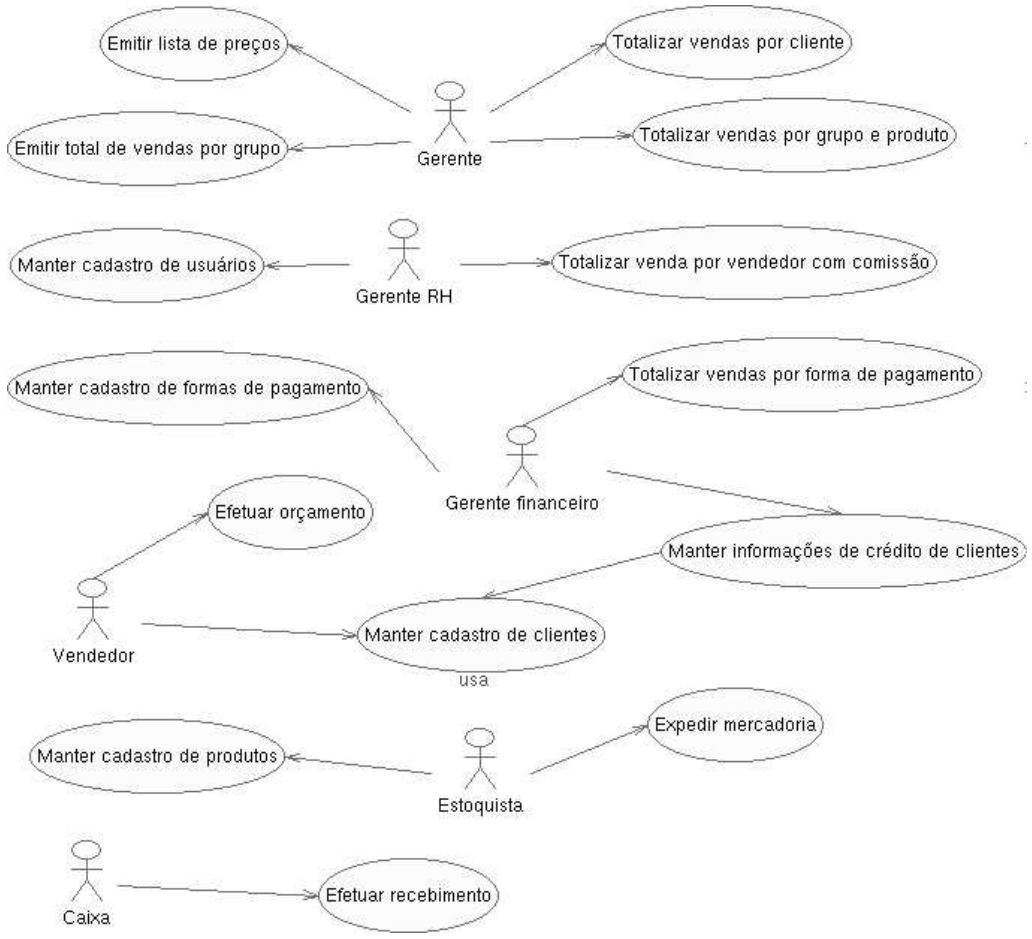
Figura 5-2. Umbrello - documentação de elementos



Depois de criado o diagrama, é possível imprimi-lo ou exporta-lo como uma imagem. Esta funcionalidade é importante para incluir o diagrama em documentos como neste trabalho. O Umbrello consegue exportar o diagrama para os formatos de imagens mais populares do mercado como png, jpeg e bmp. Para criar a imagem, basta clicar com o botão direito do mouse no meio do diagrama e selecionar a opção `Export as Picture`. Depois é só dar um nome à figura para criar o arquivo.

Apresentamos agora o diagrama de caso de uso completo para o nosso sistema. A figura foi obtida como explicado acima: ela foi exportada como uma figura e depois incorporada a este documento.

