Trabalho de formatura supervisionado - MAC499

aluno: Fábio Pisaruk N.o USP: 3467959 e-mail: pisaruk@gmail.com

orientador: Alan Mitchell Durham

IME USP - CECAE - NAEG - USP

7 de dezembro de 2004

Sumário

1 Introdução

Esta monografia discorre sobre os dois estágios realizados durante o curso de Bacharelado em Ciência da Computação(BCC).

Eles foram realizados em duas entidades da USP(Universidade de São Paulo):

NAEG(Núcleo de Apoio aos Estudos de Graduação) O estágio se iniciou em fevereiro de 2003 prolongando-se por um ano e meio, possuindo uma carga horária diária variada, totalizando cerca de 25 horas semanais, em média.

CECAE(Coordenadoria Executiva de Cooperação Universitária e de Atividades Especiais) Iniciou-se em agosto de 2004, com duração prevista de seis meses, possuindo uma carga horária estimada de, pelo menos, 30 horas semanais.

2 Estágio realizado no NAEG

2.1 Apresentação do NAEG

O NAEG é uma entidade da Universidade de São Paulo responsável por realizar pesquisas e análises estatísticas sobre, e para a Universidade como um todo, além de realizar projetos para institutos ou mesmo pessoas específicas.

Para que os estatísticos do Núcleo possam realizar suas tarefas, é responsabilidade dos estagiários de computação, a manutenção de uma infra-estrutura computacional de suporte: bancos de dados, rede, etc., além de desenvolver interfaces de busca e de disponibilização dos resultados das pesquisas.

2.2 Organização do NAEG

O núcleo possui uma equipe formada por estagiários das áreas de estatística e computação, além de uma secretária, um analista de sistemas e um coordenador.

Durante o estágio, estive sob a supervisão do analista de sistemas do Núcleo: Daniel Oliveira Dantas que, além da sua responsabilidade inerente de avaliar meu trabalho, muitas vezes, participava das atividades propondo soluções.

O acompanhamento do andamento das atividades ocorria pessoalmente, não havia relatórios pois, o núcleo tinha poucos funcionários, permitindo este tipo de abordagem.

2.3 Atividades desenvolvidas

As seguintes tarefas foram desempenhadas durante este estágio:

- Manutenção da rede `Linux`.
- Manutenção da rede `Windows`.
- Instalação e configuração de máquinas com a distribuição `Debian` do `Linux`.
- Instalação e manutenção do banco de dados `Sybase`.
- Construção de páginas `Web` utilizando `PHP`.

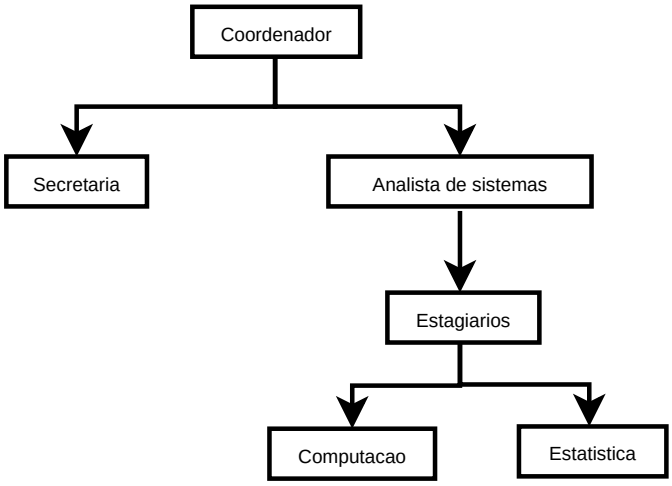


Figura 1: Organograma do NAEG

- Construção de um sistema de gerência e acompanhamento de solicitações, o MetaNaeg.

Uma vez que o sistema MetaNaeg tomou a maior parte do meu tempo durante o estágio, tratarei de dar ênfase total a ele durante este texto.

2.3.1 MetaNaeg: Sistema de Gerenciamento de Solicitações

Histórico

O NAEG é um núcleo que existe há mais de dez anos e, durante este tempo, muitas pesquisas e projetos foram realizados, os quais ficavam organizados em pastas, armários, entre outros meios, o que dificultava a consulta por informações. Além disso, não havia outro meio de se fazer um pedido de projeto ao NAEG que não fosse pessoalmente ao coordenador do Núcleo.

Como o núcleo havia experimentado uma ampliação, tanto no quadro de funcionários quanto no de atividades, surgiu a necessidade de se desenvolver um sistema que fosse capaz de gerenciar as solicitações que entravam nele. Surge então a idéia de se criar o MetaNaeg.

O idealizador do MetaNaeg, o professor Adilson Simonis, coordenador do Núcleo, ansiava por um sistema que o ajudasse na tarefa de gerenciar as atividades realizadas pelo núcleo, razão pela qual solicitou que tal sistema fosse projetado e construído. Embora eu não tenha sido o idealizador, muitas das idéias presentes no sistema também foram de minha autoria passando, é claro, pela aprovação do coordenador e do analista de sistemas.

Estimativas de tempo e acompanhamento

O sistema MetaNaeg passou por diversas fases, começando como um pequeno repositório de projetos e terminado como um sistema complexo de gestão de projetos e acompanhamento de solicitações.

A seguir, uma estimativa grosseira do tempo de desenvolvimento para cada etapa importante da construção do sistema:

1. Primeira versão do sistema(repositório de projetos) - três meses.

2. Versão final(sistema de acompanhamento completo) - seis meses.

O projeto deveria levar nove meses e foi concluído em doze mas, levando-se em conta que, além do sistema, eu possuía outras
atribuições, posso dizer que os prazos foram praticamente cumpridos.

Desafios O principal desafio do sistema foi, sem dúvida, permitir níveis de acesso diferenciados. Era essencial que cada usuário possuísse um certo nível de acesso por razões de segurança e responsabilidade.
Um desafio secundário, mas não menos importante, foi definir o fluxo de ações pelas quais uma solicitação passa dentro do NAEG. Precisávamos escolher os estados de maneira que o processo de gerenciar uma solicitação fosse o melhor possivel. Afinal, o objetivo do sistema era facilitar este controle.

Além destes, existe um outro, que pode ser considerado como dependente dos anteriores: validar a execução de ações no sistema de acordo com o estado da solicitação e os papéis do executor.

Soluções Utilizamos a idéia de papéis no sistema. Para entender melhor esta abstração pense, por exemplo que o sistema seja um carro. Neste caso, existem os seguintes papéis: motorista, passageiro, frentista, mecânico, etc. Cada papel interage com o carro de uma maneira diferentes e, de acordo com o estado em que o carro se encontra: se o carro estiver quebrado então o mecânico interage com ele, se estiver sem gasolina então é a vez do frentista, etc. Cada papel possui um conjunto de ações que pode realizar: o frentista não pode, por exemplo, consertar o carro e, nem o mecânico encher o tanque.
Cada pessoa no sistema recebe um conjunto de papéis e, como cada papel define um conjunto de ações permitidas, temos seu nível de acesso configurado.

Um autômato foi criado para controlar o fluxo de estados pelos quais uma solicitação passa. Para escolher quais estados e transições fariam parte do autômato, foram analisados os estados pelos quais uma solicitação passava atualmente no Núcleo, além de analisarmos a possibilidade de incluir novos estados que pudessem melhorar o gerenciamento das solicitações.

Vale lembrar que, a permissão de execução de uma determinada ação no sistema está ligada ao nível de acesso do executor. Se a ação for sobre uma solicitação, além do nível de acesso temos que analisar o estado no qual ela se encontra pois, cada estado define um conjunto de ações permitidas.

Requerimentos

Os principais requerimentos do sistema foram:

Fazer cadastro de pessoas O sistema deveria fornecer uma interface de cadastro de pessoas.

Existe no sistema pessoas que têm o papel **Gerenciador De Grupos** autorizadas a: adicionar, remover e alterar dados das pessoas.

Além disso, qualquer pessoa poderia obter o seu cadastro via `Web`, sendo que neste caso, ela entraria no sistema com o papel **Solicitante**.

Fazer cadastro de solicitações O sistema deveria fornecer uma interface de cadstro de solicitações.

Todo usuário cadastrado no sistema que possuir os papéis adequados, pode cadastrar uma solicitação.

Gerenciar o fluxo pelo qual uma solicitação passa dentro do NAEG O sistema deveria permitir ao usuário, executar ações que mudassem os estados das solicitações e, garantir que este fluxo [1] fosse uma execução do autômato(2) definido.

Por exemplo: uma solicitação muda do estado *Esperando Aceitação* para *Aceita* quando um usuário realiza a ação Aceitar.

Autenticar usuários via login Para ter acesso ao sistema o usuário precisa efetuar login, que é feito através de uma página web.

Autorização de usuários para a realização de ações Para realizar uma determinada ação é preciso possuir permissão dada pelo nível de acesso.

Uma vez que a solicitação é cadastrada, ela passa por diversos estágios. Durante cada estágio, algumas ações podem ser executadas sobre a mesma. A fim de realizar estas ações, as pessoas precisam possuir determinados papéis. Por exemplo: para se definir a equipe que desenvolverá o projeto, é preciso possuir o papel **Escalador de Equipe** e, a solicitação precisa estar no estado[2] *Aceita*.

A partir disso, observamos que a permissão para execução de ações sobre uma solicitação e/ou projeto depende tanto do seu estado quanto dos papéis do usuário.

[1] Um fluxo, nada mas é que um passeio pelo grafo do autômato partindo-se do estado inicial.

O seguinte passeio: *Esperando Aceitação*, *Rejeitada* e *Não existe mais* é um exemplo de uma execução.

[2] Cada estado do autômato 2 possui um certo conjunto de transições. Por exemplo: no estado *Esperando Aceitação* existem as seguintes transições: Aceitar, Rejeitar e Analisar.

Permitir filtros Deveria existir alguma maneira do usuário filtrar as listagens por diversos campos, permitindo buscas exatas ou aproximadas.

Emitir notificações via e-mail Notificações deveriam ser enviadas para determinadas pessoas quando alguns eventos ocoressem.

Por exemplo: quando uma pessoa é convocada a participar do desenvolvimento de uma solicitação, quando o pedido da solicitação é alterado, etc.

Restringir o acesso às solicitações O NAEG possui alguns dados sigilosos, que não podem ser vistos e, muito menos alterados por qualquer um.

Assim, o acesso às solicitações deveria ser, de alguma maneira, restringido a certas pessoas.

Além disso, deveria existir um papel que tivesse permissão para configurar(dar ou retirar) os acessos.

Algumas nomenclaturas importantes:

Solicitação Todo pedido que entra no NAEG é considerado uma solicitação.

Por exemplo: o professor Adilson Simonis solicita ao núcleo que uma certa pesquisa seja realizada.

Projeto A partir do momento que a solicitação é completamente desenvolvida e, o resultado se torna de alguma

forma disponível ao solicitante, a mesma torna-se um projeto.

Visão geral

O sistema foi desenvolvido em duas partes: código na forma de `stored procedures` 6.1.5 e scripts `PHP`.

Não foi utilizada orientação a objetos. Ao invés disso, foi criada uma estrutura de diretórios e um conjunto de scripts que realizavam ações bem simples.

Estrutura de diretórios:

ajuda: Contém as páginas de ajuda.

alterar: Contém scripts que executam ações como: salvar, alterar, remover dados, além de ações que alteram o estado de uma solicitação: aceitar, rejeitar, etc.

documentação: Documentação do sistema, incluindo o código que está no banco de dados.

buscas: As telas personalizadas de busca. Cada listagem possui uma tela de busca.

detalhes Telas com informações detalhadas sobre pessoas, solicitações e projetos.

imagens Figuras do sistema.

listagens Telas de listagens de pessoas, solicitações e projetos.

misc Tela de login, controle de acessos, além do javascript de ordenação.

novos Telas de cadastro de novas pessoas e solicitações.

relatórios Telas com os relatórios.

utils Alguns scripts com funções de apoio.

É possível notar que esta divisão hierárquica facilitava bastante a localização dos scripts que compunham o sistema. Por exemplo: o script que apaga uma solicitação está localizado no diretório **alterar**, o que exibe uma lista com as solicitações, por sua vez, deve estar no diretório **listagens**, e assim por diante.

Além dos scripts php temos uma série de `stored procedures`, `triggers` e `views` que estão no `Sybase`.
As justificativas para o uso de `stored procedures` foram:

Transações Permitem a criação de transações. Assim, ações que envolvem vários passos podem ser desfeitas caso um erro ocorra em algum deles.

Concorrência Utilizando um esquema de "locks"[3] corretos, chamadas concorrentes às `stored procedures` são gerenciadas pelo `Sybase`.

Isolamento O cliente não executa ações diretamente sobre as tabelas, ao invés disso, executa `stored procedures` que fazem este serviço.

Desta maneira, as tabelas podem ser alteradas sem que o cliente altere seu código. Além disso, como o cliente não pode alterar as

tabelas diretamente, podemos garantir maior consistência dos dados bastando, para isso, codificar corretamente as `stored procedures`.

A validação dos dados foi feita totalmente no `Sybase` utilizando `constraints` e `triggers`.
Esta abordagem foi utilizada por exigir menos codificação no cliente[4] e para permitir que mesmo outros sistemas usem as `stored procedures` do MetaNaeg, sem que os dados sejam corrompidos.

[3] Os "locks" são feitos nas tabelas que as `stored procedures` manipulam.

[4] Aplicação que interage com as `stored procedures`.

Descrição detalhada

Vamos detalhar as partes principais do sistemas:

Papéis e ações

Analisando os requerimentos do sistema, definimos um conjunto de papéis(6.1.2) satisfazendo a quantidade de níveis[5] de acesso desejada.

Começamos com um número pequeno de papéis e, à medida que o sistema evoluiu, os papéis também se multiplicaram. Apenas para exemplificar: o papel **Alterador de solicitação** foi adicinado quando percebemos que existia um ação(alterar pedido da solicitação) no sistema que requeria um nível de acesso diferenciado. Imagine, por exemplo que exista uma solicitação com o seguinte pedido: *Listagem de alunos da FEA que possuem média maior que sete.* Se não houvesse um controle para execução desta ação, o seguinte cenario poderia ocorrer: o texto pedido é alterado e o NAEG acaba desenvolvendo um trabalho inútil. Além disso, o estado em que a solicitação se encontra também influencia na permissão de alteração do pedido. Por exemplo: o solicitante não pode alterar seu pedido se a solicitação estiver sendo desenvolvida, por razões óbvias.

A escolha das ações do sistema foi feita a partir da análise dos requerimentos e, assim como ocorreu com os papéis, aumentou durante evolução do sistema. Por exemplo: quando o sistema foi concebido, não existia a ação Reabrir projeto. Entretanto, a partir do momento em que o sistema entrou em operação, notamos que, muitas vezes, os usuários precisavam reabrir um projeto concluído para alteração de diversos itens. Com isso, decidimos adicionar esta açao.

Observando os diversos casos(6.1.3) de uso do sistema, é possível ter uma boa noção da complexidade envolvida na definição dos papéis e ações do sistema. Toda esta complexidade de refletiu no momento da codificação e, principalmente nos teste pois, era preciso testar cada caso de uso.

Autômato que descreve o fluxo de uma solicitação

Os estados possíveis para uma solicitação estão descritos na forma de um autômato com estado inicial *Esperando aceitação* e final *Não existe no sistema.*

Assim que um **Solicitante** ou **Cadastrador** cadastra uma solicitação, ela entra no estado inicial. A partir daí, pode transitar pelos estados do autômato.

Em cada estado do autômato um conjunto de transições são permitidas, sendo que as transições ocorrem da execução de determinadas ações. Por exemplo: estando no estado *Em Análise* existe uma trasição

[5]Lembre-se que o nível de acesso corresponde ao conjunto de ações que um usuário possui. Uma vez que cada usuário recebe um conjunto de papéis, seu nível de acesso fica definido.

para o estado *Rejeitada* que ocorre da execução da ação Rejeitar.

Durante o desenvolvimento do sistema, o conjunto de estados cresceu consideravelmente, a fim de abranger a realidade na qual o sistema estava embutido. Por exemplo: o estado *Apagada* foi adicionado para evitar que uma solicitação fosse apagada por engano.

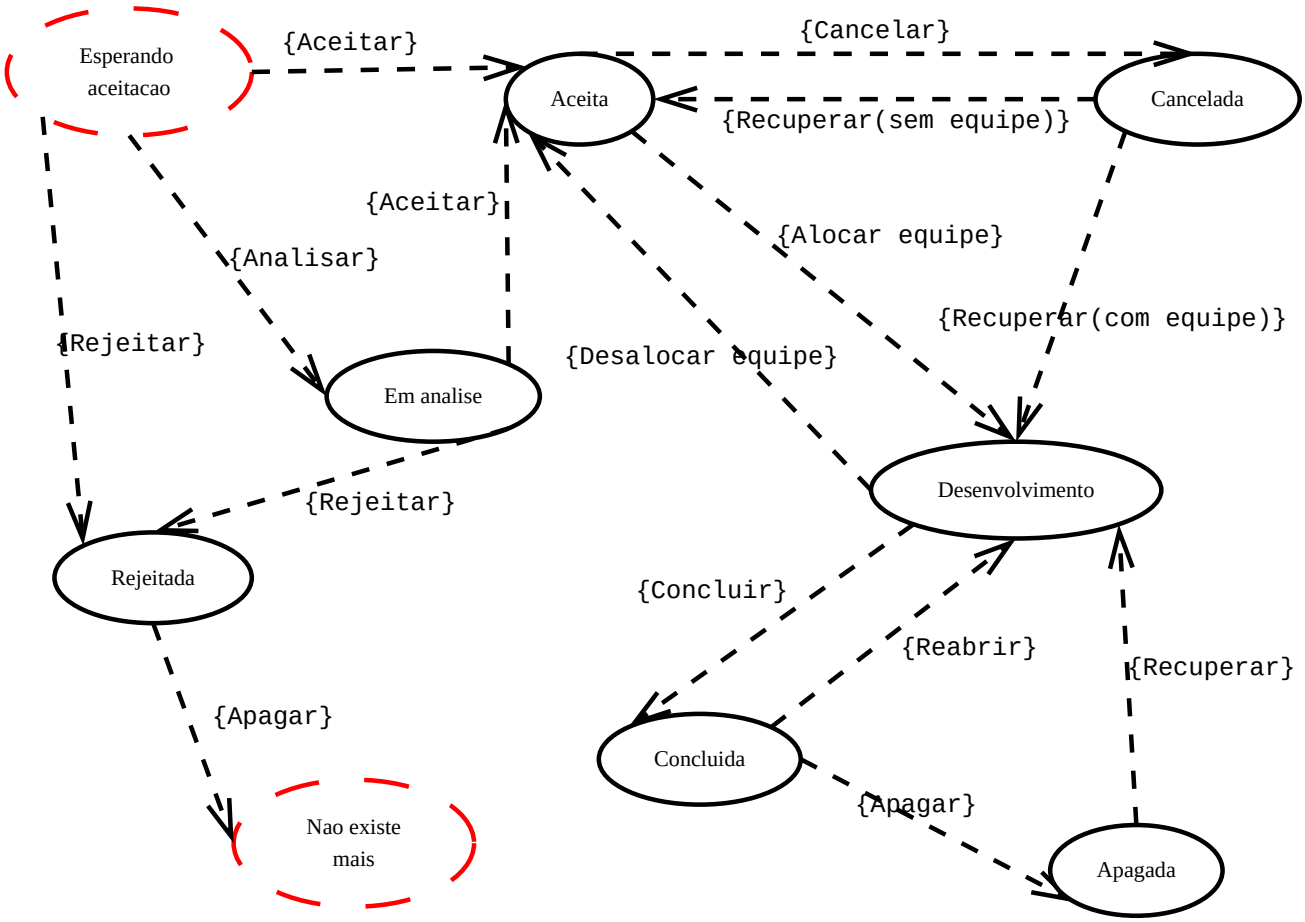


Figura 2: Diagrama que descreve os possíveis fluxos de uma solicitação no NAEG

Sistema de busca/filtragem Embora o sistema de busca[6] possa ser usado em diversas aplicações, ele foi desenvolvido para ser utilizado primeiramente pelo MetaNaeg, razão pela qual será descrito como parte integrante do mesmo
. O sistema de busca é responsável por criar uma cláusula "WHERE" a partir de um texto de busca, uma tabela e um conjunto de campos de busca.
Ele possui dois tipos de usuários:

Desenvolvedor de site Pessoa que cria as listagens de um sistema.

Ele deve criar uma listagem, configurar no sitema de busca quais campos da listagem são buscáveis e, prover um meio de enviar ao sistema de busca o texto de busca.

