UNIVERSIDADE EDUARDO MONDLANE
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE MATEMÁTICA E INFORMÁTICA

TRABALHO DE LICENCIATURA

PROGRAMAÇÃO ORIENTADA A OBJECTOS, UMA SOLUÇÃO PARA A REUTILIZAÇÃO DE CÓDIGO.

AUTOR: JORGE PAULINO BOANE

MAPUTO, JULHO DE 2008

Trabalho de Licenciatura

PROGRAMAÇÃO ORIENTADA A OBJECTOS, UMA SOLUÇÃO PARA A REUTILIZAÇÃO DE CÓDIGO.

"Criação de Uma Biblioteca de Elementos Reutilizáveis para a Gestão de Permissões de Acesso a elementos de um Software."

Autor: Jorge Paulino Boane

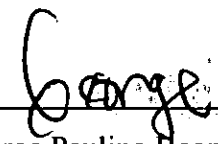
Supervisor: Zeferino Saugene

Maputo, Julho de 2008

DECLARAÇÃO DE HONRA

Eu Jorge Paulino Boane, declaro por minha honra que este trabalho é fruto das minhas próprias investigações e que o mesmo foi elaborado para ser submetido única e exclusivamente na Faculdade de ciências da Universidade Eduardo Mondlane como Trabalho de Licenciatura.

Maputo, Julho de 2008



(Jorge Paulino Boane)

DEDICATÓRIA

Dedico este trabalho a todos aqueles que fazem a minha existência ter sentido neste mundo, especialmente à minha mãe por tudo o que ela tem feito por mim; graças a ela sou hoje o que sou.

AGRADECIMENTOS

Expresso os meus sinceros agradecimentos a todos aqueles que tornaram o meu sonho em realidade, o sonho de iniciar e concluir uma licenciatura, especialmente:

Ao meu supervisor, Dr. Zeferino B. Saugene, por ter se mostrado disponível sempre que fosse necessário, para me dar orientação.

Aos meus queridos amigos da turma que durante a "batalha" me fizeram acreditar que "ela" seria vencida, nomeadamente: Mino, Sifuva, Gomba e Nello;

Aos meus colegas e amigos Márcia e Eurico pelas ideias que eles me deram para enriquecer esta dissertação.

A toda turma *"dmi 2002"*, pelos bons momentos que me proporcionaram durante o decurso do curso;

A minha querida família que cuidou muito bem de mim durante o curso; um *"muito obrigado"* especial a minha querida irmã Alice.

Finalmente expresso o meu agradecimento a aqueles que tem rodeado a minha vida de coisas boas especialmente às minhas queridas Edite, Mirka, MaEli, Epifania e Elcidia.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURA

CASE	*Computer-Aided Software Engineering (Ferramentas de Suporte à Engenharia de Software)*
DNTF	Direcção Nacional de Terras e Florestas
EXI	Engenharia e Comercialização de Sistemas de Informação
HdD	Hospital de Dia
OMT	*Object Modeling Technic (*Técnica de Modelação de Objectos*)*
OO	Orientação à objectos
SPFFB	Serviços Provinciais de Florestas e Fauna Bravia
UML	*Unified Modeling Language (*Linguagem de Modelação Unificada*)*

1. INTRODUÇÃO

Vivemos numa era em que a corrida atrás do tempo é um dos principais desafios impostos às organizações e à sociedade em geral, daí que alguns comparam o tempo ao dinheiro e, na verdade pode assim ser considerado pois, a dinâmica do mercado e do negócio em geral exigem que as actividades sejam feitas em tempo *recorde* pois quanto menos tempo a execução de tarefas levar maior será a possibilidade de ganhar um espaço considerável no mercado. Os clientes estão interessados em ter os resultados (produtos) o mais breve possível e eles optarão por quem prometa (e claro, cumpra) disponibiliza-los num intervalo de tempo relativamente curto.

Esta realidade é partilhada por quase todos os tipos de organizações pois em todos eles existe aquele que chamamos de cliente (o beneficiário) o qual não está interessado em levar tanto tempo à espera dum resultado.

Neste contexto, diversas organizações/empresas se sentem pressionadas a adoptar novos mecanismos de trabalho com vista a satisfazer o cliente cada vez mais exigente em termos de qualidade do produto e o tempo que este tem de esperar pelos mesmos. Esses mecanismos baseiam-se no uso de técnicas, ferramentas, e metodologias que flexibilizam as actividades organizacionais.

Entre as organizações mais afectadas com esta nova dinâmica do negócio encontram-se as que se dedicam ao desenvolvimento de aplicações informatizada (Leite, 2007).

A crescente procura de sistemas de informação automatizados faz com que os responsáveis pelo desenvolvimento de aplicações se vejam sobrecarregados de projectos de desenvolvimento de aplicações os quais possuem restrições muito rigorosas de tempo de execução, isto é, os prazos de entrega dos projectos são muito reduzidos daí que as mesmas têm procurado usar metodologias suficientemente flexíveis para responder a esta e outras exigências do mercado.

Com vista a responder a dinâmica do mercado e satisfazer requisitos tais como, consistência (das aplicações), flexibilidade e baixo custo (em termos de esforços de desenvolvimento, tempo de desenvolvimento, etc.), os que se dedicam ao desenvolvimento de aplicações têm vindo a empregar técnicas de desenvolvimento de aplicações, dentre as quais se podem destacar: Programação funcional, Programação lógica, Programação estruturada, a programação modular, programação orientada a eventos, entre outros métodos de concepção de aplicações. Uma das técnicas que se mostrou ser poderosa para o alcance dos atributos listados anteriormente é a programação orientada a objectos a qual até hoje é uma opção quase inevitável para o desenvolvimento de aplicações de baixo custo, flexíveis e de fácil manutenção (Larman, 1998).

A programação orientada a objectos é uma técnica de escrita de código na qual os dados são manipulados como se fossem elementos de objectos (na maior parte dos casos, objectos do mundo real), isto é, os dados são agrupados de tal modo que espelhem objectos e são manipulados como se fossem comportamentos ou estados desses objectos. (Barbara, 2004).
A programação orientada a objectos, é uma solução para muitos problemas relacionados com o processo de desenvolvimento de aplicações, razão pela qual, as entidades dedicadas ao desenvolvimento de aplicações tem apostado nesta técnica com vista a maximizar os ganhos referentes ao tempo (de desenvolvimento), a manutenção, e sobretudo, a flexibilidade, visto que esta técnica permite que códigos escritos sejam reutilizados em várias aplicações e desta forma poupar tempo e esforço de trabalho. Para além disso, a reutilização garante consistência das aplicações uma vez que as componentes reutilizáveis são testadas e depuradas de forma centralizada (Deitel, 2003).

Este trabalho, descreve os aspectos que envolvem esta técnica de escrita de programas bem como a aplicação da mesma no desenvolvimento de aplicações flexíveis, consistentes, de fácil manutenção, de baixo custo, entre outros atributos desejáveis em um sistema de informação bem concebido.

O trabalho, além de fazer uma descrição de conceitos e elementos relacionados com a técnica de programação orientada a objectos, inclui uma aplicação prática do uso desta técnica.

1.1. Definição do problema

Um dos grandes desafios imposto às organizações que se dedicam ao desenvolvimento de aplicações é a concepção de aplicações em tempo recorde (Mangan, et al., 1999); mas na maioria dos casos o que tem se verificado são atrasos significativos, isto é, os projectos de desenvolvimento levam muito mais tempo do que o planificado.
Vários factores estão por detrás desses incumprimentos a saber:

1 Estimativas de tempos de execução de tarefas mal feitas: no momento da elaboração do plano do projecto o tempo alocado às tarefas individuais tem estado muito deslocados da realidade, isto é, o tempo atribuído a essas tarefas não tem sido suficiente para a sua execução.
2 Deficiência das metodologias de execução de tarefas, o que na maioria dos casos resulta do

incumprimento de algumas regras da organização por parte de alguns membros envolvidos na equipa de desenvolvimento;

3 Falta de acompanhamento rigoroso do plano de execução definido: na maioria dos casos os responsáveis pelo projecto não seguem rigorosamente o plano estabelecido;

4 Falta de recursos humanos qualificados para a execução de tarefas;

Para além destes factores, existe um grande factor que está na origem do problema levantado que é o "desperdício" de práticas já desenvolvidas em outros projectos; isto é, um comportamento/funcionalidade de uma aplicação, pode estar presente em mais de uma aplicação, mas o mesmo é desenvolvido novamente nas próximas aplicações pois os mesmos não são concebidos com previsão de serem reutilizados em projectos futuros. Este factor causa outros problemas, como é o caso de elevados custos de manutenção das aplicações, uma vez que as aplicações são desenvolvidas de forma independente o que implica numa manutenção independente de módulos que tenham comportamentos similares;

O atraso na conclusão de projectos de desenvolvimento em muitos casos traz consequências indesejáveis. Por exemplo, uma organização que deseja obter uma aplicação em muitos casos tem seus planos já traçados (com a aplicação) e a demora na conclusão do projecto de desenvolvimento (por parte da organização responsável pelo desenvolvimento) pode comprometer os planos da organização (beneficiaria da aplicação/dona da aplicação); em alguns casos algumas organizações desistiram de ter uma aplicação devido ao tempo de espera que teriam de suportar até a conclusão de projecto;

Alguns sistemas mesmo depois de concluídos, não respondem aos requisitos dos utilizadores, devido a não observância das práticas de concepção de aplicação por parte dos envolvidos no desenvolvimento o que na maioria dos casos é causado pela falta de recursos humanos qualificados.

1.2. Objectivos

Geral

- Criar um modelo de um sistema de gestão de permissões de acessos a elementos de um software baseado em componentes reutilizáveis concebidos aplicando técnicas de programação orientada à objectos.

Específicos

- Estudar e apresentar os principais conceitos que envolvem as técnicas de reutilização;
- Estudar e apresentar os elementos principais que envolvem as técnicas de programação orientada a objectos e avaliar a sua aplicação na reutilização do código;
- Estudar e analisar o processo de atribuição de privilégios de acesso a uma aplicação;
- Elaborar um modelo orientado à objectos que os generalize.
- Elaborar protótipos de um sistema que utilizem bibliotecas reutilizáveis baseadas na técnica de programação orientada a objectos;
- Avaliar a aplicabilidade do modelo/protótipo em aplicações baseadas na mesma plataforma.

2. METODOLOGIA DO ESTUDO

Para o alcance dos objectivos deste trabalho, foram usados os seguintes recursos (meios e ferramentas:

A revisão de aspectos relacionados com os sistemas de informação em geral, processos de Software, qualidade de software, técnicas de programação, foi baseada na consulta bibliográfica existente;

Foram igualmente estudados conceitos relacionados com programação orientada a objectos e seus componentes/elementos básicos; Conceitos relacionados com a reutilização de componentes. Este estudo, foi baseado na consulta de diferentes fontes de informação (físicos e electrónicos) existentes, como é o caso de livros, manuais, artigos, revistas, etc.

Para o entendimento de Privilégios de acesso à aplicações, foi feita uma análise criteriosa dos elementos que envolvem os níveis de acesso a um sistema de informação computadorizado, foi também feita uma comparação entre os diferentes sistemas de informação com vista a encontrar partes comuns no que diz respeito ao *Controle de níveis de acesso.* Esta análise, para além de basear-se na ideia do autor, fundou-se também em estudos feitos por outros autores;

Foram igualmente usadas ferramentas de suporte para a elaboração de todos os elementos envolvidos no estudo, nomeadamente, editores de textos, aplicações de design, linguagens de programação, sistemas de gestão de bases de dados, entre outras ferramentas que se julgaram necessárias para a concretização dos objectivos do estudo. Essas ferramentas são apresentadas a seguir:

1 O ambiente computacional usado foi o *Windows (versão XP Pro English)* pelo facto de autor estar muito familiarizado com este sistema operativo em relação aos outros existentes no mercado;
2 Para a documentação do estudo: foi usado o *Microsoft Word 2003* devido à flexibilidade que oferece para a edição de textos.

3 Para o desenho de elementos envolvidos no estudo: foi usado *Microsoft Visio 2003* devido a

variedade de diagramas que o mesmo suporta;

4 Para a modelação dos elementos do protótipo, foi usada a metodologia de modelação OMT associado a linguagem de modelação UML por ser uma linguagem gráfica e pelas facilidades que ela fornece para a representação, especificação, construção e documentação de elementos de objectos (Deitel, 2003), além disso, está linguagem é suportada pelo *Microsoft Visio 2003.*

5 Para a concepção do protótipo, foi usada a linguagem de programação JAVA (JAVA 5) por ser uma linguagem orientada a objectos, simples, robusta, portátil e sobretudo gratuito e disponível com muita facilidade na Internet.

6 Para a concepção da base de dados, foi usado o MySQL™ 5.0, por ser robusta, rápida, portátil (MySQL™ Reference, 2003) e sobretudo por possuir uma versão gratuita a qual foi utilizada neste projecto.

3. REUTILIZAÇÃO

O conceito reutilização, é aplicado em quase todas as actividades rotineiras do homem; as pessoas reutilizam conhecimentos, ideais até mesmo componentes criados ou adquiridos na implementação duma solução a um problema específico.

No âmbito de software, a reutilização é definida como sendo a aplicação de componentes concebidas em projectos anteriores em novos projectos (Barbara et. al, 2001) essas componentes, segundo Tavares (2007) podem ser: especificações, arquitectura, desenho, algoritmo, código, interface, casos de teste, documentação, conhecimento, metodologias, entre outras. Resumindo, a reutilização é os mecanismos pelo qual uma solução é aplicada em mais de um problema os quais podem surgir em momentos diferentes mas que sejam similares.

A reutilização, conforme veremos mais tarde, traz benefícios no processo de desenvolvimento de software, nomeadamente: o aumento de níveis de qualidade e produtividade, a redução de custos e tempo de desenvolvimento (Tavares, 2007), daí que muitos desenvolvidores têm vindo a empregar padrões que lhes garantam a reutilização.

O conceito reutilização surgiu na engenharia de software desde finais da década 60; ela foi considerada como uma solução para a crise de software.

A reutilização pode ser introduzida em todas as fases do ciclo de desenvolvimento de software desde a análise até a implementação.

As diferentes formas de reutilização que podem ser integradas em cada fase do ciclo de vida do software estão alistadas a seguir:

- *Análise de domínio*: A reutilização a este nível é feita através do uso de conhecimentos/experiência adquiridos em outros projectos. Além disso segundo Loucopoulos & Karakostas (1995), podem ainda serem reutilizados (ou concebidos para a reutilização) literaturas especializadas.
- *Engenharia de requisitos*: em engenharia de software os requisitos podem ser reutilizados; para isso, segundo Tavares (2007), é necessário que os mesmos sejam concebidos com um nível de abstracção muito maior. O que significa que, a análise e a gestão dos requisitos deve ser feita tendo em conta não apenas um caso restrito mas casos mais abrangentes. Existem muitas situações onde os requisitos podem ser reutilizados, por exemplo: os relativos ao *domínio de aplicação*, os

quais não dizem respeito às funcionalidades do sistema mas sim especificam objectos, regras e limitações de domínio. Também os *requisitos que especificam a interface do utilizador* podem ser reutilizados em todos os sistemas duma organização. Este tipo de reutilização, padroniza elementos da aparecia e minimiza as dificuldades de utilizadores no caso de passarem dum sistema para outro. Por fim, temos os requisitos referentes às *regras ou politicas da organização*, tais como, politicas de segurança, os quais podem ser reutilizados em todos os sistemas da organização.

- *Desenho:* Nesta fase podem ser reutilizados os padrões de desenho;
- *Implementação:* Podem ser reutilizadas livrarias e classes de objectos;
- *Testes:* Na fase de teste pode ser reutilizados os casos de teste.

3.1. Componente reutilizável

"Façamos como os engenheiros de hardware! Não está certo começar cada novo desenvolvimento sempre do zero. Devia haver catálogos de software, tal como há catálogos de circuitos integrados: para construir um novo sistema, começávamos por encomendar esses componentes e depois combinávamo-las, em vez de reinventar a roda. Dessa maneira, escreveríamos menos software, mas talvez escrevêssemos melhor. Não desapareciam então os problemas de que toda a gente se queixa – custos elevados, atrasos, falta de fiabilidade? Porque é que não é assim?" ("Mass-Produced Software Components", Doug McIlroy, 1968.)*

A citação acima mostra o objectivo principal da reutilização; em engenharia de software, esta ciência tem como objectivo imitar a reutilização de componentes de forma idêntica como esta é feita em outras engenharias como a engenharia de hardware.

Para que essa reutilização seja aplicada, é necessário que os componentes sejam concebidos tendo em mente que os mesmos poderão ser integrados em projectos futuros.

Em engenharia de software, um componente reutilizável é qualquer coisa, tangível ou não, o qual pode ser integrado em outros projectos de desenvolvimento de software diferentes do original (projecto para o qual foi concebida inicialmente). Esse componente pode ser (Tavares, 2007): Especificação de Software, arquitectura, desenho, algoritmos, código, interface, casos de teste, documentação, conhecimento, metodologia, práticas, etc.

Para que um componente seja reutilizado deve observar alguns requisitos ou características que as

* Citado por Tavares(2007).

quais o tornarão reutilizável; essas características são apresentadas na subsecção seguinte.

3.2. Características dum componente reutilizável

Para que um componente seja reutilizável são necessários que sejam observados alguns atributos, nomeadamente (Tavares, 2007): Abstracção/Generalidade, Simplicidade, Independência e utilidade.

3.2.1. Componente Genérico/Abstracto

Um componente genérico ou abstracto, é um componente que é concebido para solucionar um problema específico mas tendo em conta que o mesmo poderá solucionar outros problemas similares; sendo assim o componente deve ser concebido separando atributos relevantes dos que não são (Barbara, 2004). O componente deve ser visto como um agrupamento de várias características comuns a vários problemas idênticos. O programador deve ter capacidade de se abstrair de modo a encontrar uma solução genérica para problemas similares.

