UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
ESCOLA POLITÉCNICA
DEPARTAMENTO DE ELETRÔNICA E DE COMPUTAÇÃO

Service Oriented Plone User Management

Autor: _____
Nelson Iiboshi Hargreaves Costa

Autor: _____
Vicente Roxo Mundim

Orientador: _____
Marcelo Luiz Drumond Lanza

Examinador: _____
Edilberto Strauss

Examinador: _____
Flavio Luis de Mello

DEL

Julho de 2008

Preâmbulo

AGRADECIMENTOS

Agradeço ao Criador, à minha familia e aos meus estimados amigos.

Nelson Iiboshi Hargreaves Costa

Agradeço à minha famíla e meus amigos, que sempre me apoiaram durante o desenvolvimento desse projeto. Em especial à toda a comunidade Plone que disponibiliza gratuitamente um software de alta qualidade, com excelente documentação, e com isso permite o desenvolvimento de projetos como esse. Por fim, agradeço ao Departamento de Engenharia Eletrônica, por me auxiliar no caminho percorrido até aqui.

Vicente Roxo Mundim

Agradecemos aos examinadores pela credibilidade ao nosso trabalho.

Os Autores

DEDICATÓRIAS

Dedicamos esse trabalho ao povo brasileiro e em especial aos alunos do Departamento de Engenharia Eletrônica e de Computação. Este projeto é uma pequena forma de retribuir o investimento e confiança dados a nós.

Os Autores

RESUMO

O sistema criado neste projeto tem como finalidade simplificar a gerência da base de usuários de uma rede UNIX. Para alcançar tal objetivo foi utilizado a interface de um sistema de gestão de conteúdo. Foi construído uma arquitetura orientada a serviços para unir a extremidade UNIX ao sistema web. A primeira implantação em um ambiente de produção foi realizado no Departamento de Eletrônica e de Computação da Universidade Federal do Rio de Janeiro em Agosto de 2008, para esse ambiente foram criados tipos específicos de estruturas de dados assim como elementos de design.

ABSTRACT

The system built on this project intends to remove the complexity of managing the user base of an UNIX network. To achieve this goal, a content management system graphic user interface was used. An extensible service-oriented architecture was built to bridge the UNIX network backend to the content management system. The first production environment deploy was made on the Electronic and Computing Engineering Department on the Federal University of Rio de Janeiro on August of 2008, by creating a packaging with the specific data-types and design.

PALAVRAS-CHAVE

- Sistemas de Gestão de Conteúdo
- Arquitetura Orientada a Serviços
- Aplicações Web
- Gerenciamento de Usuários.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

Capítulo I – Introdução

Nós, seres da escola de ciências exatas, temos um fascínio especial pela informação. Estamos sempre em busca das melhores formas de manipular, visualizar e armazenar informações. Empregamos grande parte do nosso tempo tentando formular processos eficientes para lidar com o volume incomensurável de informações que nos é servido diariamente.

Nada mais natural do que criarmos soluções para tornar mais eficientes nossas horas, suavizar um pouco a avidez por ser à prova de falhas e delegar o trabalho de memorização às máquinas que tomam cada vez mais parte de nosso cotidiano: os computadores.

I.1 MOTIVAÇÃO

O Departamento de Eletrônica e de Computação (DEL) da Universidade Federal do Rio de Janeiro (UFRJ) atende a uma grande quantidade de alunos. Periodicamente certa quantidade de alunos deixa o departamento e outra é incorporada. A cada um destes alunos está relacionada uma quantidade de informações que precisa ser armazenada de alguma forma. Alunos cursam matérias, realizam aulas práticas em laboratórios, etc. Professores lecionam matérias, lançam notas, etc. Este conjunto de informações é atualizado todo período, incluindo também as grades de horários e as informações sobre as matérias oferecidas pelo departamento. É possível enumerar e classificar um grande volume de informações relacionadas com o departamento.

Além destas informações, o gerenciamento da rede de computadores do DEL é diretamente afetado pela entrada e saída de alunos do departamento. Novas contas de acesso à rede precisam ser abertas no início de cada semestre, senhas individuais devem ser cadastradas para os novos usuários, contas de ex-alunos precisam ser bloqueadas/canceladas, listas de distribuição de mensagens eletrônicas precisam ser criadas, etc. Os procedimentos necessários para cumprir estas tarefas são repetitivos, tediosos e lentos. Além disso, quando realizados manualmente pelos administradores, estes procedimentos são suscetíveis a erros.

A principal motivação para o desenvolvido deste projeto foi melhorar e facilitar a administração da rede DEL, integrando todas as informações sobre os alunos, professores, funcionários e demais pessoas relacionadas com o departamento.

I.2 HISTÓRICO

Em 2002, foi desenvolvido, pelo aluno Pedro da Fonseca Vieira, um sistema que visava integrar a gerência das informações dos alunos, funcionários, professores e demais pessoas relacionadas com o departamento com a gerência da rede de computadores do DEL. Este sistema, denominado Sistema Integrado do DEL (SID) [1], foi concebido como projeto

de final de curso e foi implementado utilizando a linguagem de programação PHP (*Pre Hypertext Processor*). Embora tenha atendido às necessidades iniciais do DEL, este sistema rapidamente se tornou obsoleto por utilizar tecnologias e arquitetura que não eram adequadas ao tipo de sistema que se desejava criar, pois não permitia uma evolução eficiente do software e tampouco uma manutenção de baixo custo.

O projeto desenvolvido neste trabalho de final de curso é denominado *Service-Oriented Plone User Management* (*Soplum*) e tem como objetivo suplantar a implementação do SID, utilizando tecnologias mais adequadas para a construção de um sistema integrado de gestão e buscando principalmente a criação de um software manutenível e extensível.

I.3 OBJETIVOS

O objetivo principal deste trabalho foi, com a utilização de software livre, criar uma arquitetura extensível e resistente ao tempo, se apoiando em tecnologias e paradigmas respeitados pela comunidade de desenvolvedores ao redor do mundo, com vistas à construção de uma ferramenta capaz de gerenciar usuários. Dentre estas tecnologias, devem ser destacados os Sistemas de Gestão de Conteúdo (*Content Management Systems* – CMS) e a Arquitetura Orientada a Serviço (*Service-Oriented Architecture* – SOA).

A proposta do *Soplum* é manter uma base de usuários única que possa servir para gerenciar as informações dos integrantes de um departamento e que seja integrável a outros serviços externos através do uso de extensões modulares que possam ser adicionadas ao sistema quando necessário. O propósito deste trabalho não foi a simples entrega de um sistema que cubra todos os casos de uso. Queríamos sim, utilizar esta oportunidade para explorar a fundo e utilizar metodologias e técnicas para criar um software de qualidade e que tivesse utilidade numa situação real. Em suma, o foco principal do projeto se dividiu no processo de produção do software e no sistema de gerenciamento de usuários extensível, para que futuramente outros projetos possam ampliar a abrangência de serviços externos.

Para que a implementação desse projeto fizesse sentido, utilizamos o DEL como fonte para os requisitos do sistema. Utilizamos a arquitetura extensível que foi modelada e codificada visando criar módulos correspodentes aos sistemas externos básicos existentes na rede de computadores do departamento.

Foi criado um portal, onde todos os alunos, professores e funcionários podem se registrar e manter uma página pessoal. Este portal possui o módulo *Soplum* instalado e através dele são administrados os serviços da rede de computadores DEL.

Vale ressaltar neste ponto, que o *Soplum* não é o portal DEL. O portal utilizará o *Soplum*, que é um sistema de natureza genérica e que é adequado às necessidades do departamento.

O desenvolvimento deste trabalho se deu em três divisões claras:

i. Interface web para a administração de usuários e cliente de serviços externos, onde módulos clientes de serviço poderão ser integrados.
ii. Provedor de serviços base, onde os módulos de serviço poderão ser integrados.
iii. Implementação dos serviços de gerência de contas *Unix* e *Samba*, de listas de mensagens eletrônicas (*Mailman*) e de cotas de espaço em disco.

A metodologia utilizada para desenvolver o projeto é baseada na metodologia *Ágil* [2] denominada *Scrum* [3], onde o projeto é desenvolvido em iterações curtas, com participação intensa do cliente nas decisões e na evolução do sistema.

I.4 ESTRUTURA DA DOCUMENTAÇÃO

Este capítulo traz a introdução, objetivos e motivação do projeto. O Capítulo II apresenta a revisão da literatura e fundamentos teóricos. No Capítulo III é feita a comparação de alguns sistemas de gestão de conteúdo assim como a definição do sistema de gestão base para o projeto. O Capítulo IV se refere a parte do projeto de software e a implementação do sistema base, seguido do Capítulo V com a implementação dos serviços e as características do Portal DEL. O documento tem sua conclusão no Capítulo VI.

O Apêndice A concentra os acrônimos e siglas utilizadas no relatório. No Apêndice B encontram-se os requisitos mínimos para o desenvolvimento do sistema. O Apêndice C tem as informações de requisitos para instalação e uso do software resultante do projeto. O Apêndice D contém aos passos de instalação do software. O Apêndice E tem as referências para o manual do usuário.

O Capítulo IV deste documento utiliza alguns tópicos das normas ANSI/IEEE 830 [2] e ANSI/IEEE 1016 [3] tidos como relevantes para uma organização da seção de projeto de software.

Capítulo II - Revisão da Literatura e Fundamentos Teóricos e Práticos

Este capítulo engloba os tópicos que foram de grande importância na tomada de decisões durante a modelagem e a implementação dos módulos do projeto.

II.1 METODOLOGIA ÁGIL PARA DESENVOLVIMENTO DE SOFTWARE

Uma metodologia Ágil agrega um conjunto de melhores práticas utilizadas no processo de desenvolvimento de software e gestão de equipe. Este movimento teve como ponto de partida a publicação do *Agile Manifesto* [4], quando um conjunto de gestores de equipes de desenvolvimento de software uniu suas experiências para formular um melhoramento na forma de trabalho de suas equipes. Duas metodologias Ágeis que se destacam no mercado hoje são o *Scrum* [5] e o *Extreme Programming* (XP) [6].

Tais melhores práticas prezam o melhor aproveitamento da capacidade de análise e produção de código no lugar da criação de infindáveis volumes de documentação. Todos concordavam que o excesso de formalismo das metodologias correntes como o *Rational Unified Process* (RUP) [7] trazia muitas medidas antiprodutivas e "engessava" os analistas de sistemas. Este engessamento limitava o potencial dos analistas de sistemas que eram obrigados a gerar uma quantidade excessiva de documentação, perdendo assim a oportunidade de exercitar seu potencial criativo e se tornando operadores de ferramentas de *Computer-Aided Software Engineering* (CASE).

Para o movimento *Ágil* vale a premissa de que é mais importante entregar software testado e funcionando, do que gerar uma grande quantidade de documentação que potencialmente se tornaria obsoleta a cada nova funcionalidade adicionada ao software. O cliente constantemente muda de opinião e de vontade. Documentar de antemão o projeto inteiro, pode ser um desperdício incalculável do bem menos abundante, o tempo da equipe.

Vale frisar que a metodologia Ágil não deixa de produzir artefatos de documentação. Porém, o faz de maneira racionalizada quando comparado às metodologias como o RUP. Os artefatos são gerados somente sob demanda, nunca tentando "prever", mas sim procurando documentar somente o que está concluído. O *Scrum* é aderente ao ISO 9001 e ao CMM nível dois, que são normas que primam por atestar a qualidade de um produto. Para receber estas certificações o produto deve ter uma base bem documentada.

A distribuição de desenvolvedores em cubículos, isolados fisicamente dos colegas de equipe, também foi alvo de críticas, pois a falta de interação entre os membros do projeto fazia com que os mesmos não tivessem uma clara idéia do *todo* do qual faziam *parte*.

O *Scrum* e outras metodologias Ágeis, como o *Extreme Programing* (XP), prezam a interação da equipe através de reuniões diárias de curta duração (não devem passar de quinze minutos), onde os participantes seguem um roteiro, que de maneira prática se resume a responder aos colegas três simples perguntas: “O que eu realizei desde a reunião passada?”, “Quais serão as minhas atividades de hoje?” e “Existe algo impedindo meu trabalho?”. Dessa forma, todos ficam a par da evolução até o momento da entrega do produto. Além disso, caso um dos participantes tenha facilidade com uma dada atividade, o mesmo poderá prontamente contribuir com seu conhecimento, aproveitando melhor as competências disponíveis.

II.2 PLATAFORMA WEB PARA APLICAÇÕES

Num intervalo de dez anos, assistiu-se uma pequena empresa criar um mecanismo de busca, apoiado em maciço emprego de trabalho automatizado de indexação de informações, e tomar de assalto a cena da Internet. O Google, numa inovação disruptiva, trocou o trabalho de vários homens que passavam horas catalogando páginas em diretórios e categorias, por um algoritmo de avaliação de páginas e um poderoso mecanismo de busca distribuído. Dessa forma, uma rede com milhares de computadores “comuns” interligados poderia automaticamente indexar páginas da web e torná-las buscáveis através de palavras chave.

Recentemente, esta mesma empresa, que hoje não é mais vista como visionária ou vanguardista, mas sim como agente formador de opinião e tendências, apostou no *web desktop*. Nesta plataforma, a partir do seu navegador predileto, se tem acesso a todas as ferramentas de produtividade necessárias para organizar, gerenciar, armazenar e executar trabalhos. Este conjunto de ferramentas agrega desde correio eletrônico até calendários colaborativos.

O *web desktop* é tão apelativo para os usuários graças a sua acessibilidade. Os programas “estão” na Internet, dispensando a instalação ou a atualização nos computadores de onde se deseja utilizá-los. Todos os programas estão no servidor, são naturalmente multi-plataformas e podem ser executados em qualquer computador com um navegador. Outra vertente que naturalmente surgiu com os sistemas *on-line* foi à vertente da colaboração. A idéia é ter várias pessoas trabalhando num documento, até mesmo simultaneamente, com fluxos de trabalho sendo orquestrados por sistemas integrados, distribuição e delegação de responsabilidades a nível pessoal.

A *web* é um meio rico, um ambiente que expande as possibilidades do trabalho em equipe e do funcionamento de estruturas organizacionais não convencionais. Agilidade, produtividade e auto-organização são palavras que soam bem para qualquer organização, independente da sua natureza. É esta plataforma que o projeto elaborado neste trabalho utiliza como interface para o usuário, aproveitando todas as suas possibilidades e deixando um arcabouço de conhecimento para que outros alunos possam dar continuidade a este trabalho.

II.3 SISTEMAS DE GESTÃO DE CONTEÚDO

Os CMS's foram criados com a intenção de minimizar o esforço de manutenção de um portal na Internet. São programas que possuem modelos e ferramentas para adicionar artigos, notícias, documentos e outros conteúdos ricos, de maneira simples, sem a necessidade de especialização do mantenedor do portal.

II.3.1 DEFINIÇÃO DE CONTEÚDO

Um dos conceitos chaves dos CMS é o conceito de *conteúdo*. Entende-se como *conteúdo*, todo e qualquer item armazenado pelo sistema. Uma abstração comum seria pensar que um *conteúdo* é uma ficha de registro genérica, nela existem campos específicos que descrevem esse item. Aos campos específicos damos o nome de *metadados*. Estes *metadados* possuem um tipo comum como data, número, caractere ou um tipo especial, que pode ser denominado *relacionamento* e que define uma associação entre dois *conteúdos*.

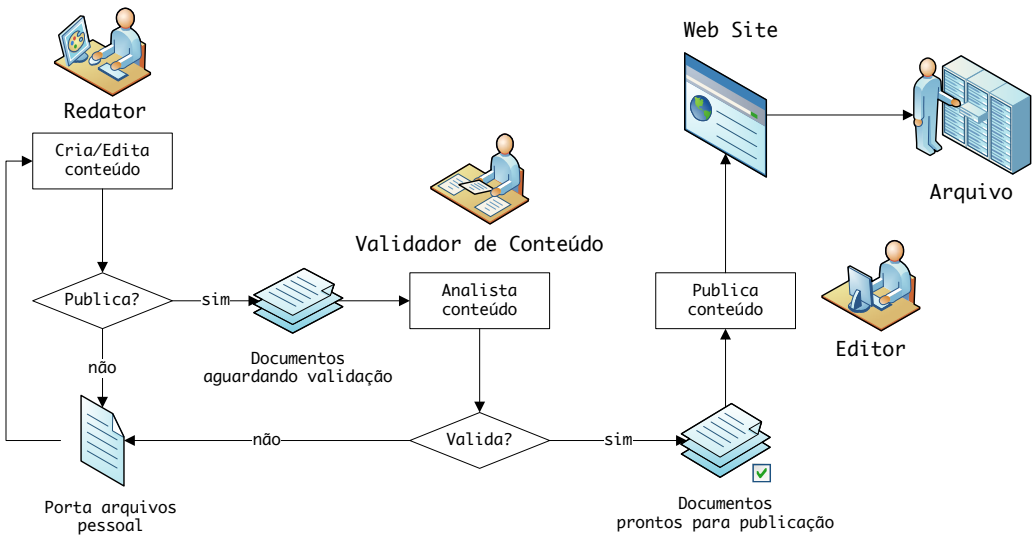
Para exemplificar o conceito de conteúdo: um livro possui informações particulares como: título, autor, data de publicação e código ISBN. Poderíamos criar um tipo de conteúdo chamado Livro e adicionar a ele estes *metadados*, criando um catálogo. Criaríamos um tipo de conteúdo chamado Pessoa onde definiríamos *metadados* que descreveriam/identificariam uma pessoa. A seguir criaríamos um *relacionamento* entre Livro e Pessoa para relacionar um autor a sua obra.

II.3.2 FLUXO DE TRABALHO

Um fluxo de trabalho é o processo pelo qual um conteúdo é submetido durante a sua existência. Cada conteúdo pode ou não passar por um fluxo de trabalho. Este processo é sempre representado por estados, transições entre estados e atores envolvidos. Os atores envolvidos fazem com que um conteúdo transite entre um estado e outro. Cada transição dessas dispara um evento, como numa máquina de estados finita.

O uso de fluxos de trabalho permite a obtenção de um controle granular e distribuído sobre o que acontece num portal. Como um exemplo, em uma empresa de comunicações, um jornalista publica uma matéria (que é um conteúdo). Esta matéria entra no estado "aguardando validação" e espera numa triagem para que um editor possa revisar a matéria e então publicá-la, alterando o estado deste conteúdo para "conteúdo publicado". O editor pode também, decidir em qual seção do portal a matéria irá aparecer. Uma vez publicada, a matéria irá aparecer para os visitantes do portal.

Note que se distribuiu a responsabilidade de maneira adequada. O jornalista precisa escrever matérias, não precisa saber como isto será publicado. O editor pode validar as matérias e decidir em qual categoria e posição na página inicial este conteúdo irá aparecer. Dessa forma cada pessoa desempenha seu papel no portal de forma colaborativa. Uma ilustração do processo pode ser vista na figura II.1.