Usuário do site Quem executa as filtragens passando ao sistema de busca o texto de busca.

Vamos tomar o seguinte exemplo de uso: o desenvolvedor de um site cria uma listagem onde ele deseja que o usuário possa aplicar filtros.
Suponha que a listagem esteja baseada na tabela de Solicitações e, que ele tenha cadastrado os seguintes campos como buscáveis: id, nome, responsável, idade, data cadastro, data entrega.
Quando o usuário do site aplicar o filtro, o sistema de busca precisará receber os seguintes dados:

nome da tabela Tabela onde a busca será realizada.

separador String que separa os elementos do texto de busca.

caracter de união O sistema não separa elementos que estejam delimitados por este caracter. O padrão é usar aspas como caracter de união.

texto da busca O texto passado pelo usuário, o qual será utilizado na construção da cláusula "WHERE".

No momento em que um usuário do site criado realizar uma busca, a string de busca é passada ao sistema juntamente com o nome da tabela.
A partir daí, ela é analisada, dividida em elementos menores e a cláusula "WHERE" construída. Uma vez retornada a cláusula "WHERE", a listagem precisa ser recarregada utilizando-a como filtro.
Doravante chamaremos de *tabela de busca* a tabela cadastrada pelo desenvolvedor do site que é passada para a função
"montaQuery[7]" e, de *campo de busca* a cada um dos campos selecionados desta tabela que devem entrar

[6] Busca e filtro estão sendo usados como sinônimos neste texto.

[7] Função principal do sistema de busca.

na busca.

O funcionamento da principal função do sistema de busca, a "montaQuery", pode ser assim resumido:

Divisão O texto de busca é dividida em outras menores utilizando o caractere separador passado e, respeitando um caractere
de união que define que todo o conteúdo dentro deste limitador não deve ser separado.

Classificação Cada elemento é analisado quanto ao tipo. Os tipos são: texto, inteiro, real, bit e data.

Relacionamento Cada tipo primitivo do `Sybase` está mapeado para um dos cinco tipos acima. Desta maneira, nesta etapa, para cada elemento do texto de busca, escolhe-se todos os campos de busca com o mesmo tipo e gera-se um cláusula "WHERE" relacionando ambos.

Considere que o texto de busca seja *Fabio 12 "Fabio Pisaruk"*.

Campos buscáveis da tabela

campo	**tipo**
id	inteiro
nome	texto
responsável	texto
idade	inteiro
data cadastro	data
data conclusão	data

Classificação da string de busca

elemento	**tipo**	**campos relacionados**
Fabio	texto	nome, responsável
12	inteiro	id
Fabio Pisaruk	texto	nome, responsável

Sendo assim, a cláusula "WHERE" gerada pelo sistema de busca seria:

(UCASE(nome) LIKE "%FABIO%"OR UCASE(responsável) LIKE "%FABIO%") AND
(id = 12) AND
(UCASE(nome) LIKE "%FABIO PISARUK%"OR UCASE(responsável) LIKE "%FABIO PISARUK%")

É possível realizar buscas exatas ou aproximadas. Caso se opte por procuras exatas então o *LIKE* é substituído por = e há distinção entre maiúsculas e minúsculas.

O sistema de busca foi utilizado em praticamente todas as listagens[8] do MetaNaeg.

Ordenação via `javascript`

[8] Veja as listagens 6.1.1 do sistema para entender melhor o sistema de busca.

Foi utilizado um pequeno `javascript` a fim de permitir que as colunas das listagens fossem ordenadas pelo próprio usuário. Optamos por usar `javascript` para não sobrecarregar o servidor com consultas que retornassem dados ordenados.
O uso desse script se mostrou muito eficiente, sendo que a única desvantagem estava na grande quantidade de codificação necessária para seu uso.

Tecnologias utilizadas

Na criação do sistema foi utilizado `php3`, `Sybase` e `JavaScript`.
Foi necessário desenvolver o sistema utilizando estas ferramentas pois, toda a infraestrutura do NAEG estava fundamentada nelas.
Desta maneira, não há como argumentar pela escolha, uma vez que não houve uma.

Equipe de desenvolvimento

O projeto foi desenvolvido por mim e pelo Guilherme Tozo de Carvalho, aluno do BCC / IME.
O Guilherme chegou a participar do desenvolvimento quando o MetaNaeg ainda era apenas um repositório de projetos, ou seja, quando ainda não havia um controle de fluxo e nem de permissões para execução de ações.

Melhoras possíveis

Existem muitas melhoras possíveis, dentre elas:

Sessões: Utilizar sessões ao invés de `cookies` para armazenar informações do usuário.
Quando o sistema foi construído só dispúnhamos do `php3` que não possui tal recurso.

Validação: Como as permissões para execução das ações são validadas no próprio código e, estão repetidas em diversos scripts, seria interessante agrupá-las de alguma maneira a fim de evitar repetição e facilitar a manutenção do sistema.

Segurança: Utilizar `SSL` sobre `HTTP` a fim de validar os dados de autenticação com mais segurança.

Código no `Sybase`: Retirar um pouco das regras de negócio que estão no Sybase, na forma de stored procedures, e passá-las para o php.
No momento do desenvolvimento, foi necessário lançar mão de `stored procedures` a fim de resolver problemas de corrupção de dados e concorrência. Uma vez que a nova versão do `php` está mais robusta e, permite tratar desses aspectos, vale a pena mover parte do código que roda no

Sybase para o PHP.

Vale aqui uma observação pertinente: este tipo de decisão deve ser norteada por justificativas que passem longe da preferência do programador. Deve-se levar em conta aspectos palpáveis e coerentes. Por exemplo: seria interessante remover grande parte do código que roda no Sybase, caso o sistema fosse projetado para funcionar em diversos Sistemas de Banco de Dados diferentes. Caso contrario, vale a pena aproveitar ao máximo as ferramentas disponibilizadas pelo Sybase.

Busca aproximada: Inserir no Sybase uma função escrita em Java que retire acentos das palavras, a fim de

permitir que buscas aproximadas sejam realizadas também em palavras acentuadas.

Estado atual

O sistema, praticamente finalizado , está em pleno uso no NAEG satisfazendo, satisfatoriamente, às necessidades dos seus usuários.

2.4 Dificuldades e frustrações

Durante o estágio encontrei diversas dificuldades. As principais estiveram relacionadas às ferramentas disponíveis para a construção
do sistema, das quais se destacaram: `Sybase 11.9` e `PHP3`.

Problemas com o `Sybase 11.9`:

Edição de colunas: Esta versão não permitia que colunas fossem removidas ou tivessem seus tipo alterados, dificultando muito as alterações que se fizeram necessárias durante a evolução do sistema.

Criação de funções embutidas: Estou chamando de funções embutidas, qualquer função que possa ser aplicada diretamente num campo da query. Por exemplo: a função "UCASE" é uma função embutida, uma vez que eu posso realizar a seguinte query "SELECT UCASE(nome) FROM Pessoas".

Se o `Sybase 11.9` permitisse a criação de tais funções então, um problema que o sistema de busca tinha em realizar buscas
aproximadas poderia ser resolvido. Neste tipo de busca, os acentos[9] devem ser ignorados, ou seja, uma pesquisa pela palavra "Fábio" deveria retornar campos com valores: "*Fabio*", "*Fábio*".

A versão mais nova do `Sybase(12.5)`, permite a criação de funções embutidas em `Java`, desta maneira, o problema anteriormente citado, poderia ser facilmente resolvido.

O `Sybase 11.9` possuía outras limitações mas, estas foram as que mais dificuldades trouxeram ao desenvolvimento do sistema.

Problemas com o `PHP3`:

Sessões: Esta versão do `PHP` não possuía este recurso logo, precisei recorrer ao uso de cookies que, além de tornarem o código menos legível, ainda trouxeram problemas de segurança ao sistema.

Recarregamento de páginas: Ocorriam diversos erros ao se utilizar o sistema de busca com esta versão do `PHP`, tanto que quando o `PHP4` foi instalado os problemas desapareceram.

Classes: Por fornecer pouco suporte ao uso de classes, a opção foi de não utilizá-las.

[9] Além de ignorar acentos, a busca aproximada não diferencia maiúscula de minúsculas mas, isso foi realizado sem problemas

Biblioteca de funções: Diversas funções que existem no `PHP4` não estão presentes no `PHP3`, por exemplo: algumas funções que realizam operações sobre "strings".

Algumas vezes foi necessário criar funções que já existiam na versão mais nova do `PHP` o que foi uma experiência desanimadora.

Foi realmente frustrante perceber que, muita força de trabalho tinha sido gasta tentando-se contornar algumas limitações das ferramentas utilizadas, ficando claro que se versões mais novas tivessem sido usadas, o tempo poderia ser melhor aproveitado, na construção de um sistema melhor.

Isto, porém, troxe à tona a seguinte reflexão: as ferramentas estão em constante evolução e, sempre haverá problemas e limitações com as mesmas,
logo, é preciso saber lidar com este fato. Desta maneira, posso dizer que foi bom me deparar com esta realidade e poder contorná-la.

3 Estágio realizado na CECAE

3.1 Apresentação da CECAE

A CECAE é uma entidade da USP, cujo objetivo é potencializar a interação entre a USP e a sociedade, por meio de parcerias internas a Universidade e com entidades da sociedade, delineando e implantando protótipos de soluções inovadoras para questões sociais e econômicas relevantes, para desenvolvimento de metodologias e demonstração.

3.2 Organização da CECAE

A CECAE é composta por coordenadores, diretores, estagiários e representantes de empresas associadas. Além disso, existem parceiros externos como por exemplo

3.3 Atividades desenvolvidas

O professor Adilson Simonis havia mostrado o MetaNaeg para o pessoal da CECAE e, uma vez que houve interesse, fui convidado a apresentar uma demonstração do MetaNaeg para os diretores do CECAE. Visto que o sistema despertou-lhes o interesse, aceitei a oferta de estagiar na CECAE com o objetivo de desenvolver um sistema similar de gestão de projetos/programas.
O estágio consistiu em desenvolver este sistema, o SGT(Sistema de Gerenciamento de Tarefas).

3.4 Histórico

A CECAE, atualmente, possui diversos meios de organizar os seus projetos/programas: fichas em papel, pequenos sistemas, planilhas,
fóruns de discussão, entre outros. Uma vez que essas abordagens não estavam mais sendo sendo suficientes para gerenciar e crescente quantidade de informações, surgiu a necessidade de um sistema que fizesse este trabalho.
O diretor da CECAE, Sérgio Oliva, precisava ter em suas mãos uma ferramenta que lhe fornecesse

informações sobre os projetos/programas, tais como: tarefas, pessoas envolvidas, estágio de desenvolvimento, além de manter um controle sobre o fluxo das tarefas de um projeto/programa.

3.5 Desafios

Diferentemente do sistema MetaNaeg, o que está sendo implementado para a CECAE, possui papéis dinâmicos por programa/projeto.
Lembre-se que no MetaNaeg, os papéis não são definidos por solicitação, ou seja, se uma pessoa tem o papel de **Gerente de Projeto**, ela o tem para qualquer solicitação do sistema. Isto funcionava razoavelmente[10] bem no NAEG mas, na CECAE, era impraticável por haver um número grande de funcionários e, porque cada um podia desempenhar papéis diferentes em cada programa/projeto.

No MetaNaeg, as etapas pelas quais as solicitações passam estão descritas por um único autômato, o que não podia ser feito para a CECAE.
Deveríamos permitir que cada programa/projeto possuísse um autômato de tarefas customizável pelos usuários pois, esta era a realidade de trabalho da CECAE.

O conceito de tarefa também não existia no NAEG mas, na CECAE, este conceito é essencial. Um programa pode ser descrito com uma série de tarefas relacionadas por um autômato. Cada tarefa possui uma equipe de desenvolvimento, responsável pela sua execução.

As tarefas deveriam possuir fichas[11] definidas pelos usuários, sendo que elas poderiam ser alteradas mesmo depois que a tarefa tivesse sido executada e a ficha tivesse sido preenchida[12]. Só para relembrar, no MetaNaeg os campos que deveriam ser preenchidos em cada solicitação eram fixos.

Por fim, algo que começou a ser implementado no MetaNaeg mas, não foi finalizado: um esquema de acompanhamento e notificação. Por exemplo: uma tarefa tem um prazo para ser completamente executada, a partir do qual, o coordenador do programa recebe uma notificação de atraso.

[10] Principalmente pelo fato do NAEG possuir poucos funcionários

[11] Este conceito será explicado mais tarde mas, só para adiantar, uma ficha é um modelo do questionário que deve ser preenchido durante a execução da tarefa.

[12] Cada execução da tarefa possui uma ficha preenchida

3.6 Soluções

Para lidar com o problema dos papéis dinâmicos decidimos que, ao ser inserido num projeto/programa, a pessoa já entra com uma determinada responsabilidade(papel) no mesmo. Por exemplo: num programa, existe um coordenador, estagiários e, um supervisor.

Para validar a execução de ações dentro do programa, verificamos o papel que o executor possui neste programa. Como foi utilizado J2EE, papéis e permissões para execução das funções podem ser definidas de maneira declarativa, eliminando a necessidade de misturar estas validações com o código da aplicação.

Como cada programa/projeto deveria possuir um autômato de tarefas definido pelo usuário, a idéia foi permitir que o usuário os criasse da seguinte maneira:

- Cadastrando as tarefas.
- Definindo as tarefas iniciais e finais.
- Para cada tarefa, escolher quais serão as tarefas subsequentes.
  Por exemplo: imagine uma tarefa chamada Atender Telefone. Após realizá-la, as possíveis tarefas a serem executadas seriam: encaminhar ligação, descartar ligação, etc.

Deixamos que o coordenador de um projeto/programa crie o seu próprio autômato.

A questão das fichas dinâmicas foi resolvida com o uso do sistema `VODU` pois, ele permite a criação de fichas utilizando `XML`.

A implementação das notificações foi feita com o uso de `Timer Services`. Para cada notificação é criado um `Timer Services` com duração especificada. Assim que ele expira, o interessado é informado e realiza a ação desejada. Por exemplo: quando uma tarefa começa a ser executada ela cria um `Timer Service` com tempo de duração igual ao tempo máximo para a conclusão da tarefa. Se ela não for concluída a tempo, o `Timer Service` notificará a tarefa que, por sua vez, tomará as atitudes necessárias.

3.7 SGT(Sistema de Gerenciamento de Tarefas)

3.7.1 Requerimentos

Durante diversas reuniões com diretores e coordenadores da CECAE, os seguinte requerimentos foram coletados:

- Permitir, via web, os seguintes cadastros:
  1. Pessoas.
  2. Projetos.
  3. Programas.
  4. Tópicos e mensagens.
  5. Fichas.
- Permitir a criação de um fluxo de tarefas para cada projeto/programa.
- Controlar permissões e acessos.
- Permitir a criação de fichas dinâmicas para cada tarefa.
- Permitir o envio de arquivo a cada uma das execuções de tarefa.
- Notificações, via e-mail, geradas por certas ações tomadas no sistema.

Antes de prosseguirmos, vamos explicar alguns conceitos:

Programa versus Projeto A CECAE possui programas e projetos sendo que diferem entre si principalmente no que diz respeito à duração: o primeiro possui longa duração enquanto o segundo não. Existem outras diferenças conceituais e de hierarquia entre eles mas, *grosso modo*, esta é a grande diferença.

Por exemplo: existe o programa Disque Tecnologia que, *grosso modo*, funciona como um serviço de tira-dúvidas.

Fluxo de tarefas Tanto programas quanto projetos, possuem um fluxo de tarefas que pode ser visto como um autômato[13].

Por exemplo: imagine que exista um projeto que trata da construção e venda de uma casa.

[13]Semelhanças com o MetaNaeg não são meras coincidência. Lembre-se que fui convocado a construir um sistema similar ao MetaNaeg

Para construir uma casa, diversas tarefas devem ser realizadas: procurar terreno, comprar terreno, desenhar planta, cotar material, comprar material, alocar pessoal, construir, demolir e vender. É fácil notar que essas tarefas devem ser realizadas numa certa ordem. Neste exemplo simples, também notamos que a execução de uma tarefa pode levar a execução de uma tarefa anterior, por exemplo: imagine que a casa esteja com vazamentos, neste caso, seria necessário demolir, cotar material, comprar material, alocar pessoal e construir novamente a casa.

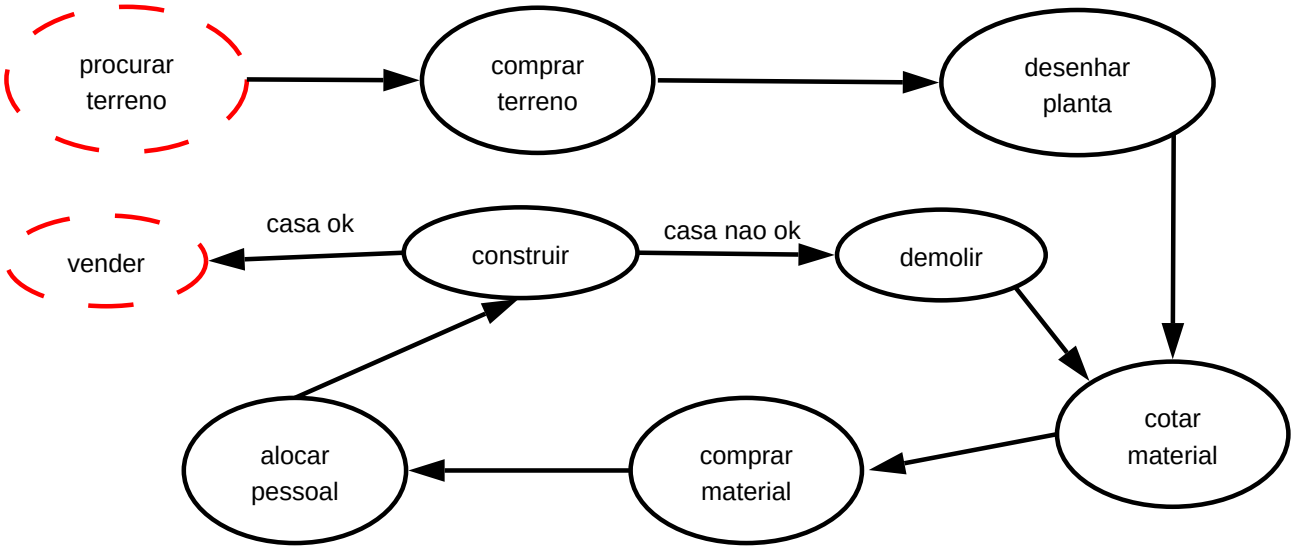


Figura 3: Exemplo de um autômato de tarefas para a construção e venda de uma casa

Tarefa versus Execução Uma tarefa é um modelo que deve ser executado. Por exemplo: suponha que exista uma tarefa chamada Atender Telefone e que ela possua os seguintes campos na sua ficha: nome de quem ligou e dúvida. Cada vez que uma pessoa atender ao telefone ela estará executando esta tarefa e, preenchendo a ficha com dados diferentes. Cada uma destas instâncias, criada usando-se a tarefa como modelo, é uma execução.

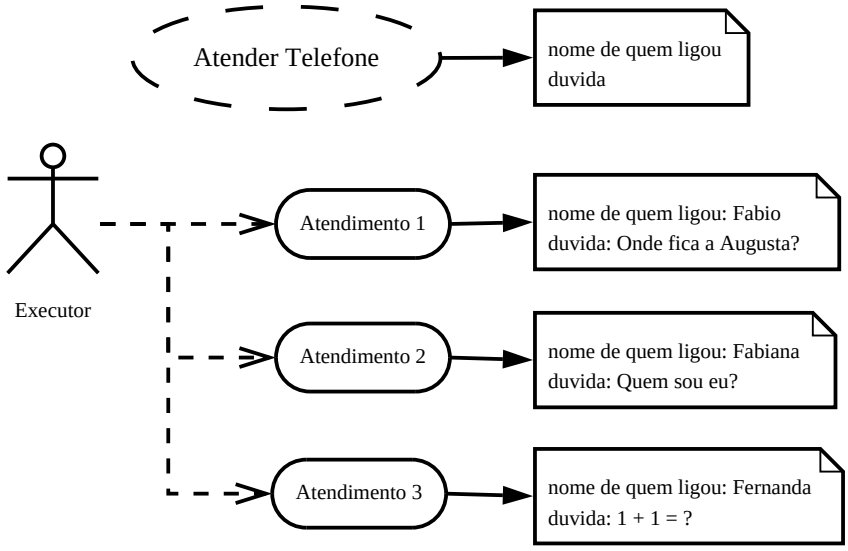


Figura 4: Exemplo de execuções da tarefa Atender Telefone

3.7.2 SGT: Visão geral

O SGT é composto de duas partes principais:

Camada de negócio[14]**:** Toda a lógica de negócio é implementada utilizando "EJBs[15]". Este código roda no servidor de aplicações, no caso o `JBoss`.