3.2.2. Simplicidade

A simplicidade é um atributo muito importante para que um componente seja reutilizável. Os outros (até mesmo quem concebeu a componente) só poderão reutilizar o componente se compreenderem a sua funcionalidade através da interface fornecida pelo mesmo. Quanto mais simples for a interface mais simples se torna a sua compreensão e utilização (Tavares, 2007).

3.2.3. Independência

A independência dum componente em relação à outros é um dos atributos fortes para que o mesmo seja reutilizável. Quanto mais dependente de outros componentes maior será o nível de complexidade para que o mesmo seja reutilizado, uma vez que para reutilizá-lo seria necessário reinvocar os componentes dos quais depende. Sendo assim, ao conceber um componente com perspectiva de ser reutilizado, é necessário que o mesmo seja pouco dependente de outros (Barbara, 2004).

3.2.4. Utilidade

A utilidade é o atributo chave para a reutilização. Se um componente não é útil, obviamente ninguém se interessará em reutilizá-lo, alias, não faria nenhum sentido reutilizar um componente sem nenhuma utilidade. Sendo assim, a reutilização só será válida se o componente for útil.

3.3. Momentos do processo de desenvolvimento com reutilização

Em um processo de desenvolvimento com reutilização encontramos dois grandes momentos ou dois grandes processos envolvidos: *development-for-reuse* que é o processo de construção dos elementos reutilizáveis e, *development-with-reuse* que é o processo de reutilização propriamente dita dos elementos reutilizáveis em novos projectos ou na manutenção de sistemas já existentes.

3.3.1. Development-for-reuse

A concepção de componentes reutilizáveis é o momento mais crítico do processo de desenvolvimento com reutilização; Este processo é conhecido como *development-for-reuse* (desenvolvimento para a reutilização); A Figura 1 mostra as diferentes actividades envolvidas no processo de desenvolvimento para a reutilização.

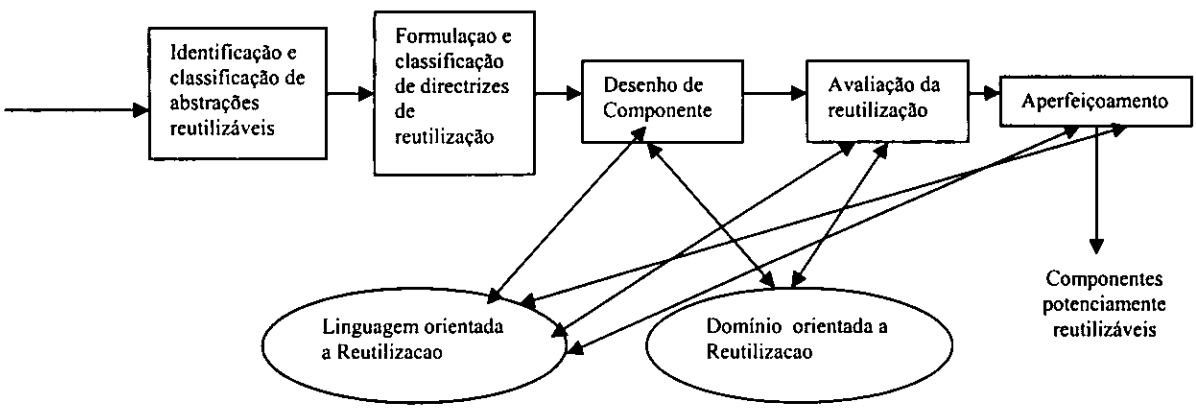


Figura 1: Processo de Desenvolvimento para a reutilização (Development-for-reuse)

O processo de Desenvolvimento para a reutilização começa com a identificação e classificação de abstracções reutilizáveis; Aqui, são identificados e classificados componentes, *frameworks,*

arquitecturas e utilitários que possuem objectivos comuns e que possam trazer um alto nível de retorno.

Na formulação e classificação de directrizes são produzidos directrizes de reutilização e os mesmos são classificados e integrados no domínio orientando à reutilização, directrizes de desenho, directrizes arquitecturais de desenho, e linguagem orientada a reutilização.

O desenho de componentes, procura garantir que os engenheiros (de software) se familiarizem com as directrizes de reutilização;

A avaliação da reutilização permite que outros engenheiros (de software) façam a inspecção do processo de reutilização;

O aperfeiçoamento faz a modificação de componentes baseando-se no relatório de avaliação da reutilização;

3.3.2. Developent-with-reuse

O processo de *Development-with-reuse (desenvolvimento* com reutilização*)*, consiste na reutilização de componentes concebidos durante o processo de desenvolvimento para a reutilização (visto na secção anterior); este processo começa com a *pesquisa* do(s) componente(s) na livraria de componentes reutilizáveis; o processo de pesquisa tem como finalidade encontrar componentes que se aproximam à solução dos problemas em causa; após a selecção dos componentes segue a selecção dos componentes mais aceitáveis para a resolução do problema; esse processo é conhecido como *compreensão.*

Algumas vezes o componente não reflectirá perfeitamente a solução do problema, havendo necessidade de modificação do mesmo de modo que possa reflectir a solução do problema em causa; a esse processo chama-se *adaptação* e, todos os componentes adaptados ou modificados devem ser reintegrados na livraria de componentes reutilizáveis de modo que os mesmos estejam disponíveis para futura reutilização.

3.4. Tipos de Reutilização

3.4.1. Reutilização Ad-Hoc

A reutilização Ad-Hoc é tida como reutilização informal, baseada nas necessidades individuais dos

projectistas (Prieto-Díaz, 1996). Durante o ciclo de desenvolvimento do projecto o projectista pode se lembrar duma solução que empregou para um determinado problema o qual assemelha-se ao problema corrente, neste caso ele pára e faz uma consulta nas livrarias de sistemas desenvolvidos anteriormente de modo a encontrar essa solução e integrá-la no novo sistema. Este tipo de reutilização, segundo Prieto-Díaz (1996), traz poucos benefícios, uma vez que dificilmente se encontrará concordância entre os requisitos actuais e a solução concebida anteriormente pois a mesma não foi projectada tendo em conta futura reutilização.

3.4.2. Reutilização sistemática

A reutilização sistemática, diferentemente da *Ad-Hoc,* faz parte de um processo planeado e bem documentado; Ela, embora tenha surgido nos princípios da década 80, só se popularizou recentemente na comunidade de projectistas.

A reutilização sistemática caracteriza-se por uma arquitectura que captura características essenciais dos sistemas a serem desenvolvidos e suas possíveis variações. Alem disso, a reutilização sistemática, exige uma continua avaliação baseada em novas métricas de reutilização para garantir melhoria contínua (Prieto-Díaz, 1996).

3.5. Vantagens da Reutilização

O objectivo principal da reutilização na engenharia de software é minimizar esforço na execução de tarefas através da aplicação de soluções para problemas anteriores em novos problemas. Se os conceitos da reutilização forem bem observados, pode se obter ganhos consideráveis no referente à:

- *Qualidade*: Uma vez que os mesmos componentes são aplicados em mais do que uma solução, com o tempo, e devido a vários testes e melhoramentos que eles vão sofrendo, estarão menos propensos a erros e alcançarão uma qualidade maior e segurança.
- *Redução de Custos*: A construção dum componente tem seus custos, e a reutilização de um componente poupa tempo e outros recursos;
- *Maior Produtividade*: Uma vez que em projecto de desenvolvimento com reutilização algumas partes do sistema serão extraídas de partes de sistemas já desenvolvidos os projectos levam menos tempo e se consegue abranger muitos projectos em pouco tempo e deste modo alcança-se maior produtividade.

3.6. Problemas da Reutilização

Embora a reutilização seja solução para muitos problemas referentes a projectos de desenvolvimento de software, ela possui alguns problemas; além disso, existem obstáculos que a tornam difícil.

O custo de manutenção de livrarias é um dos principais obstáculos da reutilização; isto deve-se ao facto de as técnicas actuais de classificação, catalogação e pesquisa serem imaturas (Tavares, 2007);

Outro problema está relacionado ao facto de a Maioria das ferramentas CASE não suportar a reutilização.

Segundo Tavares (2007), existe também o problema do factor humano; alguns engenheiros simplesmente preferem refazer um componente em vez de reutilizá-la.

4. PROGRAMAÇÃO ORIENTADA A OBJECTOS

"A programação orientada à objectos é um dos meios pelos quais se podem alcançar os atributos para a reutilização de componentes de um Software";

A programação orientada a objectos é a técnica de escrita de programas baseada na forma como se encara o mundo que nos rodeia (Pillay, 2007), isto é, em programação orientada a objectos, os problemas são modelados[†] de modo a encontrar uma solução que reflicta um comportamento análogo ao comportamento de objectos ou ideias da natureza.

Segundo Pillay (2007), em programação orientada a objectos, o software é visto como conjunto de objectos que cooperam entre si através do envio de mensagem para resolver algum problema. Ele, refere-se a programação orientada a objectos como sendo, conjunto de ferramentas e métodos que permitem aos programadores de software construir softwares seguros, amigáveis, bem documentados, reutilizáveis e que satisfazem os requisitos dos utilizadores.

Para Webopedia (2007), a programação orientada a objectos, é o tipo de programação na qual são definidos não apenas os tipos de dados da estrutura de dados mas também o tipo de operações aplicáveis sobre essa estrutura. Desta forma a estrutura de dados se torna *objecto* que inclui dados e funções; adicionalmente, podem ser criados relacionamentos entre um objecto e outro.

Em Wikipédia (2007), a programação orientada a objectos é definida como um paradigma de análise, desenho e programação de sistemas de *software* baseado na composição e interacção entre diversas unidades de software chamadas de objectos

Resumindo, pode se dizer que programação orientada a objectos é uma técnica de análise, desenho e escrita de código na qual os dados são agrupados (de acordo com o relacionamento que exista entre eles) em módulos específicos considerados objectos e as operações mais relevantes sobre esses dados são definidos dentro desses módulos. Esses módulos são vistos como objectos em analogia com os objectos do mundo real uma vez que eles possuem características (dados), comportamentos e acções (funções executadas dentro dos módulos) e além disso, eles podem

comunicar com outros (envio de mensagem e recepção de respostas de/e para outros módulos).

4.1. Objecto e Classe

O objecto é o elemento principal da programação orientada à objectos pois, em programação orientada a objectos, "tudo gira em volta" deste elemento. Tal como acontece com os objectos do mundo real, o que se considera objecto em programação orientada a objectos tem *características, estado* e *comportamento*; as características e estados, correspondem aos dados (agrupados) e o comportamento aos métodos (procedimentos e funções) sobre os dados. O módulo no qual são agrupados estes elementos (dados e métodos) chama-se *classe* e ela descreve as características e comportamentos do(s) objecto(s) a que se refere.

Uma classe pode ser definida como representação das variáveis e métodos comuns a todos os objectos de um determinado tipo (Pillay, 2007), isto é, a classe, define as características e comportamentos de todos os objectos que pertençam a ela.

Em *Unified Modeling Language* (UML)[‡] uma classe pode ser representada graficamente como mostrado na figura 2.

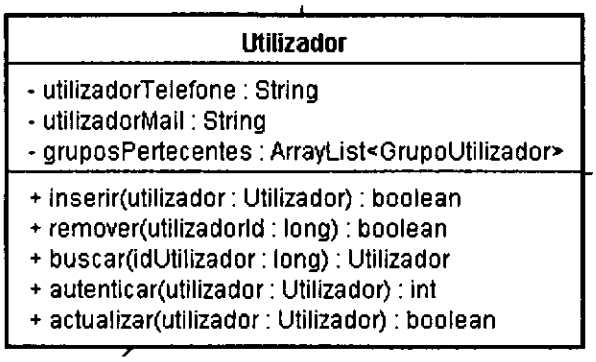


Figura 2 ***Representação duma classe em UML***

Da figura podemos identificar alguns elementos principais duma classe

Nome da classe – é o identificador (nome) da classe; é através dele que se faz referência a classe. Na figura acima o nome da classe é **Utilizador**;

Atributos – são as características da classe (ou dos objectos que serão gerados por ela); os atributos descrevem as propriedades da classe. No exemplo da figura 2 temos três atributos, nomeadamente: *utilizadorTelefone, utilizadorMail* e *gruposPertecentes;*

[†] Modelação, é o mecanismo de estudar algo através de representações simplificadas do mesmo (Pillay, et. al, 1999).

[‡] UML é uma linguagem gráfica de modelação de dados que disponibiliza uma sintaxe para a descrição e apresentação dos elementos de um software.

A definição dum atributo(na classe) possui um *modificador de visibilidade* o qual pode ser cada um destes tipos:

- *Publico (denotado por "+" em UML):* neste caso o atributo pode ser acedido e manipulado por qualquer outro objecto;
- *Privado (denotado por "–" em UML): o atributo definido com este modificador, só pode ser acedido pelo objecto no qual foi definido.*
- *Protegido (denotado por "#" em UML): neste caso, o atributo pode ser acedido pelos objectos "filhos" do objecto no qual o atributo foi definido. (O conceito de "objecto filho" é descrito na secção 4.3);*

Métodos – Os métodos definem o comportamento da classe (ou dos objectos que serão gerados por ela); são eles que manipulam os atributos da classe. A semelhança dos atributos, os métodos possuem os modificadores de visibilidade. No exemplo apresentado na figura 2 temos cinco métodos públicos nomeadamente: *inserir, remover, buscar, autenticar e actualizar;*
Como se pode observar da figura, os métodos possuem parâmetros. Um parâmetro é meio pelo qual a classe se comunica com os objectos do sistema.

Uma classe pode ser considerada como um ***molde*** (ou uma máquina) através do qual são gerados objectos com características comuns; por exemplo, a classe Utilizador pode ser considerada como um molde para gerar objectos do tipo Utilizador, os quais terão mesmas descrições e comportamentos;
A classe/objecto descreve o comportamento de entidades[§] dentro de um sistema (Ariadne, 2001). Isto significa que em programação orientada a objectos, tudo o que é entidade será definido como objecto e, os seus estados e comportamentos serão definidos dentro desse objecto (classe).
Esta abordagem traz alguns benefícios em relação a outras abordagens, uma vez que os métodos que manipulam os dados ficam embutidos dentro do módulo "dono" dos dados o qual chamamos de classe; isto significa que apenas os métodos da classe (objecto) é que podem manipular os dados o que garante maior consistência e integridade de dados. Além disso, em caso de alteração da implementação dos métodos, os efeitos sobre os outros módulos (classe/objectos) serão mínimos, uma vez que a comunicação entre uma classe e outras não é feita directamente através dos dados mas sim através de troca de mensagens.
Outro beneficio considerável é que o mesmo módulo (classe) pode possuir várias instâncias num

dado momento durante a execução da aplicação; e estes objectos terão características e comportamentos similares, uma vez que todos são gerados pela mesma classe.
Uma *instância* duma classe é um objecto concreto ou um valor concreto em analogia aos tipos de dados concretos; por exemplo, para a classe Utilizador (responsável pela geração de objectos do tipo Utilizador) um utilizador concreto, por exemplo *Jorge Paulino* seria uma instância da classe Utilizador; cada objecto é responsável pela manipulação dos seus próprios dados e os objectos afora não podem "mexer" directamente nesses dados.
Isto traz vantagem, uma vez que o objecto dono "sabe melhor" manipular os seus dados e em caso de mudanças na forma de manipulação dos dados, os efeitos em relação aos objectos afora serão mínimos.

Uma *classe* é algo estático expresso em forma de texto no código de uma aplicação, ao passo que *objecto* é algo dinâmico que é criado em tempo de execução da aplicação (Ariadne, 2001); o objecto ocupa um espaço de memória do computador no qual armazena-se o estado do objecto (ou valor) e o conjunto de operações aplicáveis aos dados (atributos) do objecto.

Para que sejam alcançadas as vantagens que advêm do uso de programação orientada a objectos, é necessário que se observem algumas regras definidas neste paradigma; uma das regras fundamentais é o encapsulamento de dados

4.1.1. Encapsulamento

Qualquer objecto possui um *núcleo* o qual é composto de informações exclusivas ao objecto; estas informações basicamente são variáveis do objecto e métodos exclusivos a ele. Ao acto de ocultar ou proteger estas informações de modo a não serem manipulados por outros objectos para além do objecto "dono" chama-se *Encapsulamento*.
Este conceito garante que a estrutura de sistemas seja mais robusta e minimiza os custos de manutenção (Pillay, 2007) uma vez que o objecto não comunica directamente com os outros através dos seus dados (pois estes estão "ocultos") mas sim através de envio de mensagens e no caso de mudança da forma como estes dados são manipulados apenas o módulo (classes/objecto) "dono" dos dados é que sofrerá alterações e as alterações nos outros módulos ou não existirão ou serão

[1] Uma entidade, é definida como algo, real ou abstracto, sobre o qual se pretende capturar e armazenar dados(Whitten, Jeffrey et. al, 1994)

mínimas.

A Figura 3 representa a forma como os objectos se comunicam tendo em conta o conceito de encapsulamento

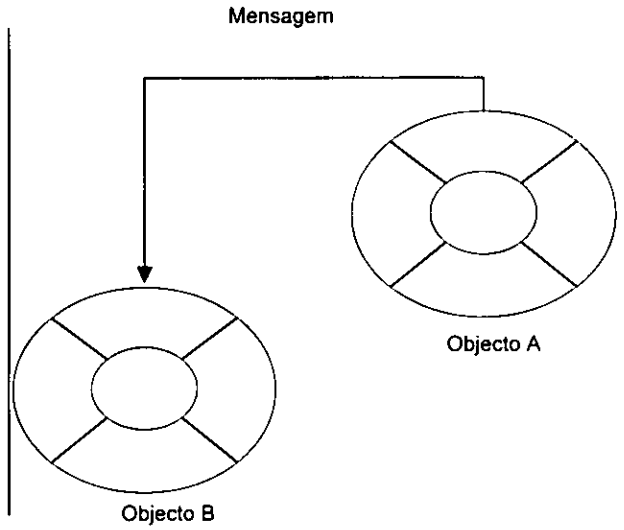


Figura 3: Comunicação entre objectos em um sistema que aplica o conceito de Encapsulamento. (adaptado de: Pillay, 2007)

Como se pode ver na figura, o *Objecto A* não tem acesso directo aos dados (interior do circulo interno) do *Objecto B* e para se comunicar com este usa mensagens. A área pintada representa uma interface através da qual o objecto comunica-se com os restantes objectos.