Fluxograma representando um fluxo de trabalho de um conteúdo de 3 estados:

1) O Redator cria documento este é iniciado no estado 'em criação', ele pode publicar ou não o documento, se decidir publicar o conteúdo passará para o estado seguinte que é 'aguardando de validação'.
2) Validador de conteúdo acessa todos os conteúdos neste estado, podendo validá-los para que o conteúdo fique disponível no estado 'pronto para publicação' ou pode rejeitá-los para que voltem ao estado 'em criação' para que o redator possa alterar seu conteúdo para que fique de acordo com o requerimento do validador.
3) O Editor têm acesso aos documentos que estão no estado 'pronto para publicação' e pode publicá-los passando o documento a ter o estado 'publicado'.
4) O mesmo editor que publicou o documento pode retirá-lo do ar, nesse ponto o arquivo passa para o estado 'arquivado'.

Figura II.1 - Representação do fluxo de trabalho de uma revista *on-line*

II.4 PERSISTÊNCIA DE DADOS

O conjunto de dados armazenados em um sistema é o que dá sentido a existência do mesmo. Um sistema funciona bem se a capacidade de manuseio dos dados se faz de maneira intuitiva. Os sistemas integrados têm, em sua maioria, o propósito de armazenar um volume grande de informações estruturadas. Isso faz da escolha de uma boa estratégia de armazenagem de dados um ponto crítico.

Além do simples fato de armazenar com eficiência e segurança, também é importante que se possam disponibilizar os dados através de protocolos de comunicação consolidados, como o *File Transfer Protocol*[1] (FTP) e o *Common Internet File System*[2] (CIFS). Estes protocolos apresentam interfaces mais amigáveis para a transferência dos dados.

É muito comum o armazenamento de arquivos binários de grande tamanho e de formatos considerados padrões do mercado como: *Microsoft Word*, *Microsoft Excel* e o *Adobe Portable Document File* (PDF). Por isso, uma boa estratégia de persistência de dados deve

[1] Protocolo mais recomendado para gerência remota de arquivos utilizando uma rede TCP (*Transmission Control Protocol*)/IP (*Internet Protocol*).
[2] Protocolo bastante utilizado por ser mais portável, podendo ser integrado aos gerenciadores de arquivos nativos como o *Windows Explorer*. Este protocolo é usado pela Microsoft para o compartilhamento de arquivos em redes *Windows*.

levar em consideração possibilitar a indexação desses tipos de arquivo, permitindo buscas pelo seu conteúdo e não somente pelos *metadados* associados ao conteúdo.

II.4.1 Estratégias comuns de Armazenamento para Objetos de dados

Como a implementação do armazenamento de dados é diretamente afetada pela adoção de uma linguagem de programação, a estratégia escolhida foi atrelada ao CMS base escolhido. O projeto busca uma solução onde facilidade de desenvolvimento não sacrifique desempenho ou usabilidade. Um comparativo das estratégias mais comuns foi feito para que se escolha uma solução adequada.

II.4.1.1 Serialização de Objetos

A serialização de objetos é a maneira mais simples de se persistir os dados de um programa. Faz-se uma cópia direta da área de memória onde o objeto está armazenado para a mídia desejada, comumente utilizando-se arquivos para tal finalidade. A implementação dessa estratégia é muito barata, não possuindo inteligência alguma, visto que não há transformação na representação de dados.

O processo de escrita dos objetos é chamado de *marshalling*, enquanto que o processo inverso é chamado de *unmarshalling*. Algumas versões deste processo utilizam uma linguagem intermediária para tornar os dados persistidos legíveis para o ser humano, inserindo alguma complexidade ao software, criando *marshallers* cuja saída é um formato proprietário. Atualmente, linguagem mais utilizada com este propósito é a *eXtensible Markup Language* (XML),

Por se tratar de um algoritmo simples, a serialização é bastante utilizada para transportar objetos em protocolos de alto nível, como no *Asynchronous JavaScript and XML* (AJAX), em soluções de AJAX utilizando *JavaScript Object Notation* (JSON) e finalmente em protocolos mais robustos de *Remote Procedure Call* (RPC) como o *Simple Object Access Protocol* (SOAP) e o XML-RPC[3].

Apesar de simples, quando se trabalha utilizando essa abordagem, toda a árvore de objetos com relacionamentos precisa ser persistida para que o sistema se mantenha consistente. Isso obriga o desenvolvedor a se preocupar toda vez que desejar relacionar um objeto a outro, pois ele deverá se certificar que o objeto para o qual está se criando o relacionamento é serializável. Por esta característica, a serialização só é aconselhável para uso com estruturas de dados simples.

Uma das maiores dificuldades da serialização é lidar com volumes grandes de dados, pois o programa é obrigado, sempre que é carregado, a acessar todos os dados, trazendo-os

[3] Apesar de ser uma união de dois acrônimos o XML-RPC é o nome de uma solução de RPC, que poderá ser vista com mais detalhes na seção “Comunicação entre Processos”.

para a memória. Por outro lado, toda vez que o programa é finalizado deve ser efetuada a persistência. Isso é impraticável para arquivos grandes, não importando a mídia de armazenamento em questão. Essa dificuldade é agravada em sistemas onde o armazenamento é remoto como, por exemplo, em uma unidade de rede. Neste caso, o programa teria que transmitir o arquivo na íntegra, mesmo que se fosse utilizar somente um trecho do mesmo.

Por estes motivos a serialização, apesar de simples, não se aplica para o projeto como estratégia de persistência. Porém apresenta-se como uma ótima solução para o tráfego de objetos numa conversação cliente/servidor.

II.4.1.2 MAPEAMENTO RELACIONAL-OBJETO

Na estratégia de mapeamento relacional-objeto (ORM - *Object-Relational Mapping*), é realizada uma transformação dos dados armazenados em objetos para dados relacionais armazenáveis em um banco de dados, através de um mapeamento bi-direcional. Uma classe é representada por uma tabela, um atributo dessa classe é representado por uma coluna na tabela, um relacionamento é representado por um identificador único do objeto relacionado e que ocupa uma coluna na tabela da classe e uma restrição de chave estrangeira. Logo, uma instância de uma classe é representada por uma linha nessa tabela.

Existem atualmente excelentes bibliotecas que fazem a lógica de negócio do ORM. O desenvolvedor deve criar os mapeamentos reversíveis, que normalmente são feitos em arquivos de configuração utilizando uma linguagem intermediária simples. O ORM permite que somente os objetos em uso estejam na memória do computador. A biblioteca tem ainda a responsabilidade de otimizar os acessos ao banco de dados, para não fazer leituras desnecessárias, e de carregar os objetos somente sob demanda. É de suma importância a utilização da memória do computador de maneira eficiente, tanto com relação à alocação, quanto com relação à liberação, mantendo um patamar adequado de objetos “carregados”. Esse dimensionamento inteligente permite que o conjunto de dados utilizados seja grande, sem que haja prejuízo para o usuário.

Os sistemas de gerenciamento de bancos de dados (SGBD) utilizam técnicas avançadas para armazenar volumes muito grandes de informação e são otimizados para obter um bom desempenho mesmo em condições extremas de altas taxas de acesso e de concorrência. Por isso, os SGBDs são um excelente meio de persistência. Além disso, devido à sua característica inerentemente transacional, são mídias confiáveis. Por todas estas razões, a estratégia de ORM é uma das mais utilizadas no meio corporativo.

O único revés dessa estratégia é obrigar que o cliente mantenha a instalação de um banco de dados relacional e a adição obrigatória de um atributo a mais em todas as entidades do sistema, ou seja, o identificador único.

II.4.1.3 BANCO DE DADOS ORIENTADO A OBJETOS

Os bancos de dados orientados a objeto são a solução ideal para o desenvolvedor que trabalha utilizando este paradigma, pois toda persistência é feita de maneira transparente, sem que haja necessidade de intervenção do desenvolvedor para a criação de mapeamentos ou de cuidados na hora de criar relacionamentos entre entidades.

Os bancos orientados a objetos são, assim como os bancos relacionais, otimizados para o acesso em condições extremas. Por razões comerciais e por ser uma tecnologia ainda emergente, poucos fabricantes de software implementam bancos orientados a objetos. Apesar disso, este tipo de banco de dados representa uma excelente aposta para o futuro.

II.5 ARQUITETURA ORIENTADA A SERVIÇO

A arquitetura orientada a serviço (SOA - *Service Oriented Architecture*) é uma evolução de outros conceitos como a programação modular e a computação distribuída. Um Serviço é um programa que oferece aos seus "clientes" um conjunto de lógicas de negócio, através de uma interface de acesso definida para um conjunto de operações conhecido (comumente chamado de "contrato").

A idéia é que cada serviço tenha uma área específica de atuação e que a sua codificação e manutenção independa do conhecimento interno de outros serviços ou de sua linguagem de programação. O que importa em um serviço é apenas conhecer suas entradas e saídas. Toda a complexidade do serviço não é exposta para o cliente.

Desta forma, com conhecimento de todos os serviços disponíveis para uso, o esforço de criação de um software de grande complexidade se dá apenas em orquestrar o fluxo dos dados entre os serviços até que se chegue ao resultado final.

Neste projeto os serviços representam os pontos de extensão do CMS escolhido. É através deles que o CMS poderá interagir com sistemas fora da sua alçada comum, como por exemplo, criar um usuário numa ambiente de rede *Unix*.

II.6 COMUNICAÇÃO ENTRE PROCESSOS

O principal pilar do SOA é o método utilizado para a troca de informações entre os serviços. A definição de uma tecnologia para a implementação dessa arquitetura é um ponto crítico, pois todos os dados serão transportados utilizado a rede definida. O ideal é que se adote uma abstração transparente para os programas cliente e que seja utilizada uma abordagem genérica e portável, isto é, que não seja difícil para o desenvolvedor criar um cliente para o serviço e que não seja difícil para o desenvolvedor levar a solução para outros sistemas operacionais.

A abordagem natural para a comunicação entre processos durante muito tempo foi a de se criar um protocolo usando *Protocol Data Units* (PDUs) convencionados, criando formatos *ad-hoc* para transferir dados entre os aplicativos. O problema dessa abordagem é a

falta de padronização. Num meio onde se têm muitos serviços, ou pelo menos se deseja prestar suporte a isso, a falta de padronização pode fazer com que as implementações dos protocolos divirjam, dificultando a criação de novos clientes. Como são módulos independentes, cada desenvolvedor tem total liberdade para criar seu próprio protocolo. Se para cada serviço novo, for necessário que se aprenda um novo "dialeto", o crescimento e a adoção dos mesmos poderiam estar sujeitas ao desinteresse das constantes curvas de aprendizado. Além do protocolo utilizado para a conversação com o serviço, o desenvolvedor poderia utilizar inúmeros transportes, como *sockets*, *pipes* ou *shared memory mappings*, o que aumenta ainda mais o grau de liberdade, que como citado anteriormente, para essa natureza de projeto não é vantajoso.

Com a popularização das aplicações em rede e a propagação dos meios interligados, inúmeras novas abordagens de comunicação entre processos foram criadas. O *Remote Procedure Call* (RPC) está entre elas e é bastante interessante no que diz respeito à maneira segundo a qual se codifica os programas, pois para o desenvolvedor não existe uma rede entre o cliente e o servidor. A troca de dados entre as pontas é feita como uma simples chamada de função. O cliente chama uma função e esta é executada no servidor. O servidor por sua vez, retorna o resultado da função executada para o cliente. Neste projeto, usaremos o RPC como recurso para a comunicação entre cliente e servidor.

Proposta a abordagem de comunicação, precisa-se de uma linguagem intermediária para trafegar os dados, uma espécie de embrulho para os dados, para que este chegue de maneira estruturada no lado remoto. Existem também para esta demanda inúmeras soluções. No projeto decidimos utilizar o XML como o *middleware* para o RPC, pois, das linguagens intermediárias, o XML é a que se encontra num estado de maturidade maior tendo interpretadores de estruturas e bibliotecas de serialização muito estáveis. O nome da solução completa é denominado pela comunidade como XML-RPC. A especificação do XML-RPC é aberta [8] e isso fez com que surgissem implementações nativas em muitas linguagens de programação [9] como C, C++, Java, Ruby e Python. Um esquema exemplificando funcionamento do XML-RPC pode ser visto na figura II.2.

II.7 CRIPTOGRAFIA

Quando se trafegam dados sigilosos - como senhas ou informações privativas, como um boletim acadêmico, numa rede, se faz necessário o uso de medidas para que esses dados não possam facilmente ser capturados e/ou corrompidos.

As soluções de criptografia envolvem algoritmos de complexidade sem igual. Muitos destes, protegidos por patentes e outros, de propriedade intelectual. Está fora do escopo deste projeto, implementar ou mesmo divagar sobre os métodos matemáticos envolvidos no processo de criptografia. Na implementação deste projeto foram utilizadas tecnologias maduras de código livre, com um nível de segurança aceitável.

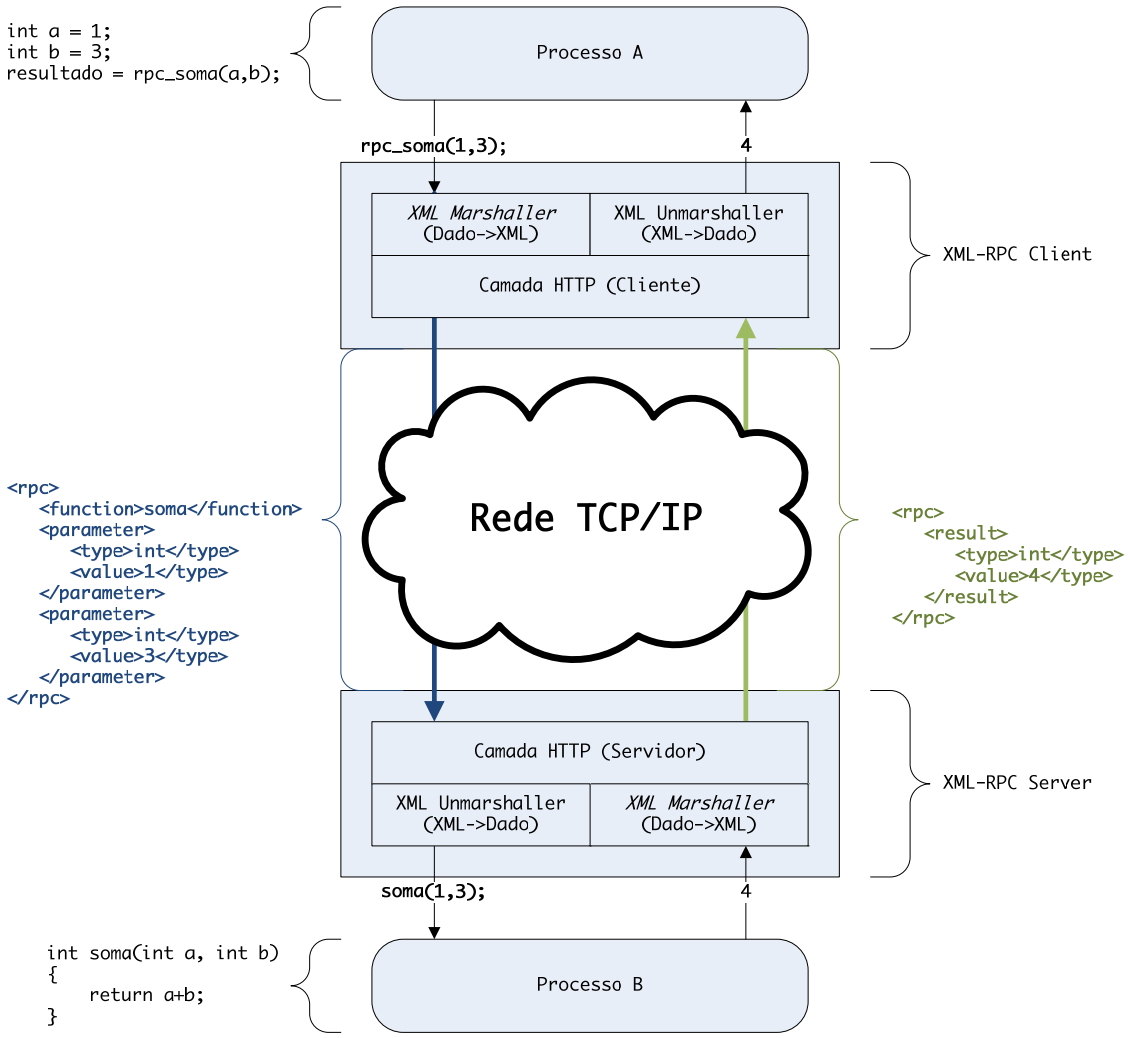


Figura II.2 - Representação de uma chamada de função com XML-RPC

Dentre algumas soluções existentes na comunidade de desenvolvimento ao redor do mundo existe uma abstração considerada uma tecnologia “de fato”. Esta tecnologia, *Secure Sockets Layer* (SSL), permite criar uma camada forte de criptografia sobre os *sockets* comuns. Para a utilização dessa camada, basta o desenvolvedor iniciar os chamados "contextos seguros". A seguir, a codificação é feita de maneira transparente para o desenvolvedor, que pode codificar seus programas como se estivesse utilizando *sockets* puramente, sem custo adicional ao desenvolvimento.

A implementação de SSL utilizada no projeto é a OpenSSL [10], que possui código fonte aberto e é a biblioteca considerada padrão nos ambientes *Unix*.

II.8 AMBIENTE UNIX

O sistema multiusuário *Unix* tem como um forte pilar a transparência de funcionamento. Existe uma abundante quantidade de documentação que vêm desde especificações de arquitetura de código a roteiros passo-a-passo sobre a implantação de um

dado serviço. Este projeto tem a intenção de ser utilizado em qualquer plataforma. No entanto, o foco estará nos serviços específicos encontrados em plataformas *Unix*, como NIS (*Network Information Service*), Samba, NFS (*Network File Service*), etc. Em outras plataformas, como *Windows* e *Mac*, também será possível utilizar o software, mas será necessário criar implementações de serviços específicas para estes ambientes.

Os ambientes da família *Unix* por si só já clamam por uma variedade enorme de plataformas. Desde a família BSD (*OpenBSD*, *FreeBSD*, etc) até as principais distribuições de *Linux* (*Red Hat*, *Ubuntu*, *Gentoo*, *Suse*, *Debian*, etc), cada tipo de sistema possui pecularidades que devem ser levadas em conta, principalmente na implementação dos serviços.

II.9 PADRÕES DE PROJETO EXTENSÍVEIS E COM BAIXO ACOPLAMENTO

A utilização de Padrões de Projeto, que diminuem o acoplamento entre os módulos de um sistema, tem como objetivo reduzir a dependência direta dos módulos em tempo de compilação. Isso permite uma verdadeira separação do projeto em módulos e o tratamento dos mesmos como subsistemas autônomos.

Uma segunda vertente do desenvolvimento de baixo acoplamento é a vocação para a extensibilidade. Como o acoplamento em tempo de compilação é diminuído, no caso ideal para zero, o desenvolvedor ganha a liberdade de poder criar extensões que modifiquem o comportamento "padrão" do sistema sem intervir no código. Para alterar o comportamento padrão, o desenvolvedor de um módulo externo poderá utilizar a arquitetura do tipo "plug-in" e arquivos de configuração.

As técnicas utilizadas no projeto se baseiam nos padrões de projeto: *Abstract Factory* e *Inversion of Control* (IoC). O uso dessas estruturas permite uma versatilidade que muitas vezes pode colaborar para outra boa prática: a reutilização de trechos de código. Se toda ou grande parte da lógica de negócio for baseada em interfaces, o código se torna mais genérico e reutilizável.