Figura 5-3. Diagrama de caso de uso do sistema



5.2. Diagramas de seqüência

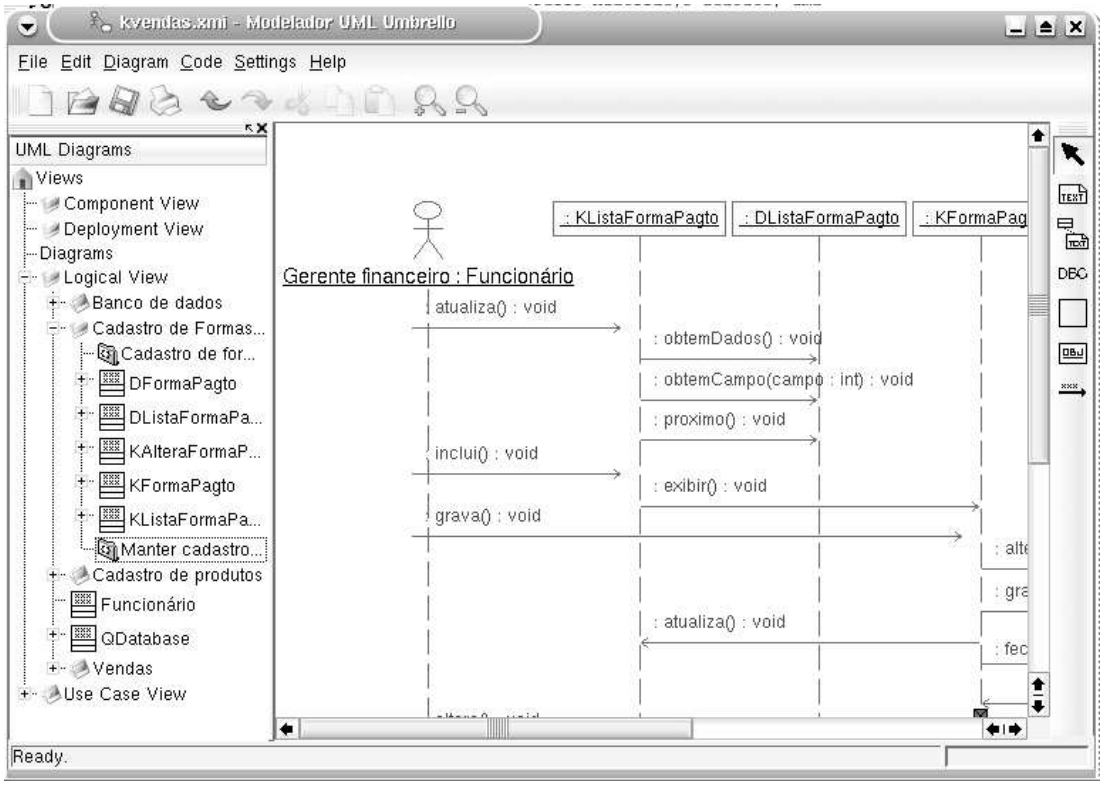
Definidas as funcionalidades do sistema, passamos a modelar o funcionamento do sistema, ou seja, como ele vai trabalhar para responder os eventos modelados pelos casos de uso. Nesta fase já modelamos as classes que vão responder aos eventos. Para cada caso de uso modelado, foi elaborado um diagrama de seqüência, porém, vamos mostrar aqui apenas o diagrama que vamos implementar: `Manter cadastro de forma de pagamento`.

Criar um diagrama de seqüência no Umbrello não é uma tarefa difícil, basta selecionar o menu `Diagram - New - Sequence Diagram`. A área de trabalho do Umbrello é limpa e fica pronta para desenharmos o diagrama. Para organizar melhor nosso trabalho, podemos criar pastas para agrupar o diagrama e as classes que o compõem. Vamos então criar uma pasta com o nome `Cadastro de formas de pagamento` e tanto o diagrama quanto as classes que o compõem serão criados dentro desta pasta.

Para criar uma classe, basta clicar no botão correspondente a classe na barra de ferramentas e clicar no diagrama. Note que este procedimento vai criar uma classe e colocá-la na raiz da pasta `Logical View`.