O encapsulamento é conseguido usando *modificadores de visibilidade* (vistos na secção 4.2) privativos o que torna impossível o acesso aos dados definidos como privados pelos módulos (classes/objectos) externos ao módulo "dono" dos dados.

O conceito de encapsulamento, sugere que o objecto seja manipulado apenas por métodos definidos como públicos e estes métodos apenas devem fornecer a interface de comunicação e não os detalhes de implementação. O que significa que os objectos que se comunicam com este objecto apenas precisam de fornecer os dados de entrada e receber as respostas e não precisam saber como os dados são processados.

4.2. Herança

Herança é um dos conceitos fundamentais da programação orientada à objectos; ela é definida como mecanismo pelo qual uma classe herda (uma parte ou todos) os atributos e funcionalidades da classe

da qual descende (Pillay, 2007), isto é, se uma classe descende doutra, ela possuirá os atributos e funcionalidades da classe a que descende. Esta última pode ser considerada filha (descendente) e a primeira Mãe (ancestral). Neste tipo de relacionamento, diz-se que a classe filha estende a classe Mãe.

O mecanismo de herança, aumenta os ganhos da programação orientada à objectos, uma vez que permite que classes escritas sejam reutilizadas sem necessidades de serem "remexidas".

A classe filha poderá acrescentar (a ela mesma) atributos e funcionalidades, os quais não foram contemplados na classe Mãe; além disso, a classe filha poderá reescrever algumas funcionalidades da classe Mãe, isto é, implementar novamente funcionalidades definidas na classe Mãe.

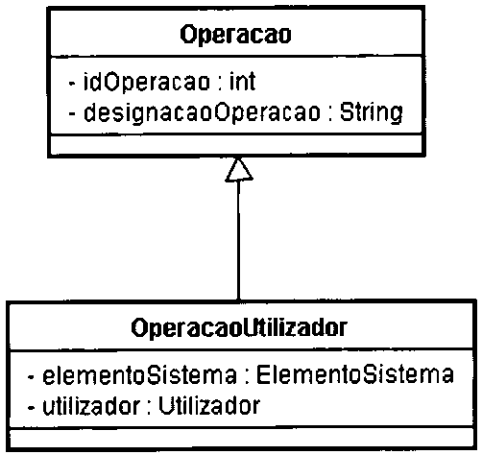


Figura 4: Representação de Herança entre duas classes; a classe OperacaoUtilizador Herda da Classe Operacao.

Na Figura 4, pode ser observado um relacionamento de Herança entre duas classes, a classe *Operacao* e a classe *OperacaoUtilizador*; A classe *OperacaoUtilizador* estende (descende) a classe *Operacao*; diz-se que a classe *OperacaoUtilizador* é sub classe da classe *Operador* e esta última é super classe da Classe *OperacaoUtilizador*.

Uma classe pode ser estendida por mais de uma classe, isto é, várias classes podem descender da mesma classe e cada uma das classes descendentes (ou filhas) irá igualmente herdar as características e comportamentos da classe mãe (Super classe)

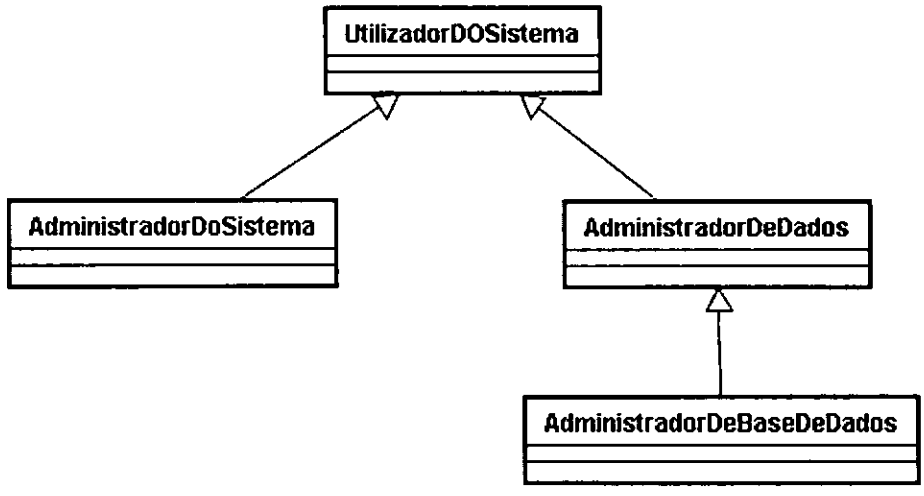


Figura 5: Múltipla Herança.

Da Figura 5 pode se notar que a classe *AdministradorDoSistema* e *AdministradoDeDados* estendem a classe *UtilizadorDoSistema* e desta forma herdam as características (variáveis) e comportamentos (funcionalidades/métodos) dela.

A classe *AdministradorDeBaseDeDados* é filha da classe *AdministradorDeDados* o que implica que ela herdará desta as suas características e funcionalidades e visto que *AdministradorDeDados* herda de *UtilizadorDoSistema*, *AdministradorDeBaseDeDados* também irá herdar de *UtilizadorDoSistema*, isto é, as características de *UtilizadorDeSistema* serão herdadas também pelo *AdministradorDeBaseDeDados*.

4.2.1. Tipos de Herança

Dependendo das regras definidas na forma como a classe filha se relaciona com a classe mãe, podem ser definidos os seguintes tipos de herança:

- *Extensão* – Neste tipo de Herança, a subclasse (classe filha) herda os atributos e funcionalidades da super classe podendo acrescentar a ela mesma novos atributos e funcionalidades. A super classe é concebida sem ter em conta as subclasses, isto é, a super classe, implementa todas as suas funcionalidades à sua maneira;
- *Especificação*: Neste tipo de Herança, a sub classe herda apenas a especificação da super classe; isto é, a super classe apenas define o que a subclasse deve fazer mas ela não implementa essas funcionalidades; diz-se que a super classe fornece uma interface à

subclasse.

- *Combinação de Extensão e Especificação*: Neste tipo de Herança, a super classe, especifica algumas funcionalidades para a classe filha mas também implementa algumas as quais podem ser herdadas pela subclasse.

4.2.2. Regras de Herança

O uso de herança, traz muitos benefícios, mas se este conceito for mal implementado poderá trazer problemas, como é o caso de problemas de manutenção (Ariadne, 2001). Para evitar esses problemas é necessário que sejam observadas algumas regras de Herança entre as classes, a saber (Ariadne, 2001):

- *A Regra de 100%:* Esta regra sugere que se uma classe estende outra, deve aplicar todos os atributos e funcionalidades oferecidas pela super classe, isto é, todos os atributos e funcionalidades definidas na super classe devem ser aplicáveis à sub classe.
- *A Regra "é do tipo de"*: Esta regra é uma simples forma de testar se a Herança é válida; ela é expressa em simples palavras "A classe *B* é do tipo *A*". Se esta frase não fizer sentido é sinal de a Herança entre *B* e *A* é desnecessária e poderá trazer problema de manutenção. Por exemplo, na Herança *Veículo-Carro*(Carro estende veículo), a frase "O carro é do tipo veículo" faz sentido; mas na Herança *Animal-Carro*(Carro estende Animal) a frase "Carro é do tipo Animal" não faz sentido, logo esta herança não é válida.

4.3. Polimorfismo

A programação orientada a objectos permite-nos reutilizar códigos escritos anteriormente através da invocação de classes escritas ou através do conceito de herança. No conceito de Herança, as subclasses (duma classe especifica) podem reescrever métodos definidos (e implementados) na super classe, e cada uma das classes pode implementar estes métodos à "sua maneira". Em momento de execução, caso a sub classe invoque um método da super classe o qual faz referencia a algum método reescrito, é da responsabilidade do compilador determinar qual método deverá ser invocado e o método invocado será o reescrito na classe corrente; a este mecanismo define-se polimorfismo, onde não se sabe de antemão que método será executado em determinadas situações

e essa decisão é feita pelo compilador em momento de execução dependendo do tipo corrente de dados envolvido. Só existe polimorfismo se o método reescrito tiver exactamente mesmos parâmetros de entrada que os da super classe;

O polimorfismo é um importantíssimo mecanismo pois permite que sejam definidas funcionalidades a um alto nível e as mesmas sejam implementadas de várias formas em classes derivadas.

4.3.1. Classes e Métodos Abstractos

O conceito de polimorfismo garante que métodos escritos em super classes funcionem perfeitamente nas subclasses sem necessidade de serem "remexidos" directamente da classe mãe. Graças a este conceito, estes métodos podem ser reescritos (nas subclasses) e serem invocados como se eles estivessem na super classe.

Para tornar o relacionamento entre super classes e subclasses mais simples e transparente, existe o conceito chamado *Classe Abstracta.* Uma vez que se detecte que um determinado método terá implementações distintas nas suas subclasses, o mesmo é apenas definido (sem implementação) a assume-se que a implementação será feita nas classes de nível abaixo. Desta forma, todas as subclasses imediatas desta classe deverão obrigatoriamente implementar esta funcionalidade. A esta funcionalidade ou método definido na classe mas que será implementada por suas subclasses chama-se método abstracto e, a classe na qual se define este método chama-se classe abstracta.

A Figura 6 representa a notação UML para classe e método abstracto.

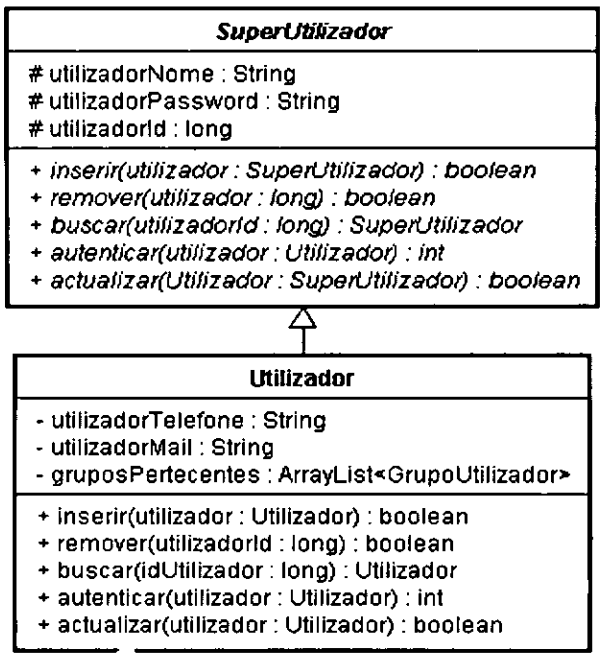


Figura 6: Notação UML para classe e método Abstracto.

A *classe SuperUtilizador* é uma classe Abstracta, isto é, define funcionalidades que deverão ser implementadas pelas suas subclasses; em UML este tipo de classes é representado em *Itálico* (conforme pode ser observado na figura - *Classe SuperUtilizaodr);* O *autenticar()* (Na classe *SuperUtilizador*), representa uma funcionalidade definida na classe "*SuperUtilizador*" mas que será implementada nas suas subclasses. Na figura, a Classes *Utilizador* estende a classe *SuperUtilizador* e obrigatoriamente implementa o todos os métodos definido na classe *SuperUtilizador.*

5. TÉCNICAS DE MODELAÇÃO DE OBJECTOS

Existem várias técnicas de modelação de objectos, mas para este trabalho interessa apenas descrever aquela que foi usada para a modelação do sistema objecto de estudo. Para a concepção do caso de estudo deste trabalho foi usado a metodologia de desenvolvimento orientada a objectos *Object Modeling Technic(OMT)* a qual é descrita neste capítulo.

5.1. A metodologia OMT

OMT é uma metodologia para a modelação de sistemas baseados na tecnologia OO (Orientação a objectos); esta metodologia foi sugerida por James Rumbaugh, Michael Blaha, Willian Premerlani, Frederick Eddy e Willian Lorensen em 1991.

A metodologia OMT divide o ciclo de desenvolvimento de sistemas em três fases, a saber: a fase de análise, projecto de sistema e projecto de objectos.

A fase de análise tem como objectivo a compreensão do sistema e seu domínio, bem com a sua modelação;

A fase de projecto, procura organizar o sistema em subsistemas adicionados ao domínio da solução dos problemas identificados;

A última fase, tem como finalidade a elaboração de modelos de análise com objectivo de produzir os elementos do projecto prático; esta fase inclui também a refinação e optimização dos mesmos modelos.

A metodologia OMT segundo RUMBAUGH at al(1991) possui as seguintes características:

1. O esforço de desenvolvimento concentra-se na análise;
2. As estrutura de dados é mais enfatizada que as funções;
3. O processo de desenvolvimento é contínuo;
4. O processo de desenvolvimento é interactiva, isto é, não é sequencial.

5.2. Modelos da Metodologia OMT

Segundo os autores desta metodologia, o desenvolvimento de um projecto de sistema deve seguir os três modelos seguintes:

- Modelo de objectos;
- Modelo dinâmico;
- Modelo funcional;

O objectivo destes modelos é fornecer uma descrição completa do sistema através da descrição dos aspectos que envolvem o sistema como um todo.

As secções seguintes detalham estes modelos.

5.2.1. O Modelo de objectos da OMT

O modelo de objectos da metodologia OMT tem como objectivo a especificação da estrutura dos objectos que compõem o sistema. Nesta identificação, é ainda feito o levantamento dos atributos (dos objectos/classes), operações (métodos), identidade e relacionamentos entre os diferentes objectos identificados.

O resultado final desta fase, é o diagrama de classes, o qual especifica a estrutura de objectos, sua hierarquia, e associações entre as classes identificadas. Este diagrama é a base para o modelo seguinte, o Modelo dinâmico.

Os elementos modelados nesta fase já se encontram descritos a mais alto nível no capítulo 4 deste trabalho. A seguir apresenta-se a descrição simplificada destes elementos(alguns novos em relação aos que foram descritos no capitulo 4). Esses elementos são:

- Classe de objectos: descreve as características comuns de um conjunto de objectos;
- Atributo: é uma característica duma classe, ele guarda (em um objecto) um valor correspondente a uma característica do objecto;
- Operação/Método: operação definida na classe e executada sobre um objecto concreto;
- Relacionamento/Associação: Representa uma associação entre duas classes (ou objectos);
- Classes associativas: relacionamento entre classes modelada como uma outra classe;
- Generalização: Relacionamento entre uma classe e as suas classes derivadas;

A figura 7 representa a notação dos elementos do Modelo de Objectos OMT.