II.9.1 DESACOPLAMENTO COM O USO DE INTERFACES

O foco principal do desenvolvimento de software extensível com fraco acoplamento se dá nas estruturas chamadas *interfaces* (classes abstratas). As *interfaces* definem *contratos*. Nesses *contratos* encontramos informações de que funções uma classe - que *implementa* essa interface - deve responder. Como um exemplo prático, suponha que uma interface *IAnimal* defina em seu *contrato* a função *makeSound()*. Uma classe concreta *Duck* que implementa essa interface responde à chamada de função retornando *"quack"*, enquanto a classe concreta *Cat* retorna *"meow"*. O diagrama de classe deste exemplo pode ser visto na Figura II.3.

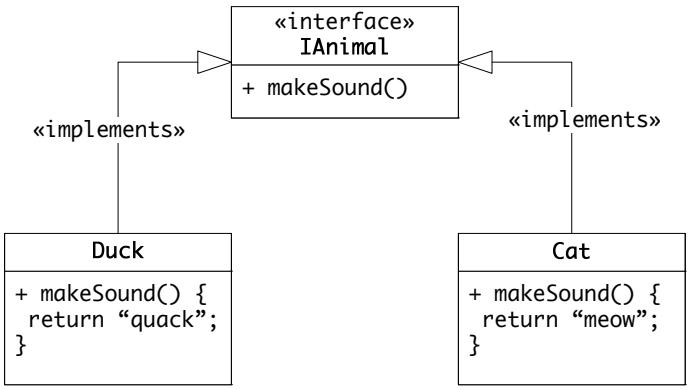


Figura II.3 - Interface IAnimal

Isso permite que o desenvolvedor não precise se comprometer a utilizar uma determinada classe diretamente. Todo o desenvolvimento é baseado nas interfaces. Note que neste ponto já se evita o acoplamento de dependência de classe concreta. Quem acessa a interface *IAnimal* não precisa conhecer necessariamente a classe concreta *Duck* ou *Cat*.

Uma modelagem modular bem feita tem muito mais o foco sobre as interfaces do que sobre as implementações das classes de negócio, pois é possível modelar toda a lógica de negócio conhecendo apenas as interfaces.

Linguagens orientadas a objeto como Java, C++, Object Pascal e linguagens dinâmicas como Python e Groovy possuem em seu núcleo o suporte às interfaces, com algumas variações na forma de declaração.

Java:

IAnimal.java:

```
public interface IAnimal
{
	public String makeSound();
}
```

Cat.java:

```
public class Cat implements IAnimal
{
	public String makeSound()
	{
		return "meow";
	}
}
```

C++:

IAnimal.h:

```
virtual class IAnimal():
{
	public std::string makeSound();
}
```

Cat.h:

```
class Cat(IAnimal):
{
	public std::string makeSound();
}
```

Cat.cpp:

```
public std::string Cat::makeSound()
{
	return "meow";
}
```

II.9.2 PADRÃO DE PROJETO *ABSTRACT FACTORY*

O padrão de projeto *Abstract Factory* tem um papel muito importante no desenvolvimento com fraco acoplamento. Ele permite que o desenvolvedor não se comprometa na escolha direta de qual classe concreta utilizar no momento em que se deseja uma nova instância. Esta responsabilidade é passada para a classe que implemeta a *Abstract Factory*.

Quando se cria uma nova instância de uma classe diretamente pelo seu nome, cria-se neste momento uma dependência de classe. O *Abstract Factory* tem como objetivo remover esse acoplamento, tomando para si a responsabilidade de decisão de qual classe utilizar.

Seguindo na mesma linha de raciocínio de ilustração do item anterior *interface*, o desenvolvedor da classe que implementa a interface *IPetShop* necessita de uma nova instância de animal, então pede para a *IAnimalFactory*, esta por sua vez lhe retorna um *IAnimal* (figura II.4).

Note que *IAnimalFactory* pode ser implementada de maneiras diferentes. No nosso exemplo temos duas implementações, para uma loja de gatos e de patos. A interface *IPetShop* não tem conhecimento algum de que classe concreta lhe será retornada, tampouco precisa, visto que foi criado um isolamento através da interface *IAnimal.* As figuras II.5 e II.6 mostram este isolamento. Nestas figuras alteramos o modelo da loja de animais de estimação representado na figura II.4 para ser uma loja ou de gatos ou de patos.

II.9.3 PADRÃO DE PROJETO INVERSION OF CONTROL (IOC)

O conceito de *Inversion of Control (IoC)* completa o conjunto das técnicas que normalmente são utilizadas para o desenvolvimento de software orientado a interfaces. Sua importante função é servir como um elo entre as configurações, que definem as classes concretas e as interfaces utilizadas pelo programa e que são determinadas em tempo de compilação.

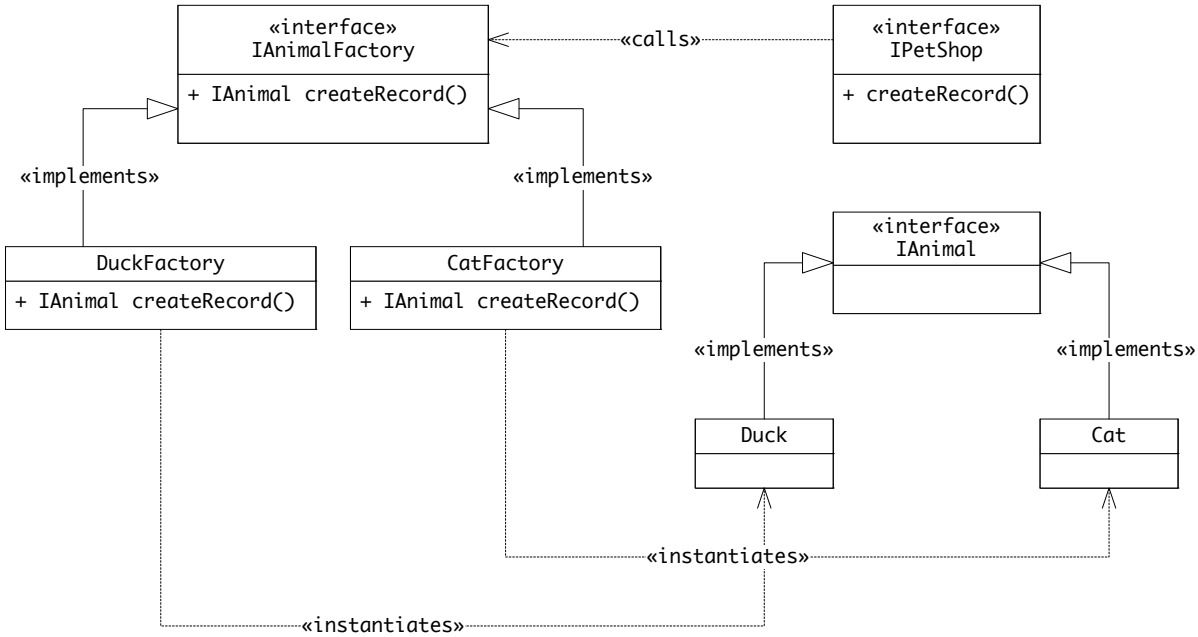


Figura II.4 Modelo de classes para loja de animais

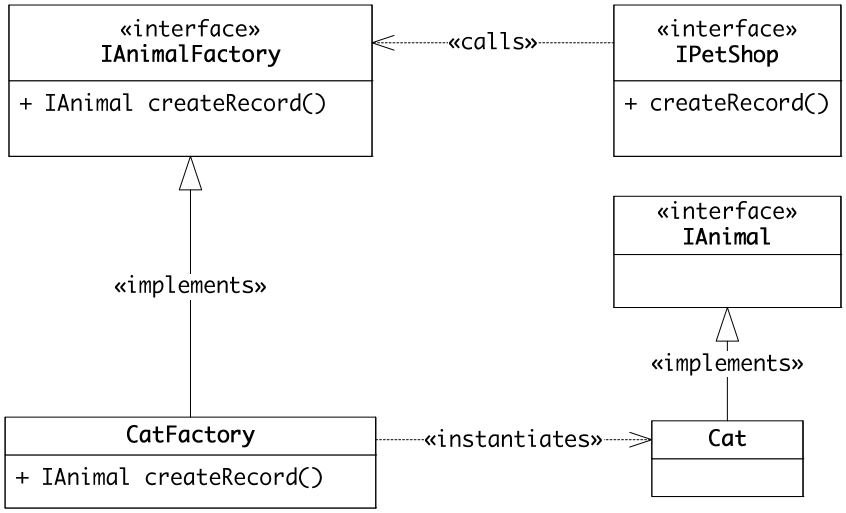


Figura II.5 - Modelo de classes para loja de gatos

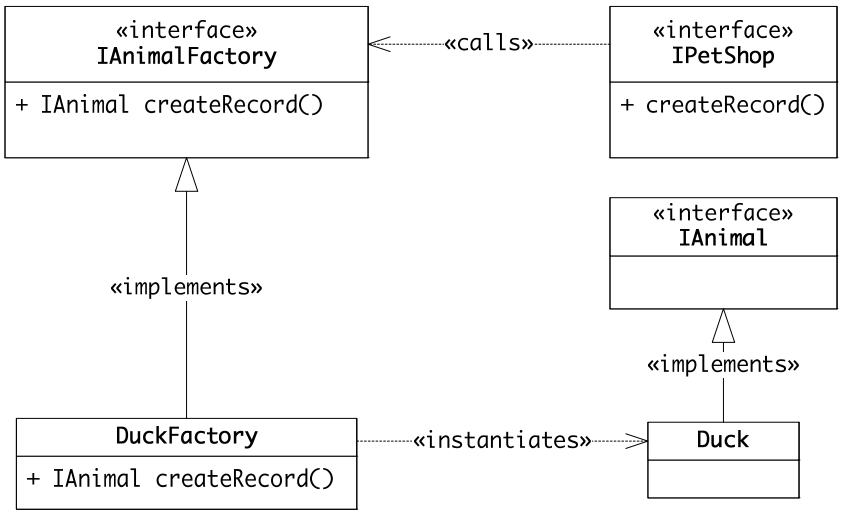


Figura II.6 - Modelo de classes para loja de patos

Uma das formas de implementação do padrão IoC é a utilização de uma classe especializada para prover o atrelamento entre a classe abstrata e a implementação utilizada a partir da fábrica de instâncias (instância de *Abstract Factory*). Essa classe especializada é chamada de *ServiceLocator* . A classe *ServiceLocator* utiliza algum meio externo para se configurar. Este meio externo deve ser definido em tempo de execução e deve ser configurável. Normalmente é utilizado um arquivo de configuração, o que não compromete o desacoplamento desejado pelo sistema.

Na figura II.7 temos um exemplo aplicado seguindo a mesma linha dos desenvolvidos anteriormente. O *ServiceLocator* tem a responsabilidade de indicar para a interface *IPetShop* que a classe concreta que implementa a interface *IAbstractFactory* que deverá ser utilizada é a *CatFactory*. Essa informação foi retirada do arquivo de configuração, que a classe *ServiceLocator* carrega no momento em que é construída.

Note que em tempo de compilação o código da *IPetShop* independe de *CatFactory*, pois essa informação foi retirada do arquivo de configuração. Isso demonstra o desacoplamento das classes.

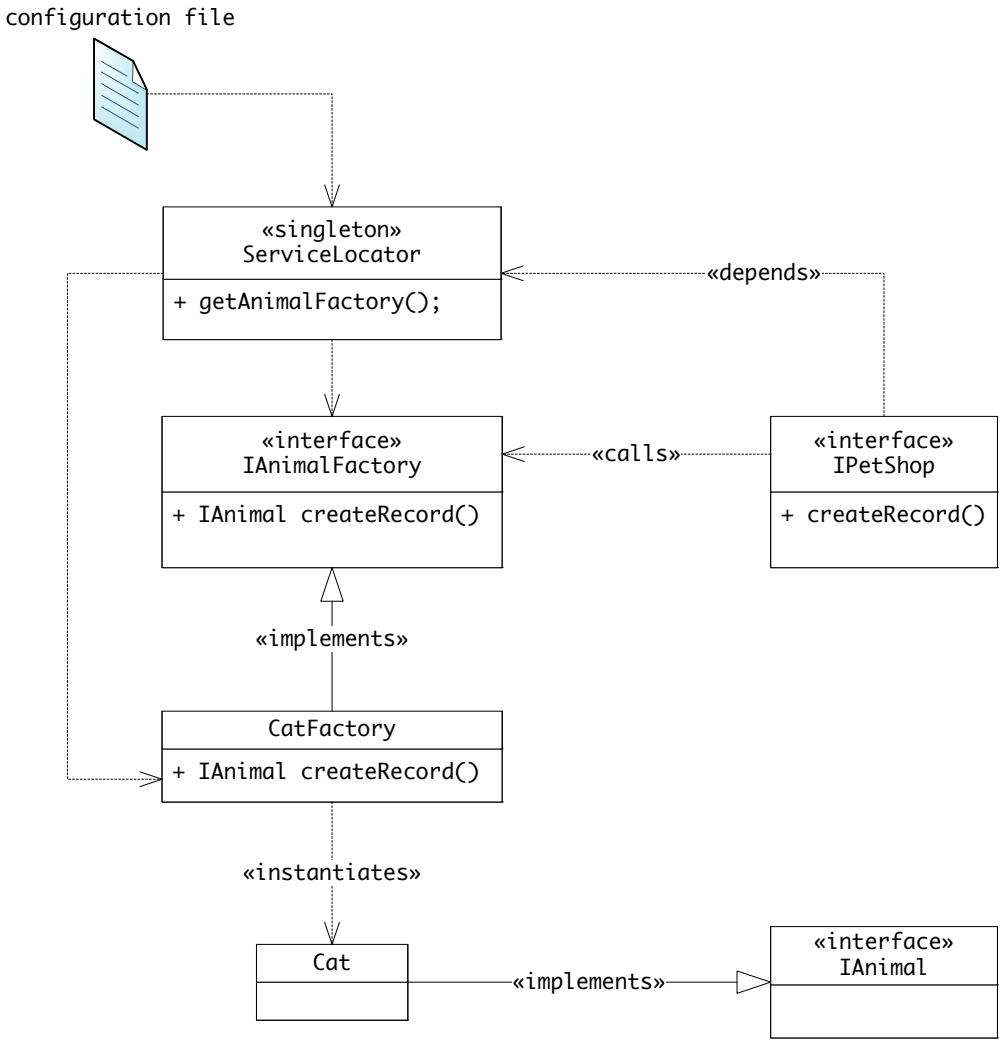


Figura II.7 – Service Locator utilizando arquivo de configuração

Capítulo III - Definição do CMS Utilizado

A base para o desenvolvimento do sistema alvo deste projeto é um CMS. Além de prover uma base conceitual de abordagem para estruturas organizacionais genéricas, com fluxos de trabalhos diversos, um CMS também pode prover um modelo com uma implementação que atenda aos requisitos do sistema integrado. Não menos importante, é a utilização do CMS como um espaço para os usuários publicarem seus artigos e sua página pessoal.

Uma dos requisitos não negociáveis para o CMS é o controle de versão de conteúdo, o que dá capacidade do usuário de acompanhar as alterações de um determinado dado e se desejado desfazê-las. Para um sistema colaborativo esse tipo de controle é indispensável.

Foi realizado um levantamento e uma comparação das características dos principais CMS utilizados na Internet. Desta forma, foi possível optar pela solução que mais se aproximou do ideal buscado.

Neste capítulo primeiramente é feita uma rápida análise qualitativa dos CMS estudados. Inicialmente foi realizada uma análise genérica de um grupo de CMS, seguida por uma análise mais específica para o sistema implementado neste trabalho, incluindo características que julgamos ser imprescindíveis para o mesmo.

Uma das fontes de pesquisa utilizada foi o site CMS Matrix[11] que pode comparar uma infinidade de sistemas através de suas características principais. Por concentrar uma grande quantidade de dados é importante ter em mente que tipo de CMS se deseja buscar.

III.1 PLONE

O *Plone* é um CMS baseado no servidor de aplicações *Zope*, que por si já carrega um enorme conjunto de ferramentas para a construção de aplicações web. O *Zope* difere dos demais servidores de aplicação web por possuir um banco de dados orientado a objetos embutido, o que torna o processo de persistência de dados muito simples. Obviamente, isto muda de forma radical o paradigma de desenvolvimento de software, indicando, para os desenvolvedores, uma curva de aprendizado um pouco mais inclinada no início, mas que com o tempo tende a se estabilizar.

Este CMS é bastante utilizado na comunidade, inclusive em sites com bastante volume de acessos e dados como os portais implantados pelo Serviço Federal de Processamento de Dados (SERPRO). A maioria dos sites do governo federal utiliza o *Plone*. Um exemplo de página que mostra a robustez e a versatilidade desse sistema é o Portal do Governo

Brasileiro[1]. Tanto o *Zope*, quanto o *Plone* são sistemas maduros e que já atingiram um nível bastante refinado.

Modularização é uma palavra chave no *Plone*. Este CMS é composto por vários componentes menores, denominados “produtos”. Cada produto é responsável por uma tarefa dentro do sistema como um todo. O *Plone* apenas agrupa estes produtos e atua como mediador para que funcionem em harmonia uns com os outros. Sua linguagem de implementação, o *Python* é uma linguagem dinâmica orientada a objeto, multi-plataforma por *design*, de código livre e bastante consolidada e difundida. Existem muitos adeptos ao *Python*, o que contribuiu para a criação de uma série de produtos para o *Plone*, desde sistemas de comércio eletrônico e portais corporativos até sites de distribuição de arquivos.

III.2 JOOMLA

O *Joomla* é um produto também em estado de desenvolvimento maduro, fruto da dissolução de uma equipe de desenvolvimento de um projeto anterior chamado *Mambo*. Escrito em PHP, também possui uma arquitetura extensível. O fato da linguagem PHP não ter uma curva de aprendizado acentuada fez com que o *Joomla* fosse bastante difundido. Este sistema também possui adeptos de peso como o Portal do Ministério da Educação[2].

III.3 MAGNÓLIA

Segue uma tendência recente de projetos com dupla licença, possui uma versão *community* que é open source e presta suporte comercial vendendo a versão *enterprise*. O *Magnólia* é um *framework* que segue à risca uma série de boas práticas de desenvolvimento de software.

Sua linguagem de implementação, o Java, tem um vasto contingente de desenvolvedores, e é comum nesta comunidade discussões sobre uma arquitetura bem desenhada ser transformada em uma especificação pela comunidade. Os padrões Java têm o prefixo "JSR" (*Java Specification Requests*) seguido de uma numeração. Existem cerca de novecentas especificações de arquitetura para o Java incluindo as próprias melhorias para a linguagem. O *Magnólia* utiliza a especificação JSR170 que é o *Java Content Repository*, a maneira que a comunidade Java e seus “Gurus” formularam como a melhor forma de se armazenar conteúdos que devam ser gerenciados.

O *Magnólia* possui uma variedade de características que o levam bem próximo do *Plone* quanto à flexibilidade e à capacidade de extensão, porém dos CMS estudados é o que tem a menor base de entusiastas e utilizadores. Um fator que se constatou durante a busca é

[1] Portal do Governo Brasileiro. http://www.brasil.gov.br.

[2] Portal do Ministério da Educação - http://portal.mec.gov.br.

que existem outras soluções Java para a gestão de conteúdo que atendem melhor ao mercado como o Alfresco [12].