Para mover a classe para nossa pasta recém criada, basta arrastá-la para dentro da pasta. Para criar um ator, devemos criar uma classe e dizer ao Umbrello que ela deve ser exibida como um ator. Se desejarmos colocar no diagrama uma classe que já exista, basta arrastá-la para o diagrama.

Figura 5-4. Umbrello - Diagrama de seqüência

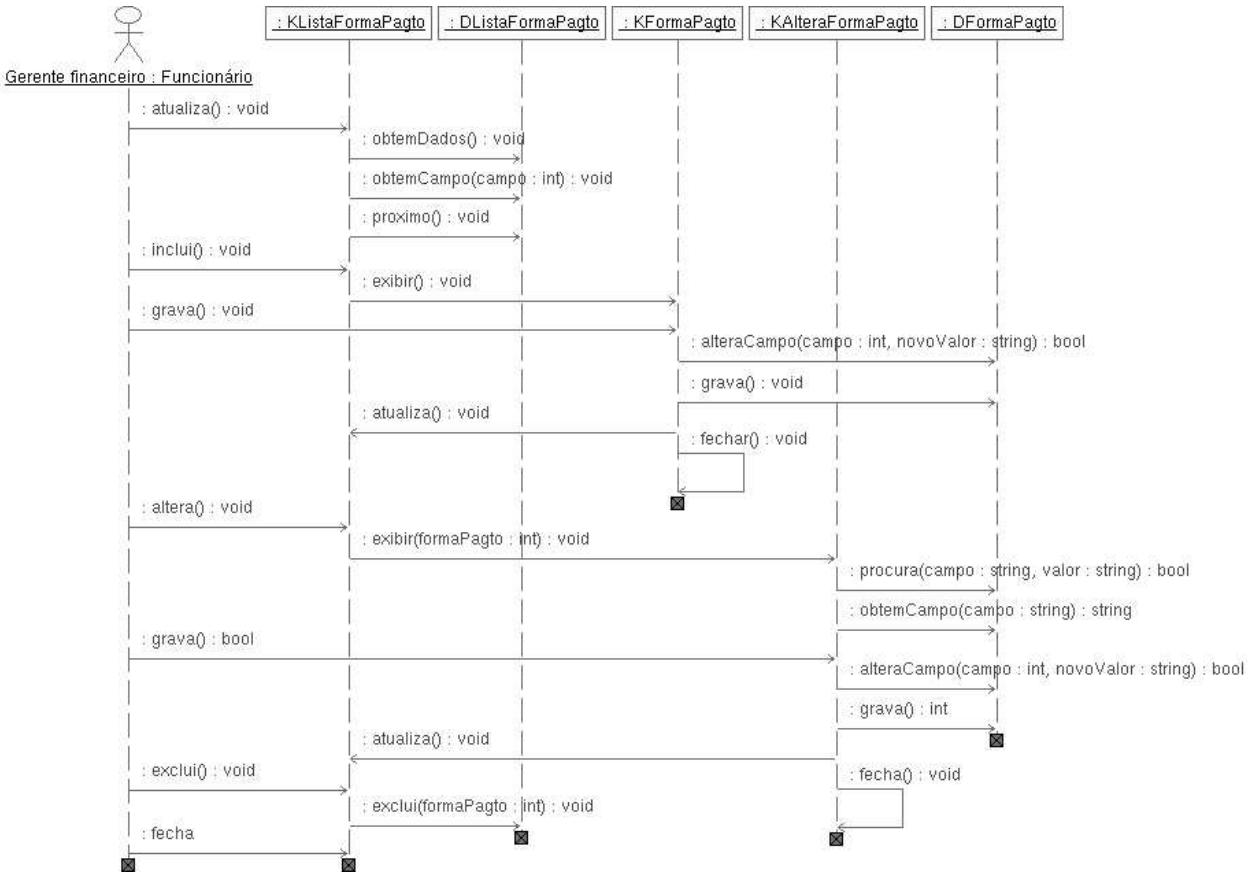


A troca de mensagens entre as classes ação modeladas clicando no botão `Message` na barra de ferramentas e em seguida na classe que solicita a mensagem e depois na classe que vai responder a mensagemn. Depois da linha criada, dê um duplo clique para dar o nome à mensagem criada. e a classe já tiver algum método declarado, o Umbrello os apresenta para você escolher qual deles representa a mensagem. Você pode ainda modelar uma mensagem diferente selecionando a opção `custom operation`[2].