A lógica de negócio é implementada por `Entity Beans`(persistência), `Session Beans`(controle de fluxos) e objetos auxiliares(`Value Objects, Util Objects.`).

Camada de apresentação[16]**:** É composta pelas interfaces a partir das quais o cliente usa o sistema.

Para desenvolvê-la utilizamos: `jsp`, `html`, `JavaScript`.

Tratamos de não colocar nenhuma regra de negócio nesta camada da aplicação, a fim de permitir a criação de outros tipos de cliente sem a necessidade de reprogramar as regras.

O sistema foi construído utilizando a arquitetura `J2EE`[17] pois, durante o desenvolvimento do MetaNaeg, nos deparamos com diversos problemas que, utilizando-a, poderiam ser resolvidos mais facilmente.
As principais razões desta desta escolha foram:

Lógica versus Apresentação É mais fácil fazer a separação entre a lógica do negócio e a apresentação ao cliente.

[15] Enterprise JavaBeans: são os componentes que rodam no servidor de aplicações.
[17] Java Enterprise Edition

Transações O servidor de aplicações fica encarregado de gerenciar as transações.

Concorrência O servidor de aplicações também cuida deste aspecto. O desenvolvedor nem deve inserir código[18] que manipula concorrência dos "EJBs".

Segurança É possível definir permissões de acesso e execução de ações com o uso de papéis.
O importante é notar que isto é feito de maneira declarativa, à parte do código da aplicação.
No caso do MetaNaeg a validação das permissões estava no próprio código.

Persistência O servidor de aplicações faz a persistência automaticamente utilizando-se Persitência Gerenciada pelo Container("CMP").

Código "limpo" A maior parte destes aspectos pode ser configurada em arquivos chamados "deployment descriptors".
Desta maneira, o código do sistema fica livre de código que trata destes aspectos ortogonais[19] à aplicação.
Além disso, podemos alterar os "deployment descriptors", mudando assim alguma característica do sistema sem, no entanto, precisarmos alterar uma linha de código do mesmo. Por exemplo: podemos alterar uma permissão de execução de uma determinada função, sem precisarmos alterar código do sistema, bastando para isso, alterar o "deployment descriptor".

Desta maneira, podemos notar que o desenvolvedor pode se concentrar na lógica do negócio, deixando a cargo do servidor de aplicações a gerência dos aspectos ortogonais à aplicação citados. Além disso, como no código do sistema não encontramos código que trata de aspectos ortogonais à aplicação, o código torna-se mais fácil de ser alterado e entendido.

3.7.3 SGT: detalhes

Durante o desenvolvimento, algumas arquiteturas foram propostas e analisadas, sendo que duas delas mereceram destaque, por terem sido implementadas.

Apenas relembrando, a aplicação cliente é responsavel por disponibilizar uma interface gráfica para interação do usuário com o sistema e, a aplicação servidora, é responsável por efetuar as tarefas do

[18] O uso de métodos synchronized não é permitido.
[19] Segurança, concorrência, entre outros.

sistema: regras de negócio, validações, persistência, etc.

A principal diferença entre as arquiteturas propostas está no tipo de acesso entre a aplicação cliente e a servidora:

Local Neste caso, ambas devem estar rodando no mesmo servidor de aplicações e, a comunicação entre elas é feita apenas com o uso de interfaces locais.

Remoto Cada uma pode estar rodando em servidores de aplicações diferentes e, a a comunicação entre elas, ocorre com o uso de interfaces remotas.

Não existe um tipo de acesso melhor, isto é decidido pela especificação do sistema. Se houver a necessidade de distribuição de carga, por exemplo, então é interessante que se use acesso remoto e que ambos fiquem em servidores diferentes, caso contrário, é melhor usar acesso local por ser menos dispendioso, do ponto de vista computacional.

Arquitetura para acessos remotos Primeiramente desenvolvi uma arquitetura que permitia o uso de clientes remotos, ou seja, que estavam rodando em outros servidores de aplicações.
Esta abordagem se deveu ao fato do NetBeans possuir um ambiente excelente de depuração, além de possuir um servidor `TomCat` embutido. Desta maneira, os `Servlets` rodavam no servidor `Tomcat` acessando os "beans" que estavam no servidor `JBoss`.

Para que este acesso remoto fosse bem sucedido, foi necessário que os beans possuíssem interfaces remotas e estivessem registrados no servidor "JNDI".
Além disso, era necessário que a aplicação cliente(cliente `Web` que roda no `TomCat`) possuísse estas interfaces.
Por isso, elas foram disponibilizadas através de um arquivo "jar".

Nesta arquitetura, o acesso aos "beans" está bem mascarado, de forma que o cliente nem sabe que está utilizando-os.
Para acessá-los, utiliza-se uma classe utilitária e, os dados dos `Entity Beans` são passados da aplicação cliente a servidora(e vice-versa) na forma de `Value Objects`.

Para o desenvolvimento do sistema, foi preciso criar uma série de "beans": `Entity` e `Session`, responsáveis pela persistência de dados e por controlar fluxos de transformações, respectivamente.
Além disso, padrões `J2EE` foram utilizados para garantir maior flexibilidade, poupar esforço e melhorar a performance.
Os principais padrões utilizados foram: `Remote Adapter`, `Value Object` e `Facade`.
O primeiro existe mais para poupar o trabalho de criar um "bean"(localizá-lo e retornar um `EJBObject`), o segundo está relacionado à performance, por diminuir o tráfego de rede e, o terceiro aumenta a flexibilidade[20] do sistema.
Os dois primeiro padrões foram criados diretamente por uma ferramenta de geração de código chamada `XDocLet`.
Para isto, bastou definir algumas "tags" para que, as classes que implementavam os padrões, fossem geradas automaticamente.

Classes[21], a partir das quais, as outras[22] foram criadas:

`PessoaBean:` Um `Entity Bean` que representa uma pessoa no sistema. Ela é um objeto persistente gerenciado pelo servidor de aplicações.

`PessoaAgentBean:` Uma Fachada para um `PessoaBean`.

O uso de uma fachada, permite que a classe `PessoaBean` seja alterada sem que o cliente precise mudar seu código, já que ele acessa apenas a fachada.

[23]

Vamos analisar algumas classes deste diagrama, lembrando que com exceção das classes: `PessoaBean` e `PessoaAgentBean`, todas as outras foram geradas automaticamente pelo `XDocLet`.

A partir da classe `PessoaBean`, o `XDoclet` gerou as seguintes classes:

`PessoaCMP:` Estende a classe `PessoaBean`, implementa os métodos da interface `Entity Bean` que não

[20] Como a aplicação cliente só se comunica com as Fachadas que, por sua vez se comunicam com os `Entity Beans`, uma alteração na estrutura dos `Entity Beans` não acarreta a necessidade de mudança na aplicação cliente.

[21] Classes que foram codificadas manualmente

[22] Classes utilitárias.

[23] O uso de uma fachada para cada `Entity Bean` não é recomendado. Neste exemplo, criamos uma fachada que acessa apenas um "bean" mas, no sistema, a maior parte das fachadas opera sobre diversos `Entity Beans`.

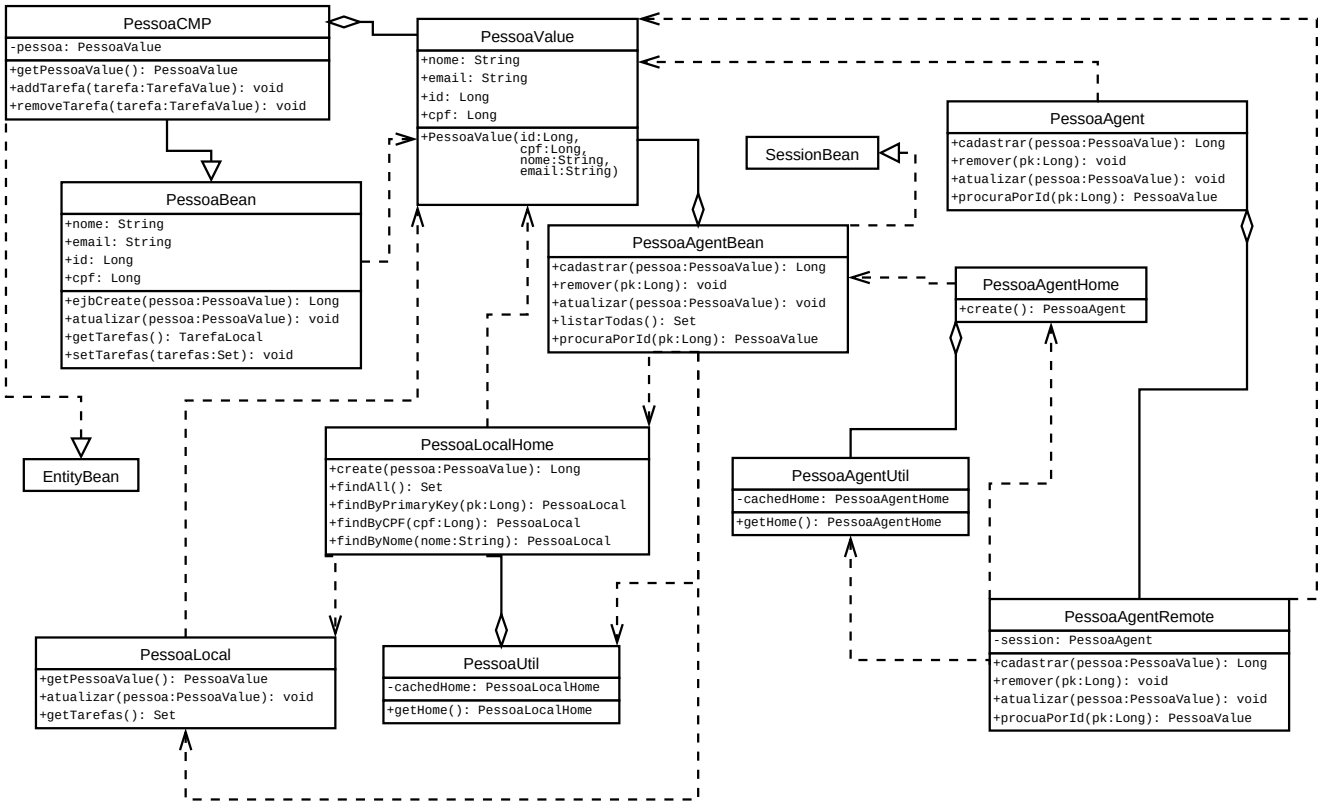


Figura 5: Diagrama UML simplificado mostrando as classes que estão diretamente relacionadas com as classes: `PessoaBean` e `PessoaAgentBean`

foram implementados pela classe `PessoaBean` e, realiza algumas operações com os `Value Objects`, dentre elas, retornar uma `Pessoa` como uma `PessoaValue`.

`PessoaLocal:` Interface "local" da classe `PessoaBean`.

`PessoaLocalHome:` Interface "home" da classe `PessoaBean`.

`PessoaUtil:` Possui métodos para retornar a interface "local home" da classe `PessoaBean`.

Além disso, ela guarda numa variável a classe "home" retornada, funcionando como um "cache".

`PessoaValue`[24]`:` Agrupa alguns campos da classe `PessoaBean` a fim de poder transmitir os dados da classe `PessoaBean` empacotados pela rede, de modo a diminuir o tráfego de rede.

As seguintes classes foram geradas pelo `XDoclet` a partir da classe `PessoaAgentBean`:

`PessoaAgent:` Interface remota do bean PessoaAgent.

`PessoaAgentHome:` Interface home do bean PessoaAgent.

`PessoaAgentUtil:` Classe utilitária do bean PessoaAgent que possui métodos para retornar sua interface home, além de manter um cache do home retornado.

`PessoaAgentRemote:` `Remote Adapter` para o `PessoaAgentBean`. Esta classe é acessada diretamente pela aplicação cliente e, é responsável por repassar as chamadas ao "bean" `PessoaAgent`. Para cada chamada que ela recebe, as seguintes tarefas são realizadas:

1. Obtém a interface "home" do "bean".
2. Cria um objeto através da interface "home".
3. Repassa a chamada ao "bean".
4. Recebe a resposta e retorna ao cliente.

Os primeiros dois passos são executados apenas na primeira chamada pois, a classe guarda um "cache" com o objeto criado, para repassar futuras chamadas ao mesmo.
Esquecemos de mencionar que a classe `PessoaagentRemote` é `Singleton`.

Arquitetura voltada ao acesso local Com o passar do tempo, notamos que o uso de acessos remotos não era necessário e, apenas trazia ônus ao sistema.
Decidimos então, utilizar apenas acessos locais e, deixar a aplicação servidora e cliente rodando no mesmo servidor de aplicações.
Entretanto, caso no futuro o acesso remoto seja desejável não parece ser muito custoso implementá-lo[25].
Nesta nova arquitetura, o padrão `Remote Adapter` foi excluído. Desta maneira, a aplicação cliente precisa ter em mãos, além das interfaces "locais" e "home", as classes utilitárias[26].

Na arquitetura anterior, a aplicação cliente tinha acesso às fachadas indiretamente, através dos Adaptadores Remotos. Agora, o acesso é direto, não há mais um despachante[27].

[25] Observe como os dois diagramas são parecidos para se convencer disto

[26] Na verdade, na arquitetura anterior, as classes utilitárias também precisavam ser passadas à aplicação cliente(via "jar") mas, eram apenas utilizadas pelas classes que implementavam os `Remote Adapters`

[27] Os Adaptadores remotos são bem mais que meros despachantes, elas escondem do cliente a complexidade das chamadas aos "beans"

Como ocorreu na arquitetura anterior, a aplicação cliente continua trocando dados dos `Entity Beans` com a aplicação servidora através de `Value-Objects`.

A seguir, a estrutura de classes utilizada. Neste caso, com execeção das classes: `PessoaBean`, `PessoaFacadeBean`, todas as outras foram geradas automaticamente pelo `XDocLet`.

Vale lembrar que a principal diferença entre as classes `PessoaFacadeBean` e `PessoaAgentBean` reside no fato da primeira ser um `Session Bean` "Stateful"[28] e a segunda "Stateless"[29].

Por ser bastante parecido com o diagrama da arquitetura anterior, não achamos necessário uma explicação mais prolongada. Note que a classe `PessoaFacadeBean` possui um atributo chamado `PessoaLocal`. Esta é a chave da diferença entre as arquiteturas.

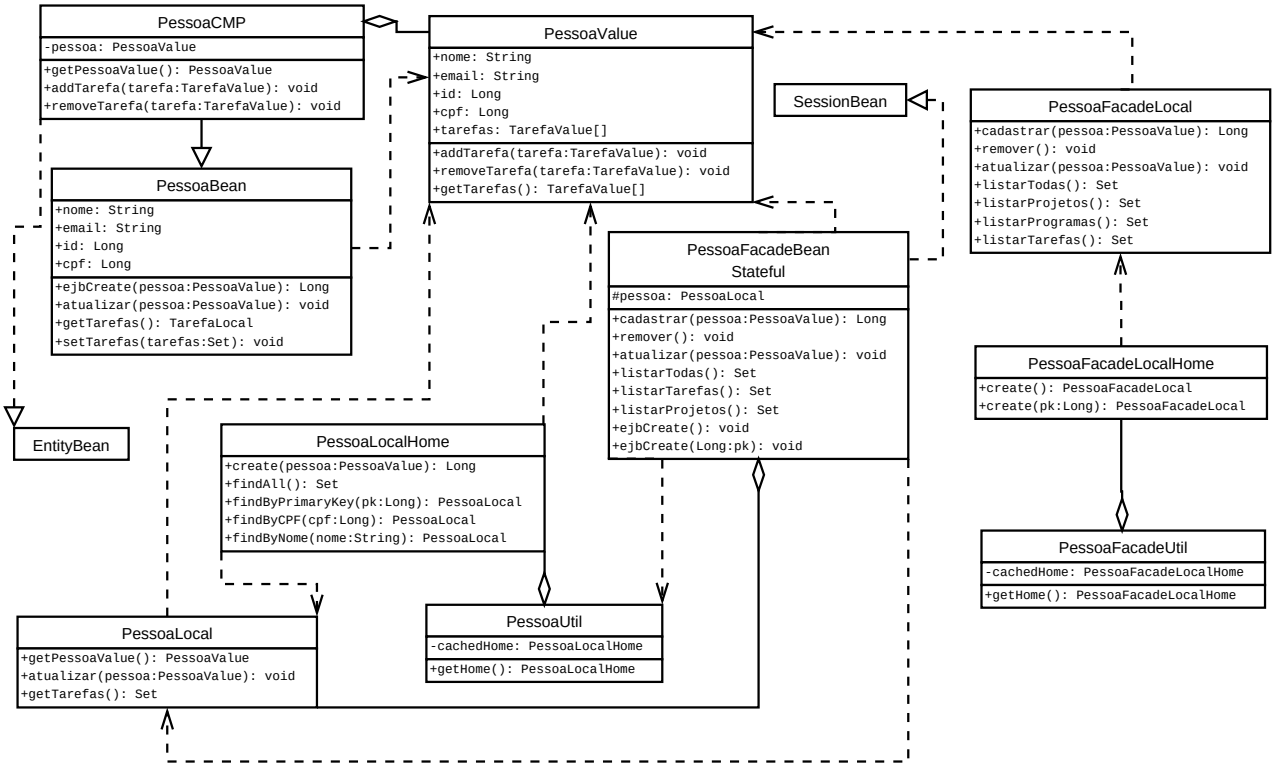


Figura 6: Diagrama `UML` simplificado mostrando as classes que estão diretamente relacionadas com as classes: `PessoaBean` e `PessoaFacadeBean`.

[28] Mantém estado entre as chamadas de cada cliente

[29] Não mantém.

Esta variável armazena o `Entity Bean`[30] que a fachada está manipulando e, para o qual está repassando as chamadas.
Isto é uma grande vantagem pois, se várias chamadas forem realizadas por um mesmo cliente, teremos um ganho de performance, uma vez que não é necessário localizar o "bean" novamente.

Value-Objects Consideramos importante detalhar melhor como os `Value-Objects` foram utilizados no SGT.

Um `Value-Object` é uma classe utilitária, criada a partir de um `Entity Bean`, que possui como atributos os campos persistentes do "bean", além de prover métodos de manipulação dos relacionamento do mesmo.

Criamos, para cada `Entity Bean`, uma série de `Value-Objects`, cada qual possuindo um subconjuntos dos campos persistentes do "bean".

No exemplo a seguir, o "bean" `TarefaBean` possui diversos atributos: nome, nomeFicha, equipe e execuções. No entanto, cada `Value-Object` contém apenas um subconjunto destes campos.
Esta divisão dos campos, se justifica pelo fato de que, nem sempre precisamos lidar com todos os campos do "bean". Por exemplo: numa tela que permite a escalação de pessoas para uma tarefa, não precisamos das execuções da mesma logo, não há razão para desperdiçar recursos computacionais alocando um `Value-Object` que contenha estes dados.

Relacionamentos O sistema é composto por uma série de `Entity Beans`, cada qual representando uma entidade persistente. Para que o sistema atenda aos requerimentos foi preciso relacioná-los de uma certa maneira.
A seguir podemos visualizar alguns relacionamentos entre os "beans": Pessoa, Projeto, Programa, Arquivo, Tarefa, execução, Topico e Mensagem. O servidor de aplicações controla todos esses relacionamentos, bastando para isso, que algumas regras sejam seguidas.