III.4 OUTROS CMS ESTUDADOS

Na busca por CMS adequados ao projeto incluímos também na pesquisa alguns candidatos de menor expressão no mundo web, Mencionaremos suas características e a razão pela qual estes sistemas não são aceitáveis para a implementação do projeto.

III.4.1 DRUPAL

Segue a mesma linha do Joomla e também é escrito em PHP, o que aumenta sua base de desenvolvedores, porém com um apelo menor de "site fechado". Este CMS preza pela construção de sites colaborativos mais abertos para a comunidade.

III.4.1 PHP NUKE

Dos analisados parece ser o mais amador, apesar de não ser o mais novo. O núcleo do PHP Nuke parece ter graves problemas de projeto, por não ter adotado boas práticas no processo de desenvolvimento. Muitos colaboradores reclamam da sua estrutura fraca, sem a adoção de padrões de projeto. Em comparação aos outros CMS é o que menos tem extensões já inclusas como *blog*, *wiki* e comércio eletrônico.

III.5 COMPARATIVO DE CARACTERÍSTICAS BÁSICAS

Compilando algumas características chave dos CMS estudados, é possível concluir que o *Plone* oferece a melhor plataforma de desenvolvimento para o projeto. Isto ficará mais claro a seguir.

Na Tabela III.1 agregamos características do processo de desenvolvimento do CMS levando em conta a linguagem de implementação, a maturidade do processo e o sistema de persistência de dados.

A comparação de linguagem de programação utilizada é importante em todas as fases do software, desde o seu desenvolvimento até a sua instalação. As linguagens utilizadas nos CMS’s escolhidos são todas multi-plataforma, sendo que a linguagen Java e Python possuem uma melhor reputação entre os desenvolvedores de software com complexidade alta.

Nessas linguagens, existe uma série de projetos e bibliotecas robustas e prontas para serem utilizadas em ambientes de produção. O PHP, apesar de possuir uma comunidade ativa imensa, carece de bons *frameworks*. A orientação a objetos em PHP é ainda muito recente, enquanto que as outras duas linguagens nasceram orientadas a objetos. Esses fatores fazem com que a adoção de um sistema escrito em PHP seja desfavorecida em relação aos sistemas escritos em *Python* e Java.

Tabela III.1 - Comparativo de linguagem de programação, *framework* de testes e unidade de armazenamento

CMS	Linguagem	*Framework* de Testes	Armazenamento
Plone	Python	possui	ZoDB, extensível
Joomla	PHP	não possui	MySQL (SGBD)
Drupal	PHP	possui	MySQL, Postres (SGBD)
PHP Nuke	PHP	não possui	extensível (SGBD)
Magnólia	Java	possui	JCR, extensível

Neste projeto a prioridade não é o desempenho, uma vez que aplicações web, por natureza, possuem seu desempenho limitado às capacidades da rede e ao acesso aos dados, e por isso, muitas vezes não é notável a diferença no desempenho entre linguagens de programação.

Em um projeto de software livre é comum que muitos dos colaboradores não se conheçam pessoalmente, nem tampouco trabalhem no mesmo ambiente. As porções de código são adicionadas à base principal, as correções são submetidas livremente, mas o que assegura a qualidade do software? Um processo maduro, convenções de código e uma suíte de testes que dê cobertura as regras de negócio do sistema podem nos dar essa garantia.

A falta de erros em tempo de compilação não assegura que o software esteja livre de erros em tempo de execução, os chamados *bugs*. O mais importante a ser verificado é a lógica de negócio que se deseja, garantindo-se a seguir que erros antigos não voltem a assombrar o código. Para isso é importante uma *framework* de testes de unidade, testes funcionais e testes de regressão.

Por estes motivos a característica "*Framework de Testes*" se faz presente nesta listagem comparativa. Ao acolher um CMS que não atenda a esse requisito o projeto está mais exposto a erros em sua lógica de negócio.

Na característica "*Armazenamento*", foram pesados os aspectos citados no item Persistência de Dados (seção II.4). Na análise são utilizadas as opções: ZoDB, JCR e SGBD. O uso de um banco de dados relacional apesar de interessante para entidades de negócio se mostra confuso muitas vezes e não trás intrinsecamente a disponibilidade em outras mídias que não permitam o uso de uma conexão TCP/IP e o acesso por comandos SQL (*Structured Query Language*). Neste ponto o ZoDB e o JCR ganham destaque, pois foram concebidos exatamente com o propósito genérico de armazenamento de conteúdos e a capacidade de

servi-los em inúmeros meios como CIFS, FTP, HTTP (*Hyper Text Transfer Protocol*) entre outros.

Um fato que pesa a favor do ZoDB é que ele é uma excelente implementação de banco de dados orientado a objetos (uma das poucas a ser utilizada realmente em larga escala em ambientes de produção). Um dos obstáculos dos desenvolvedores que é ter que mapear tabelas em objetos, não é necessário num banco de dados orientado a objetos, pois neste a lógica de transformação dos dados para serem armazenados fica a cargo do *framework* de armazenamento, e não do desenvolvedor. Sendo assim é muito simples a codificação de entidades persistentes usando o ZoDB, o que reduz de maneira drástica a complexidade do projeto.Na Tabela III.2 faz-se um comparativo dos CMS baseado nas 3 características descritas abaixo.

Tabela III.2 - Comparativo de capacidade de internacionalização, suporte a metadados e *workflows*

CMS	Internacionalização	Metadados	Fluxo de trabalho
Plone	possui	possui	Possui
Joomla	possui	não possui	não possui
Drupal	possui	não possui	limitado
PHP Nuke	não possui	não possui	não possui
Magnólia	possui	possui	possui

Visto que a Internet é um meio onde não existem fronteiras, é importante não criar restrições aos visitantes de um portal. Quando se tem como público alvo um visitante que não é necessariamente um compatriota, devemos fazer o possível para que o visitante entenda o conteúdo do seu portal. A maneira mais fácil para isso é fornecer ao visitante um portal em seu idioma local. As abordagens mais elegantes já preparam as aplicações para suportar conteúdos multi-idiomas. O nome dessa técnica é Internacionalização (I18N) onde normalmente são criados arquivos externos de dicionário. Estes arquivos são traduzidos para os mais variados idiomas por tradutores especializados. Os tradutores não precisam entender de programação ou de sistemas web, só precisam entender o formato do arquivo e traduzir o conteúdo. O conteúdo traduzido é então disponibilizado para os visitantes. Se o CMS não tiver suporte a essa característica uma opção será alterar o código para inserir tal funcionalidade, o que representaria uma soma inviável de trabalho. Outra solução seria criar uma instância de portal para cada idioma desejado, o que seria um enorme desperdício de recurso, além das preocupações com condições de concorrência e de sincronização que isso poderia trazer. Isso elimina o PHP Nuke.

O suporte a *metadados* é importante quando se pensa em construir um sistema baseado em CMS. A capacidade de criar novos tipos de entidade através de uma meta-linguagem sem

que seja necessário mexer no código do CMS se faz necessária, pois caso haja atualizações do CMS estas poderão ser feitas sem que sejam novamente necessárias as intervenções.

Um *engine* de *workflow* para os *metadados* dão a liberdade necessária quando se trata de um sistema colaborativo. É possível modelar infindáveis casos de uso utilizando fluxos de trabalho colaborativos. O fato de um CMS ter em seu núcleo esse *engine* é uma grande vantagem, pois economiza recursos de homem/hora para o projeto.

Na Tabela III.3 temos algumas características que medem a flexibilidade do CMS.

Tabela III.3 - Comparativo de extensibilidade do sistema de autenticação e sistema de arquivos

CMS	Autenticação	*Filesystem* Virtual	"Grau de extensibilidade"
Plone	extensível	possui	*****
Joomla	não extensível	não possui	***
Drupal	não extensível	não possui	***
PHP Nuke	não extensível	não possui	*
Magnólia	extensível	possui	***

Quanto ao requisito Autenticação, o fato de um CMS apresentar opção de extensão faz com que não sejam necessárias alterações no seu código para acoplar uma extensão particular. Uma alteração desse tipo é muito impactante no tempo de entrega do projeto, portanto, com esta característica o *Plone* e o *Magnólia* se tornam excelentes opções para a implantação do projeto.

Constituir um *Filesystem* virtual no núcleo do CMS ou tê-lo como módulo externo traz uma versatilidade maior quando se trata do *download* ou do *upload* dos dados em massa. Quando uma pessoa quer popular um documento, para a seguir enviá-lo ao CMS uma sequência com 50 figuras, a interface mais natural para isso seria um cliente de FTP. Como o projeto tem a intenção de ser um grande diretório de usuários, ser capaz de fornecer esse tipo de facilidade é de grande valia.

A característica “Grau de Extensibilidade” é uma característica aferida pelos desenvolvedores do projeto, observando os meios de comunicação de cada comunidade e fazendo busca por personalizações comuns como *Blogs*, *Wikis*, módulos de *e-commerce*, integração com *Google Maps* e módulos de fórum. Nesse aspecto o *Plone* certamente se destaca, A comunidade bastante ativa faz muitos lançamentos de versão e são bastante inovadores em suas customizações. A arquitetura extensível do *Plone* se mostra bastante amigável para tal.

III.6 ZOPE, PLONE – SUBSISTEMAS E ABORDAGENS

Dentre os CMS's avaliados o *Plone* foi o que melhor atendeu às demandas do projeto em questão. Vamos enumerar as características chave desse sistema:

Distribuição e Portabilidade – A equipe do *Plone* mantém em seu site, pacotes instaladores para plataformas como *Windows*, *Linux* e *Mac*. Muitas extensões consideradas essenciais também vêm inclusas na instalação básica. A manutenção de um pacote básico bastante completo faz com que os novatos na plataforma se sintam bastante realizados com a riqueza do sistema. Uma das características marcantes é a preocupação com a qualidade nos lançamentos de versão, que se preocupam em criar instâncias limpas do software facilitando a sua manutenção. A equipe se propõe a trazer informativos de migração dos dados junto a cada lançamento de versão principal, de forma que sempre se possa atualizar seu portal sem perder as informações anteriores.

Internacionalização de Interface – A interface do *Plone* tem traduções em mais de 35 idiomas incluindo Português, Espanhol, Alemão, Francês, Japonês entre outros. Criar produtos que dêem suporte a internacionalização é uma excelente prática que é recomendada e seguida pela comunidade.

Usabilidade – A interface base do *Plone* foi cuidadosamente desenhada para disponibilizar o conteúdo com um alto grau de usabilidade e acessibilidade. Isso não se limita à exibição de um portal bonito e limpo. Aspectos de web-semântica e amigabilidade com sistemas alternativos de leitura de tela também foram contemplados no projeto do *Plone*, resultando em alta capacidade de indexação em sites de busca como o *Google*, possibilidade de acesso ao site para pessoas com deficiência visual e possibilidade tradução para idiomas cuja ordem de leitura é feita da direita para esquerda sem prejuízos ao *layout* do portal.

Interface Gráfica Altamente Flexível – o *Plone* possui modelos de exibição utilizados para a apresentação de conteúdos que podem ser personalizados. O conjunto de modelos que dão ao portal uma aparência distinta é chamado de *Skins*. Este conjunto de modelos é escrito na linguagem de modelo do Zope (*Zope Page Templates* – ZPT) junto a uma quantidade considerável de folhas de estilo em cascata (*Cascading Style Sheets* - CSS). Com estas duas frentes é possível alterar por completo a aparência do portal, sem que o conteúdo armazenado seja sequer tocado.

Sistema Extensível de Usuário – O *Plone* tem incluso um sistema completo de cadastro de usuários e através de extensões é possível fazer com que a autenticação seja feita em um domínio remoto, em um servidor de contas pré-existente como um servidor *Unix*.

Fluxos de Trabalho e Segurança – Os fluxos de trabalho controlam a lógica de processamento de conteúdo dentro do portal. Essa lógica pode ser alterada pela interface web do portal utilizando ferramentas apropriadas. Os administradores do site têm liberdade de criar fluxos complexos ou simples, como desejarem. Para cada conteúdo num portal, é

possível configurar listas de controle de acesso e decidir quem pode acessá-lo e com qual grau de interatividade (se um usuário pode alterar um determinado campo, ou se só pode visualizá-lo, por exemplo). Tudo é configurável pela interface web.

Documentação – O projeto *Plone* mantém uma extensa base de documentação, que inclui livros e artigos de referência [13].

Desenvolvimento contínuo – Uma das características mais marcantes que se pode notar no *Plone* é sua ávida e participativa comunidade de desenvolvedores e empresas que apóiam e prestam suporte ao projeto. Grande parte das extensões foi patrocinada por alguma empresa que quis uma funcionalidade em particular para o seu portal. Além disso, a licença de código livre contribui para que o sistema seja continuamente depurado e aprimorado. Existe uma grande quantidade de desenvolvedores envolvidos de alguma maneira com o projeto, seja codificando ou documentando. Basta listar todas as extensões e artigos contribuídos que o comprometimento da comunidade salta aos olhos.

Extensibilidade – O *Plone* foi concebido para servir de base para sistemas ricos em funcionalidades. Para isto adotou-se desde seu período embrionário o conceito de ser extensível por natureza. A estrutura de produtos *Plone* faz com que seja possível um produto alterar o comportamento de outro produto criando infinitas possibilidades para adequar o sistema aos requisitos. Ferramentas como o *Archetypes* também fazem com que o *Plone* seja uma poderosa ferramenta de prototipação com um rápido tempo de entrega e de resposta a demandas por modificação.

Suporte a Metadados – O *Plone* traz em sua instalação padrão um conjunto de tipos de conteúdo adequados a um CMS. Apesar de ser o suficiente para grande parte dos adeptos da plataforma, os desenvolvedores de aplicação tem total liberdade de adicionar novos tipos de dados ou novos campos a um tipo já existente.

Capítulo IV - Projeto de Software

Este capítulo abrange tópicos do projeto de software, as normas ANSI/IEEE 830 e ANSI/IEEE 1016 foram utilizadas como guias, porém não foram cobertos todos os tópicos indicados pelas normas por questões de escopo do relatório.

IV.1 OBJETIVOS

O *Soplum* é uma extensão para um CMS de forma que seja possível fazer a gestão de usuários de uma rede de computadores usando uma arquitetura orientada a serviços. Estes serviços podem ser codificados separadamente e adicionados posteriormente ao *Soplum* através de configuração. Neste projeto foram implementados inicialmente alguns serviços adequados para a estrutura organizacional da rede DEL.

O sistema comporta uma interface de administração da rede de computadores do DEL (UNIX/Microsoft Windows) e uma interface amigável para a manutenção da página do departamento. Estas interfaces foram implementadas através do CMS selecionado (Zope/Plone). Como objetivo secundário, o uso do CMS deve aumentar a colaboração e a difusão do conhecimento entre todas as pessoas envolvidas com as atividades do departamento.

Abaixo da camada de apresentação amigável, existe um programa cliente que faz chamadas remotas a um servidor que irá prover os serviços necessários para a gerência de usuários da rede DEL. A codificação dos serviços também segue a mesma filosofia da codificação do cliente, portanto qualquer premissa, postulado ou restrição se aplica igualmente para todo o código produzido.

É importante ressaltar que o projeto não tinha como objetivo criar um sistema com todas as informações que possam existir em um departamento de uma universidade. O objetivo do projeto era implantar um sistema que permitisse que outros subsistemas pudessem ser integrados a ele de forma coerente. Um desses sistemas é exatamente o sistema de gerência de usuários da rede DEL. Futuramente, outros sistemas poderão ser criados para gerenciar inscrições, processos, gerar relatórios sobre alunos, professores, disciplinas cursadas, enfim, qualquer sistema que possa agregar valor ao DEL.

IV.1.1 POSTULADOS E RESTRIÇÕES

O projeto utilizou somente software livre e não houve nenhum custo envolvido para a execução do projeto. O único investimento foi o tempo e o conhecimento dos membros da equipe.

A meta de arquitetura para o sistema era adotar um sistema que tivesse como premissas algumas filosofias de projeto de software Ágeis como *Don’t Repeat Yourself*

(DRY[1]) e *You Aren´t Gonna Need It* (YAGNI[2]), com o menor acoplamento possível de novos módulos para o sistema, e utilizando padrões de projetos adequados para tal.

Este sistema é baseado em um CMS ao qual podem ser acoplados módulos para aumentar a abrangência de serviços prestados. Foi realizada uma pesquisa e análise para a definição do CMS mais adequado para o projeto (capítulo III), assim como a análise do impacto de adoção do CMS pelo departamento.

Um serviço deve ter a capacidade de atender a chamadas assíncronas e simultâneas. Portanto o sistema deve se preocupar com condições de corrida e de concorrência.

O sistema é colaborativo e pode ser administrado por mais de um usuário. Portanto, a gravação de ações críticas para efeito de auditoria é uma premissa fundamental.

IV.1.2 REFERÊNCIAS

Como ambiente de desenvolvimento foi utilizado o *Eclipse* 3.3 [14] Europa desenvolvido pela *Eclipse Foundation*.

A linguagem de programação utilizada para a implementação do sistema foi o *Python.* Para o auxílio ao processo de desenvolvimento na plataforma *Eclipse* foi utilizada a extensão *PyDev*[3] que fornece um conjunto de ferramentas de organização e manutenção de código, junto a uma interface mais amigável de depuração.

O controle de versão de desenvolvimento foi realizado pelo software *Subversion* (SVN) [15] desenvolvido pela *Tigris*. Foi utilizada a extensão *Subclipse* que integra o

---

[1] Portland Pattern Repository's Wiki. *Don't Repeat Yourself*. http://c2.com/cgi/wiki?DontRepeatYourself

[2] Portland Pattern Repository's Wiki. *You Aren´t Gonna Need It*. http://c2.com/cgi/wiki?YouArentGonnaNeedIt

controle de versão à plataforma de desenvolvimento do *Eclipse*, o que permite um controle de revisões e alterações no código. Os artefatos gerados durante o processo de desenvolvimento e o controle de versão foram realizados pelo servidor *Sourceforge*[4], um site que oferece hospedagem gratuita a projetos de código aberto.

Não foi utilizada nenhuma metodologia formal para o processo de desenvolvimento. Foi feito apenas um controle de funcionalidades completadas através de uma planilha, disponível no repositório SVN e que pode ser consultada a qualquer momento pelo orientador.

IV.1.3 PERSPECTIVA DO PRODUTO

A interface administrativa do *Soplum* funciona como um módulo do CMS *Plone*. Desta forma, a existência dessa interface depende da devida instalação do *Plone*. O *Plone* por sua vez é um sistema baseado no servidor de aplicações *Zope*.

Além disso, a interface administrativa do *Soplum* depende da instalação dos provedores de serviços do *Soplum*.

Esta interface de serviços depende também da biblioteca *Twisted*[5] para prover acesso assíncrono e facilidades de execução em segundo plano.

O *Zope* e os serviços do *Soplum* foram implementados em *Python* que é uma linguagem dinâmica e que utiliza uma máquina virtual portável.

Na Figura IV.1 é possível observar as estruturas das duas interfaces do projeto.

---

[4] http://www.sourceforge.net – Repositório de projetos para *software opensource*.

[5] http://twistedmatrix.com/trac/ – Implementação com alta confiabilidade de bibliotecas para softwares com arquitetura cliente/servidor em diversos protocolos.