Figura 5-5. Umbrello - Associando a mensagem a um método da classe

Depois de tudo completo, o diagrama de seqüência `Manter cadastro de formas de pagamento` pode ser exportado para uma figura para ser incluído na documemntação do sistema.

Figura 5-6. Diagrama de seqüência - Manter cadastro de formas de pagamento



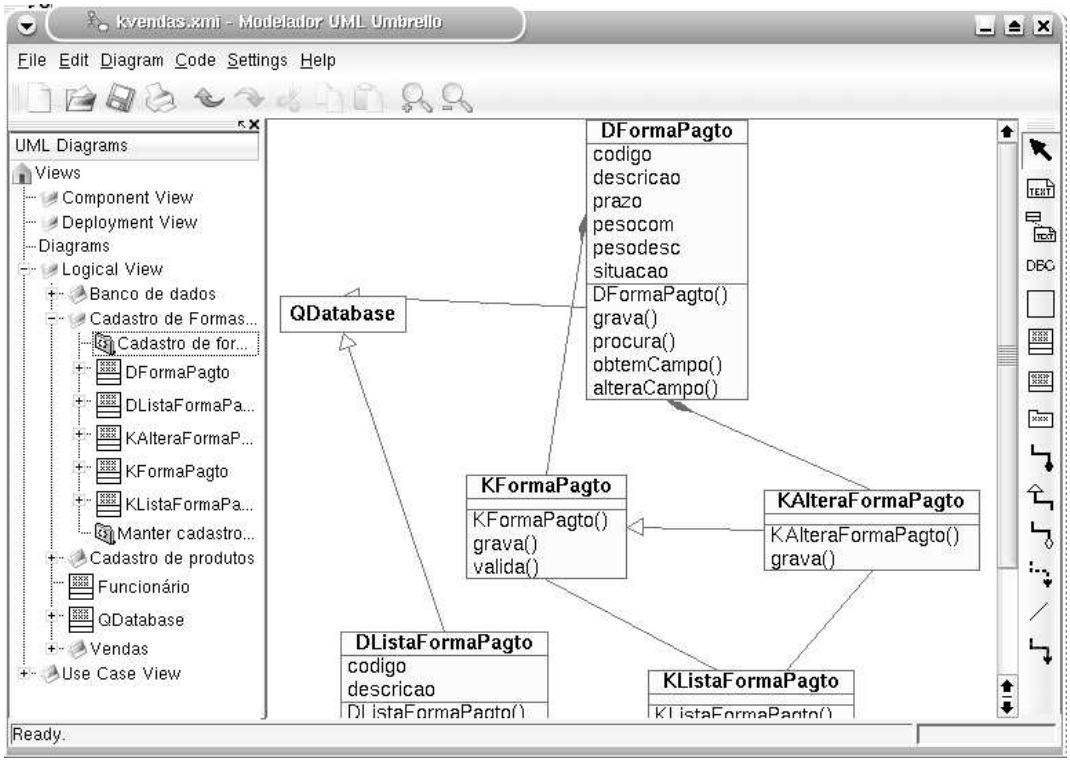
5.3. Diagramas de classes

Durante a elaboração dos diagramas de seqüência, nós definimos as classes que deverão fazer parte de nosso sistema e mostramos as trocas de mensagens entre elas. Porém algumas classes podem ser geradas baseadas em outra (herança) e este tipo de relacionamento não fica explícito no diagrama de seqüência. No diagrama de classes, é mostrado justamente o relacionamento entre as classes e não mais trocas de mensagens.