Papéis e ações permitidas

Como foi dito anteriormente, a permissão de execução das ações está ligada aos papéis que cada pessoa possui.
A seguir podemos ver um resumo das ações que cada papel disponibiliza:

[30] Na verdade, guarda um `EJBObject` que encapsula o "bean"

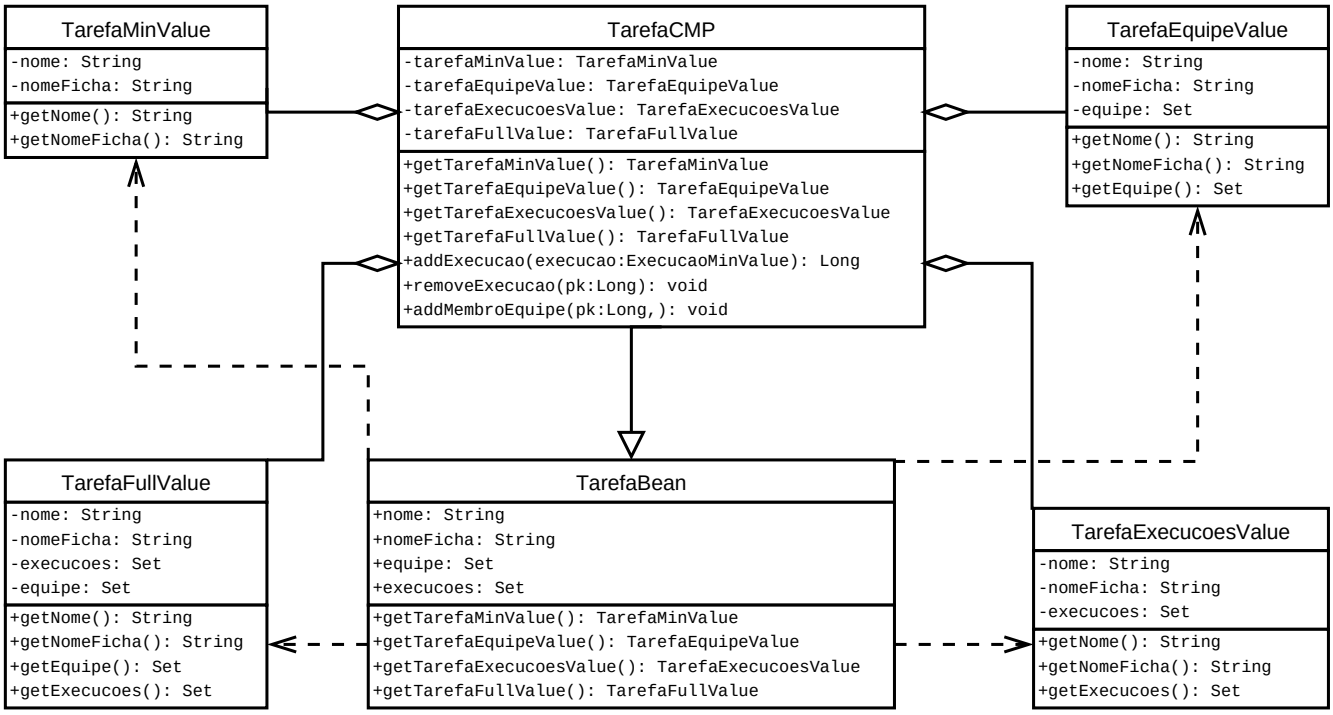


Figura 7: Diagrama `UML` simplificado mostrando os diversos `Value-Objects` criados a partir da classe `TarefaBean`. Observe como cada `Value-Object` contém apenas um subconjunto dos campos do "bean".

Supervisor: Usuário que possui acesso irrestrito ao sistema.

- Cadastrar pessoas.
- Cadastrar programas.
- Definir equipe de desenvolvimento dos programas.
- Designar coordenadores dos programa.
- Cadastrar Projeto.
- Alocar equipe de desenvolvimento do projeto. Pode escolher qualquer pessoa de qualquer programa.

Coordenador: Pessoa responsável por programas e/ou projetos

- Cadastrar tarefas, bem como definir os relacionamentos entre elas.
- Criar fichas.
- Cadastar pessoas.
- Alocar pessoas para as tarefas. Só pode escolher pessoas que façam parte do seu programa.

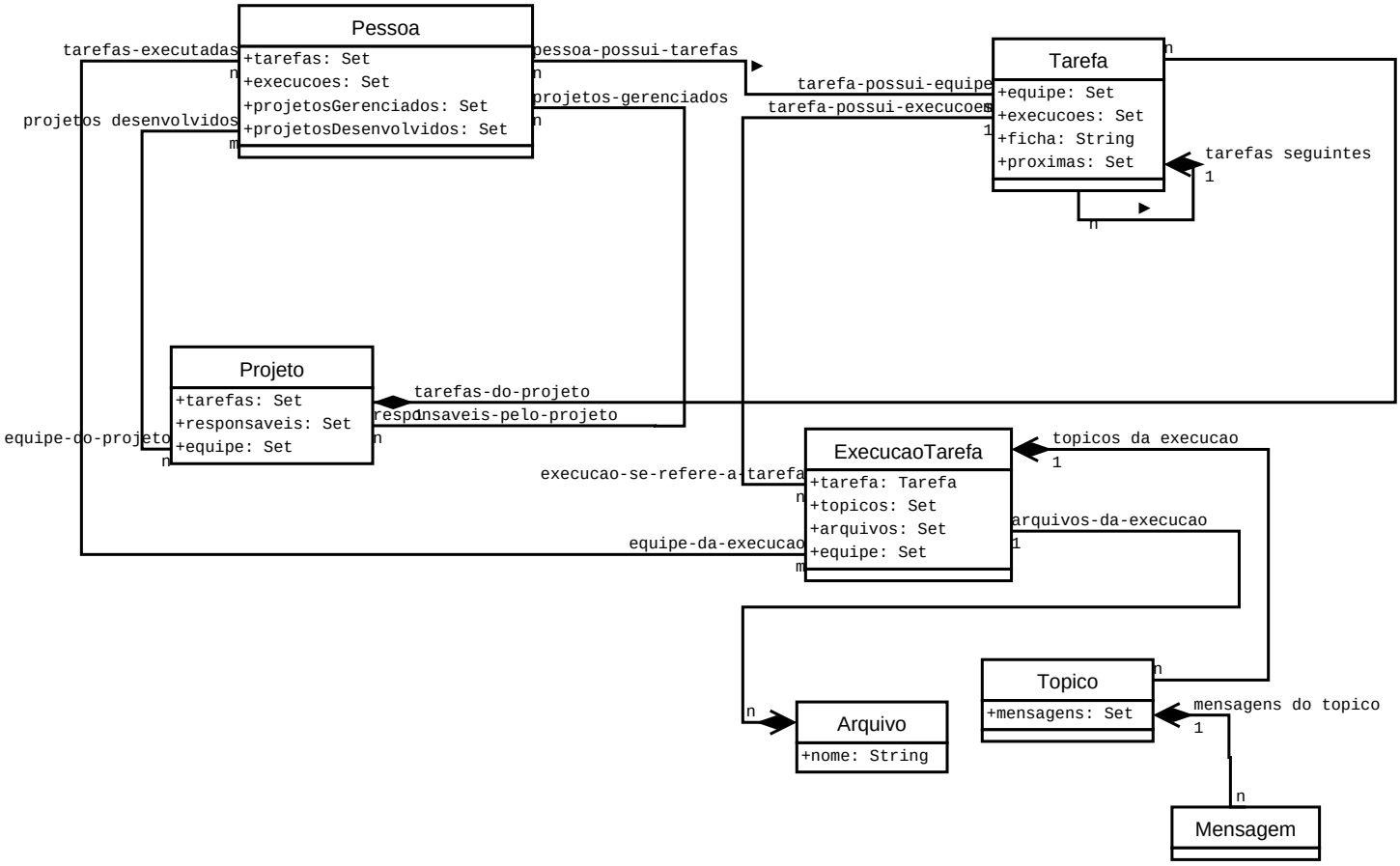


Figura 8: Diagrama simplificado com os relacionamentos entre os diversos “beans” do sistema

- Cadastrar projetos.
- Definir equipe de desenvolvimento do projeto. Só pode escolher pessoas que façam parte do seu programa.
- Definir responsável pelo projeto, sendo que este deve fazer parte da equipe de desenvolvimento de um dos programas deste coordenador.

Responsável: Pessoa responsável por um projeto

- Cadastrar tarefas do projeto pelo qual ele é responsável.
- Criar grafo de relacionamentos entres as tarefas de cada projeto sob sua responsabilidade.
- Alocar pessoas que façam parte do programa/projeto para cada tarefa.

Estagiário/Funcionário: Pessoa responsável por executar as tarefas.

- Executar qualquer tarefa na qual ele esteja alocado.
- Concluir uma tarefa que esteja sendo executada. Isto inclui selecionar a próxima tarefa a ser liberada para execução.
- Durante a execução de uma tarefa, ele pode enviar arquivos, criar mensagens e tópicos de discussão, além de preencher a ficha da mesma.

Funcionamento Coordenadores e supervisores cadastram pessoas no sistema, cada qual com um dos quatro papéis citados anteriormente([31]).
As pessoas são alocadas nos programas.

Agora, coordenadores criam as tarefas, as fichas, alocam pessoal para cada uma das tarefas e, criam o autômato de tarefas.
Deste momento em diante, cada uma das pessoas alocadas nas tarefas começa a executá-las.

Dentro de cada programa, os coodenadores podem criar projetos, alocando para cada um deles, pessoas que façam parte da equipe de desenvolviemnto do programa que ele coordena. O supervisor também pode criar projetos, com a vantagem de poder escolher, pessoas de qualquer um dos programas.

O intuito aqui foi apenas fornecer uma idéia superficial e, de alto nível, do funcionamento do sistema.

Ficha

Como já dissemos, cada tarefa possui uma ficha, que deve ser preenchida durante sua execução. A ficha é o modelo enquanto que o preenchimento dela pode ser visto como o registro. Então, enquanto uma tarefa possui uma ficha, uma execução possui um registro.

As fichas são manejadas pelo sistema `VODU`, criado por mim e pelo Élcio Koiti Nakashima, outro aluno do BCC/IME. Pense numa ficha como um pequeno questionário. Por exemplo: imagine que exista uma tarefa chamada Atender Telefone e que, durante a execução desta tarefa, a pessoa deva preencher um questionário com os seguintes dados: nome da pessoa que ligou, horário da ligação, assunto, além, é claro, do nome da pessoa com quem ela gostaria de falar. O conjunto deste dados forma a ficha

[31] 3.7.3

desta tarefa.

No sistema, o usuário pode criar diversas fichas e, atribuir a cada tarefa do sistema uma.

É vantajoso utilizar o VODU por que este permite que a estrutura das fichas sejam alteradas sem que, os dados preenchidos, sejam corrompidos ou fiquem inacessíveis. Imagine, por exemplo, que uma determinada tarefa[32] use uma ficha(3.7.3) com os seguintes dados: nome, dúvida, email. Suponha que esta tarefa tenha sido executada diversas vezes e, um certo dia a estrutura da ficha foi alterada, retirando-se o campo email. A partir deste momento, os próximos preenchimentos não irão conter o campo email mas, toda vez que uma execução passada for aberta(uma que tenha ocorrido antes da alteração da ficha) o preenchimento mostrado conterá os campos: nome, email e dúvida.
Para que isto funcione, o VODU registra junto com o preenchimento, a versão da ficha utilizada.

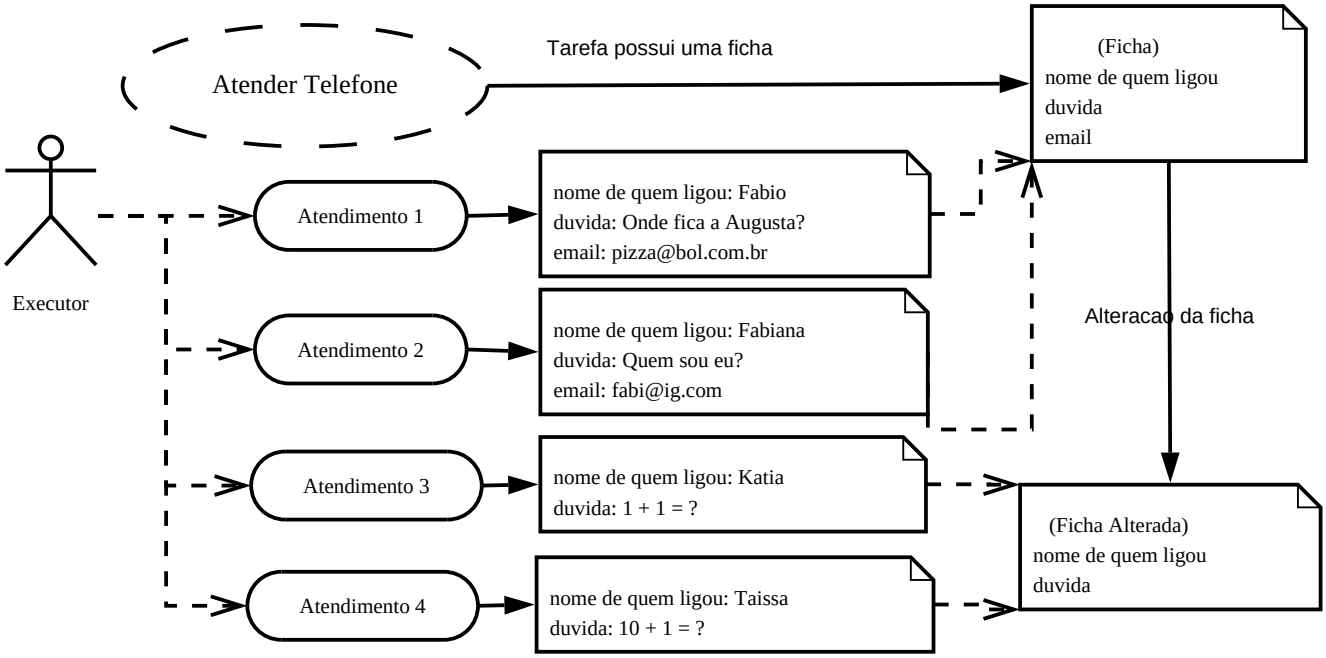


Figura 9: Exemplo de como o sistema se comporta quando a estrutura de uma ficha é alterada. As duas primeiras execuções foram baseadas na ficha que continha email, enquanto que as duas últimas forma baseadas na ficha alterada.

Ordenação de listagens

[32]Novamente a tarefa Atender Telefone.

Assim como ocorre nas listagens do MetaNaeg, as listagem do SGT podem ser ordenadas no próprio navegador do usuário, utilizando-se um código `JavaScript` de terceiros. Uma importante diferença entre este e o utilizado no MetaNaeg, reside no fato de que a quantidade de código necessária para sua utilização é bem menor, além de ser mais flexível, no que diz respeito à criação de novos tipos.

3.7.4 Tecnologias utilizadas

As "IDE's" utilizadas no desenvolvimento do sistema foram:

Eclipse Desenvolvimento dos "beans", classes utilitárias e interfaces.

Utilizamos exaustivamente o "plugin" `JBoss-IDE` que facilita bastante o desenvolvimento de aplicações `J2EE`.

Fazia parte do "plugin" o gerador customizável de códigos `XDocLet`. Este foi utilizado para gerar classes utilitárias, "deployment descriptors", entre outros.

O empacotamento das classes também podei ser feito através do `JBoss-IDE`.

NetBeans Desenvolvimento do cliente `Web` da aplicação, por possuir mais recursos que o `Eclipse` para esta tarefa, além de fornecer um ambiente de depuração melhor.

O servidor de aplicações utilizado foi o `JBoss`, sendo que a principal razão de sua escolha, foi o fato de ser gratuito.

O sistema de gerenciamento de banco de dados utilizado foi o `Sybase`. A razão foi a familiaridade que tínhamos com o mesmo.
Como não utilizamos quase nenhuma funcionalidade específica do mesmo, é possível trocá-lo por outro "SGBD" sem muito trabalho.

3.7.5 Dificuldades e frustrações

Quando comecei o estágio na CECAE sabia muito pouco sobre `J2EE`, praticamente apenas o que tinha aprendido no curso de SOD(Sistemas de Objetos Distribuídos) por isso, precisei gastar um bom tempo aprendendo esta tecnologia.
Enquanto aprendia a mesma, deparei-me com uma série de padrões e ferramentas que auxiliavam no desenvolvimento de aplicações utilizando `J2EE`, dentre elas: `JBoss-IDE` e `XDoclet`.
Passei então a utilizá-las exaustivamente pois, prometiam facilitar muito o meu trabalho, o que no entanto, não ocorreu.
Perdi muito tempo, principalmente com o `XDoclet` pois, foi preciso alterar alguns de seus templates a fim de gerar o código corretamente e, além disso, insisti em usar algumas tarefas problemáticas[33] do `XDoclet`.
A conclusão foi que o `XDoclet` se mostrou uma ferramenta muito boa para tarefas já bem estabelecidas e, muito perigosa para outras.

Além de problemas com o `XDocLet`, não foi trivial integrar `Sybase` e `JBoss`. Existem alguns problemas que, para meu grande desapontamento, ainda não tem solução. Somente para não deixar muito vago, os problemas estão relacionados ao uso de chaves estrangeiras e à criação de campos do tipo "identity".

Uma falha também foi encontrada na especificação EJB 2.0, relacionada diretamente ao problema das chaves estrangeiras[34].

Tive duas frustrações na CECAE: falta de tempo para finalizar um sistema robusto como o SGT e, não ter desenvolvido o mesmo em grupo.

[33] Por exemplo: a tarefa entity facade nao funcionava para value objects.

[34] Chaves estrangeiras eram criadas com a restrição "not null" entretanto, como os relacionamentos de um "bean" só são definidos no "ejbPostCreate", o `Sybase` retornava um erro na criação do "bean".

3.7.6 Estado atual

O sistema ainda está na fase de desenvolvimento.

A maior parte dos `Entity Beans` está completamente desenvolvida, incluindo seus relacionamentos. Faltam alguns `Session Beans`(fachadas).

A interface gráfica é, sem dúvida, a parte do sistema mais incipiente. Entretanto, não parece ser difícil desenvolvê-la tanto quanto foi criar os "EJBs".

4 Estágio e o BCC

Os estágios exigiram muito o uso da criatividade na resolução de diversos problemas que surgiram o tempo todo. Existiam problemas de concorrência, limitações que as ferramentas utilizadas ofereciam, dificuldade em entender problemas dos usuários, entre outros.

Felizmente, o BCC forma alunos que sabem lidar com problemas, uma vez que somos estimulados, durante todo o curso, a resolvê-los, pensando e usando a criatividade. Acredito que o curso esteja preparando bem seus alunos para atuarem com sucesso na área de computação, uma vez que é bem teórico, os alunos acabam saindo com uma base sólida nos fundamentos da computação o que lhes permite absorver, com grande habilidade e velocidade, as novidades da área.

Durante todo o meu período de estágio, pude perceber o quanto o BCC me ajudou a tratar dos problemas: buscando por soluções, raciocinando de uma maneira bem organizada, entre outras coisas essenciais no exercício de funções nesta área.

Algo digno de nota, que ocorreu comigo em ambos os estágios, esteve relacionado com a dificuldade em entender os clientes e travar um diálogo no mesmo jargão deles. Percebi que, para que os requerimentos fossem corretamente coletados, era preciso que o analista entendesse, muito bem, o negócio da empresa. Isto inclui também o palavreado comum aos negócios da mesma. Além disso, é interessante notar que, muitas vezes os clientes não sabem o que realmente querem e, que o analista também deve propor requerimentos, ficando quase que na posição de cliente. O analista deve, a meu ver, orientar o cliente na escolha dos seus requerimentos e, não apenas realizr um papel passivo de coletá-los. Assim, a construção de um sistema passa a ser vista como uma cooperação entre ambas as partes envolvidas.

No estágio na CECAE, foi gratificante ter proposto um modelo que, supria a maior parte das necessidades dos usuários de diversas áreas diferentes pois, no início eles achavam que cada setor deveria ter um sistema diferente.

No estágio no NAEG, fiquei muito satisfeito em ver que tinha construído um sistema, que realmente estava trazendo benefícios ao Núcleo, facilitando o trabalho do pessoal e propiciando ao coordenador, manter um controle sobre as atividades lá realizadas.

As seguintes matérias podem ser consideradas as mais importantes durante os estágios:

Engenharia de software A disciplina tratou de ensinar diversas técnicas de desenvolvimento de sistemas, guiando-me na escolha de uma metodologia para o desenvolvimento do SGT e do MetaNaeg. Além disso, aprendemos como coletar requerimentos dos usuários, testar sistemas, entre outros.

Estruturas de Dados É essencial por fornecer uma base sobre como problemas podem ser resolvidos mais fácil e eficazmente com o uso de estruturas de dados específicas.
Ensinou que a maneira de estruturar os dados é parte integrante da resolução de muitos problemas.
Durante o estágio, recordo-me de ter utilizado as seguinte estruturas de dados: Grafos e Tabelas de Espalhamento.

Introdução a linguagens formais e autômatos Nos dois sistemas construídos, forma utilizados autômatos a fim de controlar fluxos. Pensar em termos destes autômatos, facilitou muito o entendimento e a resolução dos problemas.

Programação Concorrente Os problemas de concorrência encontrados durante o desenvolvimento dos sistemas foram resolvidos utilizando-se stored procedures com locks nas tabelas e, utilizando transações.

A maior contribuição desta matéria foi nos colocar a par de uma questão que agora esta permeando a maioria dos sistemas computacionais: a concorrência, além de nos mostrar diversas maneiras de atacar o problema.