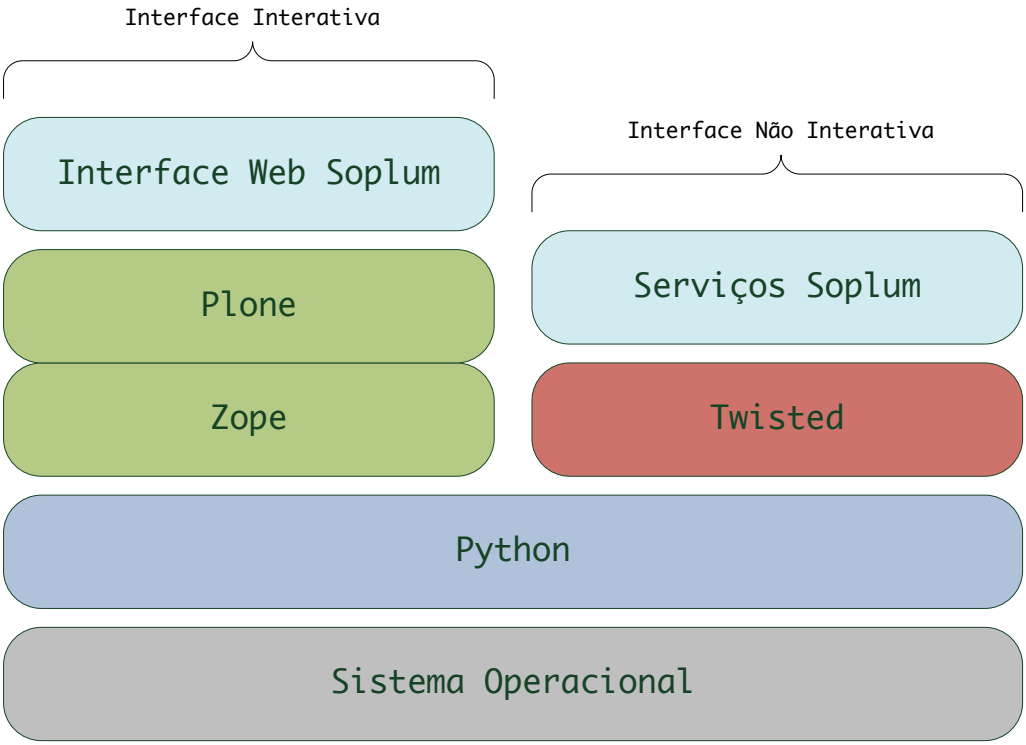


Figura IV.1 - Perspectiva em camadas do produto

IV.1.4 FUNÇÕES DO PRODUTO

O produto deste projeto deve permitir ao administrador do sistema, via interface web, gerenciar os usuários do próprio sistema Web, contas UNIX (principalmente NIS), contas Samba, listas de distribuição de mensagens eletrônicas através do gerenciador Mailman e cotas de espaço em disco.

IV.1.5 CARACTERÍSTICAS DO USUÁRIO

Os usuários da interface administrativa devem ter pleno conhecimento da estrutura organizacional do Departamento de Eletrônica e de Computação. Esta interface é usada casualmente no cotidiano e será usada periodicamente no período de entrada de alunos no departamento, quando o seu uso será intensificado, tendo um grande volume de cadastros para fazer. O sistema não apresenta restrição de idade para os seus usuários. Além disso, esta implementação só presta suporte aos idiomas português e inglês (embora o *Plone* seja traduzido para diversos idiomas, o SoplumCore só oferecerá suporte, inicialmente, aos idiomas português e inglês), podendo posteriormente ser traduzido para outros idiomas. O

único momento em que o usuário precisa ter conhecimento avançado é no momento de instalação e configuração inicial do sistema.

Os usuários da interface de serviço devem ter conhecimento da estrutura interna do *Soplum* e do funcionamento de um sistema Unix, para que possam adequadamente configurar o sistema. A instalação do sistema deve, a princípio, ser realizada uma única vez, visto que os serviços são auto-suficientes. Se desejado, o usuário poderá acessar os registros de atividade por razões de auditoria. A interface de configuração (arquivos) utiliza um conjunto de palavras em inglês técnico.

Os usuários do CMS têm acesso a uma interface multi-idioma já incluindo traduções para o inglês e o português. O *Plone* possui um grande nível de acessibilidade, sendo amigável para interfaces leitoras de tela e para que usuários portadores de necessidades especiais de visão possam "ouvir" o site. Este CMS é compatível com as práticas adotadas pelo governo dos EUA com relação à acessibilidade.

IV.2 REQUISITOS ESPECÍFICOS

IV.2.1 ANÁLISE ORIENTADA A OBJETO UTILIZANDO A ABORDAGEM ÁGIL

A metodologia de desenvolvimento Ágil preza pela leveza e pela geração de projetos “enxutos” que não tragam para os seus mantenedores um excesso de retrabalho. Por esse motivo, não foram gerados de antemão artefatos de documentação. O planejamento foi feito em pequenas etapas para que não houvesse precipitação em documentar algo e em seguida tornar o artefato produzido um artefato obsoleto.

IV.2.2 ATRIBUTOS

A interface administrativa deve funcionar de maneira que suas informações estejam sempre síncronas em relação aos domínios dos serviços. Portanto os atributos ACID (*Atomicity, Consistency, Isolation, Durability*) devem ser respeitados tanto na comunicação entre a interface administrativa e os serviços, quanto entre os serviços e o sistema mediado por ele, como por exemplo o serviço que faz a alteração de um arquivo de senhas.

A seguir, estes atributos são descritos.

- Atomicidade
  Uma transação iniciada não pode ser interrompida, ela é a menor unidade

executável, por isso chamada atômica. Para o sistema não existem estados intermediários a transação, ou toda a transação é executada ou nada.

- Consistência
  Uma transação é iniciada com o sistema num estado consistente, podendo se tornar temporariamente inconsistente durante a sua execução. Porém, o estado final deve ser sempre um estado consistente, o sistema deve estar sempre íntegro entre as transações.

- Isolamento
  Os dados manipulados em uma transação não devem afetar os dados manipulados por outra transação. Portanto, transações que almejarem alterar o mesmo conjunto de dados só o farão com exclusão mútua. Mais uma vez, nenhuma transação concorrente é capaz de acessar os dados de um estado intermediário de uma transação.

- Durabilidade
  Terminada e consolidada uma transação, a informação será persistida e estará segura mesmo nos casos em que o sistema falhar e se tornar indisponível momentaneamente.

Todos os módulos do sistema foram codificados pensando em extensibilidade com baixo acoplamento, o que facilita a manutenção da base de código futuramente. Dessa maneira outros projetos poderão contribuir com módulos sem que tenham que realizar alterações no núcleo do sistema. O baixo acoplamento é buscado para obter independência entre projetos. Assim as bases de código podem avançar em ritmos diferentes sem interferência.

IV.2.3 Plano para a Resolução de Problemas

Durante todo o processo de desenvolvimento, o projeto em questão esteve à disposição do cliente para que pudesse ser homologado de maneira devida e para que pudesse lançar mão de qualquer meio disponível para comunicar algum erro ou inconformidade com os requisitos especificados. Desta maneira estes comunicados foram avaliados através da planilha de atividades da equipe de desenvolvimento, definindo a estratégia mais adequada para a resolução do problema levantado.

IV.3 Definição de uma Arquitetura extensível

O *Soplum* é uma composição de vários subprojetos, cada um possuindo uma responsabilidade específica no sistema.

Na Figura IV.2 temos a definição dos subprojetos *Python* que formarão o Soplum:

SoplumCommons - API comum a todos os projetos, contém classes auxiliares e funções comuns a mais de um projeto.

SoplumConnectivity - Define uma API para a comunicação do cliente com o servidor.

SoplumCore - É a parte principal do projeto, responsável pela gerência dos serviços no lado do cliente e pela definição dos tipos de conteúdo que irão representar as entidades de negócio. Terá o formato de um produto do *Plone*, podendo ser instalado e configurado através dos mecanismos existentes no próprio *Plone*, seguindo, portanto, as mais novas práticas de desenvolvimento de produtos para o *Plone*. É responsável por carregar os tipos de serviços no lado do cliente. Este módulo depende do pacote *SoplumConnectivity* para que possa se conectar aos serviços remotos.

SoplumServer - É o servidor responsável por gerenciar as requisições do cliente e delegá-las aos serviços hospedados. Contém as definições das interfaces que devem ser utilizadas pelas implementações dos serviços. É responsável ainda por carregar instâncias das implementações de serviços do lado do servidor. Guarda as implementações do protocolo padrão de comunicação XML-RPC sobre HTTPS (*Hypertext Transfer Protocol Secure*) com autenticação, utilizado pelo *Soplum*, junto à implementação do sistema de auditoria em nível de aplicação (*logging*).

<*ServiceImplementation*> - Esse “pacote” serve como exemplo de como seria uma implementação das pontas cliente/servidor de um serviço. O *Soplum* procura delegar a responsabilidade de definir e seguir o contrato para as implementações dos serviços. Tais implementações precisam seguir apenas uma convenção pré-determinada e que será vista mais a diante para que estes sejam devidamente registrados no cliente (*SoplumCore*) e no servidor (*SoplumServer*). Uma vez registrados, os gerenciadores dos serviços em ambos os lados irão invocar os serviços conforme necessário, se baseando nas interfaces já definidas.

Essa disposição de pacotes serve como guia no desenvolvimento. O grau de dependência entre os pacotes é o menor possível e as dependências ilustradas são justificáveis.

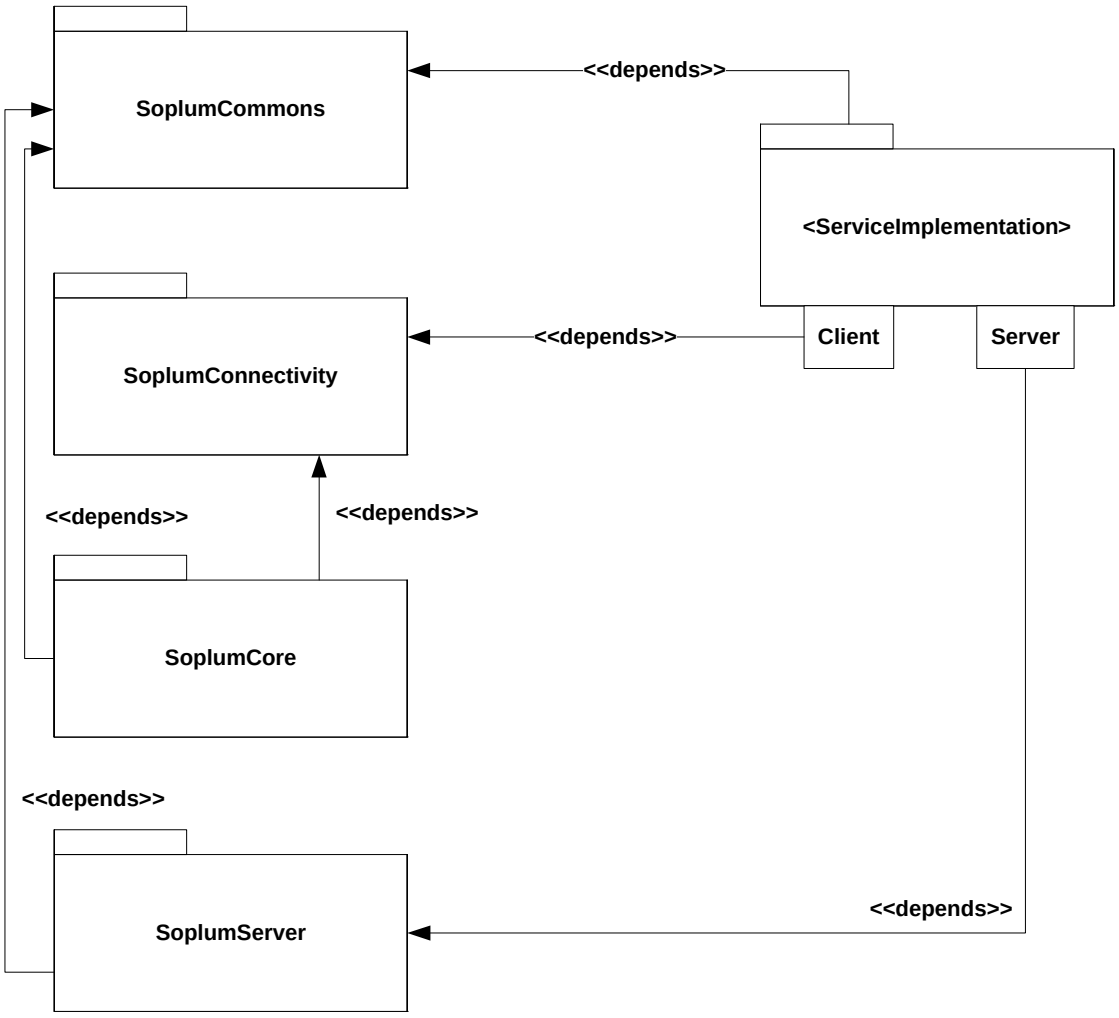


Figura IV.2 - Disposição de artefatos de código (*packages*)

IV.4 DESENVOLVIMENTO DA ARQUITETURA PARA A INTERFACE ADMINISTRATIVA

Essa seção ilustra os passos da análise necessária para a elaboração do módulo referente à interface administrativa que é uma extensão do CMS definido anteriormente.

IV.4.1 PRODUTO PLONE

Para adicionar funcionalidades a um portal *Plone* é necessário a criação de um Produto *Plone*. Este é um projeto *Python* do tipo *plug-in* que segue algumas convenções. Através de um produto, é possível alterar as funcionalidades do portal, adicionar novos tipos de

conteúdo, fluxos de trabalho, realizar alterações na camada de apresentação, etc. A forma mais eficiente de se criar um novo tipo de conteúdo no *Plone* consiste em implementar um produto apoiado no *framework Archetypes*, criado pelos desenvolvedores da equipe do *Plone* especialmente para alavancar novos subsistemas para o CMS.

O *Archetypes* utiliza uma metalinguagem onde são definidos *Schemas*. Um *Schema* define os atributos que serão utilizados por um tipo de conteúdo, bem como sua visibilidade nos diversos estágios do ciclo de vida deste conteúdo. Suponha, por exemplo, que seja necessário criar um tipo de conteúdo denominado Usuário, e este tipo possui os atributos *login* e *senha*. É possível definir que o atributo *senha* só será exibido para usuários que possuirem permissão para alterar as senhas. Além disso, o *Schema* define o tipo do atributo (*Booleano*, *Inteiro*, *String*, *Referência*, etc) e como este deve ser apresentado na interface, ou seja, se deve ser exibido em um campo texto, como um *checkbox*, etc. Todas as instâncias de um tipo de conteúdo construído pelo *Archetypes* são objetos *Python* que podem ser manipulados de forma sistemática e que podem ser persistidos no banco de dados de maneira transparente para o desenvolvedor.

IV.4.2 Modelo Básico de Usuários

O modelo de usuários do *Plone* é bem simples. Existem quatro entidades principais: *Usuário*, *Grupo*, *Papel* e *Permissão*.

Cada pessoa que visita um site *Plone* é um *Usuário*. Um *Usuário* que não esteja autenticado é chamado de usuário anônimo, e possui permissão apenas, na maior parte dos casos, para visualizar conteúdos. *Usuários* cadastrados no sistema podem efetuar o *login* e, dependendo de seu *Papel* dentro do site, podem adicionar conteúdos, revisar conteúdos publicados por outros usuários e também, visualizar conteúdos disponíveis. *Usuários* são identificados por uma sequência de caracteres curta (*String*), seu *username*.

Um site *Plone* possui uma série de *papéis*, que são uma categorização lógica de *usuários*. Esta abstração é utilizada para agilizar o trabalho de Administradores. Ao invés de configurar *permissões* para cada *usuário* cadastrado, essas *permissões* são definidas para um *papel* e este *papel* é dado a um conjunto de *Usuários*. Por exemplo, a *permissão* que determina se um conteúdo pode ou não ser visualizado é chamada "*view*". Para que um *usuário* possa visualizar um conteúdo, pelo menos um dos *papéis* que ele possui deve possuir a permissão "*view*". *Papéis* e *permissões*, conforme observado são definidos por um nome simples. Uma característica interessante no *Plone* é a possibilidade de relacionar um *papel* a um *usuário* no *contexto* de cada objeto.

Todo site *Plone* possui os seguintes *papéis* pré-definidos (é possível alterar o nome desses papéis, adicionar novos papéis ou remover papéis existentes):

- *Anonymous*: Todo usuário que ainda não esteja autenticado recebe esse papel automaticamente. Assim é possível distribuir permissões para usuários não autenticados, bastando dar ou retirar uma permissão do papel *Anonymous*. Este papel não pode ser associado a um usuário manualmente.

- *Authenticated*: Uma vez autenticado o usuário recebe esse papel automaticamente. Este papel também não pode ser associado a um usuário manualmente.

- *Owner*: Este é um papel especial, dado a um usuário no contexto de um objeto do site. Essa informação é salva no objeto, o que é feito automaticamente pelo *Plone* quando um usuário cria um conteúdo.

- *Member*: Qualquer usuário que se cadastre num site *Plone* possui este papel.

- *Reviewer*: São revisores de conteúdo, podendo alterar conteúdos inseridos por membros e publicá-los.

- *Manager*: São os administradores do site, e, por isso, não possuem restrições.

O *Plone* também conta com o conceito de *Grupos* que são meros agrupadores, normalmente utilizados para categorizar *usuários*, sem ter o viés de autorização de um *Papel*.

IV.4.3 MODELO DE USUÁRIOS DO SOPLUM

As entidades que compõem o *Soplum* utilizam uma modelagem bastante familiar quanto ao domínio de usuários *vs.* grupos. Neste ponto nos aproximamos bastante do modelo multiusuário com grupos de permissões utilizados por sistemas UNIX e pelo próprio *Plone*. Por se tratar de um sistema *web*, definimos a entidade ***WebUser*** para cada pessoa cadastrada no portal. Este ***WebUser*** possui uma relação direta com o *Usuário Plone*. De fato, um ***WebUser*** é um *Usuário Plone*. De forma semelhante, um ***WebGroup*** é um *Grupo Plone*. Este é o ponto de fusão do *Soplum* com o *Plone*. Em outras palavras, a partir desses relacionamentos fundimos a gerência de usuários de uma rede com a gerência de usuários do *Plone*.

As entidades ***WebUser*** e ***WebGroup*** possuem o mesmo comportamento de *Usuários* e *Grupos Plone*. Porém, diferente das entidades existentes no *Plone*, as entidades *Soplum* são

tipos de conteúdo, e portanto podem se relacionar com outros tipos de conteúdo, podem ter campos extras como telefone, e-mail, etc.

Para a gerência da rede, temos as entidades ***NetworkUser***, ***NetworkGroup***, ***MailingList*** e ***FilesystemQuota***. Estas compõem o conjunto de entidades utilizadas pelos diversos serviços disponíveis através do *Soplum*.

As entidades ***NetworkService*** e ***NetworkServer*** agregam as informações necessárias para o estabelecimento de conexão com os serviços.

O conjunto das entidades e seus relacionamentos está representado no modelo da Figura IV.3. Na mesma figura estão representados os seus atributos. Os modelos estão separados para que haja uma melhor clareza dos relacionamentos (e o modelo possui muitos relacionamentos), já que os atributos não influenciam o comportamento da entidade.