Para facilitar a visualização destes relacionamentos, geramos um diagrama de classes para cada diagrama de seqüência modelado. Sendo assim vamos criar o diagrama de classe dentro da mesma pasta onde criamos o diagrama de seqüência. Como modelamos apenas o diagrama de seqüência do cadastro de forma de pagamento, vamos mostrar aqui também apenas o diagrama de classes equivalente.

Para criar o diagrama, basta selecionar o menu `Diagram - New - Class Diagram`. O Umbrello então abre um novo diagrama. Como as classes já foram criadas na modelagem do diagrama de seqüência, não vamos criá-la novamente, mas apenas arrastá-la da árvore à esquerda da janela do Umbrello paradentro do diagrama criado. Neste momento podemos também documentar emontar a estrutura das classes.

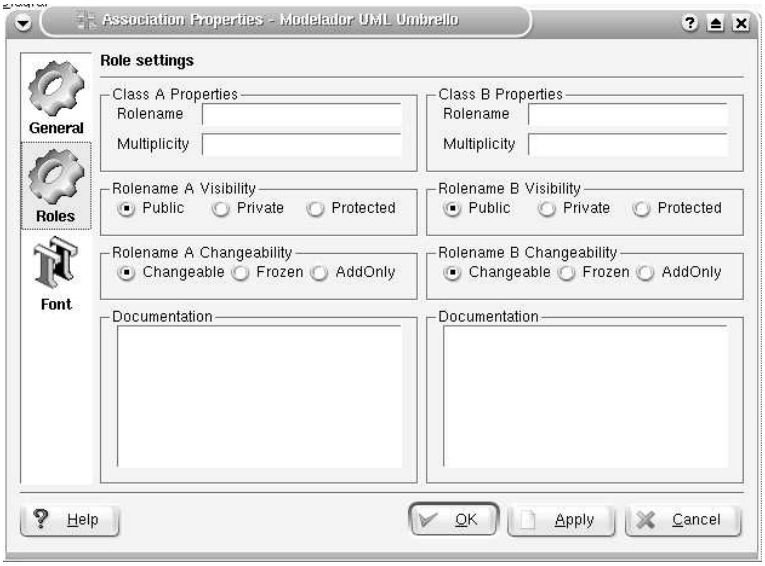
Figura 5-7. Umbrello - Diagrama de classes



Para relacionar duas classes basta selecionar o tipo de relacionamento na barra de ferrametnas à direita e depois clicar nas duas classes que serão relacionadas. Atributos do relacionamento como multiplicidade e papel podem ser alterados clicando-se sobre o relacionamento com o botão direito e, no menu que aparece, selecionar a opção `Properties`[3]. Uma janela então se abre para alterarmos as propriedades do relacionamento.

À esquerda temos algumas opções que quando selecionadas mudam a página de propriedades do relacionamento. A primeira página (*General*) altera o nome do relacionamento e pemite documentar o relacionamento. Na página *Roles* alteramos os atributos relativos a cada uma das classes do relacionamento.