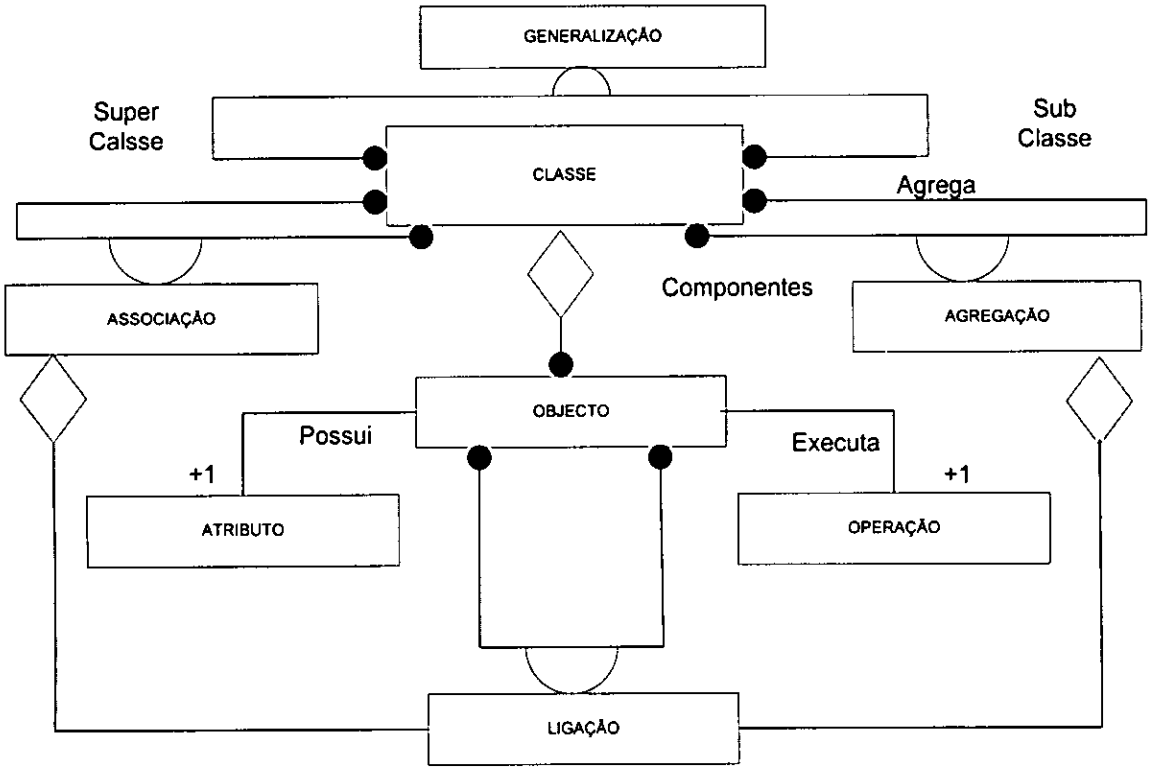


Figura 7: Modelo de Objectos de OMT

Na figura que se segue apresenta-se a notação usada neste modelo

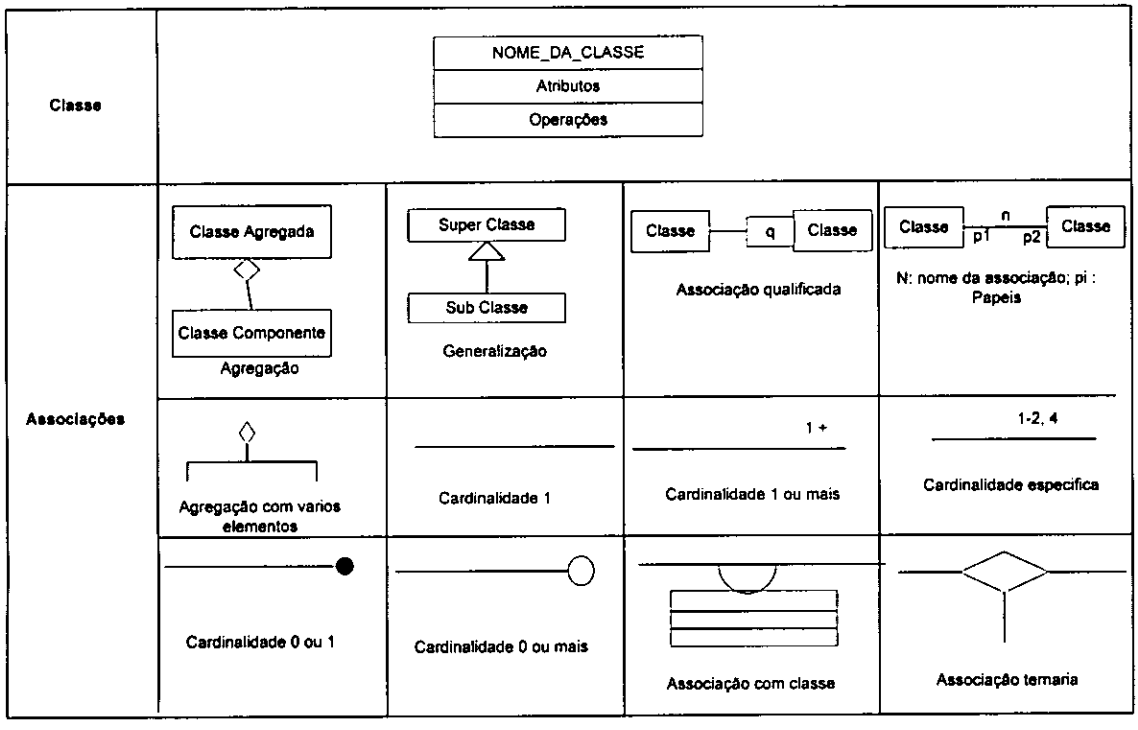


Figura 8: Notações usadas no Modelo de Objectos OMT

5.2.2. O Modelo Dinâmico de OMT

O modelo dinâmico descreve as transformações que ocorrem no sistema(nos objectos) em tempo de execução. Neste modelo são apresentadas as transformações ocorridas e a sua sequência. Os elementos principais deste modelo são o diagrama de fluxo de eventos e o diagrama de Estados. O diagrama de fluxo de eventos descreve a comunicação entre os objectos e no diagrama de fluxo de estados são descritos os diferentes estados pelos quais os objectos passam à medida que os eventos ocorrem, isto é, este diagrama descreve o ciclo de vida dos objectos dentro do sistema.
Os elementos descritos neste modelo são:

- Estado: Estado de um objecto, corresponde a uma descrição do mesmo através de valores dos seus atributos e ligações. Um estado específico de um objecto, pode estar associado a uma actividade ou a uma condição.
- Transição: diz se que houve uma transição quando um objecto passa dum estado para outro. Uma transição pode ocorrer sempre que um evento específico ocorre e as condições para a transição forem satisfeitas. Algumas transições são automáticas; uma transição é automática quando ela ocorre quando o objecto passa para o estado final através da finalização da execução da actividade associada ao estado corrente (penúltimo estado).
- Evento: um evento é um estímulo dum objecto para outro. O evento pode ser accionado dum objecto dentro do sistema para um outro objecto, também dentro do sistema ou pode provir dum ambiente externo (ao sistema) para dentro do sistema.
- Condição: é uma expressão booleana de valores de objecto; ela é válida dentro de um intervalo de tempo;
- Acção: é uma transformação que tem efeitos no objecto-alvo ou em outros objectos do sistema interligados com este objecto. Basicamente, uma transformação é uma actualização dos atributos do objecto e das suas ligações;
- Actividade: é uma operação que ocorre quando um objecto permanece num estado. A actividade é iniciada quando o objecto "entra" num estado e terminada quando ele "sai" desse estado.

A figura 9 apresenta os elementos do modelo dinâmico da metodologia OMT.

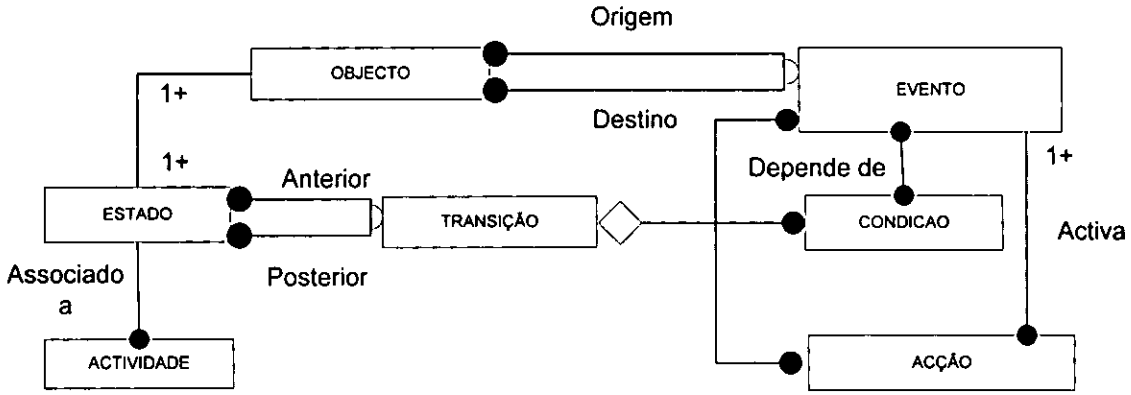


Figura 9: Modelo dinâmico de OMT.

A seguir é apresentada a notação usada neste modelo(Modelo dinâmico):

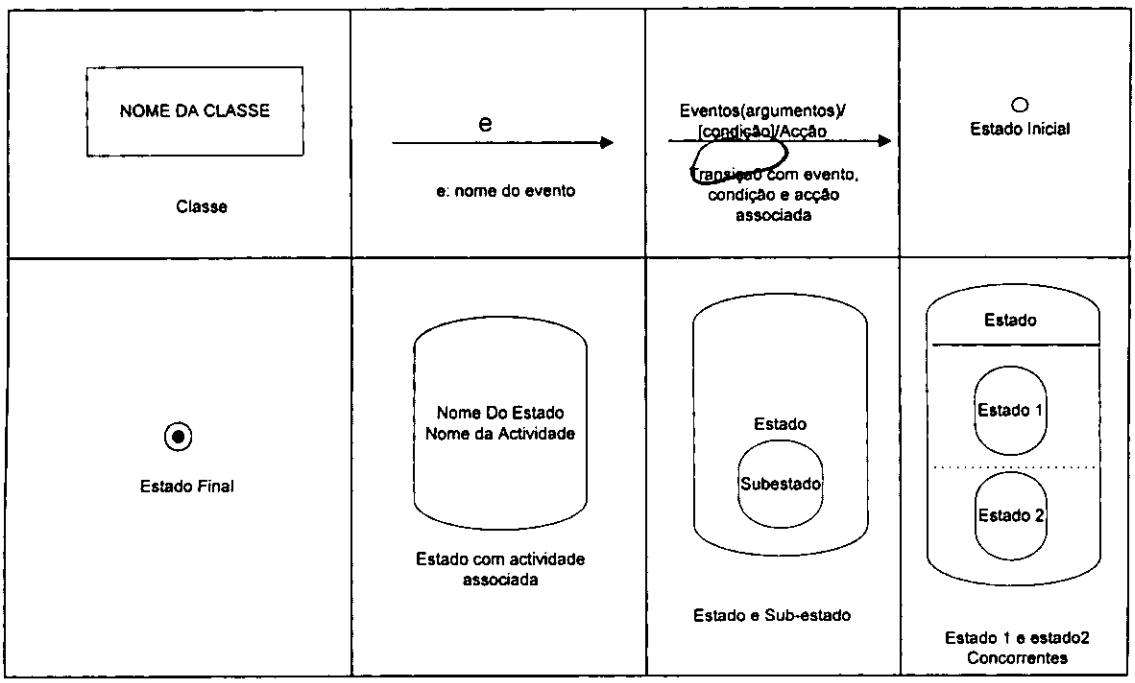


Figura 10: Notação usada no modelo dinâmico OMT.

5.2.3. O modelo funcional de OMT

O modelo funcional descreve as transformações que ocorrem no sistema (nas instâncias de objectos) através de funções, dependências funcionais, mapeamentos e restrições. Neste modelo são ainda descritos os fluxos de dados, os repositórios de dados, os processos e os actores. Os dados flúem entre os processos, actores e repositórios e o diagrama de fluxo de dados é usado para a identificação e descrição desse fluxo.

A seguir são descritos os elementos que fazem parte deste modelo apresentados na Figura 11:

- Processo: um processo é uma transformação de valores de dados. Em um objecto um processo é representado pelas operações que fazem um método;
- Fluxo de dados: é a transição de dados entre os componentes do sistema (processos, objectos);
- Actor: é uma instância de objecto (activa) que manipula os dados (produz/consome valores);
- Repositório de dados: é um objecto passivo em um diagrama de fluxo de dados o qual armazena dados para o uso posterior. O repositório responsabiliza-se pela recepção e disponibilização de dados.
- Fluxo de controlo: é um valor booleano que afecta a maneira como um processo é avaliado.

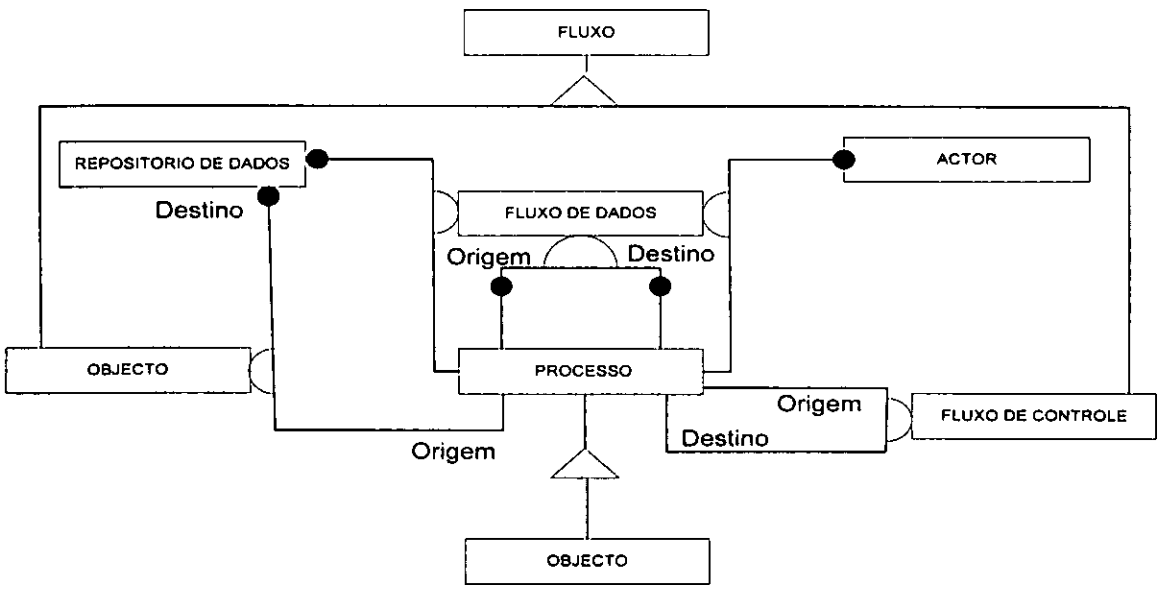


Figura 11: Modelo funcional de OMT.

A notação usada neste modelo encontra se na figura 12.

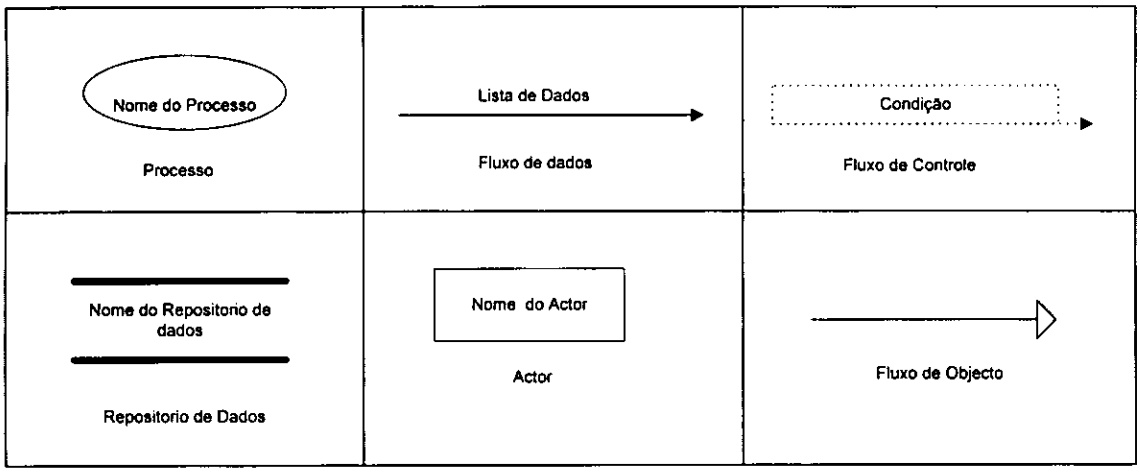


Figura 12: Notação usada no modelo Funcional de OMT.

Na base deste modelo foram modelados os elementos que compõem o caso de estudo deste trabalho. A escolha desta metodologia deveu-se principalmente ao facto de ser uma metodologia orientada à objectos e alem disso, possuir vários modelos para a descrição dos elementos do sistema para alem de ela possuir uma vasta gama de documentação.

O capítulo seguinte apresenta esta modelação a qual culminou com o desenvolvimento do protótipo do sistema sugerido.

6. CASO DE ESTUDO - MODELO DE GESTÃO DE PERMISSÕES DE ACESSO A ELEMENTOS DE UM SOFTWARE

O controlo de acesso a elementos de uma aplicação é uma das políticas de segurança de sistema que quando bem implementado pode minimizar a vulnerabilidade de um sistema de informação computadorizado (Viera, 2002), daí que a maioria de aplicações implementa mecanismos que permitem fazer a gestão de permissão de acesso às diferentes partes que envolvem as mesmas.
O controlo de permissões, garante que os utilizadores do sistema tenham apenas acesso às partes que lhes foram autorizadas.
O caso de estudo deste trabalho culminará com um protótipo de um sistema de gestão de permissões de acesso à uma aplicação, o qual será concebido aplicando os conceitos vistos no capítulo 4.
A modelação do sistema segue a Metodologia apresentada no Capitulo 5, a metodologia OMT. Segundo esta metodologia, o esforço do projecto está concentrado na análise do sistema. A seguir faz-se o levantamento dos elementos do modelo do sistema objecto de estudo aplicando os modelos da metodologia; este levantamento constitui a análise do sistema.

6.1. Elementos que envolvem a gestão de acesso

Um sistema de gestão de acesso é composto de elementos que permitem monitorar o acesso de utilizadores ao sistema. O objectivo é restringir o acesso ao sistema e às diferentes partes que o envolvem (Vieira, 2002); sendo assim, um sistema de gestão de permissões deve ter um mecanismo de *identificação e autenticação*, privilégios *de acesso e modalidade de acesso.* Além disso, o sistema de gestão de permissões de acesso deve possuir um mecanismo de *monitorar as acções/operações de utilizadores no sistema.*

6.1.1. Identificação e autenticação

A identificação e autenticação é o mecanismo através do qual o sistema permite que apenas utilizadores autorizados (registados) tenham acesso a ele. Para isso, é necessário que o utilizador se autentique através da identificação que lhe foi atribuída no momento do cadastro ou a posterior.
O modelo de classe para este elemento de gestão de acesso pode ficar como representado na Figura

13.

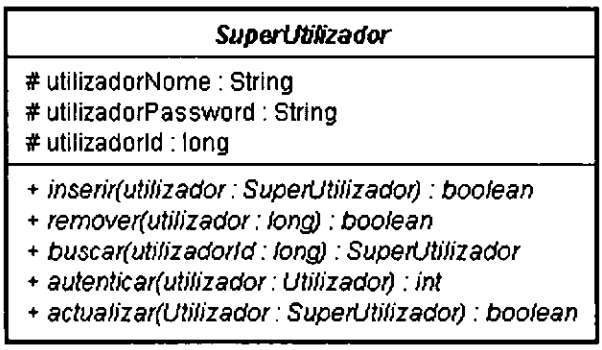


Figura 13: Classe *SuperUtilizador*.

A Figura 13 representa a classe base para o utilizador; ela será a super classe de todas as classes que representarão o utilizador. Esta é uma classe abstracta a qual possui três métodos, o método:

- *inserir()* – o qual será responsável pela inserção de utilizador no sistema;
- *remover()* – o qual será responsável pela remoção de utilizador no sistema;
- *buscar()* – o qual será responsável pela busca de utilizador do sistema;
- *autenticar()* – o qual será responsável pela autenticação do utilizador no sistema.
- *actualizar() – o qual será responsável pela actualização de utilizadores do sistema.*

Os três métodos são abstractos, pois para cada implementação, estes métodos poderão funcionar de maneiras diferentes daí que a implementação deixa-se para as classes filhas desta, as quais serão mais específicas. Como foi visto no capitulo 4, esta forma de representação traz ganhos de reutilização, pois para várias implementações da classe Utilizador (através da extensão desta) os métodos podem ser invocados sem o problema de compatibilidade.