No diagrama pode se notar a presença de algumas interfaces, que são utilizadas para desacoplar o tipo de conteúdo em si, da gerência de serviços. Quando o gerenciador de serviços atua sobre um conteúdo ele não precisa saber que tipo de conteúdo é este, basta que o tipo de conteúdo implemente a interface *INetworkServiceAware*. Um objeto *INetworkServiceAware* está "ciente" dos serviços em sua volta. Este poderá estar registrado em alguns serviços, e quando for habilitado, seus serviços serão ativados pelo gerenciador de serviços automaticamente.

IV.4.4 Entidades que implementam *INetworkServiceAware*

Durante o ciclo de vida no sistema, as entidades que implementam a interface ***INetworkServiceAware*** são submetidas à um fluxo de trabalho com estados pré-definidos. Quando disparada, cada transição entre estados irá executar um determinado conjunto de tarefas para que o estado seja de fato alterado. A Figura IV.4 demonstra o fluxo que essas entidades seguem, com todos os estados e transições. Para facilitar o entendimento do projeto, sempre que nos referirmos a uma entidade ***NetworkServiceAware*** estaremos nos referindo a uma entidade que implementa a interface ***INetworkServiceAware***. O mesmo servirá para as outras interfaces definidas neste projeto.

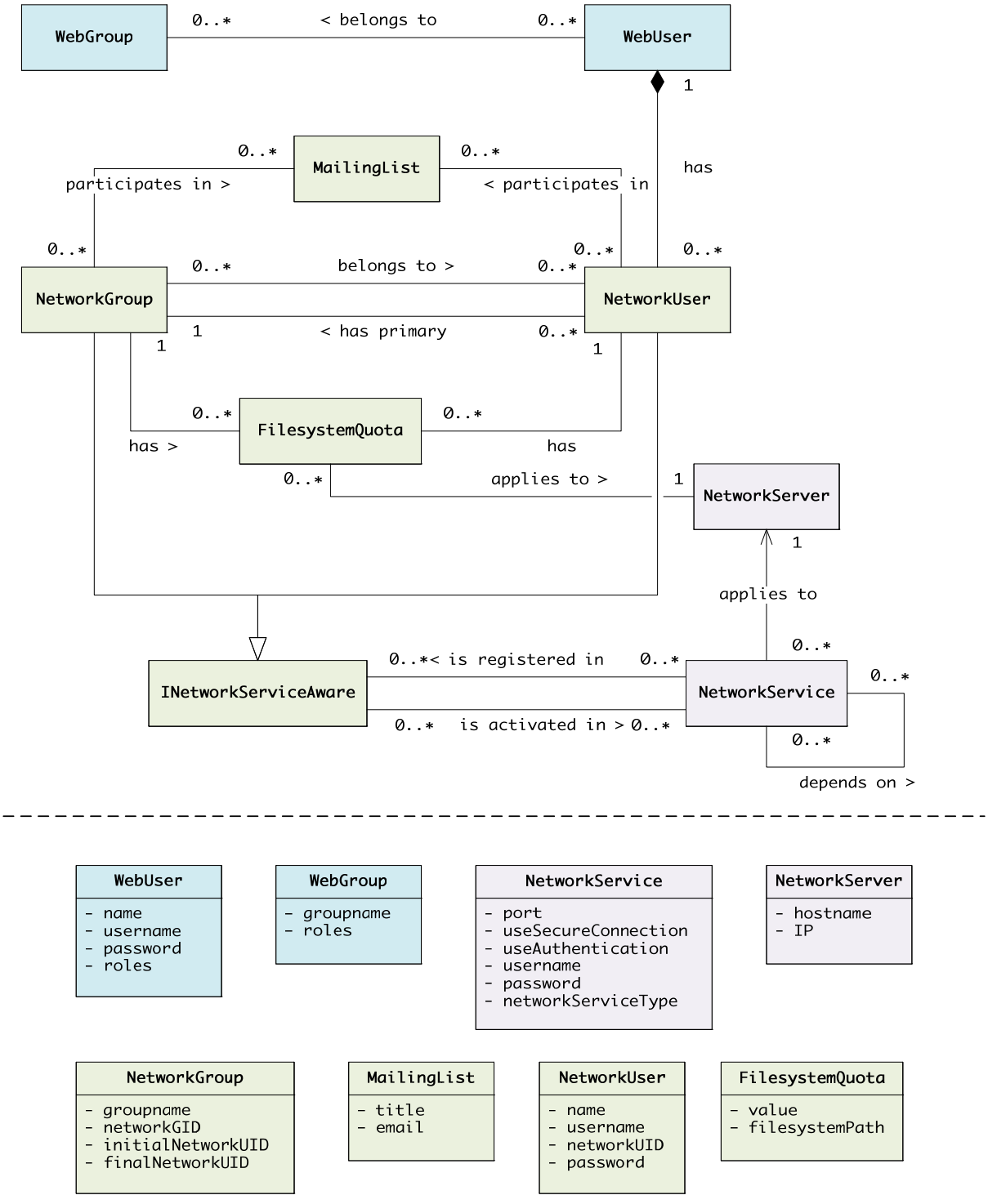


Figura IV.3 – Modelo relacional e atributos e entidades do *Soplum Core*

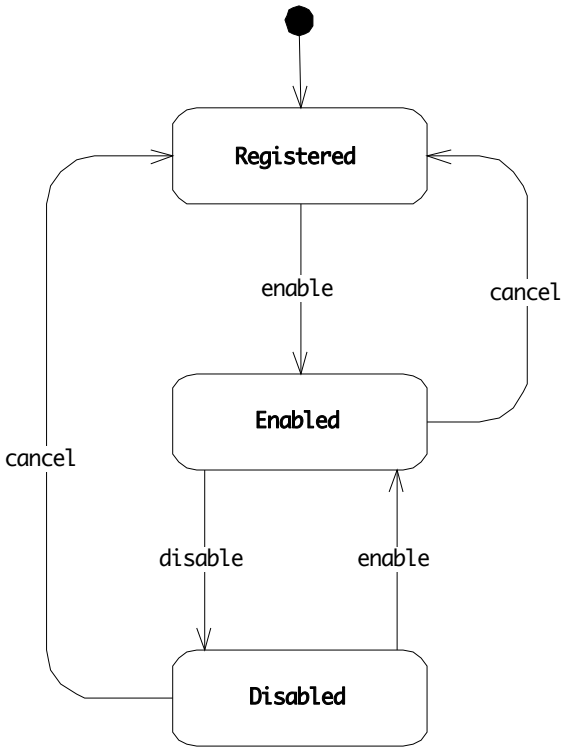


Figura IV.4 - Fluxo de entidades INetworkServiceAware

Os estados que uma entidade ***INetworkServiceAware*** pode assumir definem também seu estado nos serviços remotos:

- *Registered* - Este é o estado inicial de todas as entidades cadastradas no sistema. O estado define que a entidade está devidamente cadastrada no domínio web, porém não existe nenhum cadastro nos serviços remotos.
- *Enabled* - Uma entidade habilitada deve ter todos os seus serviços ativados.
- *Disabled* - Uma entidade desabilitada deve ter todos os seus serviços desabilitados.

Cada transição quando disparada é responsável por executar uma série de comandos. Esses comandos devem fazer com que a entidade seja transformada de maneira adequada para justificar o estado destino. Segue a lista de transições entre os estados:

- *Enable:* Habilita todos os serviços registrados para a entidade ***INetworkServiceAware***;
- *Disable:* Desabilita todos os serviços registrados para a entidade ***INetworkServiceAware***;
- *Cancel:* Cancela todos os serviços registrados para a entidade ***INetworkServiceAware***.

Para cada serviço criado, estas transições serão detalhadas na especificação de sua implementação. Em alguns tipos de serviços existe uma diferença entre cancelar e desabilitar o serviço, o que justifica a criação de duas transições aparentemente similares. Por exemplo, para um *NetworkUser*, registrado num serviço de contas de usuário NIS, desabilitar este serviço significa impedir a autenticação (*login*) na rede, através das credenciais cadastradas. Neste caso, todos os dados do usuário permanecem no banco de dados das contas de usuário NIS. Além disso, todos os arquivos do usuário permanecer no servidor de arquivos. No entanto, se cancelarmos este serviço para o *NetworkUser* então sua conta será apagada do banco de dados de contas de usuários NIS e seus arquivos serão compactados e movidos para um pasta de *backup*. É por isso que após essa transição ser efetuada, o estado do *NetworkUser* passa a ser *Registered*, ou seja, não há cadastro nos serviços remotos para este *INetworkServiceAware*.

IV.4.5 ARQUITETURA EXTENSÍVEL PARA CONECTIVIDADE COM SERVIÇOS

Antes de utilizar um serviço é preciso estabelecer uma conexão com o servidor onde ele está sendo provido. Essa conectividade pode, por decisão de projeto, ser alternada se desejado ou se for necessário devido a alguma restrição na rede, como por exemplo um servidor temporariamente indisponível.

O uso do subsistema de conectividade utiliza o padrão de projeto *Abstract Factory*. Desta maneira, sempre que um cliente de serviço precisar se conectar ao servidor, deverá solicitar ao ***ServiceConnectorFactory*** um ***IServiceConnector***. A ***ServiceConnectorFactory*** por sua vez decidirá qual implementação de ***IServiceConnector*** retornar. Para executar uma função remota de serviço o cliente precisa apenas ter conhecimento da interface ***IServiceConnector*** que é suficiente para que o sistema efetue a execução. A Figura IV.5 ilustra o modelo de classes com as interfaces citadas.

Note que este módulo consiste apenas na decisão de que tipo de conexão e qual protocolo de comunicação utilizar para se comunicar com o Serviço. O serviço em si é passado como parâmetro para a função *executeServiceAction(serviceID, actionName, data)*. O cliente do serviço usa o conector retornado pela *factory*. Os detalhes de como a comunicação será feita ficam a cargo deste conector.

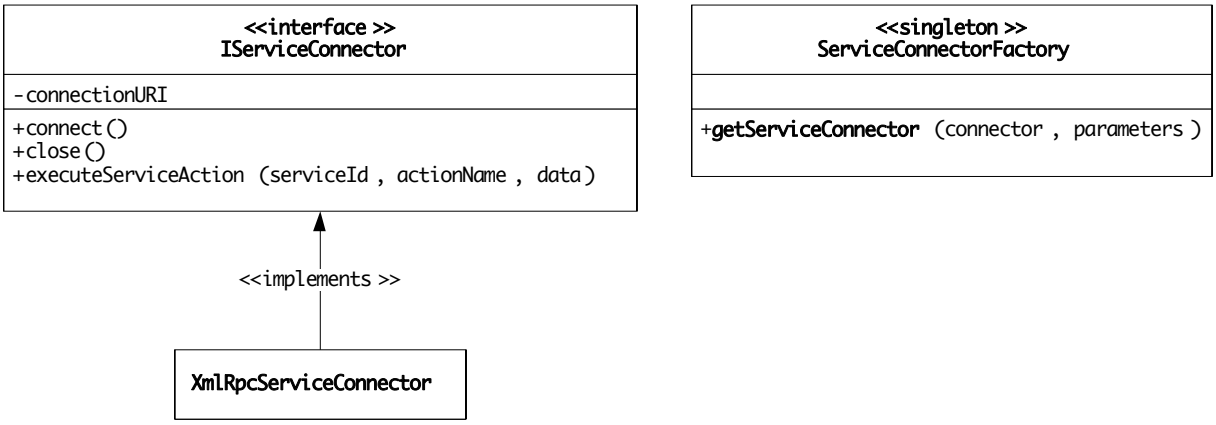


Figura IV.5 - modelo extensível de conectividade com serviços

O *Soplum* inclui uma implementação de ***IServiceConnector*** funcional e segura, com qualidade de produção. O protocolo padrão de conectividade para o *Soplum* é uma implementação de cliente XML-RPC utilizando transporte criptografado HTTPS. Portanto a configuração padrão do ***ServiceConnectorFactory*** terá um ***XmlRpcServiceConnector*** como implementação padrão de ***IServiceConnector***. Conseqüentemente, a implementação do servidor será análoga, utilizando XML-RPC sobre HTTPS.

IV.4.6 ARQUITETURA EXTENSÍVEL PARA O USO DE SERVIÇOS

Estabelecido o transporte dos dados do cliente para o servidor, é preciso definir a lógica de negócio para a execução de uma determinada funcionalidade oferecida pelo serviço. Para que isso seja feito de forma correta, é necessário que o cliente conheça a quais funções o serviço responde e em qual ordem as funções devem ser chamadas.

Neste ponto se faz claro a decisão de manter os artefatos de código do cliente e do servidor no mesmo projeto. Não faz sentido codificá-los em projetos separados frente à existência de um acoplamento implícito em tempo de execução das chamadas de funções. Isto simplifica a implementação dos gerenciadores de serviços tanto no lado cliente, quanto no

lado servidor, pois delega ao serviço a responsabilidade de cumprir um contrato que é estabelecido por ele próprio.

IV.4.7 DISTRIBUIÇÃO DE RESPONSABILIDADES

Um dos objetivos do *Soplum* é simplificar as tarefas do administrador da rede, sem que isso cause um impacto grande na manutenção do código. A distribuição de responsabilidades é um conceito muito popular em sistemas colaborativos, e foi utilizado intrinsecamente na definição da arquitetura do *Soplum*.

Além desse conceito, dois outros conceitos influenciaram a definição da arquitetura: *Don´t Repeat Yourself* (DRY) e *Convention Over Configuration*. Por que um administrador precisaria instalar um pacote com um serviço e configurá-lo tanto no servidor quanto no cliente? A abordagem do *Soplum* é utilizar uma convenção para que os serviços sejam carregados pelos gerenciadores de serviços tanto no servidor quanto no cliente.

Uma das características do *Python* é a sua capacidade de carregar módulos dinamicamente. Para isso, basta passar um caminho no sistema de arquivos até este módulo. Além disso, os pacotes *Python* possuem a capacidade de estarem espalhados no sistema de arquivos (diferentemente de pacotes Java, que são representados diretamente como diretórios no sistema de arquivos). Unindo essas duas características foi criada uma API que busca por serviços em diretórios configurados no sistema de arquivos e os carrega.

Este mecanismo funciona da seguinte maneira:

- É configurado nos gerenciadores de serviços, tanto no servidor quanto no cliente, um conjunto de caminhos no sistema de arquivos que deve ser utilizado na busca de serviços.

- Para cada caminho configurado, a API irá buscar por pacotes de serviços que tenham, em seu arquivo de inicialização, denominado __init__.py, uma função de inicialização do serviço. No lado do cliente essa função é chamada de *initializeSoplumClientServices()*, enquanto que no lado do servidor esta função é chamada de *initializeSoplumServerServices*. Note que o nome das funções é construído de tal forma que não haja colisão com nomes comuns em outros pacotes *Python*. Dessa forma, não há perigo de se chamar funções de pacotes não relacionados por engano. É de responsabilidade da implementação do serviço inserir o código necessário para que este seja registrado no servidor ou cliente.

- Uma vez registrado adequadamente, o serviço será chamado quando necessário pelos gerenciadores de serviços.

A seguir temos um exemplo de arquivo de configuração para uma implementação de cliente de serviço:

```
"""A <ServiceName> Service."""

import os


from SoplumCommons.services import getClientServiceType


def initializeSoplumClientServices():

    """Initialize the soplum client services."""

    # Repare que o pacote Products.SoplumCore.services foi extendido, e o pacote
do serviço agora

    # se encontra dentro deste pacote, sem a necessidade de se colocar o pacote
fisicamente no mesmo

    # diretório que o pacote de serviços do gerenciador de serviços.

    from     Products.SoplumCore.services.<ServiceName>Service.<ServiceName>Client
import <ServiceName>Client


    from Products.SoplumCore.services.manager import registerServiceType


    type = 'ServiceType'


    registerServiceType(type, <ServiceName>Client)
```

IV.5 DESENVOLVIMENTO DA ARQUITETURA DA INTERFACE NÃO INTERATIVA DE SERVIÇOS

O desenvolvimento de um programa não interativo requer uma abstração um pouco diferente da comum. Um sistema não interativo nada mais é do que um grande reator, uma máquina de estados internos e pré-determinados, que reage a determinados comandos pré-definidos. Dessa maneira, podemos começar a modelagem com um diagrama de alto nível do processo de inicialização do módulo Servidor do sistema de Serviços, mostrando o que ocorre desde sua a inicialização até o momento em que o mesmo se torna um processo não interativo (*daemon*). (Figura IV.6).

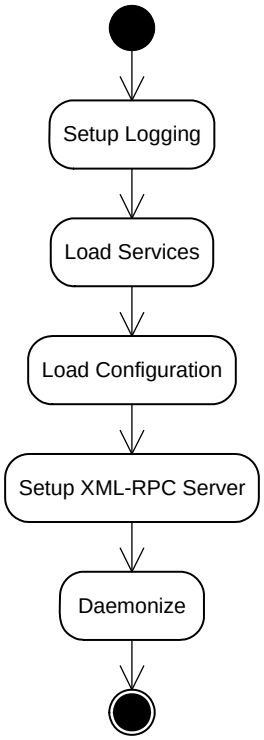


Figura IV.6 - *Bootstrap* do Servidor

Os passos de inicialização são:

1. "*Setup Logging*" - Prepara o sistema de auditoria em arquivo texto.
2. "*Load Configuration*" - Busca no arquivo de configuração informações de localização dos serviços, certificados de chave pública e privada para criptografia

no servidor HTTPS, interface de rede e porta para aguardar por conexões, além de informações de autenticação do servidor HTTPS.

3. "*Load Services"* - Inicia uma busca nos diretórios indicados nos parâmetros de configuração por serviços, preparando-os para a instanciação em tempo de execução.

4. "*Setup XML-RPC Server*" - Prepara os contextos SSL, baseado nas configurações de certificados, e o servidor HTTP usando o transporte seguro SSL (HTTPS) do servidor XML-RPC propriamente dito.

5. "*Daemonize*" - Na configuração do servidor de produção, o processo é levado para segundo plano (*background*) e deixa de ser interativo ou dar retorno ao terminal, caracterizando o programa como um *Unix Daemon.*

IV.5.1 MODELO DA APLICAÇÃO

Esquematizada na Figura IV.7 as classes indicadas na cor cinza são as classes fundamentais da biblioteca *Twisted* que implementa uma robusta plataforma genérica e assíncrona para aplicações com intuito de prover algum serviço de rede, como servidores web, servidores de arquivos, entre outros. Essa biblioteca possui uma classe genérica para aplicações que rodem em segundo plano, sendo assim bastante oportuna para o uso no projeto.