Figura 5-8. Propriedades do relacionamento - página *Roles*



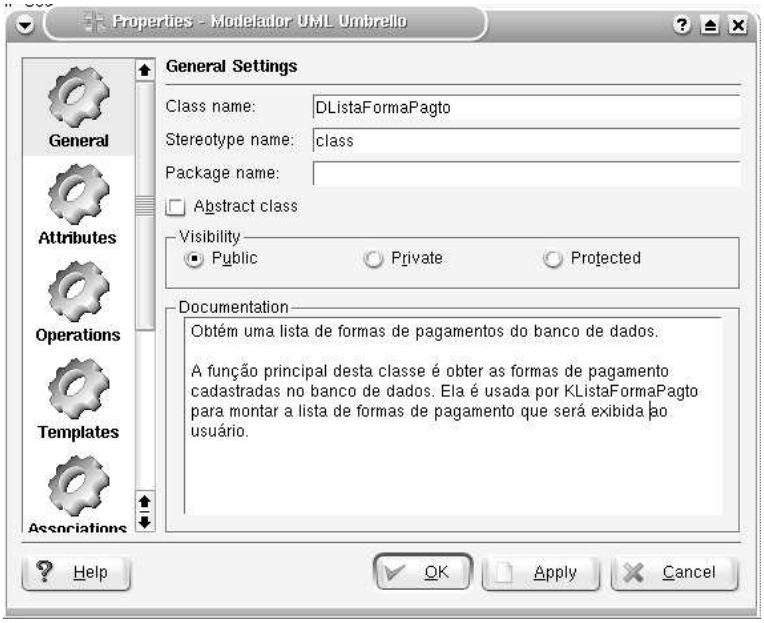
Para documentar as classes, podemos proceder da mesma forma: clicar sobre a classe com o botão direito e depois selecionar a opção *Properties*. O Umbrello então mostra uma janela similar às propriedades do relacinamento, porém com opções diferenciadas. Na página *General*, nomeamos a classe e a documentamos. Na página *Attributes* inserimos os atributos da classe. Na página *Operations* inserimos os métodos da classe. Se os relacionamentos já foram feitos no diagrama, a página *Associations* vai exibi-los.

A definição das classes no Umbrello é muito importante porque baseado nestas informações, ele gera o código fonte contendo o esqueleto da classe. No nosso caso, como vamos usar o C++ na implementação, o Umbrello já vai gerar o s arquivos headers (.h) e os arquivos fonte (.cpp). O Umbrello coloca nos arquivos além do esqueleto da classe, as documentações tanto da classe como de cada elemento da classe que estiver documentado. Esta documentação gerada nos fontes palo Umbrello segue o formato do DOxygen.

Para ilustrar o funcionamento do Umbrello, vamos mostrar como definir a classe DListaFormaPagto. Esta classe é responsável por obter os dados das formas de pagamento do banco de dados para que seja elaborada uma lista que será exibida ao usuário. Vamos então abrir a janela de propriedades desta classe clicando nela com o botão direito e selecionando a opção *Properties*.

A primeira tarefa é documentar a classe, indicando para que ela serve e uma visão geral sobre seu funcionamento. Não é necessário neste momento entrar em detalhes sobre os atributos e métodos porque eles terão sua própria documentação.

Figura 5-9. Umbrello - documentando uma classe



Feita a documentação da classe, vamos agora definir os atributos, bastando para isto selecionar a página *Attributes* e clicar no botão *New Attribute*. Agora é só definir as propriedades do atributo e confirmá-lo. O Umbrello volta a tela anterior com o atributo criado na lista. Para documentá-lo, basta clicar sobre ele e digitar a documentação na área de texto *Documentation*.

Figura 5-10. Umbrello - Criando um novo atributo

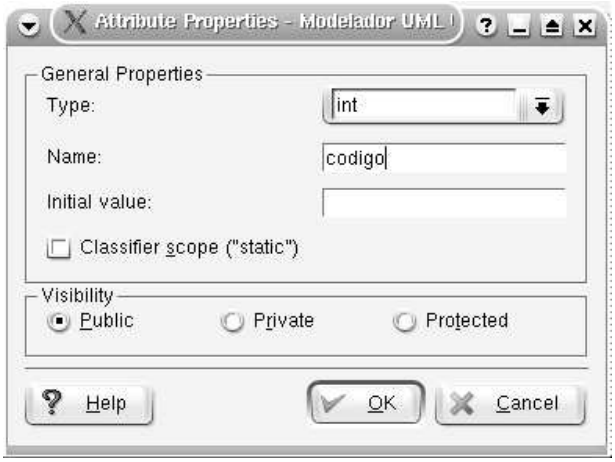
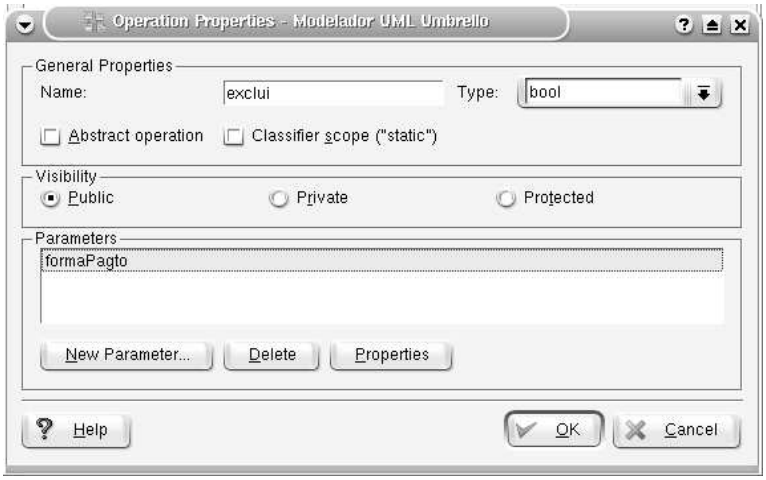