Sistemas de Banco de Dados Para construir o MetaNaeg foi necessário ter um bom conhecimento sobre o funcionamento, operação e manipulação de estruturas presentes em bancos de dados: tabelas, views, triggers, etc. Esta matéria nos deu uma boa noção sobre tudo isto.
Além desta visão pragmática, nos pudemos aprender como funcionam índices, cluster, como os dados são armazenados, além de entender, um pouco, sobre as estruturas com as quais o BD trabalha: b-arvores, páginas, etc.
Todo este conhecimento foi de grande valia no momento de escolher as colunas que deveriam receber índices, entre outras questões de eficiência.

Sistemas de Objetos Distribuídos Nesta matéria aprendemos quais são os elementos que fazem

parte de um arquitetura de objetos distribuídos. Isto foi extremamente importante para entender e trabalhar com a arquitetura J2EE, a qual foi usada no desenvolvimento do SGT.

Programação Orientada a Objetos: Nesta disciplina aprendemos como raciocinar e bolar soluções utilizando o modelo de programação de orientação a objetos.
Além disso, aprendemos também sobre padrões e UML.
Durante o estágio, utilizei diversos padrões e, pude verificar na prática, suas utilidades e benefícios.

Fiz uso também de diagramas UML e, com isso, foi possível ter uma idéia bem melhor do funcionamento do sistema. Pude notar o quão rico ele é na passagem de informações de um modo extremamente resumido.

Tópicos Avançados de Programação Orientada a Objetos Aqui, foi prossível aprofundar o conhecimento deste paradigma. Os pontos principais abordados, a meu ver, foram: reflexão e aspectos. A idéia de aspectos ajudou a entender como os servidores de aplicações lidavam com questões ortogonais à aplicação.

Laboratório de Programação II Nesta disciplina aprendemos, dentre outras coisas, a linguegem Java.
Isto foi relevante para o estágio na CECAE já que o SGT está sendo escrito nesta linguagem. Além disso, demos os primeiros passos no aprendizado do modelo de programação orientado a objetos, nesta disciplina.

Muitos dizem que a sua experiência de trabalho é bem diferente das obtidas nos trabalhos propostos pela faculdade. Uma grande diferença que eu vejo se refere à quantidade de trabalho: nos EPs[35] exigisse mais raciocínio e criatividade, enquanto que nos trabalhos realizados em empresas exige-se mais trabalho e menos criatividade. É claro que não possível generalizar esta idéia mas, na maioria das vezes é o que ocorre: o programador aprende uma linguagem, a empresa já adotou uma série de padrões de codificação e desenvolvimento de sistemas que ele deve seguir. Comigo não ocorreu isso. No NAEG e na CECAE tive total liberdade para definir metodologias e padrões, o que em parte foi muito bom, pois pude usar bastante a criatividade aplicando metodologias e técnicas aprendidas no BCC mas, em parte perdi um pouco, uma vez que poderia ter aprendido utilizando padrões e metodologias novas.

[35] Exercício - Programa.

Uma frustração que tive foi a de não ter desenvolvido o projeto em grupo. Perdi a oportunidade de ganhar experiência de trabalho em grupo. Com certeza isso enriqueceria muito a minha formação fornecendo-me uma boa idéia de como lidar com diferenças de pensamento, desenvolvendo uma grande habilidade de argumentação além, é claro, de promover uma troca de experiências.

No BCC, eu pude ter todas essas experiências, realizando os trabalhos em grupo mas, gostaria muito de ter levado isto para fora dos limites do BCC.

Algo que realmente senti falta, foi trabalhar com pessoas com formação acadêmica diferente da minha.

5 Conclusão

Encarei os estágios como oportunidades ímpares de colocar à prova todo o conhecimento adquirido durante o curso. Durante os mesmos, foi necessário relacionar conhecimentos de diversas matérias, a fim de buscar um solução para um determinado problema, o que mostrou o quanto as matérias estão relacionadas, embora sejam ministradas durante o curso sem uma preocupação maior em salientar tal.

Algumas lições que pude tirar foram:

Aprender leva tempo Aprender uma ferramenta, tecnologia, metodologia, etc, leva um bom tempo. Com pouco tempo já é possível obter resultados mas, para que ela se torne realmente produtiva e eficiente, é necessário bastante treino. Por exemplo: no SGT o uso do XDoclet atrapalhou mais que ajudou, uma vez que encontrei muitos erros e tive que alterar alguns templates. Achei que a ferramenta me pouparia tempo mas, ao invés disso, acabou dispendendo mais ainda.

Perigo de usar ferramentas de terceiros Usar software de terceiros sempre é arriscado. Durante os estágios precisei alterar o código de muitas ferramentas utilizadas, o que me tomou bastante tempo pois, precisava entender como elas foram desenvolvidas. Assim, ferramentas que deveriam ajudar acavaram atrapalhando.

Geradores de código O uso deste tipo de software deve ser feito com muito cuidado e análise.

O próprio manual do XDocLet recomenda que, devemos utilizar ferramentas de geração de código, apenas par gerar codigo que, alguma vez já criamos manualmente.

Isto pode parecer óbvio mas, não é. Às vezes, nos sentimos impelidos a usar geradores de código exaustivamente, até mesmo para gerar código que nunca escrevemos antes. Este tipo de atitude pode levar a um problema sério: acaba-se gastando tempo demais no entendimento do código gerado.

Também é importante lembrar que, nem sempre o código gerado por estas ferramentas atende às necessidades. Neste caso, é preciso criar o código manualmente ou, alterar a ferramenta.

Durante o desenvolvimento do SGT houve casos em que foi necessário alterar o código da ferramenta de geração e, outros onde desistimos de utilizá-la, passando a escrever o código manualmente.

Importancia da documentação Pude perceber, na prática, o valor que a documentação de um sistema tem. À medida que o sistema se torna mais complexo, fica cada vez mais difícil lembrar detalhes da sua implementação, do modelo utilizado, dos requerimentos, etc.

Acredito que uma maneira de prever, o sucesso ou o fracasso de um sistema, seja avaliando a qualidade da sua documentação. Isto porque, todo sistema precisa de manutenção e, a dificuldade principal em manter um sistema é entendê-lo sendo que, auxiliar no entendimento de um sistema é um dos papéis principais da documentação.

Sendo assim, se a documentacao nao for bem feita, o sistema dificilmente evoluirá e, com o tempo passará a nao ser mais útil ao cliente[36].

Durante o desenvolvimento dos projetos, gastei bastante tempo documentando os sistemas. Como o trabalho foi individual, o resultado ficou aquém do esperado mas, mesmo assim já pude notar a sua importancia, principalmente quando precisava alterar ou adicionar alguma funcionalidade aos projetos.

Embora tenha criticado muito o uso de ferramentas de terceiros estou, na verdade, frisando a necessidade de tomar cuidado ao fazê-lo e, mostrando que se deve estar preparado para o caso de precisar alterar e aprendê-las mais profundamente pois, muitas vezes, isso é necessário.

Uma vez que uma ferramenta é bem aprendida e configurada, é possível obter grandes ganhos nos seus usos futuros.

[36] O fracasso de sistemas por esta razao é, o que mas acontece. Razao pela qual empresas estao refazendo tantos sistemas.

6 Apêndice

6.1 MetaNaeg

6.1.1 Telas

Nesta tela a pessoa pode alterar seus dados pessoais.

Uma pessoa com papel **Gerenciador de grupos** pode alterar dados de quaisquer pessoas, além de definir os papéis de cada uma entretanto, ela não pode alterar seus próprios papéis.

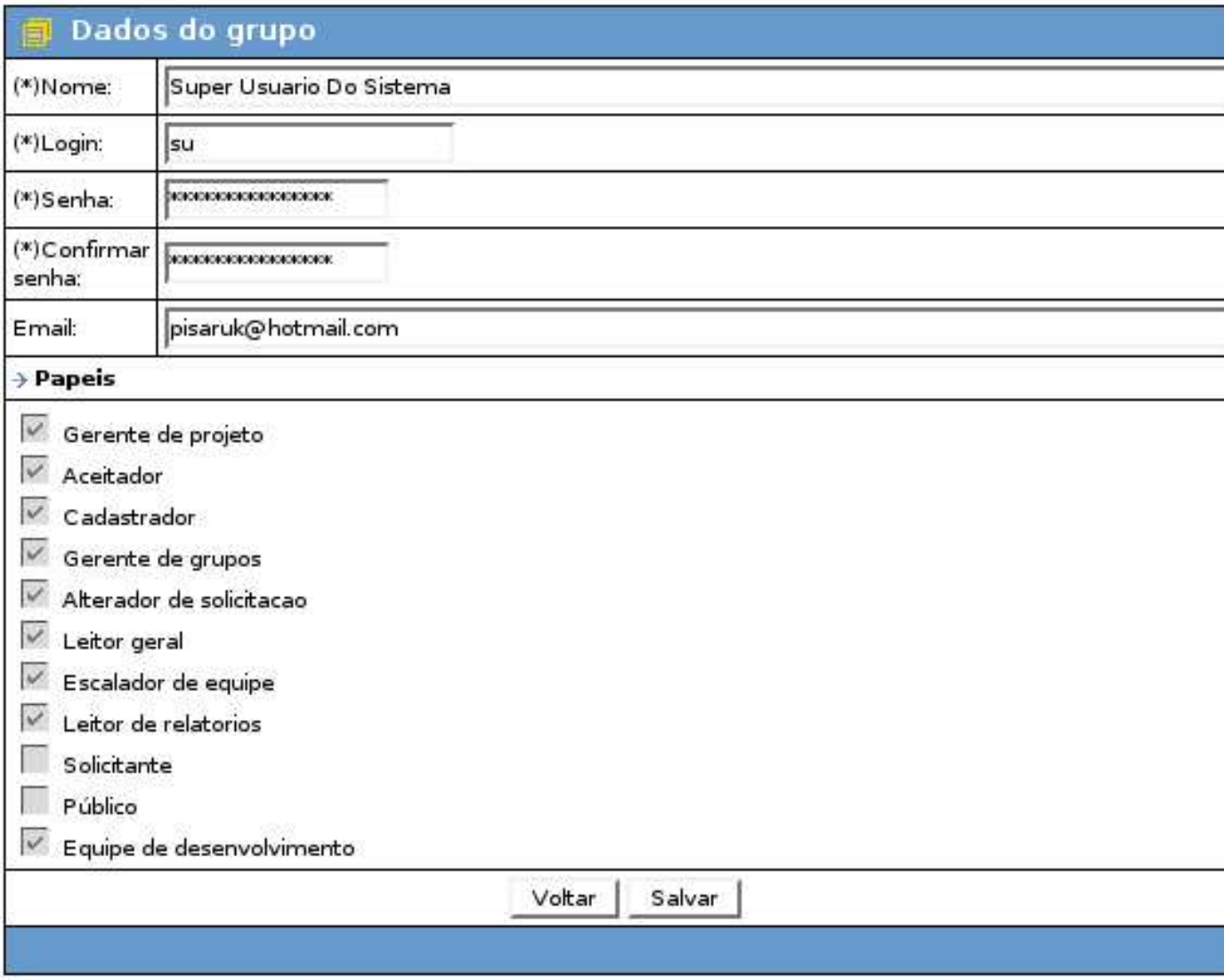


Figura 10: Alterar dados do grupo

Lista os arquivos que foram enviados à solicitação.

Um usuário com papel **Gerente de projeto** pode enviar arquivo a qualquer solicitação, bem como apagá-los.

Um usuário com papel **Equipe de desenvolvimento** pode enviar arquivos apenas às solicitações nas quais ele foi escalado para desenvolver. Além disso, ele só pode remover os arquivos que enviou.

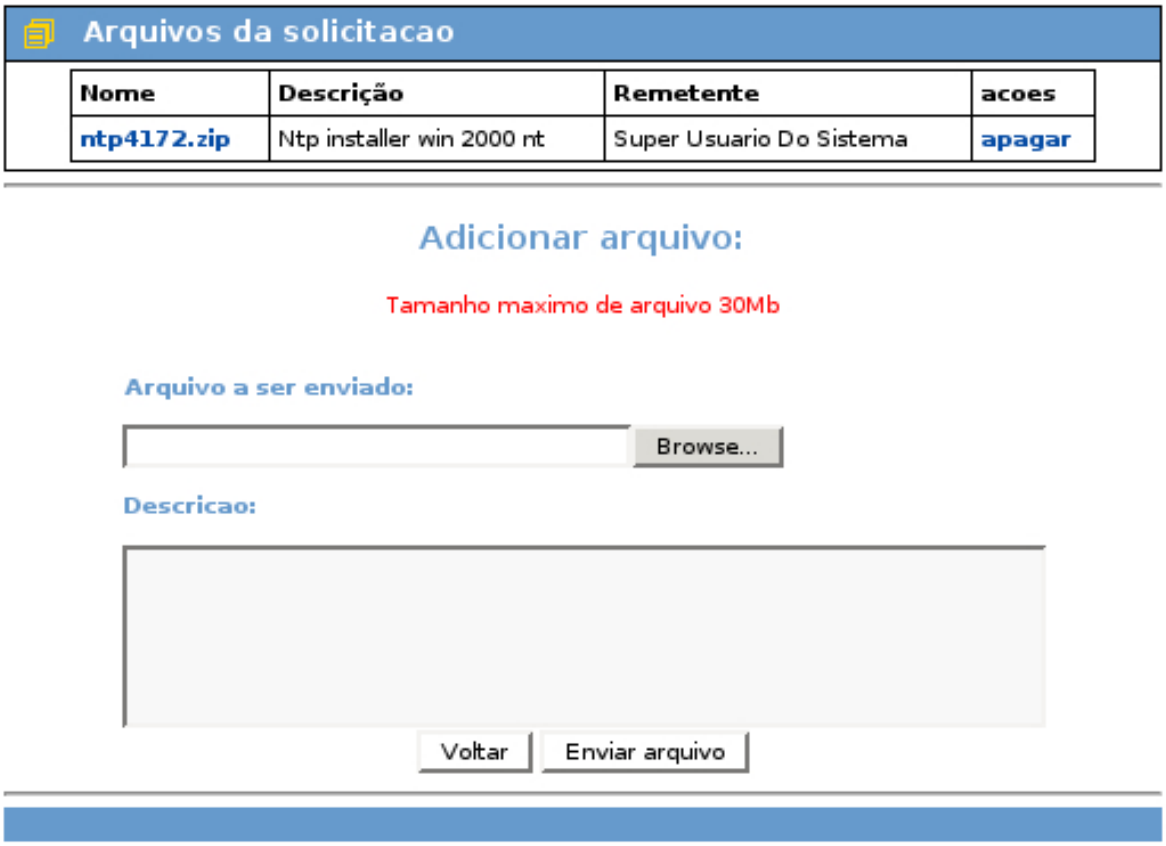


Figura 11: envio de arquivo

Nesta tela o usuário decide quais campos de cada listagem são buscáveis. O usuário desta tela é, na verdade, a pessoa que está desenvolvendo o site e, deseja utilizar o sistema de busca/filtragem. Só relembrando, o sistema de busca realiza a busca somente nos campos que o usuário selecionar como buscáveis.

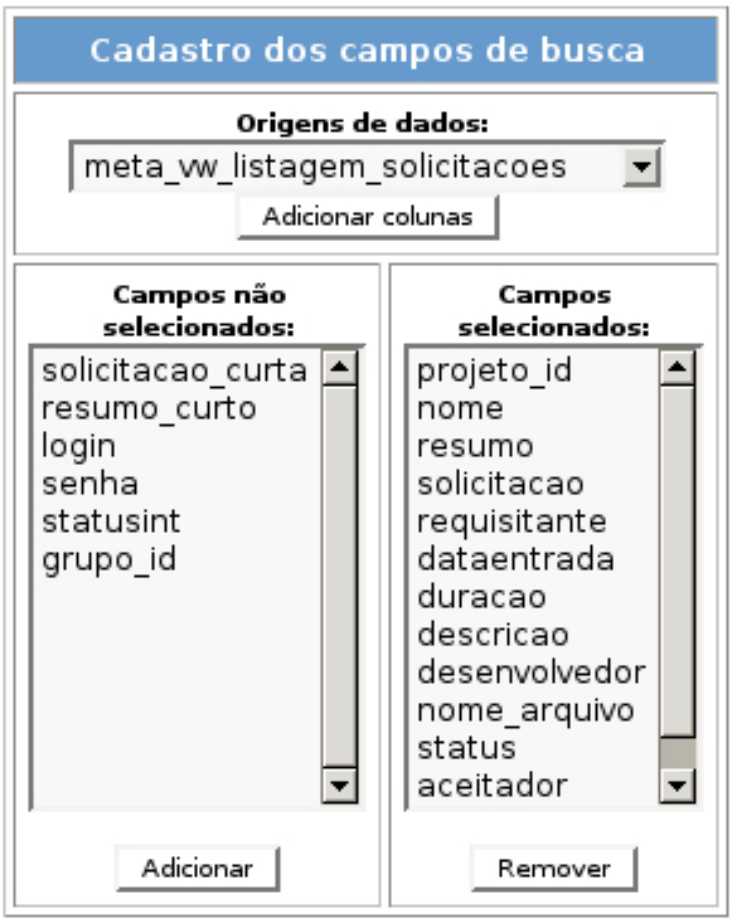


Figura 12: busca

Exibe algumas informações sobre cada grupo/pessoa do sistema. Um grupo pode se referir a um conjunto de pessoas que compartilham um mesmo login/senha ou a uma única pessoa. O sistema não faz qualquer distinção entre os dois casos.

Figura 13: detalhes do grupo

Exibe algumas informações sobre um determinado projeto.

Dependendo do nível de acesso do usuário, alguns botões são ou não visíveis.

A validação da ação não ocorre apenas na visibilidade dos botões, ocorre também em cada script chamado.

Entretanto, a visibilidade dos mesmos também pode ser considerada como parte da segurança.

Figura 14: detalhes do projeto

Exibe algumas informações sobre um determinado projeto.

Dependendo do nível de acesso do usuário, alguns botões são ou não visíveis.

A validação da ação não ocorre apenas na visibilidade dos botões, ocorre também em cada script chamado.

Entretanto, a visibilidade dos mesmos também pode ser considerada como parte da segurança.

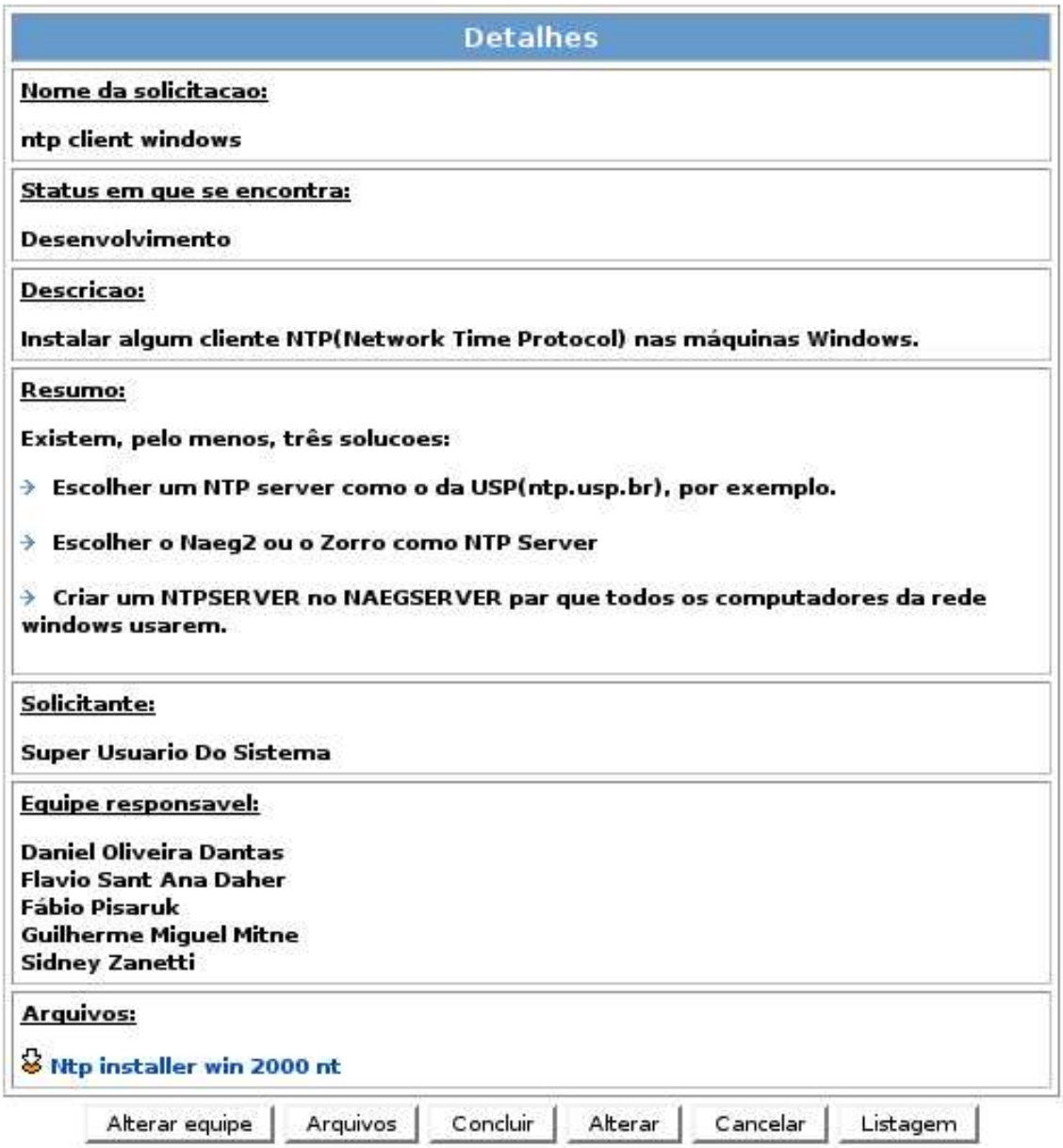


Figura 15: Detalhes da solicitação

Aqui uma pessoa com o papel **Escalador de Equipe** escala as pessoas que farão parte da equipe de desenvolvimento de uma determinada solicitação.