A classe possui ainda três atributos ou variáveis de instância a saber:

- *utilizadorNome:* o qual representa o nome do utilizador;
- *utilizadorPassword:* representa a password ou chave do utilizador;
- *utilizadorId:* representa a identificação do utilizador;

Os três atributos possuem o modificador de visibilidade *protected* o que permitirá que todas as classes que estenderem a classe *SuperUtilizador* tenha acesso às mesmas.

Para o caso de estudo deste trabalho a implementação para os métodos da classe utilizador respectivamente *inserir(), remover() e buscar()* será feita sobre uma tabela de base de dados cujo modelo é apresentado na secção 6.5.

A seguir, apresenta-se na Figura 14 a extensão da classe SuperUtilizador a qual irá incluir a implementação dos métodos definidos nela.

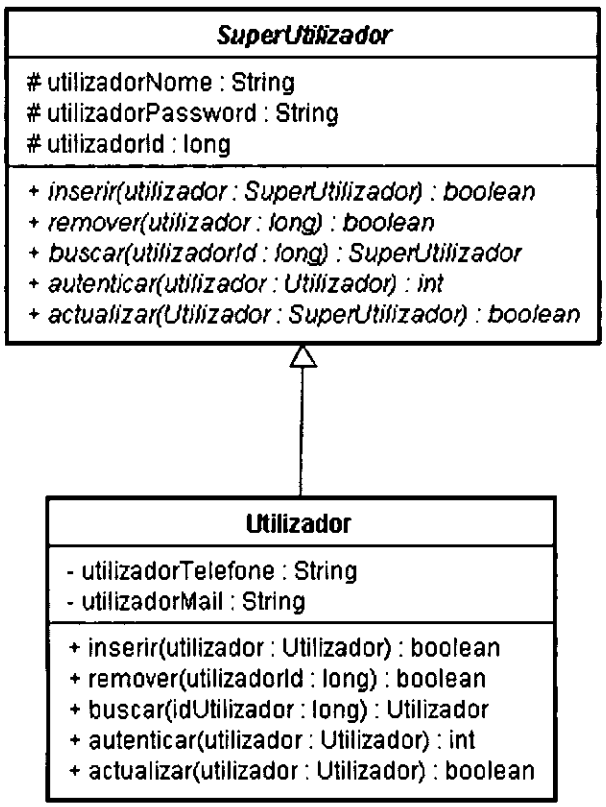


Figura 14: Classe Utilizador; Extensão da classe SuperUtilizador.

A classe Utilizador implementa os métodos definidos na classe *SuperUtilizador*. Para esta implementação, temos ainda dois atributos, os quais não aparecem na super classe, estes são atributos específicos do Utilizador do caso do estudo; uma outra implementação poderá ter outros atributos e métodos específicos a ela; "esta é uma vantagem da programação orientada à objectos."

Esta classe será responsável pela manipulação de utilizadores o que inclui a inserção, remoção, actualização, busca e autenticação. Estas operações fazem parte do processo de restrição do acesso ao sistema como um todo.

6.1.2. Modalidades de acesso

Um sistema de informação computadorizado é composto de vários elementos que são as partes que o compõe. Cada parte é responsável pela manipulação de uma certa informação suportada pelo sistema. A gestão de permissão de acessos ao sistema envolve o controlo de acesso a essas

informações (Vieira, 2002), o que significa que o sistema de gestão de acessos para além de controlar ou restringir o acesso ao sistema como um todo, deve também controlar o acesso às diferentes informações suportadas pelo mesmo, pois, diferentes utilizadores poderão ter privilégios de acessos diferentes a essas informações. Segundo Vieira (2002), as modalidades de acesso podem ser:

- *Leitura:* O utilizador apenas pode ler e visualizar a informação mas não pode alterá-la;
- *Escrita:* Este tipo de informação permite agregar dados, modificar ou remover alguma informação que tenha sido processada.
- *Execução:* Permite ao utilizador executar uma funcionalidade do sistema;

Tendo em conta estes requisitos, o conjunto de classes para representar modalidade de acesso a um elemento do sistema ficaria como mostrado na Figura 15

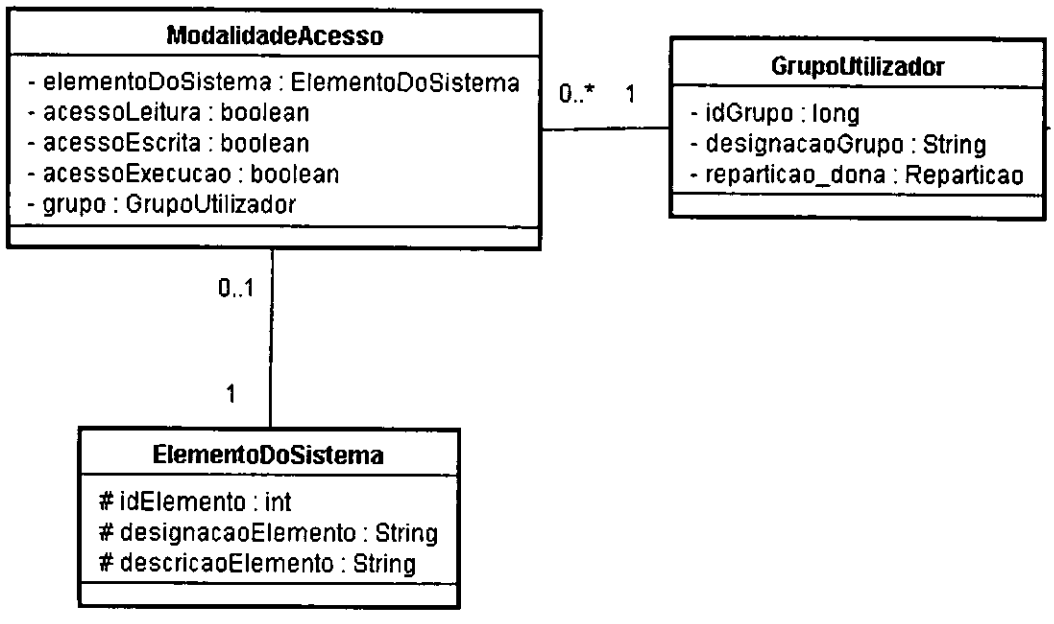


Figura 15: Conjunto de classes que representam a modalidade de sistema.

A Figura 15, representa o conjunto de classes que permitem controlar as modalidades de acesso. A seguir descreve-se cada uma delas.

ElementoDoSistema

Representa qualquer elemento acessível do sistema o qual pode ser uma tabela da base de dados, uma funcionalidade do sistema ou qualquer outro elemento;

GrupoUtilizador

Representa um elemento que acede ao sistema, este normalmente é um utilizador associado a este elemento, isto é, o utilizador estará associado a um grupo de utilizador e este é que determinará o acesso aos elementos do sistema. A Relação entre o grupo e o Utilizador está representada na figura 16.

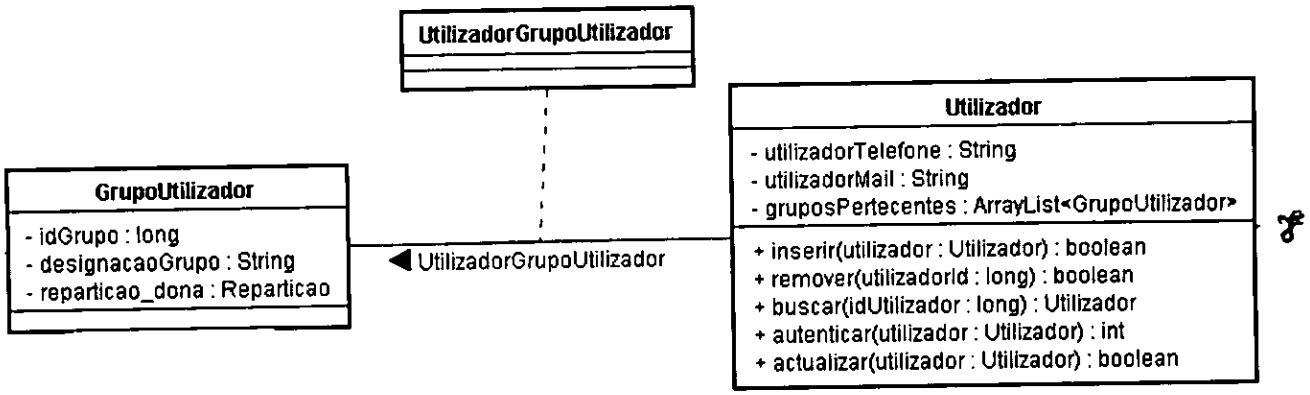


Figura 16: Relação entre as classe Utilizador e GrupoUtilizador.

Em alguns casos há necessidade de restringir o acesso a repartições específicas que fazem parte do sistema. Por exemplo em um sistema bancário, utilizadores de uma agência específica podem não ter acesso às informações registadas noutras agências. Deste modo há necessidade de se criar um elemento que represente uma repartição ou um repositório virtual no qual as informações contidas serão acedidas por utilizador (através do grupo de utilizador) cujo acesso foi atribuído.

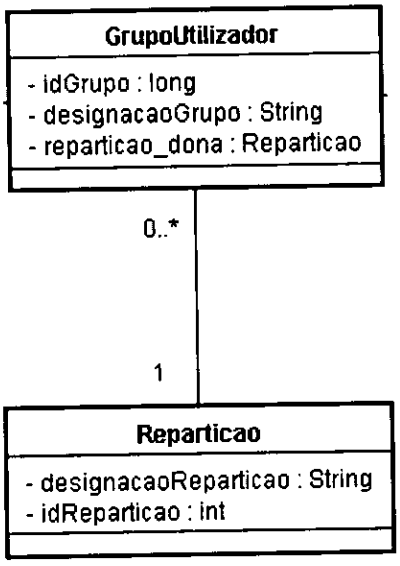


Figura 17: Classe *Repaticao* que representa um repositório virtual do sistema.

Desta forma, além de o *GrupoUtilizador* fazer referência às operações para as quais os utilizadores têm acesso, fará também referência às informações (em repartições) para as quais os utilizadores

têm acesso.

ModalidadeDeAcesso

Representa o privilegio de acesso de um grupo a um elemento de sistema. O acesso a um elemento de sistema pode ser de Leitura, Escrita ou Execução.

6.1.3. Controle de operações de utilizadores no sistema

A gestão de permissões de acesso a um sistema deve incluir também o Controle das operações dos utilizadores sobre o sistema (Vieira, 2002), isto é, as operações dos utilizadores ou aquilo que os utilizadores fazem sobre o sistema, deve ser controlado. Sendo assim, o sistema de controle de gestão de permissões deve incluir uma funcionalidade que permita colectar informações das operações executadas pelos utilizadores no sistema. O esquema a seguir representa a classe para a colecção das operações de utilizadores no sistema.

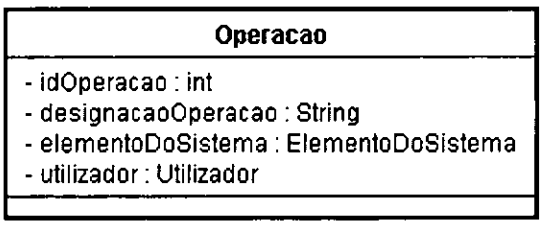


Figura 18: Classe Operação, para monitorar as operações dos utilizadores no sistema.

Os elementos apresentados até aqui, representam a base do sistema de gestão de permissões de acesso a um sistema, sugerido pelo autor deste trabalho. A Figura 19 apresenta o esquema integrado (Modelo de Objectos da Metodologia OMT) de todos esses elementos.

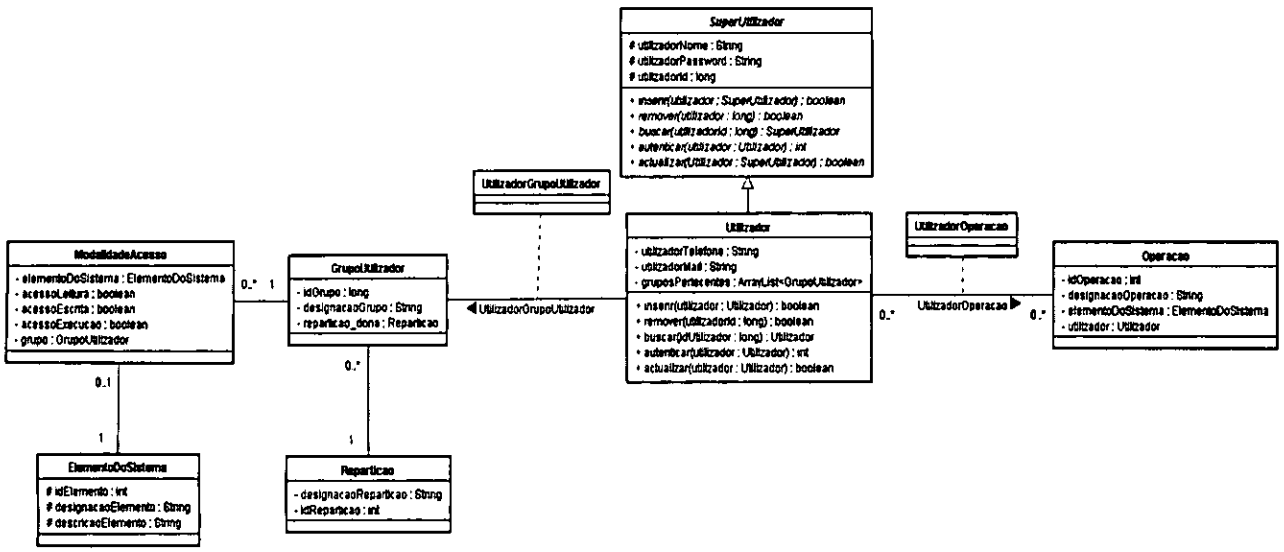


Figura 19: Modelo de Objectos dos elementos básicos do sistema de gestão de permissões proposto.

6.2. Modelo Dinâmico do Sistema proposto

A metodologia OMT sugere que sejam também descritos os aspectos comportamentais do sistema (RUMBAUGH at al,1991) . Como foi visto na subsecção 5.2.2, esta descrição é feita através do modelo dinâmico o qual é responsável pela descrição da comunicação entre os diversos objectos do sistema, os eventos, as actividades, entre outros aspectos relacionados com o comportamento do sistema.

Nesta secção espelha-se aquilo que constitui o modelo dinâmico do sistema proposto.

A figura seguinte representa as comunicações básicas entre os elementos (objectos) do sistema.

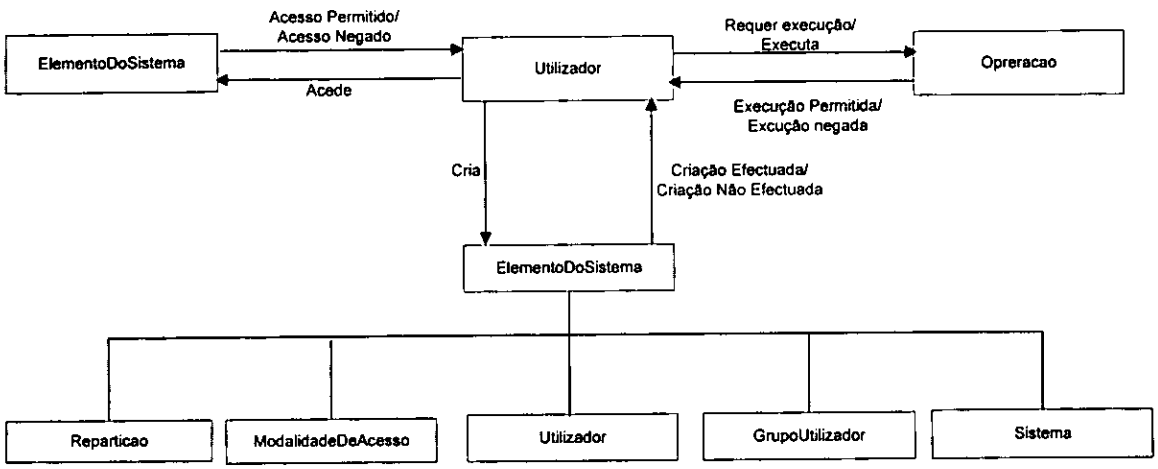


Figura 20: Modelo dinâmico dos elementos básicos do Sistema proposto – Fluxo de eventos.

A figura 20 descreve a comunicação básica entre os elementos do sistema proposto. Conforme pode ser visto(da figura) o Utilizador é o centro das comunicações. O utilizador *executa ope*rações, *acede a elementos do sistema* e *cria elementos.* Em cada comunicação, este elemento (Utilizador) recebe resposta das entidades "contactadas".

6.3. Modelo Funcional do Sistema proposto

Conforme referido antes (na subsecção5.2.3) o modelo funcional descreve as transformações ocorridas no sistema através de funções e dependências, mapeamentos e restrições. O objectivo é mostrar como os dados flúem pelo sistema. Para o sistema objecto do estudo deste trabalho o diagrama de fluxo de dados é apresentado na figura 21.