Figura 5-11. Umbrello - documentando um atributo

Depois de incluídos e documentados os atributos, podemos incluir os métodos da classe. Para isto o primeiro passo é selecionar a página *Operations*. Depois clicamos em *New Operation* para criarmos o nosso método. Na tela das propriedades do método, podemos incluir também os parâmetros que o método vai precisar para seu processamento. Para incluir um parâmetro, basta clicar em *New Parameter*. A tela de propriedades do parâmetro é similar à das propriedades de atributo. Ao confirmar os dados do parâmetros, o Umbrello volta às propriedades do método e, ao terminar de definir o método, basta confirmar e o Umbrello volta à tela de definição da classe.

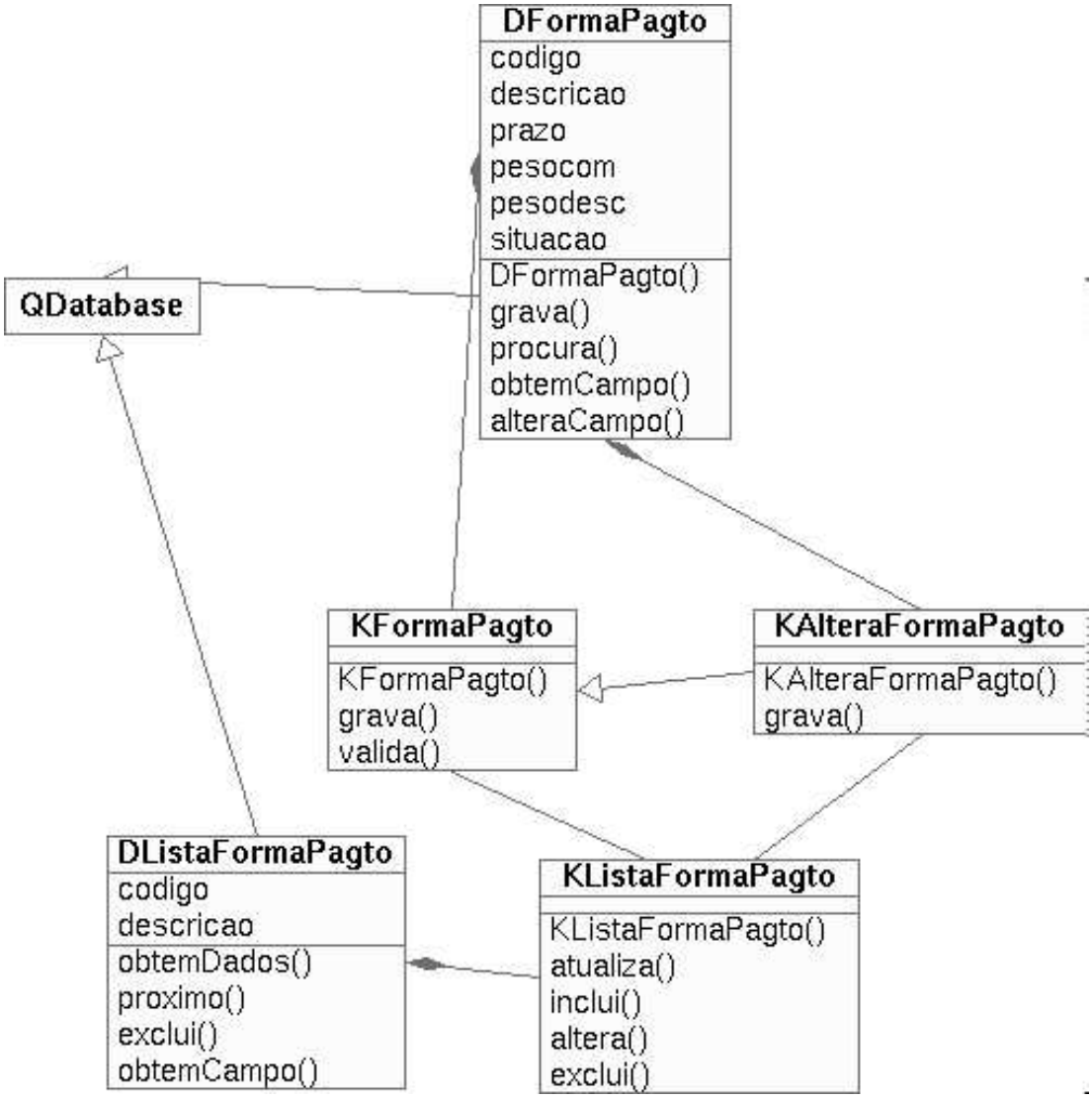
Figura 5-12. Umbrello - definindo um método