Quando alguém é alocado ou desalocado, um email é enviado notificando-a.

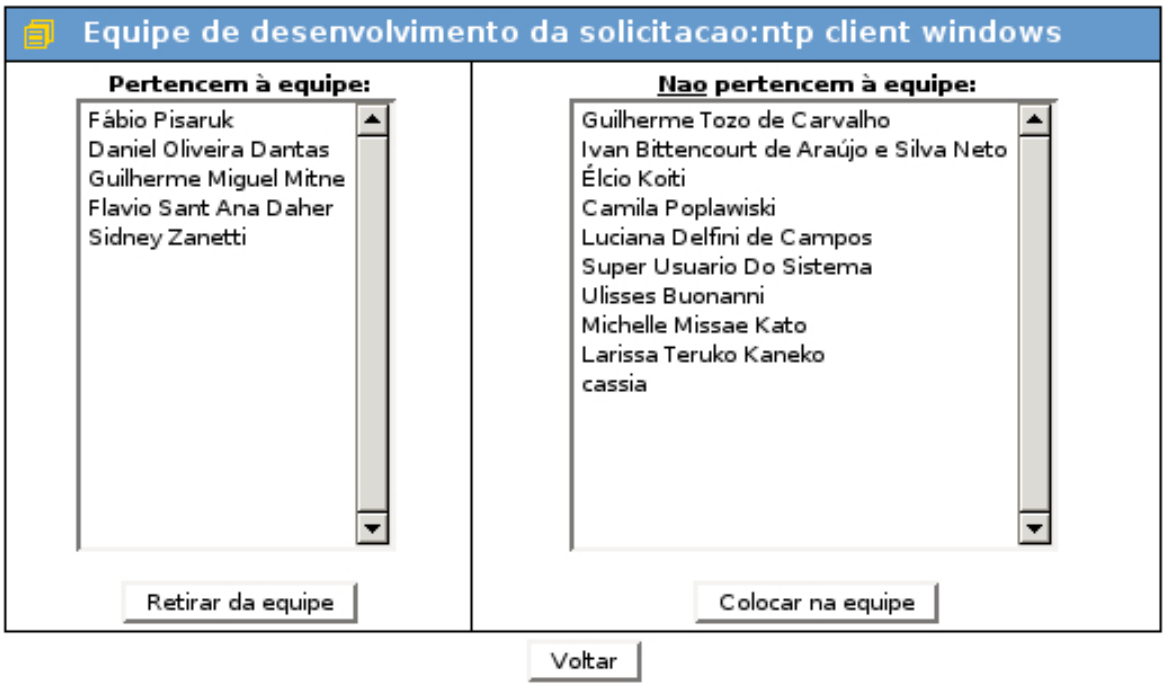
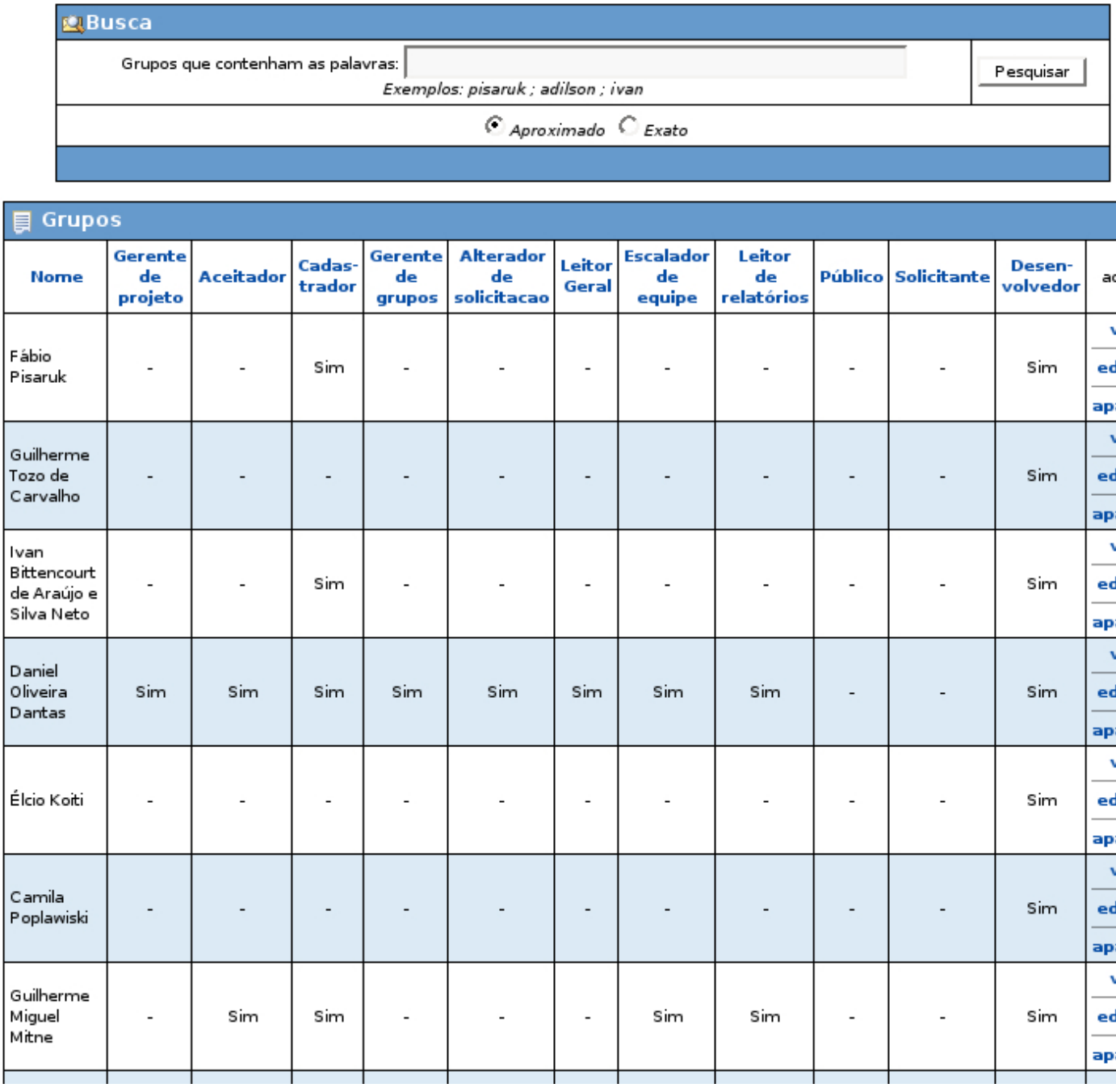


Figura 16: Tela de escalação da equipe de desenvolvimento

Listagem dos grupos/pessoas que estão cadastradas no sistemas.

Esta listagem só é visível às pessoas que possuem o papel de **Gerenciador de Grupos**.

No topo da tela encontramos um campo de busca. O usuário pode realizar filtragem dos dados da listagem entrando um texto neste campo.



Nome	Gerente de projeto	Aceitador	Cadas-trador	Gerente de grupos	Alterador de solicitacao	Leitor Geral	Escalador de equipe	Leitor de relatórios	Público	Solicitante	Desen-volvedor	aco
Fábio Pisaruk	-	-	Sim	-	-	-	-	-	-	-	Sim	ve edit apa
Guilherme Tozo de Carvalho	-	-	-	-	-	-	-	-	-	-	Sim	ve edit apa
Ivan Bittencourt de Araújo e Silva Neto	-	-	Sim	-	-	-	-	-	-	-	Sim	ve edit apa
Daniel Oliveira Dantas	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	-	-	Sim	ve edit apa
Élcio Koiti	-	-	-	-	-	-	-	-	-	-	Sim	ve edit apa
Camila Poplawiski	-	-	-	-	-	-	-	-	-	-	Sim	ve edit apa
Guilherme Miguel Mitne	-	Sim	Sim	-	-	-	Sim	Sim	-	-	Sim	ve edit apa

Figura 17: listar_grupos

Listagem dos projetos do sistema. Cada usuário pode visualizar apenas os projetos que ele tem permissão.

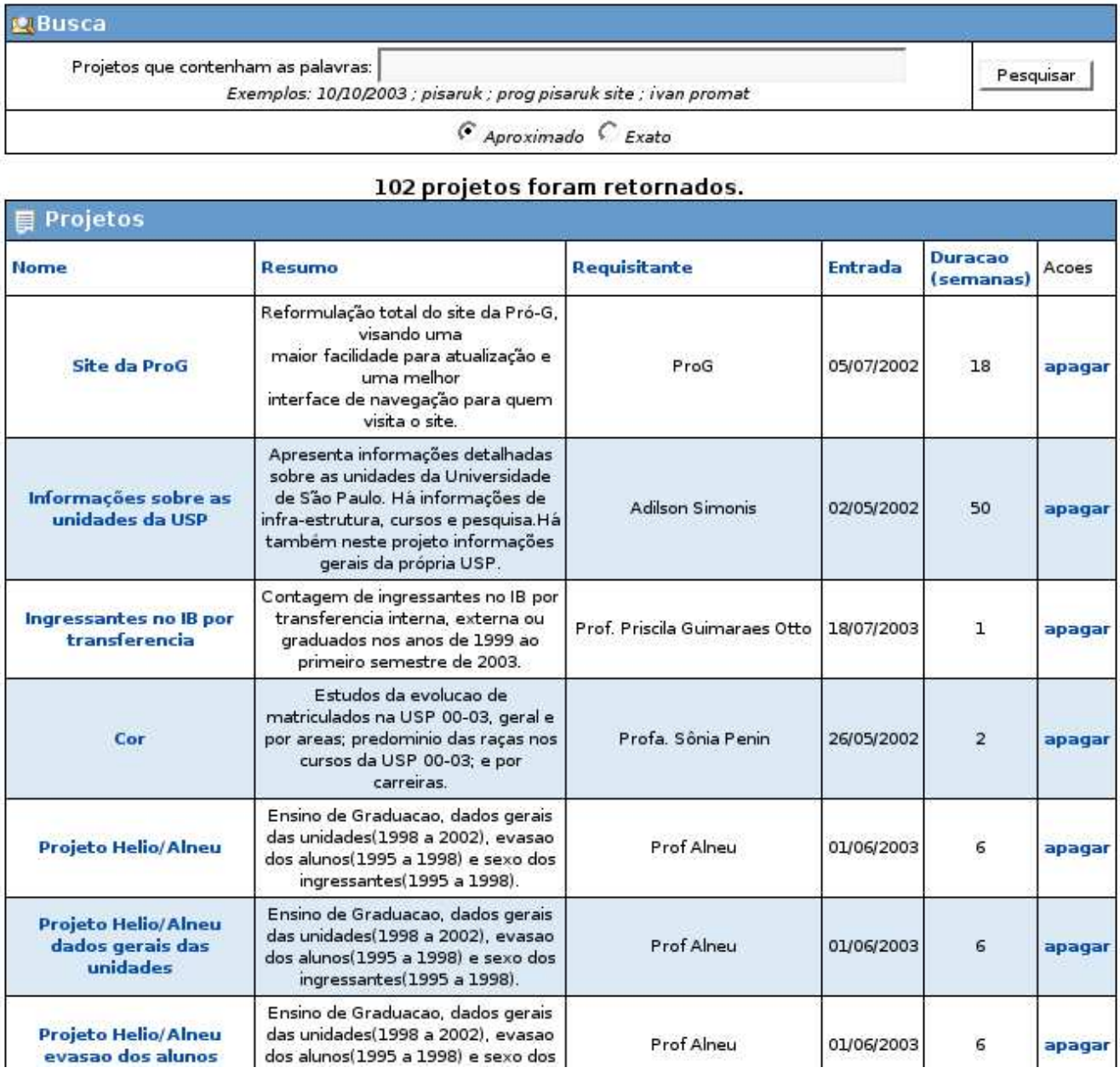


Figura 18: Listagem de projetos

Listagem dos projetos que foram apagados.

No sistema, um projeto apagado é enviado à lixeira ao invés de ser completamente excluído.

Apenas uma pessoa com papel **Gerente de projeto** tem acesso a esta listagem.

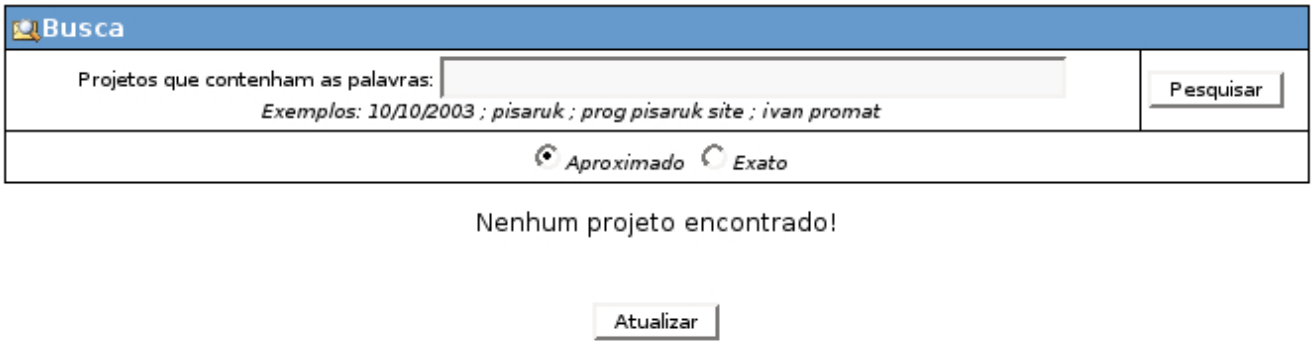
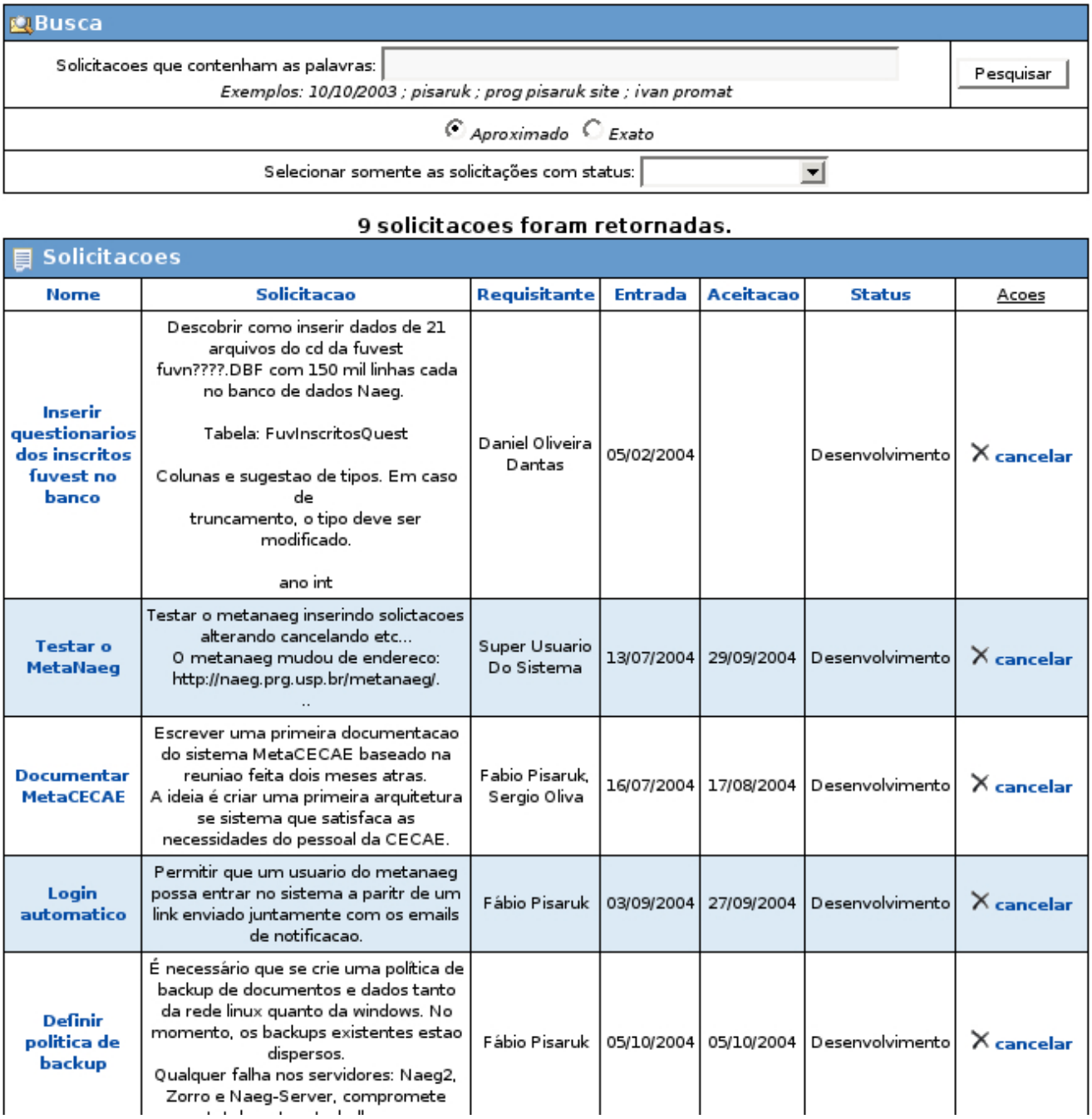


Figura 19: Listagem de projetos que estão na lixeira

Listagem das solicitações do sistema.

Cada usuário pode visualizar apenas as solicitações que ele tem permissão.



Busca

Solicitacoes que contenham as palavras: Pesquisar

Exemplos: 10/10/2003 ; pisaruk ; prog pisaruk site ; ivan promat

Aproximado *Exato*

Selecionar somente as solicitações com status:

9 solicitacoes foram retornadas.

Solicitacoes

Nome	Solicitacao	Requisitante	Entrada	Aceitacao	Status	Acoes
Inserir questionarios dos inscritos fuvest no banco	Descobrir como inserir dados de 21 arquivos do cd da fuvest fuvn????.DBF com 150 mil linhas cada no banco de dados Naeg. Tabela: FuvInscritosQuest Colunas e sugestao de tipos. Em caso de truncamento, o tipo deve ser modificado. ano int	Daniel Oliveira Dantas	05/02/2004		Desenvolvimento	cancelar
Testar o MetaNaeg	Testar o metanaeg inserindo solictacoes alterando cancelando etc... O metanaeg mudou de endereco: http://naeg.prg.usp.br/metanaeg/. ..	Super Usuario Do Sistema	13/07/2004	29/09/2004	Desenvolvimento	cancelar
Documentar MetaCECAE	Escrever uma primeira documentacao do sistema MetaCECAE baseado na reuniao feita dois meses atras. A ideia é criar uma primeira arquitetura se sistema que satisfaca as necessidades do pessoal da CECAE.	Fabio Pisaruk, Sergio Oliva	16/07/2004	17/08/2004	Desenvolvimento	cancelar
Login automatico	Permitir que um usuario do metanaeg possa entrar no sistema a paritr de um link enviado juntamente com os emails de notificacao.	Fábio Pisaruk	03/09/2004	27/09/2004	Desenvolvimento	cancelar
Definir politica de backup	É necessário que se crie uma política de backup de documentos e dados tanto da rede linux quanto da windows. No momento, os backups existentes estao dispersos. Qualquer falha nos servidores: Naeg2, Zorro e Naeg-Server, compromete	Fábio Pisaruk	05/10/2004	05/10/2004	Desenvolvimento	cancelar

Figura 20: listar solicitações

Menu principal do sistema.