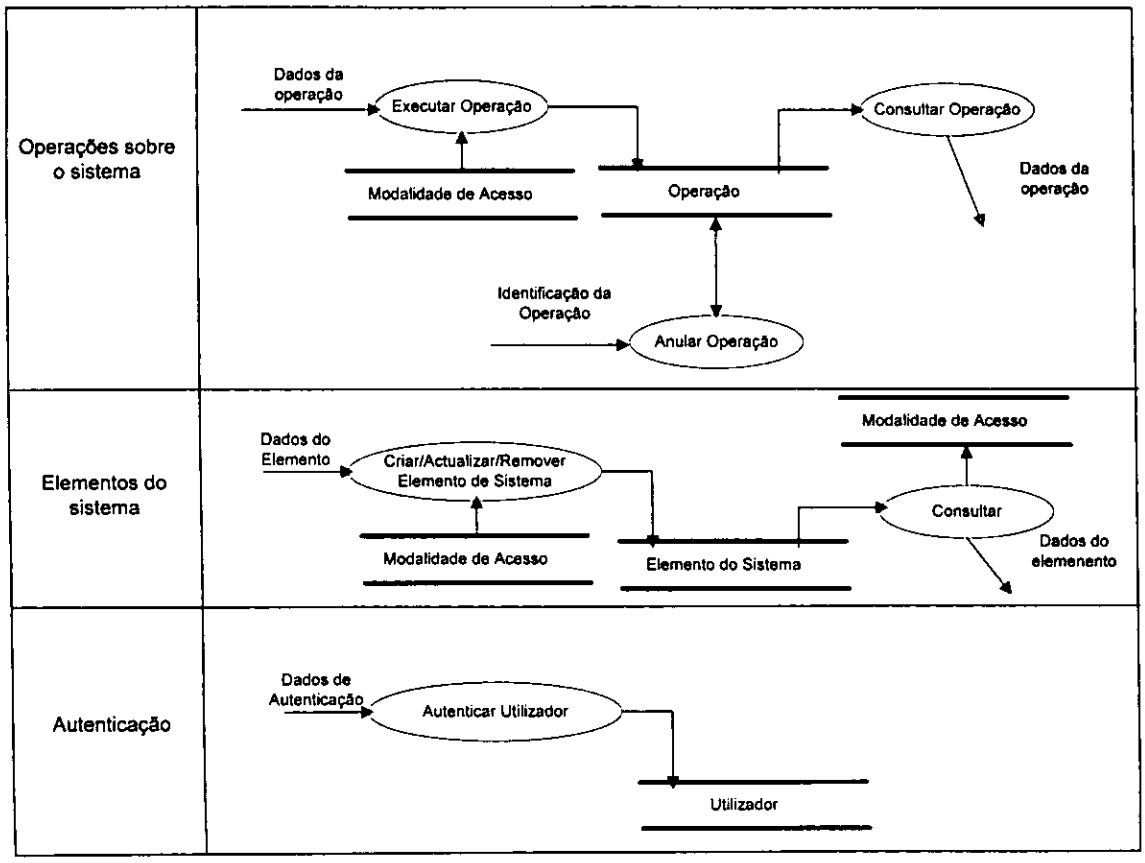


Figura 21: Modelo funcional para alguns elementos básicos do sistema proposto.

6.4. Integrabilidade dos Componentes Criados

O modelo apresentado na secção 6.2 foi a base para a concepção de componentes para o sistema de gestão de privilégios de acesso, objecto de estudo deste trabalho.

O modelo apresentado e implementado foi integrado em dois sistemas:

- Sistema de Gestão de Florestas e Fauna Bravio do Ministério da Agricultura desenvolvido na plataforma *Java Enterprise Edition* (J2EE);
- Sistema de Gestão de Pacientes em TARV e de Medicamentos nas Farmácias dos Hospitais de Dia.

As subsecções a seguir detalham a integração do modelo nos sistemas em causa.

6.4.1. Integração de Componentes no Sistema de Gestão de Florestas e Fauna Bravia

O sistema de Gestão de Florestas e Fauna Bravia apresentado aqui é uma simulação do Sistema de Gestão de Florestas e Fauna Bravia desenvolvido pela empresa EXI. Este é um sistema que permite fazer a gestão de recursos florestais e faunístico do País. Trata-se de um sistema nacional com direcções em todas as províncias do Pais. Em cada direcção provincial existem diferentes departamentos e cada departamento trata de um tipo de informação (Nota: As direcções provinciais fazem operações similares). Além disso, as diferentes operações são executadas por pessoas diferentes.[**]

Há aqui necessidade de gerir o acesso às diferentes parte que compõem o sistema, uma vez que todos os utilizadores tem acesso ao sistema como um todo mas pela lógica do sistema, nem todos os utilizadores podem ter acesso a "tudo". A integração do sistema de gestão de permissões permite restringir o acesso a cada elemento do sistema e o controle das operações dos utilizadores no sistema.

O modelo apresentado na secção 6.2 aplica-se na implementação de gestão de acessos neste sistema. Uma vez que o sistema é composto de vários elementos e que nem todos os utilizadores terão acesso a todos eles, o modelo apresentado permite mapear quais os elementos cada utilizador (através dos grupos de utilizador) tem acesso.

O sistema em causa é um sistema centralizado mas correndo também a nível provincial. Identificam-se aqui dois tipos de acesso, o acesso provincial e o acesso central, uma vez que alguns utilizadores(pela lógica) terão apenas acesso às informações que dizem respeito às províncias a que pertencem.

A componente *Repartição* representará uma província e os utilizadores (através dos grupos de utilizadores) e será feito o mapeamentos das repartições (províncias) para as quais utilizadores tem acesso.

6.4.1.1. Repositório de dados

O modelo apresentado na secção 6.2 representa apenas as classes do sistema. Mas é necessário criar um repositório de dados que permita salvar todas as configurações e informações que serão captadas e processadas usando as classes. A partir do modelo funcional do sistema (Apresentado na secção 64) podem-se identificar as seguintes tabelas: UTILIZADOR, GRUPO_UTILIZADOR,

[**] Neste trabalho não será feito um estudo profundo do sistema apresentado, pois esse não é o objectivo do mesmo. Apenas será apresentados os elementos principais do mesmo.

ELEMENTO_DO_SISTEMA, MODALIDADE_DE_ACESSO, OPERACAO, REPARTICAO.

O conteúdo das tabelas e seus relacionamentos encontra-se representado na tabela 22.

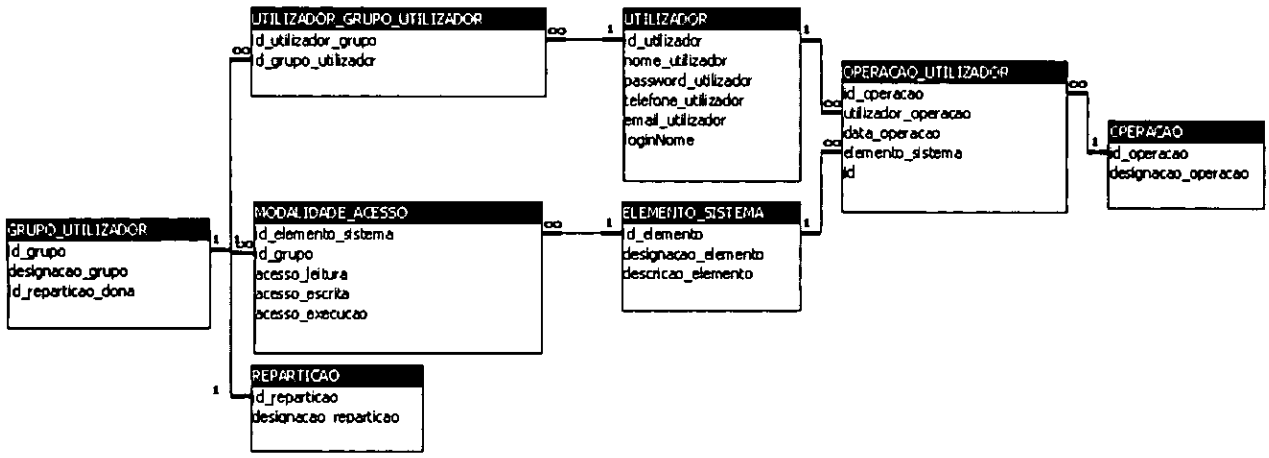


Figura 22: Esquema de tabelas para o modelo de sistema de gestão de Permissões de acesso.

Descrição do Modelo

TABELA	DESCRIÇÃO
UTILIZADOR	Local onde ficará guardada a informação de todos os utilizadores do sistema.
GRUPO_UTILIZADDOR	Local onde ficarão guardados todos os grupos de utilizadores possíveis
ELEMENTO_SISTEMA	Todos os elementos do sistema ficarão guardados nesta tabela, seja opção do sistema, tabela, etc.
REPARTICAO	Ficarão aqui armazenadas as diferentes repartições do sistema; neste caso concreto, ficarão registadas as diferentes direcções provinciais onde o sistema corre.
UTILIZADOR_GRUPO	Representa o mapeamentos dos grupos de um utilizador
MODALIDADE_ACESSO	Representa o mapeamento de acessos dos grupos aos elementos do sistema
REPARTICAO_PERMITIDA	Mapeiará as repartições permitidas para um grupo de Utilizadores

Tabela 1: Elementos do modelo de dados

6.4.1.2. Interface do sistema

A interface do sistema para o modelo apresentado segue o padrão do sistema para o qual é integrado, isto é, a interface do sistema de gestão de permissões está integrada no sistema principal (Sistema de gestão de Floresta e Fauna Bravia)[††]. A seguir são apresentados os elementos desta interface.

6.4.1.2.1. Cadastro de Utilizador

O cadastro do utilizador é composto por 3 elementos, o registo, a consulta e a actualização. O registo e a actualização baseiam-se no mesmo formulário o qual é apresentado na figura 23. O objectivo é recolher informação de utilizadores que tem (ou terão) acesso ao sistema.

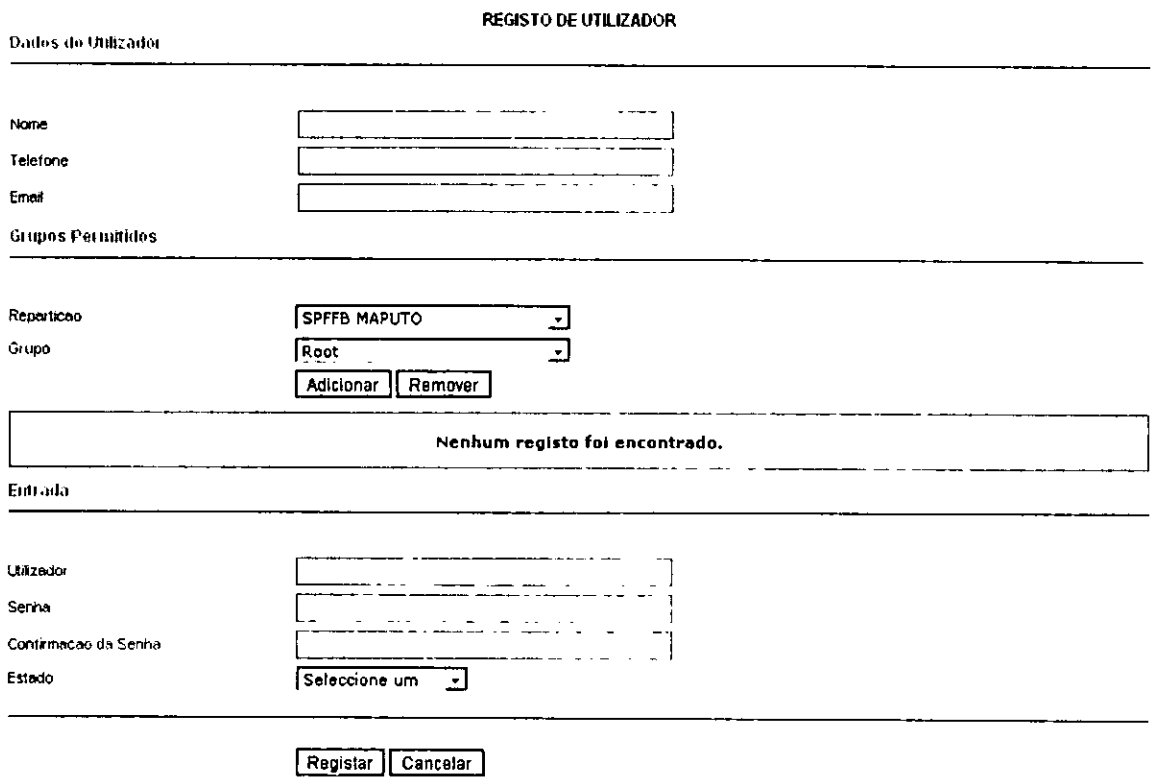


Figura 23: Formulário de registo de utilizadores.

Este formulário é responsável pela captação dos dados referentes ao utilizador. Na primeira secção

[††] Uma vez que o sistema principal é uma simulação, serão aqui apresentadas apenas algumas partes do mesmo, pois objectivo é mostrar o sistema de gestão de permissões e sua integrabilidade no sistema principal.

temos os dados referentes a identificação dos utilizadores; na segunda secção é onde se faz o mapeamento dos grupos a que o utilizador pertence e na última secção encontra se os dados de autenticação do utilizador.

Na base dos dados deste formulário são captadas informações para a criação dos objectos das classes apresentadas na secção a saber: Utilizador e os Grupos de Utilizadores a ele mapeados (representados no esquema pela classe *UtilizadorGrupoUtilizador*)

6.4.1.2.2. Consulta de Utilizadores

A consulta de utilizadores baseia-se nos métodos de consulta definidos na classe Utilizador. A interface do sistema para a consulta pode ser vista na figura 24.

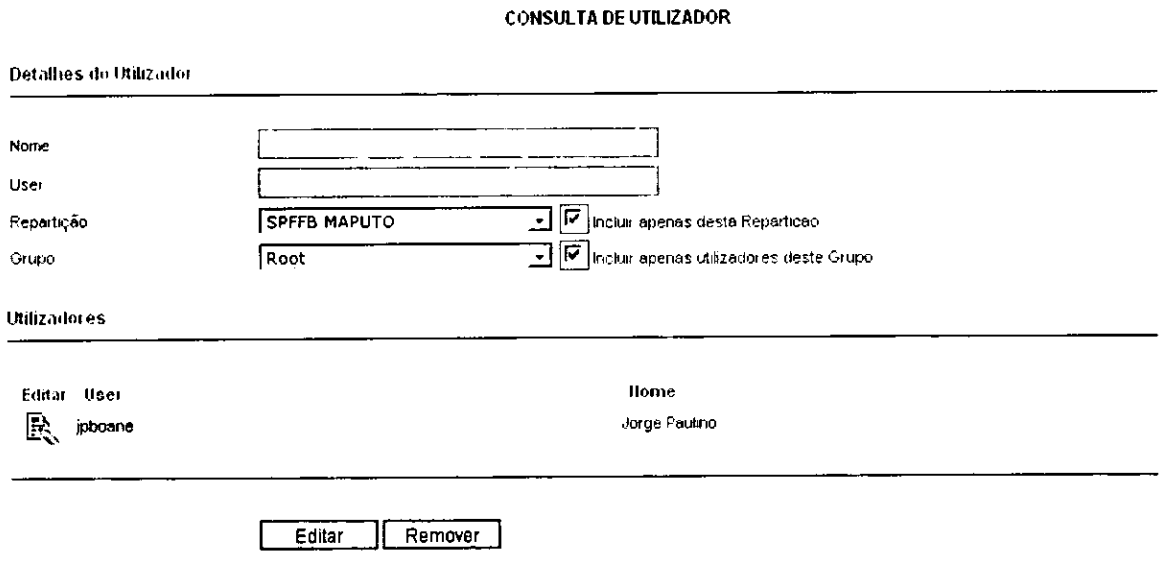


Figura 24: Interface para a consulta de utilizadores cadastrados.

Conforme se pode ver na figura 24, a consulta de utilizadores baseia-se em dois critérios: *Repartição e Grupo (de utilizadores);* Em um dado momento os utilizadores listados serão (apenas) utilizadores que pertencem a uma certa Repartição ou Grupo específico.

6.4.1.2.3. Actualização de Utilizadores

A interface para a actualização de utilizadores é a mesma para a adição de utilizadores. A partir da interface da consulta, é possível seleccionar um utilizador específico e em seguida editar os dados do mesmo.

6.4.1.2.4. Cadastro de Grupos de Utilizadores

A figura 25 ilustra o formulário do cadastro de utilizadores. Na secção inicial encontra-se uma interface para a inclusão de novos grupos e na parte mais abaixo estão listados os grupos já cadastrados. Os métodos que permitem manipular os grupos de utilizadores foram apresentados na secção 6.2.

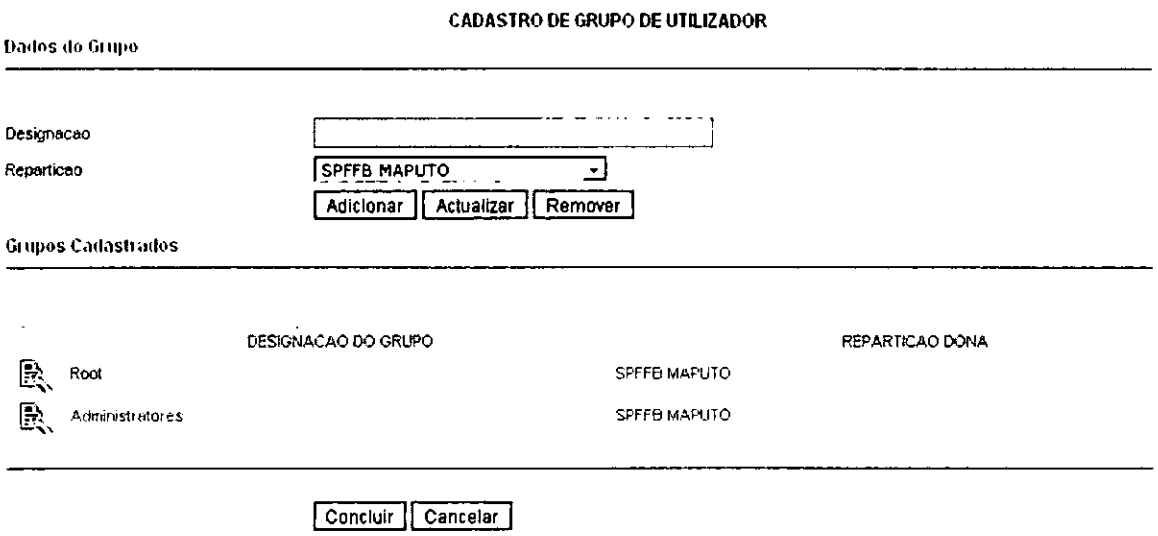


Figura 25: Formulário para cadastro, consulta, actualização e remoção de grupos.