Assim como na criação de atributos, depois de onnfirmado um método, o Umbrello acrescenta o método à lista e para documentá-lo, basta clicar sobre ele e preencher o texto *Documentation*.

Depois de definidos todos os atributos e métodos das classes, nosso diagrama de classe ficará como mostrado abaixo. Note que na figura o Umbrello está mostrando a classe em detalhes, o que pode ser alterado conforme o desejo do usuário. Porde-se por exemplo mostrar apenas o nome da classe, ou o nome e relação de atributos. Para alterar o que deve ser mostradom, na janela de propriedades da classe, selecione a página *Display* e selecionar o que você quer que seja exibido no diagrama.

Figura 5-13. Diagrama de classes usado no cadastro de formas de pagamento



5.4. Falhas do Umbrello

Durante a modelagem do sistema, notamos alguns comportamentos "estranhos" do Umbrello. Vamos enumerá-los aqui para evitar surpresas na hora da utilização.

Arrastar e soltar na árvore de elementos

Algumas vezes ao tentar mover elementos entre pastas o Umbrello não faz a movimentação, sendo necessário repetir a operaçã;o. Alguns movimentos realmente não são possíveis, como passar um diagrama de caso de uso de *Use caseview* para *Logical view*, mas estes movimentos restritos são sinalizados pelo próprio Umbrello. O problema é que ele não respeitou movimentos que ele mesmo sinaliza comosendo permitidos.

Associação mensagem - método

No diagrama de seqüência, ao criar uma mensagem, se você seleciona um dos métodos descritos para associar a mensagem, depois de fechar, o Umbrello não se lembra da associação e coloca a operação selecionada como `custom operation`.

Documentação de elementos

Ao documentar alguns elementos, o Umbrello "perde" o que foi digitado no campo *Documentation*. Este problema ocorre principalmente com o último elemento acrescentado à classe (método ou atributo). O problema foi parcialmente contornado, acrescentando um elemento por vez, fechando a janela e salvando o arquivo.

Apesar das falhas, o Umbrello se comportou muito bem durante a modelagem, e tranqüilamente pode ser adotado com satisfação. Um grande diferencial do Umbrello em relação às outras ferramentas analisadas, foi o fato de ele ser mais completo. Além de implementar o diagrama de seqüência, ele também gera código-fonte em várias linguagens.

Notas

1. A versão do Umbrello usada neste trabalho não estava traduzida para o português, por isto o termo em inglês. É possível que o Umbrello já tenha uma versão traduzida
2. custom operation = operaçatilde;o personalizada
3. Properties = Propriedades