Os itens do menu variam de usuário para usuário dependendo do seu nível de acesso.

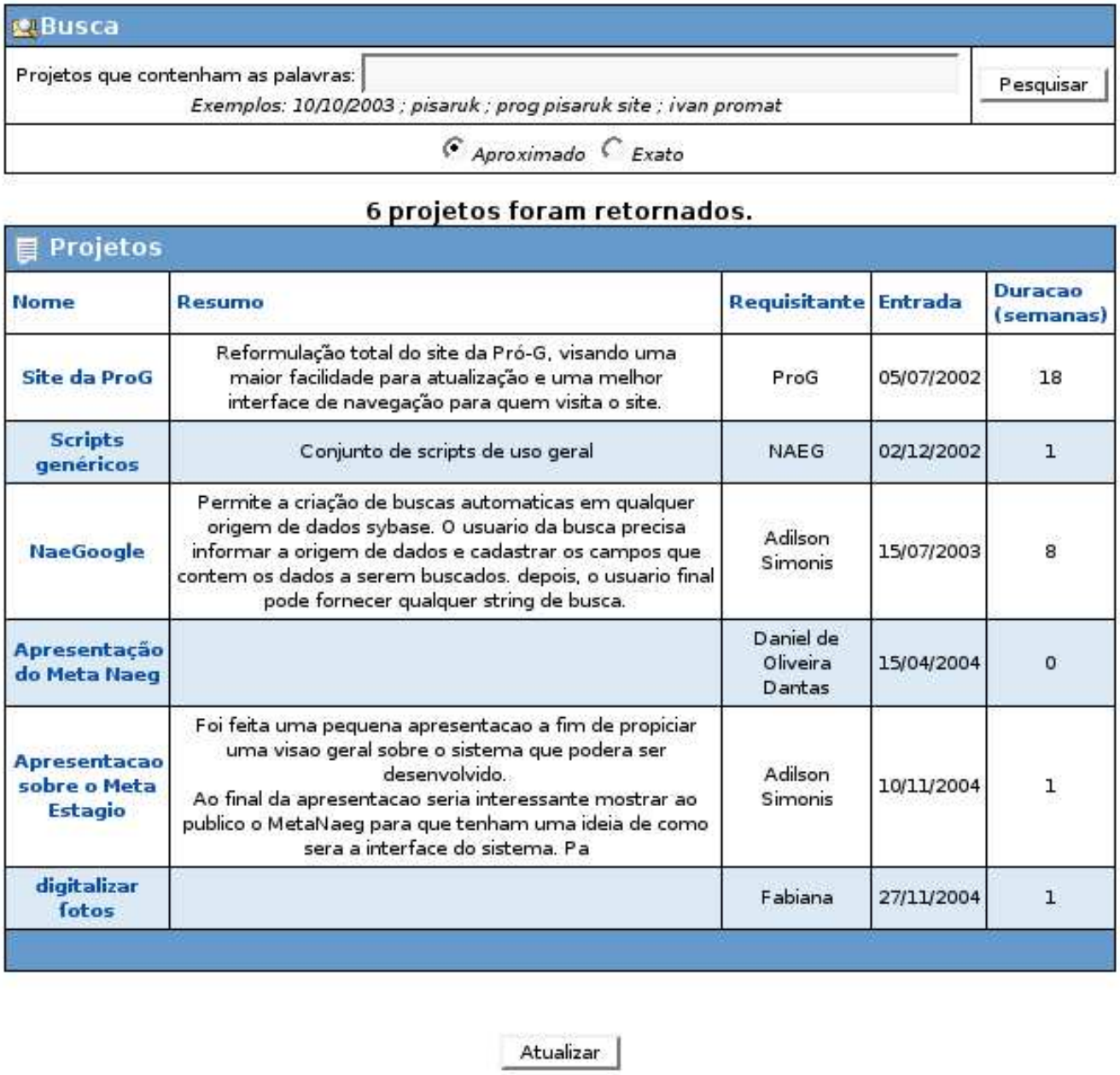


Figura 21: menu

Tela que permite o cadastro de uma solicitação.

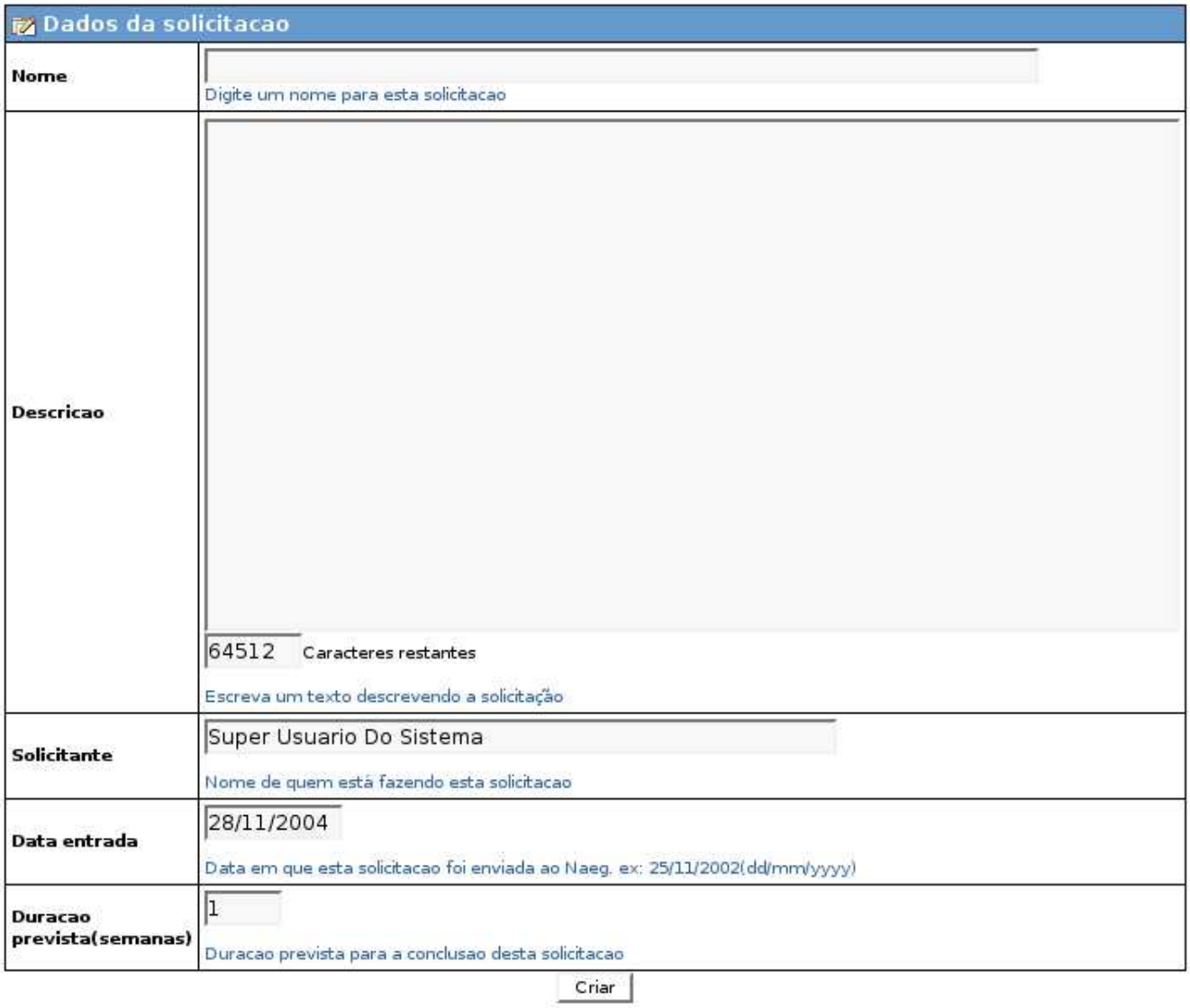


Figura 22: nova solicitação

Tela de cadastro de um novo grupo/pessoa.

Somente pessoas com papel **Gerenciador de grupos** podem cadastrar pessoas/grupos definindo para cada um o nível de acesso através dos papeis selecionados.

Na verdade, qualquer um pode criar um login no sistema entretanto, neste ultimo caso o usuário só possui o papel **Solicitante**.

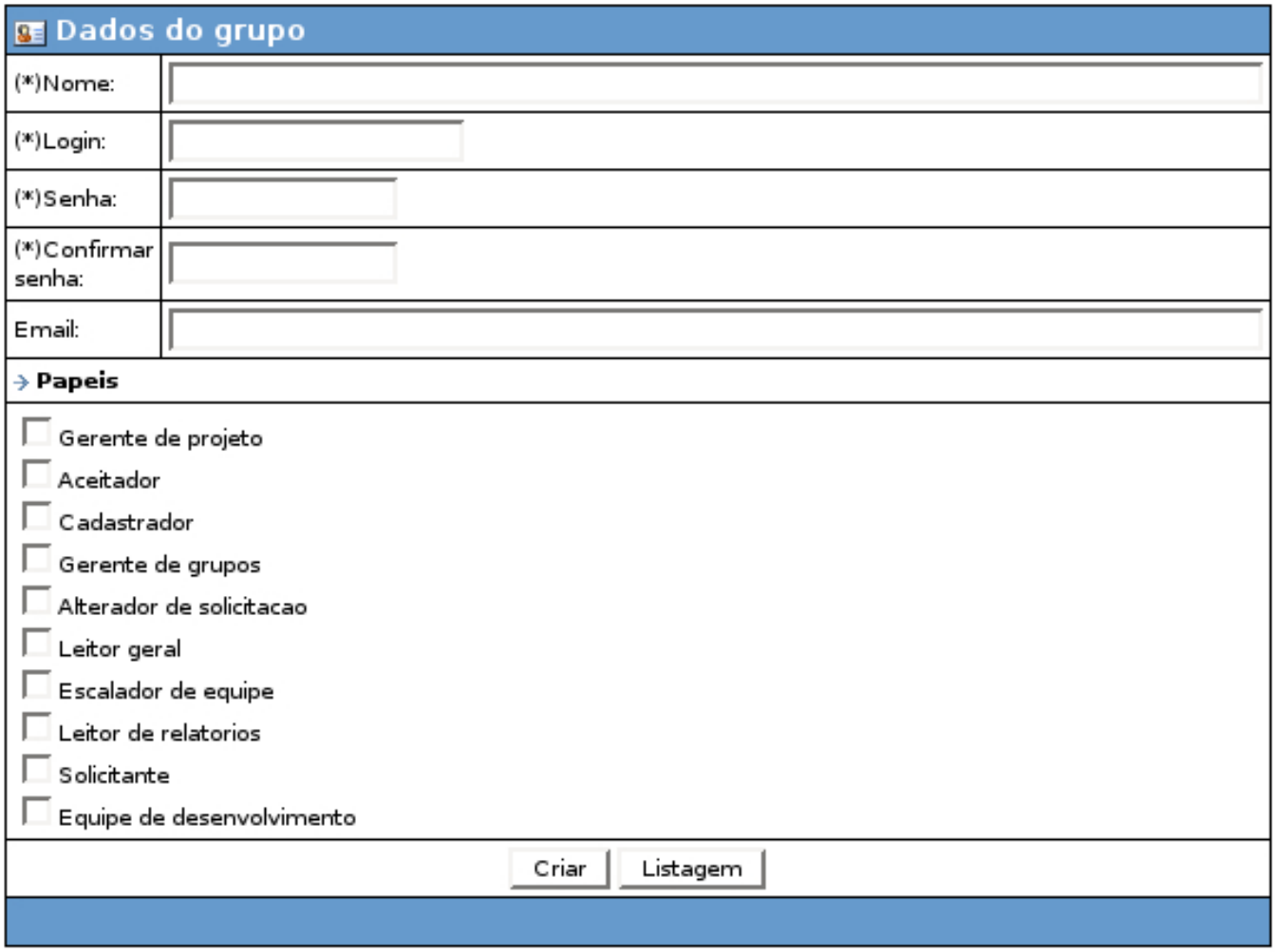


Figura 23: Cadastro de pessoa/grupo

Tela de entrada no sistema.

Aqui existe uma listagem de projetos de domínio público e uma tela de login[37].

Criamos no sistema um usuário chamado público[38] e lhe demos acesso aos projetos que não continham dados sigilosos.

[37] Infelizmente a parte de login acabou não sendo capturada...

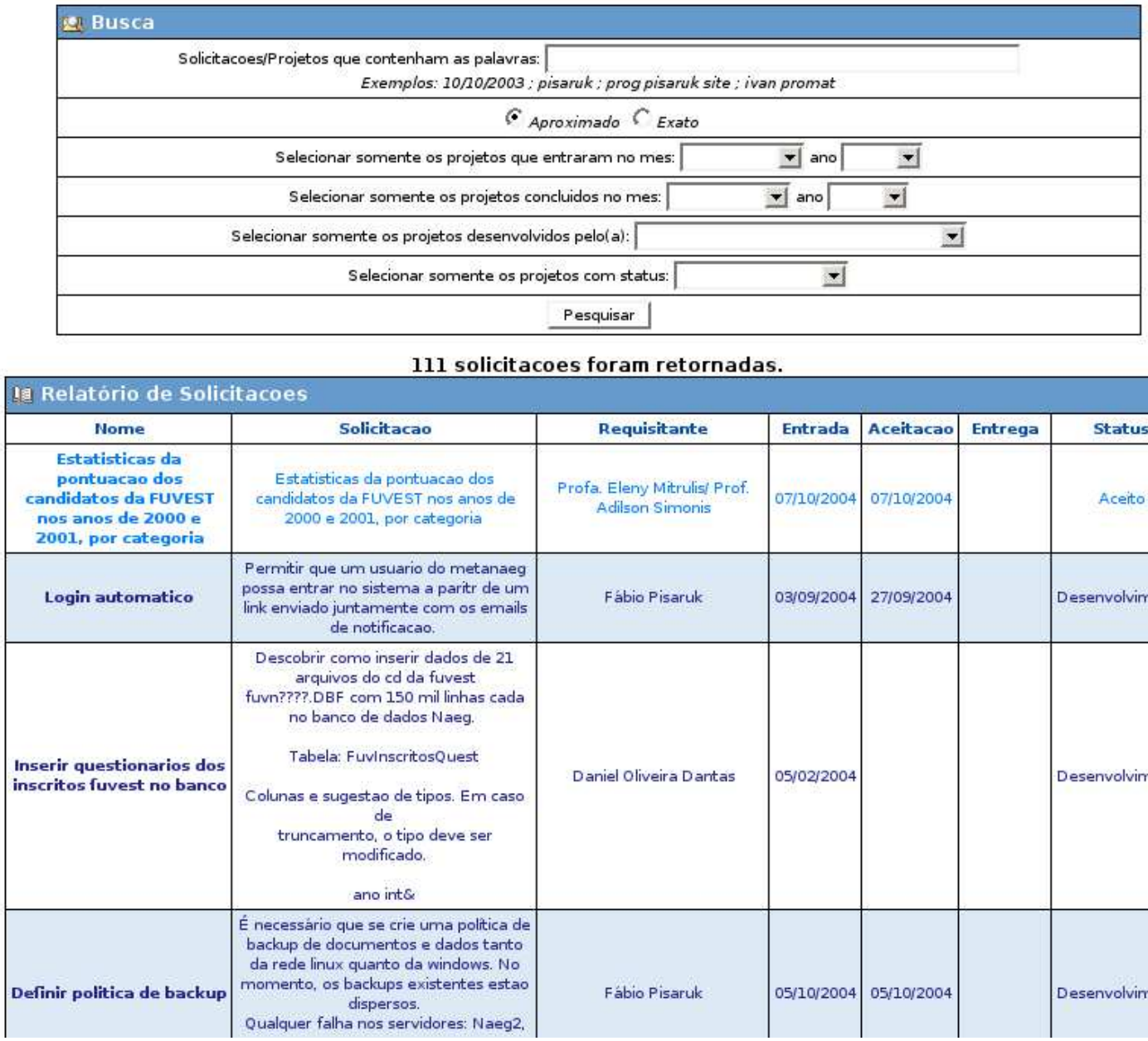
[38] O usuário público só tem o papel Público

Figura 24: principal

Listagem que contém tanto projetos quanto solicitações.

Possui uma série de opções avançadas de filtragem funcionando assim, como uma página de relatórios.

O usuário pode exportar a saída para csv de modo a enviar os dados ao Excel, por exemplo.



Busca

Solicitacoes/Projetos que contenham as palavras:

Exemplos: 10/10/2003 ; pisaruk ; prog pisaruk site ; ivan promat

Aproximado *Exato*

Selecionar somente os projetos que entraram no mes: ano

Selecionar somente os projetos concluidos no mes: ano

Selecionar somente os projetos desenvolvidos pelo(a):

Selecionar somente os projetos com status:

Pesquisar

111 solicitacoes foram retornadas.

Relatório de Solicitacoes

Nome	Solicitacao	Requisitante	Entrada	Aceitacao	Entrega	Status
Estatisticas da pontuacao dos candidatos da FUVEST nos anos de 2000 e 2001, por categoria	Estatisticas da pontuacao dos candidatos da FUVEST nos anos de 2000 e 2001, por categoria	Profa. Eleny Mitrulis/ Prof. Adilson Simonis	07/10/2004	07/10/2004		Aceito
Login automatico	Permitir que um usuario do metanaeg possa entrar no sistema a paritr de um link enviado juntamente com os emails de notificacao.	Fábio Pisaruk	03/09/2004	27/09/2004		Desenvolvime
Inserir questionarios dos inscritos fuvest no banco	Descobrir como inserir dados de 21 arquivos do cd da fuvest fuvn????.DBF com 150 mil linhas cada no banco de dados Naeg. Tabela: FuvInscritosQuest Colunas e sugestao de tipos. Em caso de truncamento, o tipo deve ser modificado. ano int&	Daniel Oliveira Dantas	05/02/2004			Desenvolvime
Definir politica de backup	É necessário que se crie uma política de backup de documentos e dados tanto da rede linux quanto da windows. No momento, os backups existentes estao dispersos. Qualquer falha nos servidores: Naeg2,	Fábio Pisaruk	05/10/2004	05/10/2004		Desenvolvime

Figura 25: Relatório de solicitações

6.1.2 Papéis e ações

Lista de papéis presentes no sistema, bem como as suas respectivas funções.

Gerente de Projeto:
- Cancelamento de solicitação.
- Conclusão de projeto.
- Exclusão de projeto.
- Listagem de projetos/solicitações.
- Criação de um login.
- Login.
- Adicionar arquivos, alterar resumo e listar projetos.
- Reabertura de projeto.
- Recuperação de solicitação.

Aceitador:
- Listagem de todos os projetos/solicitações.
- Criação de um login.
- Login.
- Aceitação de uma solicitação.
- Rejeição de uma solicitação.
- Analisar solicitação.

Cadastrador:
- Cadastro de uma solicitação.
- Listagem de projetos/solicitações.
- Criação de um login.
- Login.

Gerente de grupos:
- Alterar grupos e acessos.
- Listagem de projetos/solicitações.
- Criação de um login.
- Login.

Alterador de solicitação:
- Alterar solicitação *Aceita*.
- Alterar solicitação *Esperando aceitação*.
- Alterar solicitação *Em análise*.
- Alterar solicitação *Em desenvolvimento*.
- Listagem de projetos/solicitações.
- Criação de um login.
- Login.

Leitor geral:
- Criação de um login.
- Login.
- Listagem de todos os projetos/solicitações.

Escalador de equipe:
- Definição da equipe de desenvolvimento.
- Criação de um login.
- Login.

Leitor de relatórios:
- Relatório de solicitações.

Público:
- Login.
- Listagem de projetos.

Solicitante:
- Cadastro de uma solicitação.
- Listagem de projetos/solicitações.
- Criação de um login.
- Login.
- Alterar solicitação *Esperando aceitação*.
- Cancelamento de solicitação.

Equipe de desenvolvimento:
- Adicionar arquivos,alterar resumo e listar projetos.
- Listagem de projetos/solicitações.
- Criação de um login.
- Login.

6.1.3 Casos de uso

Uma lista de casos de uso foi criada explicitando a maneira pela qual o usuário realiza as ações e, exibindo as restrições para tal.

Cada caso de uso possui um ou mais atores, onde cada ator representa um papel existente no sistema. Sendo assim, se o ator for, por exemplo, **Solicitante** então, qualquer pessoa que possuir este papel, pode executar este caso de uso, dependendo, é claro, de outras restrições impostas pelo mesmo, como por exemplo, o estado em que a solicitação se encontra.