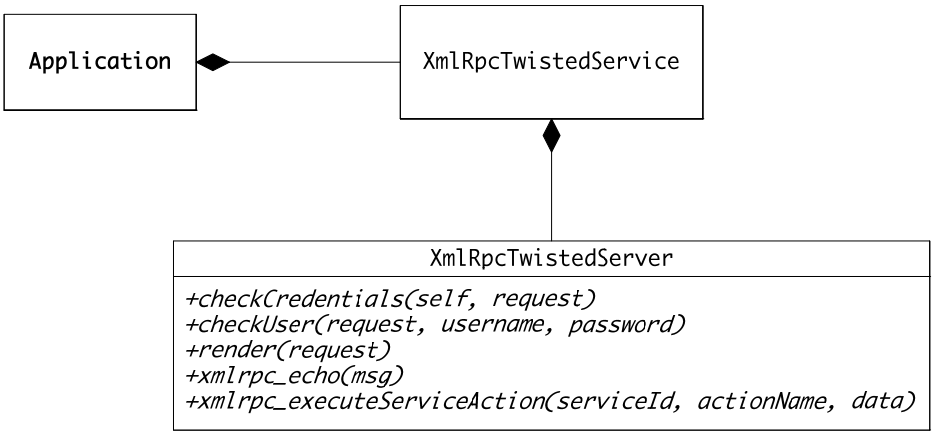


Figura IV.7 - Modelo de Classes da aplicação servidor

A classe *XmlRpcTwistedServer* de fato é a classe que atua no papel de Servidor. Essa classe agrega as funções de autenticação (*checkCredentials(), checkUser()* e *render()*) e o servidor HTTP com as extensões para servir também em SSL (HTTPS), além do fato de ser responsável por receber as chamadas remotas. A função *xmlrpc_echo()* serve como uma verificação de "pulso" do servidor. Ela é utilizada quando se quer verificar se o servidor está respondendo, e se o canal de comunicação está estabelecido. Esta função retorna qualquer dado que tenha sido enviado a ela. O método *xmlrpc_executeServiceAction()* é a porta de entrada dos comandos enviados pelo cliente. Esta delega ao gerenciador de serviços a execução e tratamento de erros da ação propriamente dita.

IV.5.2 ARQUIVO DE CONFIGURAÇÃO

A interface não interativa requer uma configuração prévia feita em um arquivo com sintaxe simples. O interpretador do arquivo é uma classe da biblioteca nativa *Python*.

Arquivo de configuração:

```
[directories]

base_dir = /opt/SoplumServer

test_files_dir = %(base_dir)s/tests/files

certificates_dir = %(base_dir)s/etc

services_dirs = /opt/services

[network]

hostname = localhost

port = 8051

[xmlrpc]

enable_authentication = True

username = dummyUser

password = $1$AMOwVqbx$YwPGIbXkxz1qVlZ1ZzSiH1

user_credentials_filename = /opt/SoplumServer/conf/user_credentials_file
```

Cada seção é determinada por um cabeçalho delimitado por um par de colchetes. No arquivo existem três seções: *directories*, *network*, *xmlrpc*. Dentro das seções são definidos valores para parâmetros que são acessados no processo de inicialização.

- *Directories* - todas as suas entradas são caminhos no sistema de arquivos.

  base_dir - local onde o servidor está instalado;

  test_files_dir - caminho de arquivos utilizados pela suíte de testes do servidor;

  certificates_dir - caminho dos certificados de chave pública e privada utilizada na comunicação criptografada;

  services_dir - uma lista de caminhos, separados por vírgula, por onde o sistema deve iniciar a busca por serviços instalados.

- *Network* - define parâmetros relacionados a conexões de rede.

  hostname - endereço que determina a interface de rede na qual o serviço deve fazer o *binding* da interface de rede;

  port - número da porta em que o serviço deve aguardar por conexões.

- *XmlRpc* - define parâmetros relacionados ao protocolo de chamada remota de função.

  enable_authentication - *flag* que determina se será utilizado o sistema de autenticação ou não;

  username e password - definição rápida de um usuário;

  user_credential_files - caso se queira definir uma infinidade de usuários deve se utilizar um arquivo externo, cujo nome deve ser definido nesse parâmetro.

IV.5.3 A ARQUITETURA EXTENSÍVEL DE SERVIÇO

A arquitetura de extensibilidade de serviços é análoga àquela do cliente, também utilizando arquivos de configuração para determinar os tipos de serviços expostos para o cliente.

A seguir, um exemplo de arquivo __init.py__ para um serviço <ServiceName>. O que faz com que esse serviço seja encontrado é a existência da função initializeSoplumServerServices().

IV.5.4 Tipos e Instâncias de Serviços

Uma coisa que pode parecer confusa a princípio são os conceitos de Tipo e Instância de Serviços. Os serviços do *Soplum* têm como função realizar alterações nos servidores da rede (NIS, *Samba*, HTTP, Mailman, cotas, etc). Por isso, as configurações desses serviços ficam armazenadas e são utilizadas apenas pela implementação do serviço no lado do servidor. O cliente, neste cenário, só precisa implementar o protocolo de chamadas às ações existentes no servidor conforme sua própria lógica de negócio.

```
"""A <ServiceName> Service."""

import os

from SoplumCommons.services import getServerServiceInstances

def initializeSoplumServerServices():
    """Initilize the soplum server services."""
    # We need to import this using the SoplumServer.services namespace because
    # when this method is called the UnixAccount namespace cannot be found by
    # the python import machinery.
    from SoplumServer.services.<ServiceName>Service.<ServiceName>Server
        import <ServiceName>

    from SoplumServer.services.manager import registerServiceInstances

    instances = getServerServiceInstances(os.path.dirname(__file__))
```

É por isso que quando estamos inicializando serviços no lado do cliente nos referimos apenas ao seu tipo, enquanto que quando o fazemos no lado do servidor nos referimos às diversas instâncias desse serviço. Cada instância possui um identificador único e um conjunto de configurações específicas para aquela instância. Com isso é possível termos duas instâncias de um mesmo serviço no servidor, utilizando configurações diferentes. Suponha que existam dois servidores HTTP, um utilizado pelos alunos e outro utilizado por professores. Poderíamos utilizar o mesmo serviço de contas HTTP para manipular ambos os servidores HTTP, alterando apenas a configuração para apontar para os arquivos de banco de dados de usuários correspondentes.

IV.6 DESENVOLVIMENTO DE MÓDULOS PARA O SOPLUM

Esta seção contém a análise necessária para a implementação da arquitetura base em que módulos cliente/servidor dos serviços oferecidos por uma rede UNIX utilizarão.

IV.6.1 ABORDAGEM

Num ambiente *Unix*, existem algumas maneiras de gerir a base de usuários de um serviço. A mais comum é interagir através de comandos no interpretador de comandos (*Shell*), utilizando comandos de manipulação da base de usuários ou de grupos de um determinado serviço. Tais comandos devem ser providos pelo serviço local no sistema operacional. Uma abordagem similar seria a utilização da própria API de manipulação de serviços do *Unix*, porém esta iria requerer compilação de programas. Isto dificultaria sua implantação pois o processo de compilação e link-edição com as bibliotecas do sistema são particulares para cada sistema operacional e mesmo se tratando de sistemas aderentes ao padrão *Portable Operating System Interface* (POSIX) temos uma variedade incontável de sistemas do tipo *Unix*, tornando a prática inviável. Outra forma que se tem é a manipulação direta do arquivo onde são armazenados os dados, mas para tal é necessário que se conheça o formato do arquivo e depois se informe ao serviço que sua base foi alterada. Levando-se em conta que podemos utilizar uma linguagem interpretada multi-plataforma como o *Python* para manipular arquivos, podemos criar uma solução multi-plataforma a partir desta última abordagem.

No desenvolvimento do *Soplum*, a abordagem tomada é preferencialmente a de manipulação direta dos arquivos por diversos motivos. Dentre eles está a facilidade de manipulação dos mesmos, uma vez que seguem um padrão bem definido, muito similar ao padrão *Comma-Separated Values* (CSV). Além disso, caso ocorra algum erro durante o processo, é possível facilmente colocar a base de dados consistente, bastando para isso restaurar um *backup* anterior do arquivo.

O processo de manipulação é determinado pelo seguinte protocolo: em primeiro lugar o arquivo é interpretado e levado para o domínio programático de objetos, manipulado por código e em seguida salvo novamente em arquivos. Essa abordagem é uma implementação simplória de serialização de objetos citada na seção de estratégias persistência de dados.

No entanto nem sempre é possível obter controle total do serviço apenas manuseando seus arquivos de dados, muitas vezes são necessárias as execuções de alguns comandos do interpretador de comandos. O *Soplum* também utiliza, nesses casos, a interação com comandos do interpretador. Isto é conseguido através da utilização da biblioteca *PyExpect*,

que permite que seja aberto um pseudo-terminal onde são executados comandos, e seu retorno é então tratado através do uso de expressões regulares. A cada saída do comando executado é possível executar uma ação, como retornar um erro, ou enviar um *username* e uma senha. Resumidamente é possível interagir de forma automática com um programa que necessitaria de interação humana para ser executado.

IV.6.2 Dependência entre Serviços

Um dos conceitos que o *Soplum* utiliza é o fato de que alguns serviços dependem de outros serviços. Este é um requisito real de alguns tipos de serviços, como, por exemplo, o *Samba*, que requer uma conta de usuário local ou NIS criada para cada conta *Samba*. No *Soplum* isso é mapeado de forma flexível, podendo ser configurado pelo administrador do sistema. Isto é necessário uma vez que podemos ter serviços similares em mais de um servidor. Suponha um cenário onde se tenha dois servidores, A e B, e que ambos ofereçam serviços NIS e Samba. O cadastro dos serviços seria feito da seguinte forma:

- um serviço NIS registrado no servidor A chamado de NisServiceA;
- um serviço Samba registrado no servidor A chamado de SambaServiceA;
- um serviço NIS registrado no servidor B chamado de NisServiceB;
- um serviço Samba registrado no servidor B chamado de SambaServiceB.

Note que se a relação de dependência não pudesse ser configurada por serviço, teríamos ambos os serviços SambaServiceA e SambaServiceB dependendo dos serviços NisServiceA e NisServiceB. Os serviços NisServiceA e NisServiceB teriam que estar ambos registrados e ativados para que se pudesse registrar e ativar os serviços SambaServiceA e SambaServiceB.

Uma vez que a relação de dependência pode ser configurada, basta que o administrador defina que o serviço SambaServiceA depende de NisServiceA e que SambaServiceB depende de NisServiceB.

Esta relação de dependência define a ordem em que os serviços são ativados e desativados. Ao se ativar um serviço, é necessário em primeiro lugar que todas as suas dependências que ainda não estão ativadas devem ser ativadas. Caso alguma dependência possua outra dependência, então esta também deve ser ativada antes, e assim sucessivamente. Qualquer erro na ativação de uma dependência (ou da dependência desta) resultará na falha de

ativação do serviço. Uma mensagem de erro é então retornada ao usuário. Note que outros serviços que não estejam incluídos nessa árvore de dependências serão ativados normalmente. A falha de ativação de um serviço não implica na não-ativação de serviços independentes.

Na desativação de serviços o processo é invertido. Ao se desativar um serviço que possua dependentes, primeiro todos seus dependentes devem ser desativados, para só então este ser desativado. Novamente a falha na desativação de um dependente (ou dependente deste) resultará na falha de desativação do serviço e uma mensagem de erro apropriada será retornada ao usuário. Da mesma forma que na ativação, a falha de desativação de um serviço não implica na não-desativação de serviços independentes.

Capítulo V - Implementação dos Serviços

Este capítulo agrega a análise necessária para a implementação dos Serviços disponíveis rede UNIX e os quais se deseja obter controle através do Soplum.

V.1 LINUX / FREEBSD ACCESS CONTROL SERVICE

Esta seção analisa o controle de acesso de um sistema UNIX (distribuições de Linux e FreeBSD) este é o principal módulo da gerência de usuários e grupos de rede.

V.1.1 DESCRIÇÃO

Este serviço é o responsável pela criação de usuários e grupos em sistemas Linux. Ele oferece suporte a contas locais de usuários, mas também é possível gerenciar contas NIS com este serviço.

V.1.2 FUNCIONALIDADES

- Criação de contas locais e NIS.
- Utilização opcional do arquivo *shadow*. Este arquivo é utilizado como medida de segurança. O arquivo principal de senhas, comumente chamado *passwd*, pode ser lido por todos os usuários do sistema, e por isso é considerado inseguro. Já o arquivo *shadow* só pode ser lido pelo usuário *root*. Este contém todas as senha encripitadas dos usuários. Dessa forma nenhum usuário do sistema, a não ser o *root*, tem acesso às senhas de outros usuários.
- Utilização opcional do arquivo *netgroup*. Este arquivo tem como função simplificar o controle de acesso de usuários NIS em estações de trabalho e servidores.
- Definição flexível de comandos. É possível definir de forma flexível que comandos devem ser executados após a base de dados ser alterada.
- Definição flexível de diretórios de usuários. É possível definir que diretórios devem ser criados para cada tipo de usuário, bem como a permissão padrão para este diretório.

V.1.3 CONFIGURAÇÃO

Para este serviço temos as seguintes configurações, organizadas por seções:

1. *Paths:* Nesta seção são definidos os diversos caminhos de arquivos manipulados por este serviço.
   a. *passwd_dir*: Diretório onde se localiza o arquivo principal com as contas de usuários.
   b. *passwd_filename*: O nome do arquivo principal com as contas de usuários.
   c. *shadow_dir*: Diretório onde se localiza o arquivo shadow, contendo as senhas encriptadas dos usuários. (opcional)
   d. *shadow_filename*: O nome do arquivo shadow, contendo as senha encriptadas do usuários. (opcional)

e. *group_dir*: Diretório onde se localiza o arquivo de grupos de usuários.

f. *group_filename*: O nome do arquivo de grupos de usuários.

g. *netgroup_dir*: Diretório onde se localiza o arquivo netgroup, utilizado por servidores NIS. (opcional)

h. *netgroup_filename*: O nome do arquivo netgroup, utilizado por servidores NIS. (opcional)

i. *backup_dir*: O diretório onde devem ser armazenados os *backups* dos arquivos manipulados por este serviço.

j. *user_backup_dir*: O diretório onde devem ser armazenados os *backups* das pastas dos usuários quando estes são removidos.

k. *user_null_home_dir*: O diretório que deve ser utilizado por contas de usuário que não devem possuir diretórios pessoais (utilizado principalmente para contas de máquinas).

2. *Ordering*: Nesta seção são definidos os padrões de ordenação dos arquivos manipulados por este serviço. Para todas as configurações nesta seção, deixar seu valor vazio ou utilizar um valor diferente dos valores possíveis desabilita a ordenação do arquivo.

   a. *passwd_field*: O campo utilizado para ordenar o arquivo principal com as contas de usuários. Pode ser 'uid', 'username' ou 'name'.

   b. *group_field*: O campo utilizado para ordenar o arquivo de grupos. Pode ser 'gid' ou 'groupname'.

   c. *shadow_field*: O campo utilizado para ordenar o arquivo shadow, contendo as senhas encriptadas dos usuários. Pode ser apenas 'username'.

   d. *netgroup_field*: O campo utilizado para ordenar o arquivo netgroup, utilizado por servidores NIS. Pode ser apenas 'group_abbreviature'.

   e. *netgroup_user_field*: O campo utilizado para ordenar os usuários pertencentes a um netgroup dentro do arquivo netgroup. Pode ser 'host', 'username' ou 'domain';

3. *Shells*: Nesta seção são definidos os *shells* utilizados por usuários habilitados e desabilitados.

   a. *enabled_shell*: O *shell* utilizado por usuários habilitados.

   b. *disabled_shell*: O *shell* utilizado por usuários desabilitados.

4. *Netgroups*: Nesta seção são definidos alguns parâmetros utilizados na manipulação de arquivos netgroup.

   a. *domain*: O domínio a ser usado por novos usuários.

   b. *one_user_per_line*: Esta configuração booleana indica se o arquivo deve ser salvo colocando um usuário de cada grupo por linha, ao invés de colocar tudo na mesma linha. É mais uma questão estética do que funcional.

c. *Commands*: Este é um tipo de seção especial. Para cada comando que deve ser executado ao final de uma transação, i.e., sempre que uma transação está sendo finalizada (*commited*). Para cada comando deve-se criar uma seção que inicie com '*command_*'. O identificador do comando será o texto após o '_'. Cada seção deve definir duas configurações, o *path*, que é o diretório onde o comando deve ser executado, e *command* que é o comando em si. Por exemplo, para um comando make (executado sempre após a criação de uma conta NIS), temos a seguinte configuração:

```
[command_make]
path = /var/yp
command = make
```

5. *User Directories*: Da mesma forma que a seção acima, esta também é especial. Aqui são definidos os diretórios que devem ser criados para cada usuário, além do diretório pessoal do usuário. Uma explicação detalhada está colocada no arquivo de configuração de exemplo que acompanha o pacote com o serviço.

V.1.4 GERÊNCIA DE USUÁRIOS

Este serviço é capaz de adicionar, habilitar, desabilitar, alterar e remover usuários. A seguir estão listadas as ações que são executadas para cada transição realizada no cliente:

- *Activate:*
  1. Verifica-se se o usuário está cadastrado na rede:
     1.1. Caso o usuário não esteja cadastrado na rede:
        1.1.1. É realizado o cadastro do usuário na rede;
     1.2. Caso o usuário já esteja cadastrado na rede:
        1.2.1. Verifica-se se este já está habilitado:
           1.2.1.1. Caso não esteja, ele é habilitado.
        1.2.2. Atualiza-se os dados do usuário;
  2. Atualiza-se os grupos desse usuário na rede.
- *Update:* O mesmo algoritmo utilizado na ativação é utilizado na alteração.
- *Deactivate:*
  1. Verifica-se se o usuário está cadastrado na rede:
     1.1. Caso esteja cadastrado:
        1.1.1. Verifica-se se este já está desabilitado:
           1.1.1.1. Caso não esteja, ele é desabilitado;

- *Cancel:*

1. Verifica-se se o usuário está cadastrado na rede:

    1.1. Caso esteja cadastrado ele é removido:

V.1.5 GERÊNCIA DE GRUPOS

Este serviço também é capaz de adicionar, alterar e remover grupos de usuários. A seguir estão listadas as ações que são executadas para cada transição realizada no cliente:

- *Activate:*

1. Verifica-se se o grupo está cadastrado na rede:

    1.1. Caso o grupo não esteja cadastrado na rede, ele é cadastrado;

        1.1.1. Atualiza-se a lista de usuários pertencentes a este grupo.

- *Update:* O mesmo algoritmo utilizado na ativação é utilizado na alteração.
- *Deactivate:* O mesmo algoritmo utilizado no cancelamento é utilizado na desativação.
- *Cancel:*

1. Verifica-se se o grupo está cadastrado na rede:
    1.1. Caso esteja cadastrado ele é removido.

V.2 FILE SYSTEM QUOTA SERVICE

Esta seção traz a análise do módulo de serviço de cotas de disco para usuários e grupos.

V.2.1 DESCRIÇÃO

O serviço de cotas de disco serve para racionalizar o uso de um recurso compartilhado por todos os usuários do servidor, o espaço em disco. A manipulação das cotas é feita sempre em nível de cota por usuário, podendo utilizar grupos como forma de adicionar/remover cotas em lote, ou seja, quando se adiciona uma cota para um determinado grupo de pessoas o que se está realmente fazendo é adicionando a cota para cada pessoa do grupo. A cota é definida com duas importantes características:

Soft limit: A quantidade de espaço em disco que pode ocupar em um determinado filesystem, sendo um limite permissivo, onde o usuário pode ultrapassar essa quantidade até esbarrar no limite de corte, que é inflexível.

Hard limit: Quantidade máxima de espaço que um usuário pode ocupar além do seu *soft limit*. Esta cota não permite que o usuário ultrapasse seu valor.