6.4.1.2.5. Mapeamento de Privilégios de acesso

Para manipular os privilégios de acesso ao elementos de sistema, o sistema possui uma interface que permite mapear os privilégios de acessos dos grupos aos elementos do sistema. Conforme pode ser visto na figura 26, é fornecida a lista dos diferentes elementos do sistema a informação referente ao acesso pelos grupos de utilizadores.

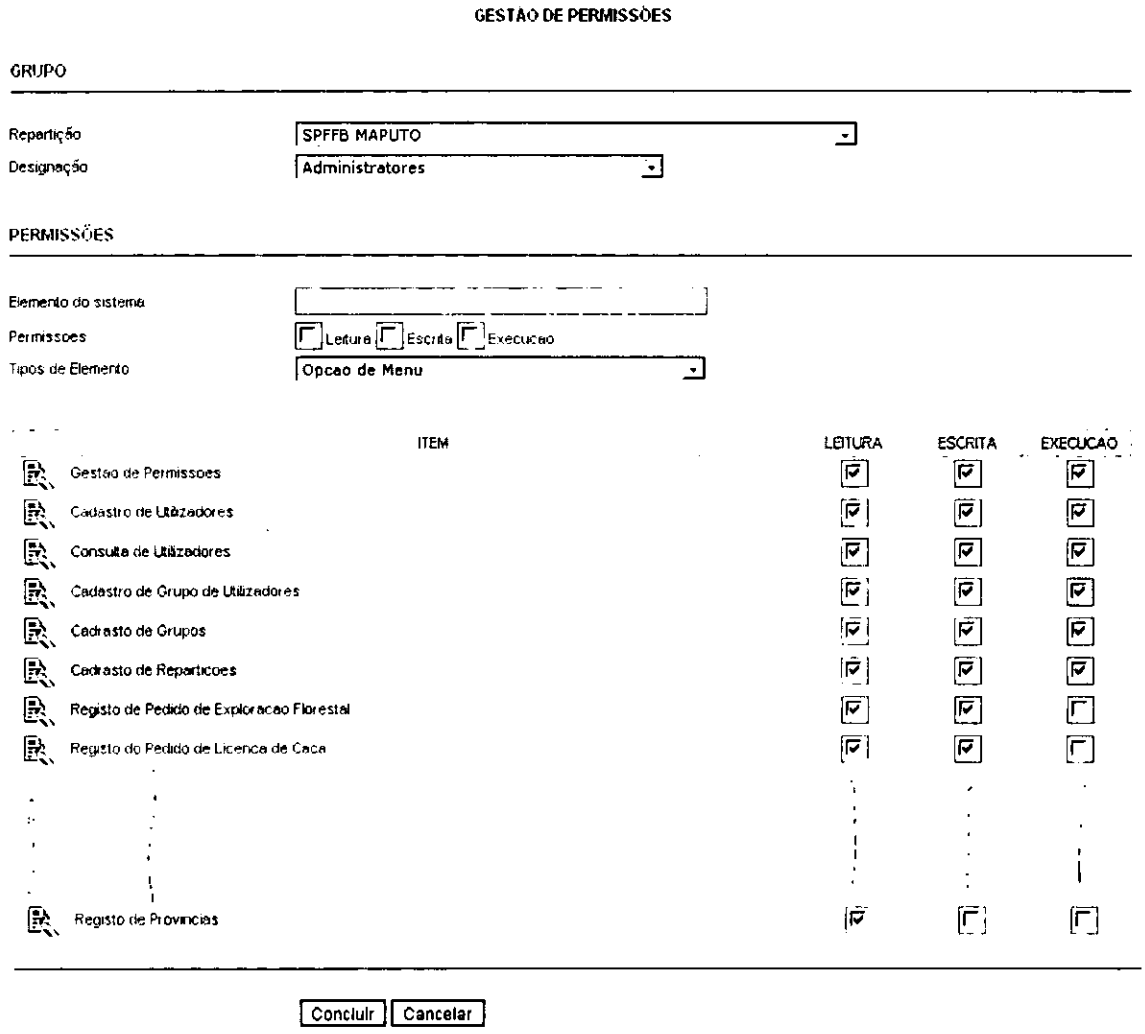


Figura 26: mapeamento dos privilégios de acesso aos grupos de utilizadores;

Na base dos mapeamentos feitos, os utilizadores associados aos grupos, terão apenas o acesso definido através desta interface. Uma parte de acesso (Acesso execução) é restringida exactamente no momento em que se carrega o menu. Isto significa que se um utilizador não tiver acesso de execução duma funcionalidade do sistema, a mesma não será exibida no menu. A figura 27 ilustra o menu do sistema o qual é dinamizado pelos acessos de execução que os utilizadores que acedem a aplicação possuem.

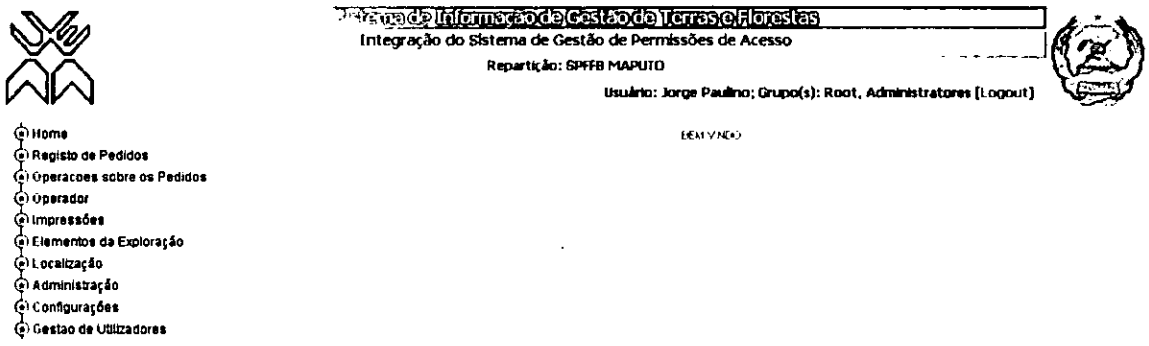


Figura 27: Menu do sistema, dinamizado pelo acesso de execução de utilizadores.

Se o utilizador corrente (por exemplo) não tiver permissão para executar as operações em "*Registo de Pedidos*", esta opção não aparecerá no menu do sistema.

Para as outras modalidades de acesso (acesso leitura e acesso de escrita) a validação é feita no momento em que o utilizador tenta ler ou escrever em um elemento de sistema, o que significa que na base do mapeamento feito o utilizador poderá não conseguir ler ou escrever sobre um dos elementos de sistema (ex. tabela da base de dados).

6.4.1.2.6. Controle de Operações de utilizadores

Conforme descrito na secção 6.2, o sistema possui um mecanismo de controlo de operações de utilizadores sobre o sistema. Tudo o que o utilizador faz, fica registado e é possível monitorar essas operações.

O interface para a monitorização de operações de utilizadores pode ser vista na figura 28.

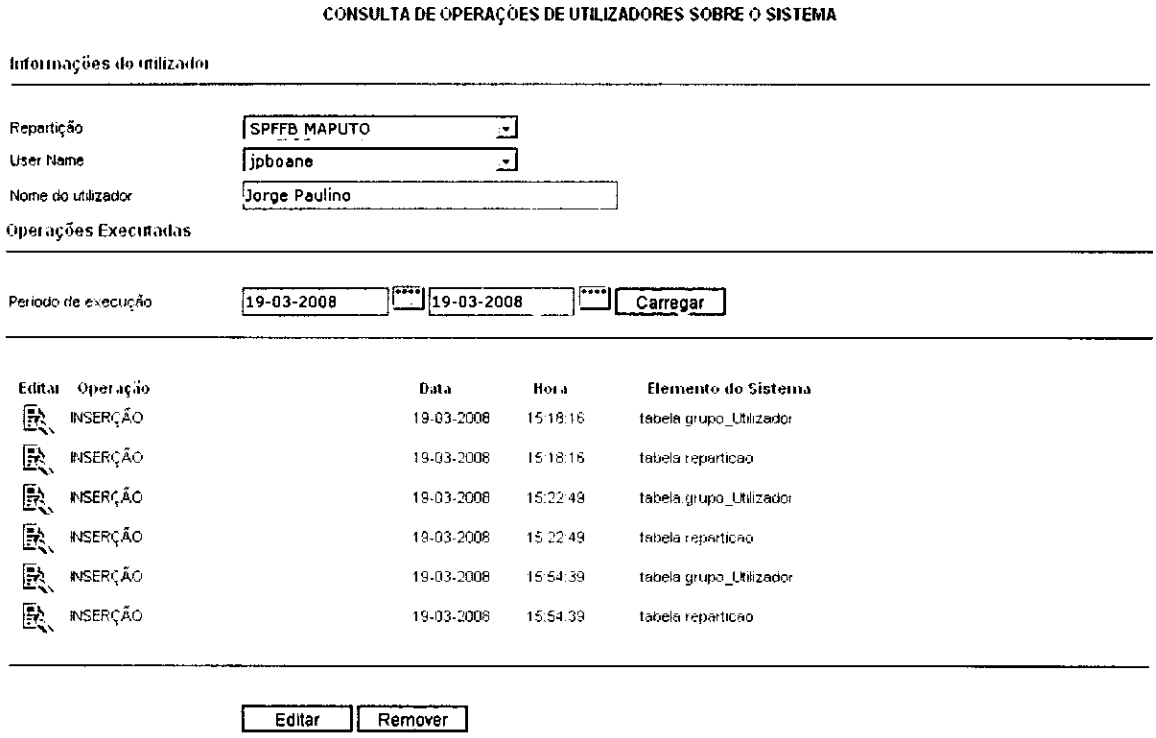


Figura 28: Formulário para a o controle das operações dos utilizadores sobre o sistema.

6.4.2. Integração de Componentes no Sistema de Gestão de Pacientes em TARV e de Medicamentos

O *Sistema de gestão de Pacientes TARV e de Medicamentos*, é um sistema que permite fazer a gestão de Pacientes em tratamento antiretroviral e de Medicamentos nas Farmácias das unidades sanitárias nacionais. O Desenvolvimento do sistema já se encontra terminado e actualmente está em curso um processo de teste do sistema o qual será seguido pela implementação piloto em uma das farmácias do Hospital Central de Maputo. Este sistema não é centralizado, isto é, não existirá comunicação entre as diferentes farmácias do País.

Pretende-se aqui neste trabalho simular a integração do módulo de gestão de permissões (apresentado na secção 6.2 deste trabalho) neste sistema. Tal como na simulação da integração do módulo no sistema de Gestão de Florestas e Fauna Bravia, aqui também serão apresentadas as funcionalidades básicas do sistema e estas não serão implementadas; apenas serão implementadas as funcionalidades de Gestão de permissões.

Tal como em qualquer sistema, o sistema em causa, será manipulado por utilizadores; e pretende-se restringir o acesso destes aos elementos do sistema e ao sistema em geral, bem como o Controlo das operações destes nos diferentes componentes do sistema.

Na base do modelo apresentado na secção 6.2 far-se-á a seguir a apresentação dos diferentes componentes que irão usar os componentes apresentados.

6.4.2.1. Repositório de dados

A figura 30 apresenta o esquema de tabelas para o repositório de dados do sistema. Conforme pode ser observado, o esquema não se difere muito do esquema apresentado na figura 22. A única diferença consiste no campo *url_foto* na tabela *UTILIZADOR* o qual não aparece no esquema anterior; este campo irá armazenar a URL da foto do utilizador. Uma vez que este campo é novo ele não foi integrado na classe *Utilizador*, apresentada na secção 6.2.1 (Figura 14). A integração deste campo nesta classe será bastante simples pois a programação orientada a objectos, conforme visto no capítulo 4, fornece o mecanismo de *Herança*, através do qual podem ser acrescidas funcionalidades a classes já concebidas sem que as mesmas sejam alteradas. Aproveitando-se deste facto será criada uma nova classe, a classe *UtilizadorHdD* a qual incluirá as novas funcionalidades exigidas neste sistema.

exigidas neste sistema.

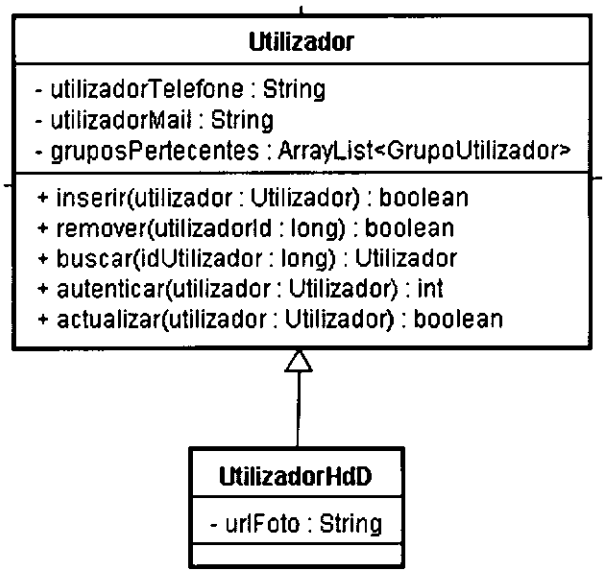


Figura 29: Classe *UtilizadorHdD* e seu relacionamento com a Classe *Utilizador*

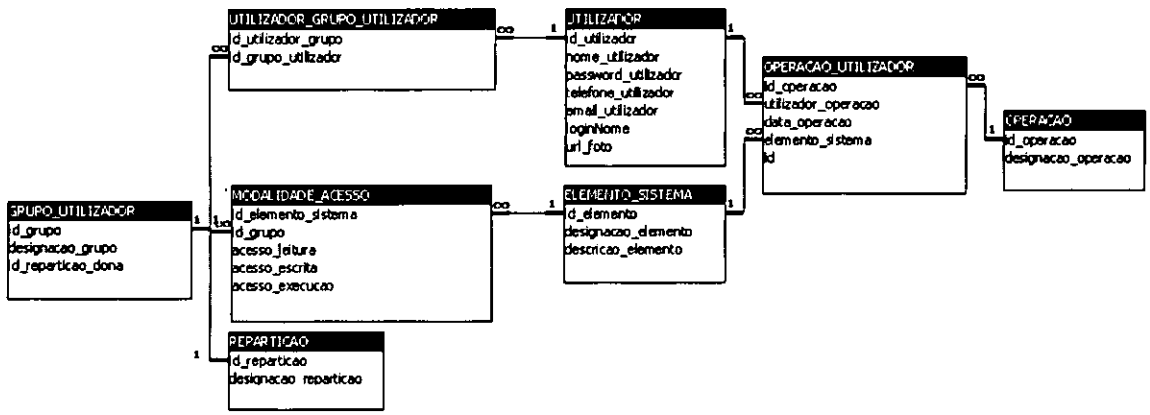


Figura 30: Esquema de tabelas do Sistema.

6.4.2.2.Interface do sistema

Tal como no sistema anterior, a interface do sistema segue o padrão do sistema mãe, e os diferentes componentes desta interface são apresentados nas subsecções seguintes.

Nota: Em relação ao sistema apresentado anteriormente, apenas mudará a forma como os dados são capturados. A forma como estes são manipulados não muda, uma vez que as classes usadas para o tal são as mesmas.

6.4.2.2.1. Cadastro de Utilizador

O cadastro do utilizador é composto por 3 funcionalidades, o registo, a consulta e a actualização. O registo e a consulta baseiam-se no formulário apresentado na figura 31. O objectivo é recolher informação de utilizadores que tem (ou terão) acesso ao sistema.

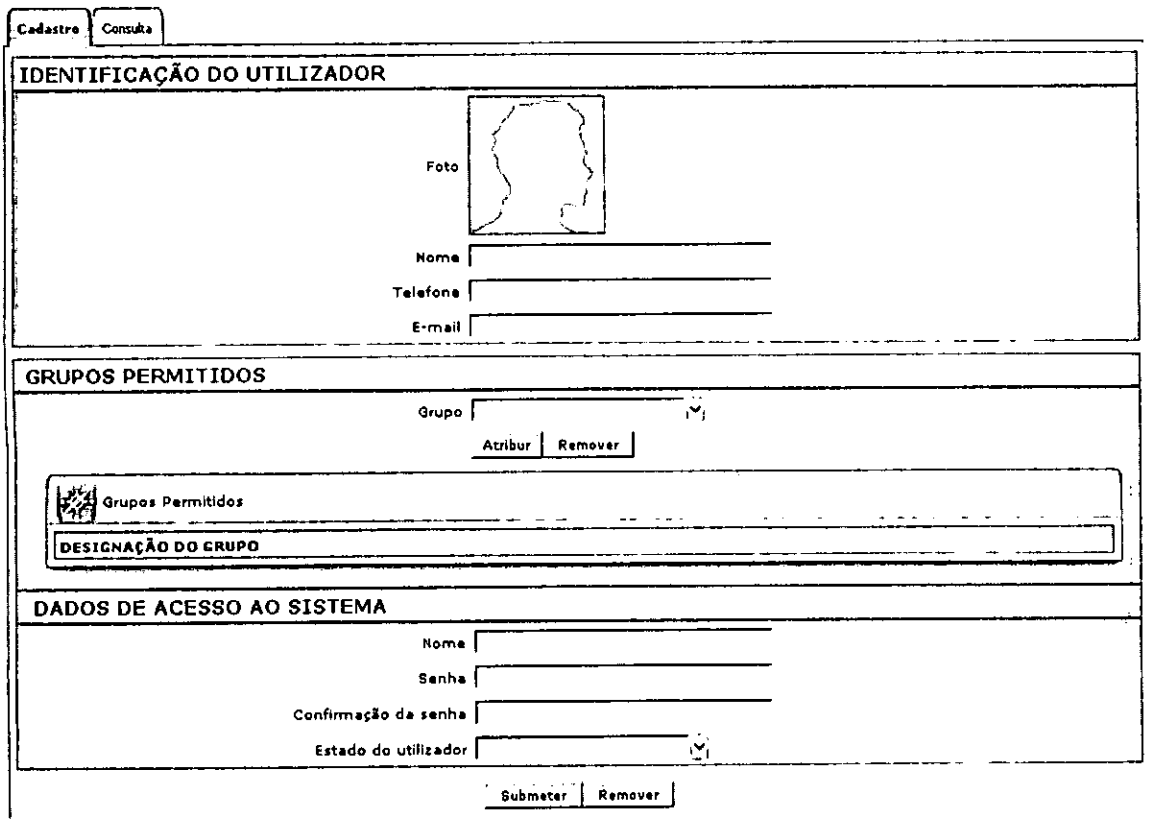


Figura 31: Formulário de registo de utilizadores.