1. Login Ator: Todos

Todo usuário do sistema deve efetuar o login.

Caso ele não possua um login ele pode criá-lo(UC 1.1) ou acessar o sistema como usuário Público.

1.1. Criação de um login Ator: Todos exceto Público

O usuário pode criar um login passando um nome, login e uma senha.

O login não pode estar cadastrado previamente.

Neste estágio, o usuário será considerado com o papel **Solicitante**.

2. Listagem de projetos/solicitações Ator: Todos exceto Público

Cada ator visualizará os projetos/solicitações que possuir acesso.

2.1. Listagem de projetos Ator: Público O ator visualizará os projetos que possuir acesso.

2.2. Listagem de todos os projetos/solicitações Ator: Leitor geral, Aceitador

O ator possui acesso a todos os projetos/solicitações do sistema.

3. Cadastro de solicitação Ator: Solicitante, Cadastrador

O solicitante preenche um formulário explicando seu pedido de projeto.

A solicitação entra no sistema com o status *Esperando aceitação*.

Automaticamente o sistema disponibiliza permissão de acesso a este novo projeto/solicitação para a pessoa que o cadastrou, leitor geral e aceitador.

Todos os gerentes de projeto, aceitadores e leitores gerais recebem um email informando a inclusão da solicitação.

Um registro de log é criado. Tipo de log: Criação.

4. Aceitação de uma solicitação Ator: Aceitador

O ator seleciona uma solicitação que esteja com status *Esperando aceitação* ou *Em analise* de uma listagem e a aceita.

A solicitação muda o status para *Aceita*.

Caso a solicitação já possua uma equipe de desenvolvimento, ela passará para o status *Em desenvolvimento*.

Observe que isto ocorre caso a descrição tenha sido alterada quando a mesma estava com status *Em Desenvolvimento*.

5. Rejeição de uma solicitação Ator: Aceitador

O usuário seleciona uma solicitação ainda não aceita de uma listagem e a rejeita. A solicitação muda o status para *Rejeitada*.

6. Alterar solicitação aceita Ator: Alterador de Solicitação

O ator seleciona a solicitação de uma listagem e edita seus detalhes.

Ela volta a ter o estado *Esperando aceitação*.

Um email é enviado a todos os gerentes de projeto, aceitadores, e leitores gerais informando a alteração.

6.1. Alterar solicitação esperando aceitação Ator: Solicitante e Alterador de Solicitação

O ator seleciona a solicitação de uma listagem e edita seus detalhes. Um email e enviado ao solicitante indicando que sua solicitação foi alterada(não funciona pois o solicitante nem sempre tem email).

6.2. Alterar solicitação Em Desenvolvimento Ator: Alterador de solicitação

O ator pode alterar todos os campos da solicitação. Mas, se alterar a descrição, ela voltará para status Esperando aceitação.

7. Definição da equipe de desenvolvimento Ator: Escalador de equipe

O ator seleciona uma equipe responsável pelo desenvolvimento de uma solicitação que foi aceita. Os participantes da equipe recebem um e-mail com o nome da solicitação que lhes foi incumbida. Assim que um projeto ganha um ou mais desenvolvedores, ele passa para o status Desenvolvimento. Quando ele ficar sem nenhum desenvolvedor volta para o status Aceito.

8. **Alterar solicitação Em analise** Ator: Solicitante e Alterador de Solicitação

   O ator seleciona a solicitação de uma listagem e muda seus detalhes. Um email e enviado ao solicitante e ao **Alterador de Solicitação** indicando que a solicitação foi alterada. Se o ator for o solicitante o status da solicitação volta a ser *Esperando aceitação*.

9. **Analisar solicitação** Ator: Aceitador

   O ator seleciona uma solicitação que esteja com status *Esperando aceitação* e muda seu status para Em analise.

10. **Conclusão de projeto** Ator: Gerente de projeto

    O ator escolhe um projeto que esteja com status *Em Desenvolvimento* e o conclui.

    Um email é enviado a todos os membros da equipe responsável e para o Adilson Simonis.

11. **Adicionar/Remover arquivos,alterar resumo e listar projetos** Ator: Equipe de desenvolvimento, Gerente de projeto

    Só é possível executar estas operações se o projeto estiver com o status *Em Desenvolvimento*.

    O ator visualiza uma lista com os projetos em que está participando ou já participou. Ele escolhe um projeto e envia arquivos para o mesmo. Caso o ator seja da equipe de desenvolvimento, ele só poderá remover os arquivos que enviou. Se o ator for **Gerente de projeto** poderá remover quaisquer arquivos.

    Todos os participantes da equipe de desenvolvimento deste projeto recebem um email informativo assim que um arquivo é adicionado, removido ou o resumo da solicitação é alterado.

12. **Alterar grupos e acessos** Ator: Gerente de grupos

    O ator cadastra,altera e remove grupos, além de definir os papéis de cada um através da tela de gerenciamento de acessos e grupos.

    Na tela de gerenciamento de acessos ele define os projetos e solicitações que cada grupo tem acesso. Observe que, quando um desenvolvedor é escalado para uma solicitação, ele recebe acesso à mesma e, na tela de gerenciamento de acessos ele não pode perder este acesso. Para que ele deixe de visualizar o projeto é necessário que seja excluído da equipe de desenvolvimento.

    O gerente de grupos não pode alterar seus papéis pois, se pudesse ele seria igual ao super usuário do sistema.

13. **Exclusão de projeto** Ator: Gerente de projeto

    O ator escolhe um projeto da lixeira e apaga definitivamente o mesmo.

Tudo referente ao projeto é excluído do sistema.

13.1 Mover projeto para a lixeira Ator: Gerente de projeto

O ator escolhe um projeto e apaga o mesmo. Neste momento ele é movido para a lixeira.

13.2 Recuperação de um projeto Ator: Gerente de projeto

O ator seleciona um projeto da listagem e decide recuperá-lo. O projeto retorna à listagem de projetos.

14.2. Relatório de solicitações Ator: Leitor de relatórios

O ator visualiza uma lista com todas as solicitações ordenados por status e data. Cada status recebe uma cor característica. O ator pode filtrar por mês e ano da data entrada/data de entrega, desenvolvedor, etc.

15. Cancelamento de solicitação Ator: Gerente de projeto e Solicitante

Um **Gerente de Projeto** pode cancelar qualquer solicitação que esteja com status: *Em Desenvolvimento* ou *Aceito*.

Um **Solicitante** pode apenas cancelar suas solicitações que estiverem com os seguintes status:

- *Esperando aceitação*.
- *Em análise*.
- *Aceita*.

15.1 Recuperação de solicitação Ator: Gerente de projeto

O ator seleciona uma solicitação que esteja *Cancelada* e decide recuperá-la. A solicitação volta a ter o status *Esperando aceitação*.

16. Reabertura de projeto Ator: Gerente de projeto

O sistema exibe o detalhe de um projeto. O ator pode então optar por reabrir o mesmo.

O projeto volta para o status de solicitação *Em Desenvolvimento*.

6.1.4 Diagrama entidade relacionamento

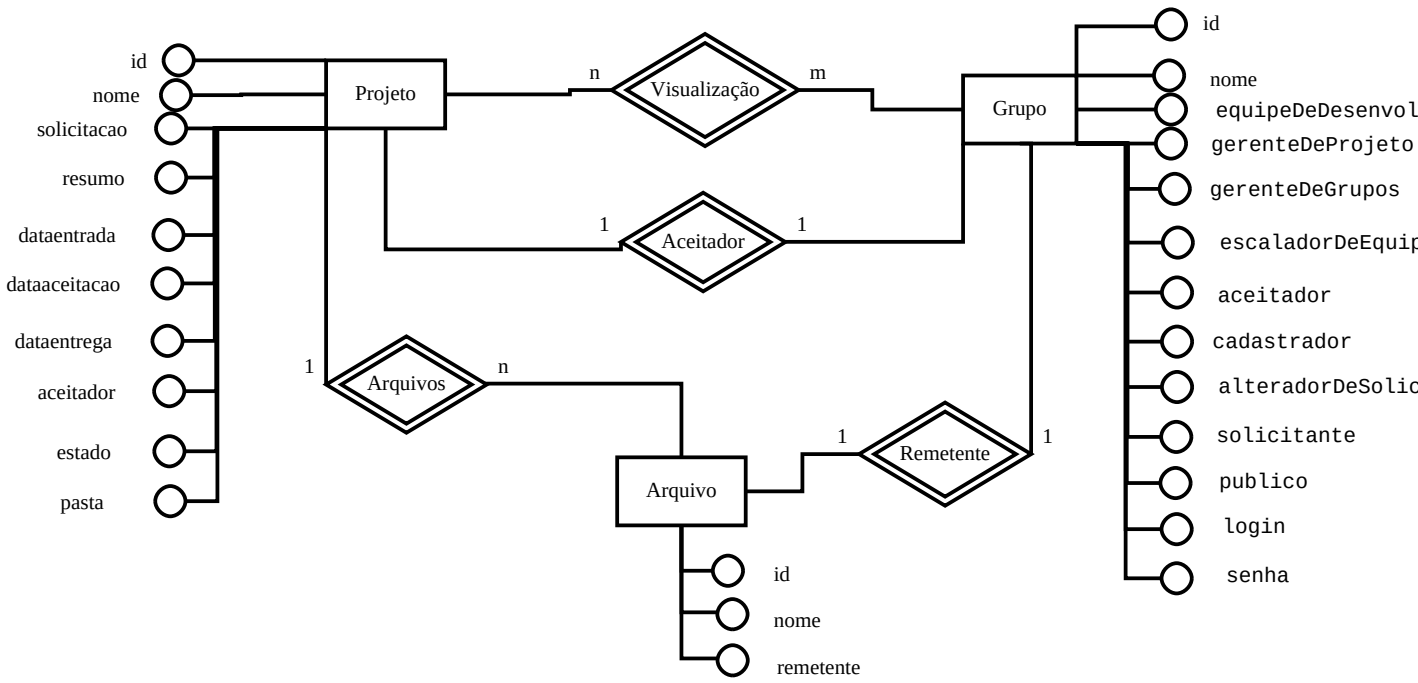


Figura 26: Diagrama simplificado dos relacionamento entre as tabelas que compõe o sistema.

6.1.5 Stored procedures

meta_insert_solicitação: Insere/altera uma solicitação.

meta_insert_arquivo: Insere/altera um arquivo.

meta_insert_grupo: Insere/altera um grupo.

meta_dar_acesso: Dá acesso de leitura a uma solicitação/projeto a uma determinada pessoa.

meta_insert_equipe_de_projeto: Escala um pessoa para a equipe de desenvolvimento de uma solicitação.

meta_apagar_grupo: Apaga um grupo.

meta_retirar_acesso: Retira o acesso de leitura a uma solicitação de uma determinada pessoa.

meta_apagar_arquivo: Apaga um arquivo

meta_aceita_solicitação: Solicitação passa para o estado *Aceita*.

meta_rejeita_solicitação: Solicitação passa para o estado *Rejeitada*.

meta_analisa_solicitação: Solicitação passa para o estado *Em análise*.

meta_concluir_solicitação: Solicitação passa para o estado *Concluído*.

meta_cancela_solicitação: Solicitação passa para o estado *Cancelada*.

meta_recupera_projeto: Projeto passa do estado *Apagado* para *Concluído*.

meta_recupera_solicitação: Solicitação passa do estado *Cancelada* para:

Em Desenvolvimento: caso já possua uma equipe de desenvolvimento.

Esperando aceitação: caso contrário.

meta_mover_lixeira_projeto: Muda o estado do projeto para *Apagado*.

meta_reabrir_projeto: Projeto passa do estado *Concluído* para *Em Desenvolvimento*.

meta_apagar_solicitação: Apaga uma solicitação.

meta_apagar_projeto: Projeto passa para o estado *Apagado*.

meta_retirar_equipe_de_projeto: Retira uma pessoa da equipe de desenvolvimento de uma solicitação.

meta_projetos_com_acesso: Retorna uma lista de projetos/solicitações para os quais a pessoa informada possui acesso.

meta_projetos_sem_acesso: Retorna uma lista de projetos/solicitações para os quais a pessoa informada não possui acesso.

6.1.6 Views

meta_vw_equipe_de_projeto Retorna a equipe de projeto de todas as solicitações.

Basta adicionar o numero da solicitação na cláusula WHERE a fim de obter de uma solicitação específica.

meta_vw_listagem_projetos Dados dos projetos.

meta_vw_proj_lixeira dados dos projetos que estão com status *Apagado*

meta_vw_listagem_solicitações Dados das solicitações.

meta_vw_solicitações Apenas dados das solicitações. A listagem anterior retorna arquivos e equipe de

desenvolvimento juntamente com os dados da solicitação.

meta_vw_proj_por_desenvolvedor Lista de projetos. Pode filtrada por desenvolvedor.

meta_vw_listagem_grupos Listagem com os dados dos grupos.

meta_vw_arquivos Listagem de arquivos. É possível filtaro resultado por solicitação ou remetente.

meta_vw_rel_solicitações Listagem com projetos e solicitações, inclusive os projetos apagados.

meta_vw_logs Logs do sistema.

6.2 SGT

6.2.1 Telas

Algums telas do sistema SGT.

Infelizmente, a maioria das telas ainda está em fase de construção e, por apresentarem alguns erros, decidimos não colocá-las.

Figura 27: Menu principal do sistema

Figura 28: Dados de uma pessoa

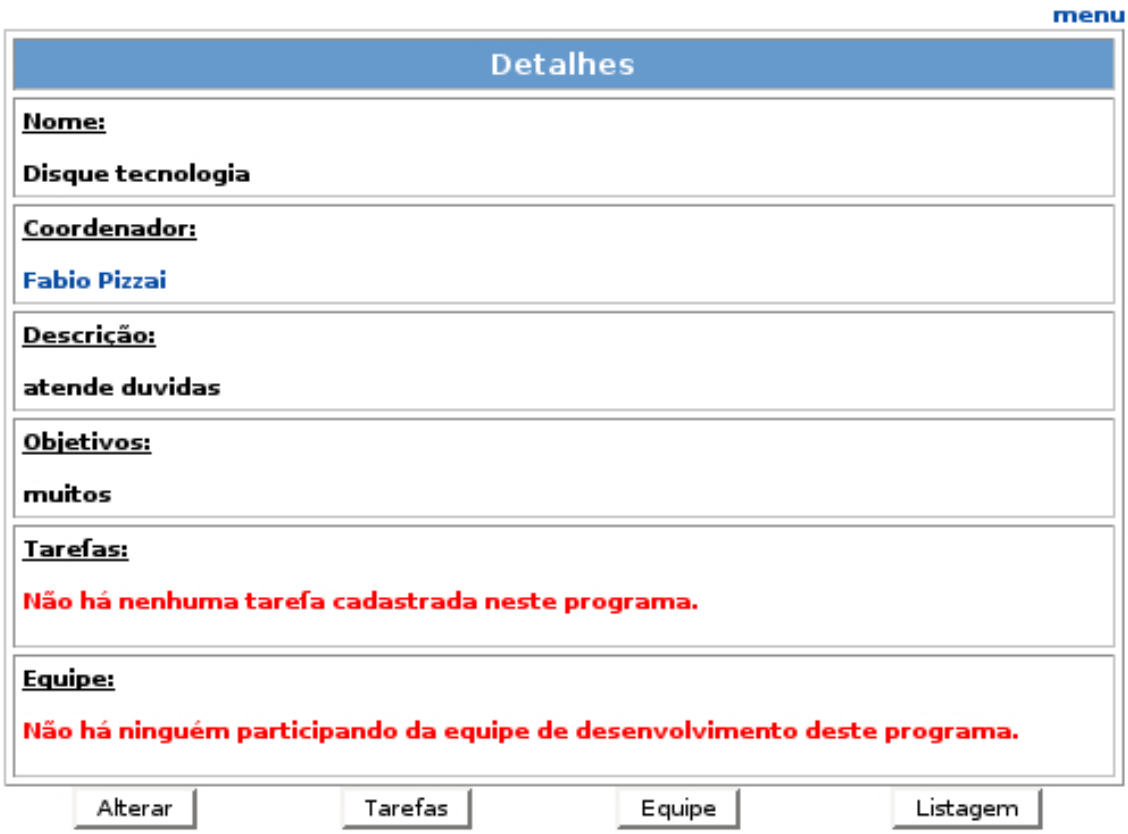


Figura 29: Dados de um programa

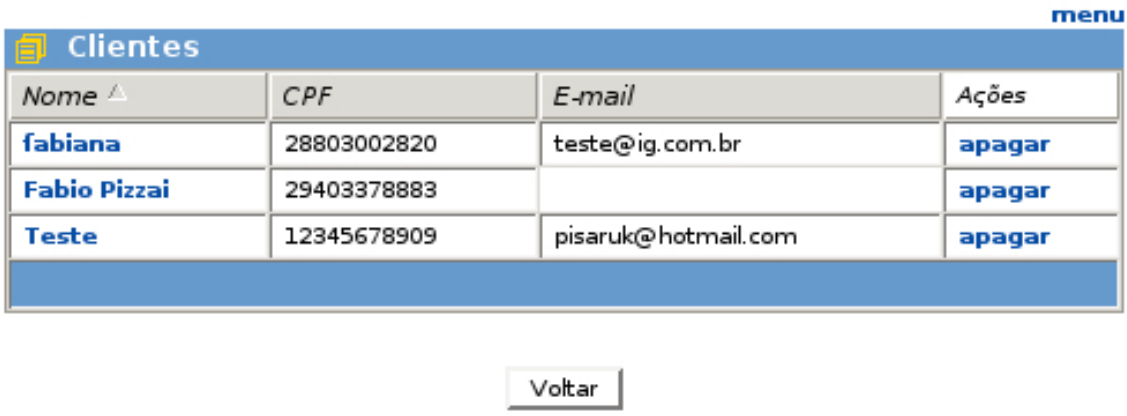


Figura 30: Listagem de pessoas. Clicando no título das colunas é possível ordená-las crescente ou decrescentemente.

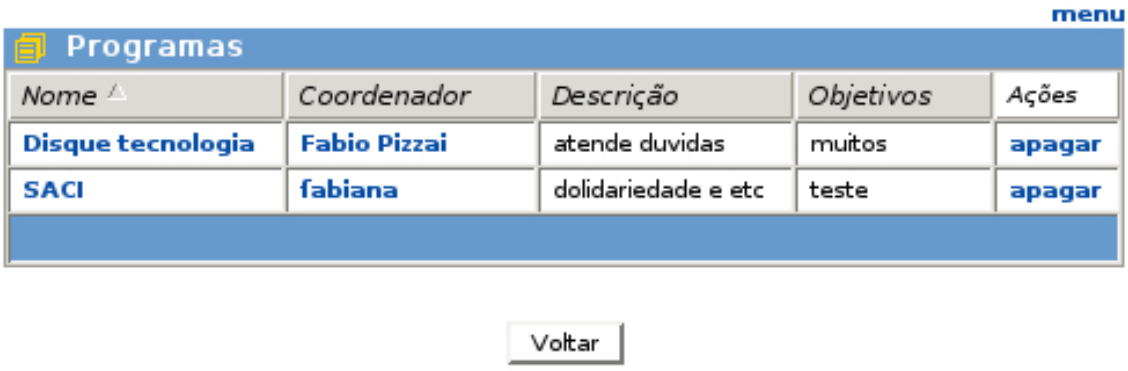


Figura 31: Listagem de programas. Clicando no título das colunas é possível ordená-las crescente ou decrescentemente.

Referências

[1] Mada Maten. Sanjeev Krishnan. Linda DeMichiel. Beth Stearns. *Applying Enterprise JavaBeans SecondEdition. Component-Based Development for the J2EE Plataform.*

[2] Monson-Haefel, Ruchard. *Enterprise JavaBeans 3rd.*

[3] Java Sun BluePrints: Core J2EE Patterns.

[4] Scott Stark and The JBoss Group. *JBoss Administration and Development Third Edition3(3.2.x Series)*

[5] Craig Walls. Norman Richards. *XDocLet in Action*

7 Agradecimentos

A Jesus pela minha vida, pela inteligência que me deu e, por esta maravilhosa oportunidade de estudar numa faculdade gratuita e tao bem
conceituada como a USP.
A minha família, por ter sempre me apoiado, e acreditado em mim.
Aos amigos e colegas do IME por estes quatro ótimos anos de convivência.
Ao meu orientador por tudo que me ensinou mesmo nos poucos encontros que tivemos.
Ao pessoal do Naeg por criarem um ambiente de trabalho muito gostoso.
A todos que conviveram comigo durante estes quatro anos...