6.4.2.2.2. Consulta de Utilizadores

A consulta de utilizadores baseia-se nos métodos de consulta definidos na classe Utilizador. A interface do sistema para a consulta pode ser vista na figura 32.

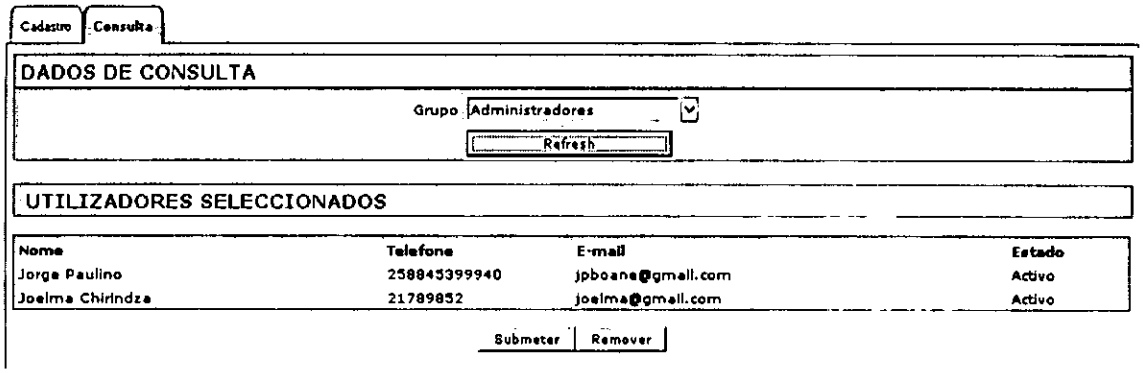


Figura 32: Interface para a consulta de utilizadores cadastrados.

6.4.2.2.3. Actualização de Utilizadores

A interface para a actualização de utilizadores é a mesma para a adição de utilizadores. A partir da interface da consulta, é possível seleccionar um utilizador específico e em seguida editar os dados do mesmo.

6.4.2.2.4. Cadastro de Grupos de Utilizadores

A figura 33 ilustra o formulário do cadastro de utilizadores. Na secção inicial encontra-se uma interface para a inclusão de novos grupos e na parte mais abaixo estão listados os grupos já cadastrados. Os métodos que permitem manipular os grupos de utilizadores foram apresentados na secção 6.2.

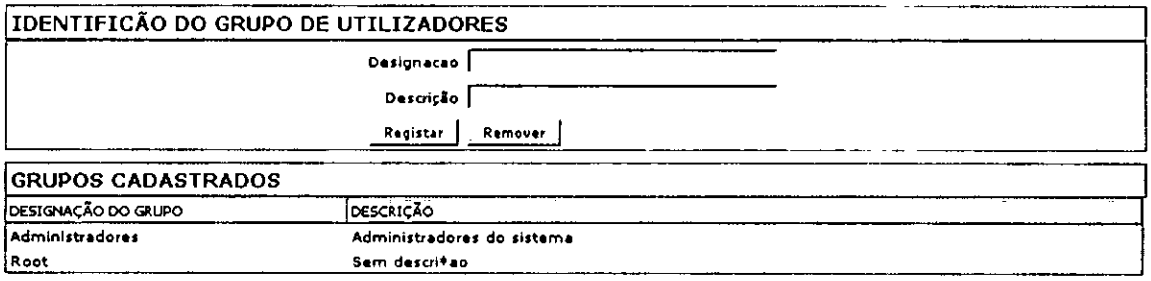


Figura 33: Formulário para cadastro, consulta, actualização e remoção de grupos.

6.4.2.2.5. Mapeamento de Privilégios de acesso

Para manipular os privilégios de acesso aos elementos de sistema, o sistema disponibiliza uma interface que permite mapear os privilégios de acessos dos grupos aos elementos do sistema. Conforme pode ser visto na figura 34, é fornecida a lista dos diferentes elementos do sistema a informação referente ao acesso pelos grupos de utilizadores.

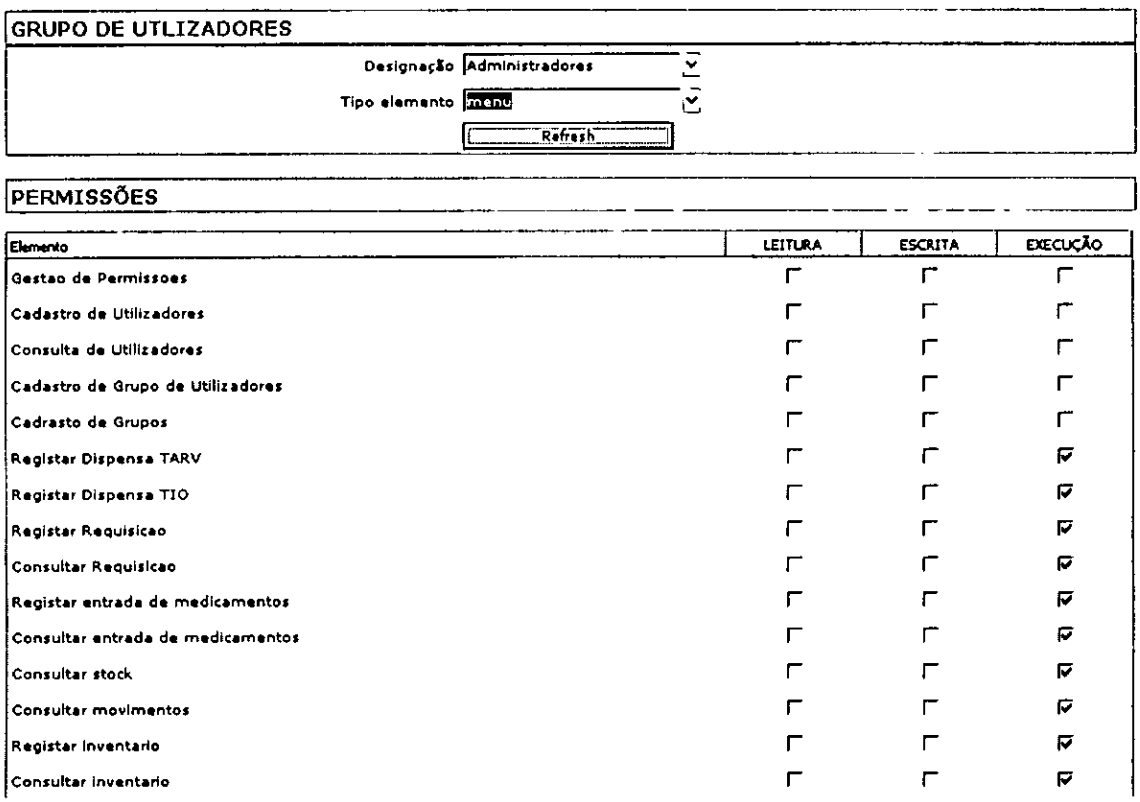


Figura 34: mapeamento dos privilégios de acesso aos grupos de utilizadores;

Na base dos mapeamentos feitos, os utilizadores associados aos grupos, terão apenas o acesso definido através desta interface. Uma parte de acesso (Acesso execução) é restringida exactamente no momento em que se carrega o menu. Isto significa que se um utilizador não tiver acesso de execução duma funcionalidade do sistema, a mesma não será exibida no menu. A figura 35 ilustra o menu do sistema o qual é dinamizado pelos acessos de execução que os utilizadores que acedem a aplicação possuem.

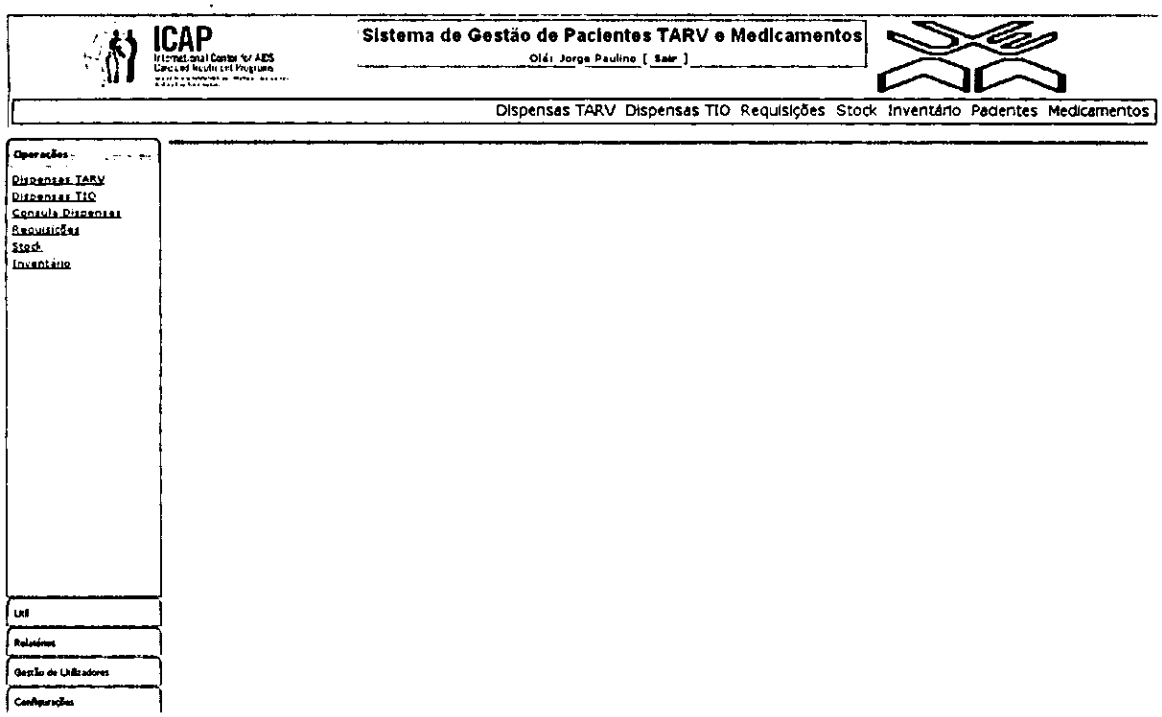


Figura 35: Menu do sistema, dinamizado pelo acesso de execução de utilizadores.

Se o utilizador corrente (por exemplo) não tiver permissão para executar as operações em "*Dispensas TARV*", esta opção não aparecerá no menu do sistema.

Para as outras modalidades de acesso (acesso leitura e acesso de escrita) a validação é feita no momento em que o utilizador tenta ler ou escrever em um elemento de sistema, o que significa que na base do mapeamento feito o utilizador poderá não conseguir ler ou escrever sobre um dos elementos de sistema (ex. tabela da base de dados).

6.4.2.2.6. Controle de Operações de utilizadores

Conforme descrito na secção 6.2, o sistema possui um mecanismo de controlo de operações de utilizadores sobre o sistema. Tudo o que o utilizador faz, fica registado e é possível monitorar essas operações.

A interface para a monitorização de operações de utilizadores pode ser vista na figura 28.

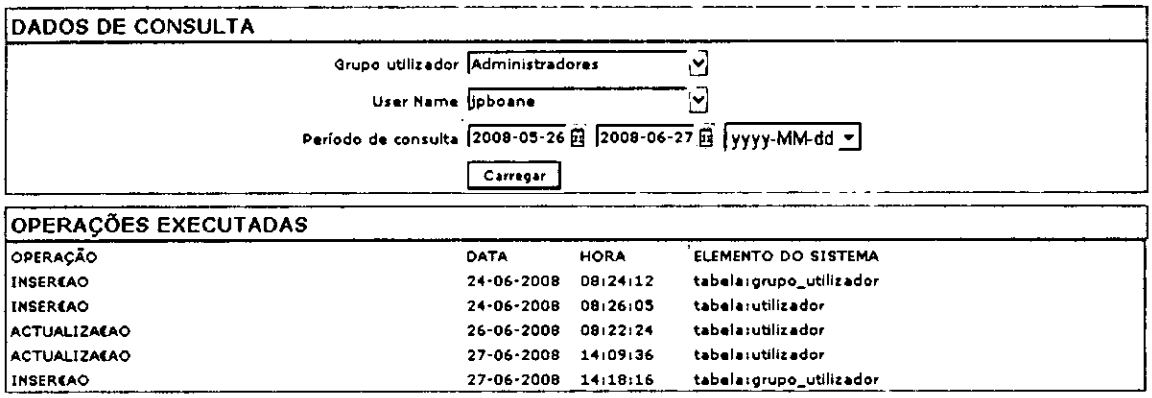


Figura 36: Formulário para a o controle das operações dos utilizadores sobre o sistema.

7. CONSIDERAÇÕES FINAIS E RECOMENDAÇÕES

7.1. Conclusões

Os objectivos traçados para o estudo deste trabalho culminaram com a concepção dum protótipo de sistema baseado na técnica de programação orientada a objectos. Esta técnica (conforme visto) tem sido uma aposta dos desenvolvedores de sistemas de informação computadorizados devido ao facto de garantir a reutilização (fácil) de componentes.

Os benefícios que advêm do uso correcto das práticas de programação orientada a objectos são palpáveis: o aumento da produtividade, a redução de custos de concepção e de manutenção e a redução dos erros (bugs) dos sistemas de informação, entre outros benefícios. Os benefícios referidos não são automáticos; é necessário que os padrões definidos para esta técnica sejam bem observados pois só assim poderão ser alcançados os benefícios que se supõe que se alcancem com o uso desta técnica.

Um dos meios pelos quais se pode seguir de perto os requisitos de modelação desta técnica é seguir alguma metodologia de desenvolvimento orientado a objectos. As metodologias orientadas a objectos fornecem uma forma de modelação propícia para a modelação de objectos. Para este estudo foi usada a metodologia orientada a objectos OMT a qual foi também descrita neste trabalho.

O protótipo elaborado, foi um exemplo de modelação e desenvolvimento de um sistema usando a técnica orientada a objectos. Embora o projecto seja simples, nele foram aplicadas as boas práticas de desenvolvimento de sistemas de informação e através dele pode se pensar em projectos mais complexos os quais podem ser concretizados com aplicação de componentes reutilizáveis.

Os componentes concebidos neste projecto mostram se ser integráveis em projectos futuros pois para além de o projecto ter seguido os princípios sugeridos pela técnica de orientação a objectos as mesmas são quase indispensáveis em quase todos os sistemas de informação que deseja implementar alguma forma de segurança de informação.

7.2. Recomendações

Os componentes concebidos em programação orientada à objectos podem ser integrados com muita facilidade em projectos posteriores baseados na mesma plataforma (de concepção dos componentes). Não obstante, os mesmos componentes podem ser integrados em projectos concebidos em outras plataformas (Plataformas diferentes da que os componentes foram concebidos).

Neste trabalho não foi possível testar esse tipo de integração daí que se recomenda a continuação deste estudo com vista a alargar mais o conceito de reutilização de componentes integrando-os em aplicações de plataformas diferentes.

8. BIBLIOGRAFIA

1. ARIADNE; **Object Oriented Analysis And Design Using The UML**

2. DEITEL, H. M; (2003) **How To Program**, Fifth Edition, New Jarsey;

3. http://www.webopedia.com/TERM/O/object_oriented_programming_OOP.html; **Object-Oriented-Programming**, (consultado em 24/10/2007)

4. LISKOV, Barbara & GUTTANG, John; (2001) **Program Development in Java.** Addison-Wesley, New York;

5. LAMARN, Graig - **An introduction to Object-Oriented Analysis and Design and Unified Process**, Second Edition;

6. LEITE, Mário. (1998) **Programação Orientada ao Objecto - Uma Abordagem Didáctica**, CESUMAR.

7. MACOME, E. (1995). **Introdução a Metodologia de Investigação**, UEM, Maputo

8. MARGAN, Souza; AURÉLIO, Marco; KAYSER, Vargas Patrícia; DENNY, Azzolin; (1999) **Técnicas para Desenvolvimento de Aplicações Orientadas a Objectos Utilizando a Linguagem Java**; Porto Alegre;

9. PILLAY, Aniban (2007) **Object Oriented Programming Using JAVA**; Durban;

10. RUMBAUGH, James; BLAHA, Michael; PREMERLANI, William, EDY, Frederick; LORENSEN, William. (1994) **Modelagem e Projectos Baseados em Objectos. SP**, Campus.

11. RUMBAUGH, James; JACOBSON Ivar; BOOCH, Grady; (1999) **The Unified Modeling Language Reference Manual**; Addison-Wesley;

12. RUMBAUGH, J; BLAHA, M.; Premerlani, W.; EDDY, F.; LORENSEN, W. (1991) **Object-Oriented Modeling and Design**. Englewood Cliffs, Ed. Prentice-Hall.

13. PRIETO-DÍAZ, Rubén; (1996) **Reuse as a New Paradigm for Software Development;** London.

14. SANTOS, Rafael; **Introdução à Orientação a objectos usando JAVA**;

15. TAVARES Joelma; (2007) **Reutilização de Software;** Porto.

16. VIERA, Pedro; (2002) Introdução à segurança dos sistemas de informação; Lisboa;

17. WHITTEN, Jeffrey; BENTLEY, Lonnie; BARLOW, Victor; **System Analysis And Design Methods,** New York, 1994.