V.2.2 FUNCIONALIDADES

- Adição e remoção de cotas de disco para usuários;
- Adição e remoção de cotas de disco para usuário utilizando grupos;
- Definição de S*oft limit* para cota feito via interface web;
- Configuração de cota *Hard limit* percentual baseada no *Soft limit*;
- Configuração flexível de comando da ferramenta para alteração de cota;

V.2.3 CONFIGURAÇÃO

1. *Main:*
    a. *Command*: define qual commando deverá ser executado para alteração das cotas de disco, podendo ser: ‘edquota’ ou ‘quotatool’ ambos utilitários mais comuns nos utilitários das distribuições de Unix/Linux;
    b. *Hard Limit Percentage:* Define o limite de corte para o armazenamento, sendo esse limiar uma porcentagem do *Soft limit*;
    c. *Soft Limit Conversion:* Multiplicador do valor de *Soft Limit*.
2. *Timeout:* Define opções de tempo de espera para o serviço.
    a. *Timeout:* Define tempo de espera máximo para execução do comando utilitário para alteração de cota, este tempo é necessário, pois a execução não se faz de maneira instantânea nos servidores.

V.3 HTTP ACCESS CONTROL SERVICE

Esta seção traz a análise do módulo de serviço de gerência de usuários do Apache.

V.3.1 DESCRIÇÃO

Esse serviço é capaz de gerenciar os usuários do servidor web, utilizados para a autenticação e autorização à recursos previamente configurados.

V.3.2 FUNCIONALIDADES

- Gerência de Usuários
- Gerência de Grupos

V.3.3 CONFIGURAÇÃO

1. Nesta seção são definidos os caminhos de arquivos que contém os dados de usuários e que serão manipulados por este serviço.

   a. *http_user:* caminho no sistema onde o arquivo com os dados de autorização dos usuários está armazenado;

   b. *http_user_filename:* o nome do arquivo de autorização de usuários;

   c. *http_group_dir:* caminho no sistema onde o arquivo com os dados de autorização para os grupos de usuário está armazenado;

   d. *http_group_filename:* o nome do arquivo de autorização de grupos.

2. *Ordering*: Nesta seção são definidos os padrões de ordenação dos arquivos manipulados por este serviço. Para todas as configurações nesta seção, deixar seu valor vazio ou utilizar um valor diferente dos valores possíveis desabilita a ordenação do arquivo.

   a. *http_user_field:* nome do campo que será utilizado para manter a ordenação das entradas de registro do arquivo de usuários, no momento só aceita a ordenação por ‘username’, podendo ter seu comportamento alterado caso desejado;

   b. *http_group_field:* nome do campo que será utilizado para manter a ordenação das entradas de registro do arquivo de grupos, aceitando no nomento somente o valor ‘groupname’.

V.4 MAILMAN SERVICE

Esta seção traz a análise do módulo de serviço de gerência do Mailman (listas de email).

V.4.1 DESCRIÇÃO

Esse serviço é responsável pela manutenção do cadastro de usuários em listas de email do Mailman.

V.4.2 FUNCIONALIDADES

- Adição e remoção de usuários em uma lista de email;
- Adição e remoção de grupos de usuário em uma lista de email;

V.4.3 CONFIGURAÇÃO

1. *Timeout:*Define opções de tempo de espera para o serviço.
    a. *timeout:* tempo de espera para a execução do comando de manutenção de lista do Mailman.
2. *Temporary_files:*
    a. *temporary_dir:* diretório onde serão armazenados o arquivo temporário para manipulação das listas de email pelo serviço;
    b. *temporary_file:* nome do arquivo onde serão armazenados os dados das listas de email quando manipuladas pelo serviço.

V.5 SAMBA ACCOUNT SERVICE

Esta seção traz a análise do módulo de serviço de gerência do Samba, que fornece contas de usuários e contas para os computadores efetuarem logon e terem acesso aos seus arquivos quando utilizando uma estação Windows.

V.5.1 DESCRIÇÃO

Este serviço é responsável pela manutenção do cadastro de usuários no domínio do Samba.

V.5.2 FUNCIONALIDADES

- Gerência dos usuários no domínio Samba
- Backup dos dados dos usuários excluídos

V.5.3 CONFIGURAÇÃO

1. *Paths:* Nesta seção são definidos os diversos caminhos de arquivos manipulados por este serviço.

    a. *smbpasswd_dir:* diretório onde estão armazenados os arquivos com dados de usuários e grupo do Samba;

    b. *smbpasswd_filename:* Nome do arquivo onde estão os dados de usuário e grupo do Samba;

    c. *backup_dir:* diretório onde serão armazenados os pacotes de cópia de segurança dos usuários desabilitados;

2. *Ordering*: Nesta seção são definidos os padrões de ordenação dos arquivos manipulados por este serviço. Para todas as configurações nesta seção, deixar seu valor vazio ou utilizar um valor diferente dos valores possíveis desabilita a ordenação do arquivo.

    a. *smbpasswd_field:* nome do campo que será utilizado para manter a ordenação das entradas de registro do arquivo de usuários, no momento só aceita a ordenação por ‘uid’, podendo ter seu comportamento alterado caso desejado;

3. *Flags:*

    a. *user_enabled:* template de flags para quando o usuário está habilitado;

    b. *user_disabled:* template de flags para quando o usuário está desabilitado;

    c. *workstation_enabled:* template de flags para quando uma conta de máquina está habilitada;

    d. *workstation_disabled:* template de flags para quando uma conta de máquina está desabilitada.

4. *Misc:*

    a. *empty_password:* template preenchedor para senhas vazias;

    b. *user_account_types:* tipos de conta que devem ser identificadas como contas de usuário pelo sistema, a implementação do Soplum traz inicialmente os tipos: ‘User Account’, ‘Student Account’, ‘Professor Account’ e ‘Employee Account’;

    c. *workstation_account_types:* tipos de conta suportados para computadores pelo sistema, a implementação do Soplum traz inicialmente os tipos: ‘Machine Account’, ‘Workstation Account’, ‘Server Account’.

Customizações extras do DEL

Esta seção demonstra as alterações feitas para comportar os tipos de dados específicos do DEL, assim como sua interface gráfica modificada.

V.6 DELSKIN E CRIAÇÃO DO PORTAL DEL

O produto DelSkin contém as customizações da interface gráfica do Plone, formando uma identidade visual para o Portal do DEL. Uma amostra do layout e disposição dos elementos pode ser observado na Figura V.I

Figura VI.1 – Página inicial do Portal DEL

V.6.1 CARACTERÍSTICAS CHAVE DO DELSKIN E DO PORTAL DEL

- Criação de estrutura do portal para suportar os mesmos dados da página anterior do DEL;
- Definição paleta de cores com as cores que representam o departamento;
- Criação de logomarca;
- Alteração dos portlets laterais para permitir um fluxo de navegação natural no sistema.

V.7 CRIAÇÃO DO CONTEÚDO INICIAL DO PORTAL DEL

No portal foram criadas seções para comportar os dados da página antiga além das adições do sistema para o gerenciamento de usuários e grupos da rede.

A estrutura do site pode ser verificada pelo link “mapa do site” encontrado na parte superior direita da página, como observado na Figura V.I. O mapa do site está ilustrado na Figura V.II.

V.8 SOPLUM DEL

Esse produto agrega os tipos de conteúdo específicos da estrutura organizacional do DEL. Tais tipos de conteúdo são a extensão das interfaces do núcleo do Soplum que fazem a ponte do sistema web com os serviços remotos. Esta implementação pode servir de base/exemplo para futuras instalações ou extensões.

O uso de tipos de conteúdo personalizados cria um aspecto de portal para o sistema, são adicionados dados pessoais como endereço, telefone, email. Além disso, são adicionados dados relacionados à instituição de ensino como: departamento, professores, alunos, disciplinas.

Figura V.2 – Estrutura de árvore do conteúdo do Portal DEL

V.8.1 CARACTERÍSTICAS CHAVE

Integração transparente com ambiente remoto através da extensão do Soplum Core

Entidades customizadas para uso específico do DEL

V.8.2 DICIONÁRIO DE ENTIDADES

- Aluno: entidade que representa a participação de uma pessoa como aluno no departamento;
- Disciplina: uma disciplina acadêmica ministrada no Departamento;

- Docente: entidade que representa a participação de uma pessoa como professor do departamento;
- Endereço: Localização geográfica do endereço de uma pessoa;
- Funcionário: entidade que representa a participação de uma pessoa como funcionário do departamento;
- Identidade: Documento de identificação de uma pessoa, expedita em algum órgão;
- Pasta de pessoas: Repositório agregador do tipo de conteúdo ‘Pessoa’;
- Situação Acadêmica: Representa a situação acadêmica de um aluno em um determinado período;
- Telefone: Número de contato.

V.9 IMPLANTAÇÃO E TESTES

O sistema foi instalado em um ambiente de homologação no mês de junho e foi testado até o mês de julho, quando foi transferido para o ambiente de produção na rede DEL. Alguns erros e inconformidade com os requisitos foram encontrados, principalmente no que diz respeito as diferenças nos arquivos de dados de usuários e grupos de rede entre o ambiente de desenvolvimento (Linux) e o ambiente de produção (FreeBSD). Tais erros foram prontamente corrigidos. O Núcleo do Soplum em sí não sofreu alterações em sua arquitetura, toda a base do sistema se mostrou estável e com um alto grau de confiabilidade. Os erros ocorreram em grande parte nas implementações dos serviços e na personalização para o DEL (SoplumDEL), a estrutura modular foi de grande valia neste aspecto.

O ambiente de produção tem cerca de duzentos cadastros e vem sendo usado pelos administradores para cadastrar os alunos do departamento.

Capítulo VI - Conclusão

O projeto foi em muitos aspectos, desafiador, seu tamanho se desenvolveu e mostrou-se muito maior do que o estimado. O fato de optarmos por uma arquitetura expansível acarretou em uma necessidade de empenho extra, tanto em código quanto em modelagem, para sempre em momentos de decisão escolher dentre as opções, aquela que levasse ao menor acoplamento possível entre os artefatos. O uso de padrões de projeto foi indiscutivelmente de grande valia no processo de desenvolvimento, visto que, a adoção de melhores práticas, diminui o esforço em curto prazo e fez com que o código seja mais manutenível. A construção de um projeto desta complexidade faz com que sejam consolidadas muitas técnicas e tal experiência supera qualquer tipo de aprendizado teórico, pois as dificuldades raramente são planejáveis.

Muitos dos conceitos aprendidos em Engenharia de Software puderam ser experimentados, pois mesmo utilizando uma metodologia Ágil, não se abre mão dos fundamentos básicos de construção e modelagem vistos nesta disciplina. A abordagem de "processo enxuto" do *Scrum* foi de grande importância, pois para os desenvolvedores que utilizam a mesma metodologia no ambiente de trabalho, a parte de análise de sistemas nunca se tornou um fardo e sim um momento descontraído de planejamento.

O limitante de execução do projeto foi o pouco tempo encontrado nos nossos dias para evoluir o projeto, onde tivemos que aprender ainda mais a gerenciar esse recurso tão escasso. O fato de o projeto ter sido realizado em dupla também exigiu o exercício de sermos educados com os contratos de chamada de função para cada biblioteca. As alterações não foram feitas a revelia do par. Como haviam dois desenvolvedores alterando concorrentemente a base de código, nada disso teria sido possível sem uma ferramenta de controle de versão.

A implementação do projeto foi concluída com sucesso e o ideal de se construir uma arquitetura extensível foi atingido, o sistema está funcional como o previsto e deixa-se uma porta aberta e um convite a outros alunos que desejarem contribuir para o aprimoramento deste projeto.

VI.1 Trabalhos futuros

Existe uma incrível quantidade de extensões que poderiam aprimorar ainda mais o pontapé inicial que representa este projeto, que foi modelado e implementado para suportar continuidade no seu desenvolvimento. Alguns dos exemplos de funcionalidades desejadas:

- Melhorias no sistema de disciplinas

Um sistema que poderia contar com ementas, grades horárias, pré-requisitos, um painel com visão geral dos períodos. Todos os dados sendo conteúdos para o CMS, para que fosse possível utilizá-los para fazer o controle de turmas e notas.

- Integração com o Sistema Integrado de Gestão Acadêmica (SIGA)

O Soplum poderia ainda obter as informações de matrícula e situação acadêmica dos alunos diretamente do SIGA através de uma integração utilizando *WebServices*.

- Sistema de pedido de relatórios utilizando fluxos de trabalho.

Isso poderia facilitar o pedido e a entrega de declarações, aliviando o trabalho dos funcionários e dos alunos.

- Sistema de acompanhamento e divulgação dos projetos integrados e dos projetos finais

Durante o período letivo o aluno poderia ter um espaço para documentar e acompanhar seus projetos acadêmicos. Na conclusão da matéria, o sistema poderia arquivar as páginas e deixá-las disponíveis num repositório para que o mundo pudesse acessar esses projetos, retribuindo à sociedade com o conhecimento desenvolvido no departamento. Além de servir como ponto de contato com outros acadêmicos.

Apêndice A – Acrônimos

ACID - *Atomicity, Consistency, Isolation, Durability*

AJAX – *Asynchronous JavaScript and XML*

ANSI – *American National Standards Institute*

API – *Application Programming Interface*

CASE – *Computer-Aided Software Engineering*

CIFS – *Common Internet File System*

CMM – *Capability Maturity Model for Software*

CMS – *Content Management System*

CSV - *Comma-Separated Values*

CSS – *Cascading StyleSheets*

BSD – *Berkley Software Distribution*

DAV – *Distributed Authoring and Versioning*

DEL – *Departamento de Engenharia Eletrônica e de Computação*

DRY – *Don't Repeat Yourself*

FTP – *File Transfer Protocol*

HTTP – *Hypertext Transfer Protocol*

HTTPS – *Hypertext Transfer Protocol Secure*

IEEE – *Institute of Electrical and Electronics Engineers*

IoC – *Inversion of Control*

IP – *Internet Protocol*

ISO – *International Standards Organization*

JSON – *JavaScript Object Notation*

JSR – *Java Specification Request*

NFS – *Network File Service*

NIS – *Network Information Service*

ORM – *Object-Relational Mapping*

PDF – *Portable Document Format*

POSIX - *Portable Operating System Interface*

PDU – *Protocol Data Unit*

PHP – *Pre Hypertext Processor*

RUP – *Rational Unified Process*

RPC – *Remote Procedure Call*

SERPRO – Serviço Federal de Processamento de Dados

SID – Sistema Integrado do DEL

SIGA – Sistema Integrado de Gestão Acadêmica

SOA – *Service-Oriented Architecture*

SOAP – *Simple Object Access Protocol*

SOPLUM – *Service Oriented Plone User Management*

SQL– *Structured Query Language*

SSL – *Secure Sockets Layer*

SVN – *Subversion*

TCP – *Transmission Control Protocol*

UFRJ – Universidade Federal do Rio de Janeiro

XML – *Extensive Markup Language*

YAGNI – *You Aren't Gonna Need It*

ZPT – *Zope Page Templates*

Apêndice B – Requisitos mínimos para o desenvolvimento do sistema

Para o desenvolvimento do sistema, basta atender os requisitos da execução do mesmo, tais requisitos estão listados no Apêndice C.

É desejável que se tenha conhecimento da linguagem Python e que seja utilizado um ambiente de desenvolvimento. A plataforma de desenvolvimento utilizada para criação do projeto está definida o Capítulo IV seção “Referências”.

Apêndice C – Requisitos mínimos para a instalação e uso do projeto

- Sistema operacional: Unix, Linux ou Windows.
- Requisitos de hardware: são dados pelo Zope/Plone/Twisted ou seja qualquer máquina que suporte esses softwares será suficiente para execução do Soplum.
- servidor de homologação tinha a seguinte configuração:
- CPU: Pentium III 800Mhz;
- RAM: 512 Mb;
- Espaço em disco: 2 Gb.

C.1 DEPENDÊNCIAS DE SOFTWARE DOS PACOTES DO SOPLUM

SoplumServer:

- Openssl – biblioteca de criptografia;
- py-openssl – biblioteca *wrapper* para a execução de funções da OpenSSL no Python;
- py-smbpasswd 1.0.1 – biblioteca para geração de senhas criptografadas pelo Samba;
- twisted 2.5 – biblioteca com as implementações dos servidores HTTP e XML-RPC, além dos serviços de daemon;
- pexpect – biblioteca que permite ao python fazer interação com o terminal;
- py-unixfilemanager – biblioteca para manipulação de arquivos de usuários e grupos dos seguintes domínios do Unix: passwd, shadow, group e netgroup;
- python 2.5 – implementação da máquina virtual Python utilizada pelo Twisted;

SoplumCore:

- Python 2.4 – implementação da máquina virtual Python utilizada pelo Zope e produtos do Plone;
- Plone 2.5 – CMS base da implementação do Soplum;
- Zope 2.9 – Servidor de aplicação de sistemas web;

SoplumCommons: biblioteca de funcionalidades básica utilizada pela suíte do Soplum, deve ser instalada tanto na máquina virtual Python 2.4 quanto 2.5;

SoplumConnectivity: biblioteca de conectividade com sistemas remotos utilizada pelo Plone, deve ser instalada na máquina virtual Python 2.4;

C.2 PRODUTOS PLONE E DEPENDÊNCIAS

SoplumCore:

- Membrane 1.0 – tratamento de membros como tipo de conteúdo.
- ATBackRef 0.1 – implementação de referências reversas entre conteúdos.

SoplumDEL:

- PopupCalendarWidget – controle amigável de calendário;
- Quota – implementação de pasta de conteúdos com controle de ocupação de espaço em disco;
- PloneLanguageTool – ferramenta de internacionalização de interface;
- LinguaPlone – ferramenta de internacionalização de conteúdo;

Apêndice D - Instalação

D.1 INSTALAÇÃO

As dependências dos pacotes do Soplum devem estar instaladas na máquina virtual Python correspondente conforme descrito no apêndice C.

Os pacotes do Soplum estão disponíveis do endereço web: http://soplum.sourceforge.net na seção de downloads.

Essa instalação supõe que todo o sistema está sendo executado na mesma máquina (serviços, servidor de aplicação, etc.). O Soplum suporta o funcionamento distribuído, porém este não será coberto por esse roteiro, que segue o modelo utilizado no DEL.

É necessário ter nível de permissão *root* para instalar todos os pontos visitados nesse roteiro.

D.1.1 INSTALAÇÃO DO SOPLUM COMMONS

Descompactar o pacote do soplum commons em uma pasta

Entrar na pasta e digitar:

```
#python2.4 setup.py install
```

Em seguida instalar na máquina virtual 2.5

```
#python2.5 setup.py install
```

D.1.2 INSTALAÇÃO DO SOPLUM CONNECTIVITY

Descompactar o pacote do soplum connectivity em uma pasta

Entrar na pasta e digitar:

```
#python2.4 setup.py install
```

D.1.3 PREPARAÇÃO DO SERVIDOR DE APLICAÇÃO E PORTAL PLONE

- Criar uma instância do Zope;
- Instale os produtos que compõem o Plone na instância recém criada;
- Inicialize o Zope e crie um Site Plone através da interface administrativa do Zope;

D.1.4 INSTALAÇÃO DO SOPLUM CORE

- Descompactar o pacote na pasta de produtos da instância do Zope onde estiver também a instalação do Plone;
- Reinicie o Zope;
- Acesse a interface de administração do portal Plone e ative os produtos do Soplum Core;

D.1.5 INSTALAÇÃO E CONFIGURAÇÃO DO SOPLUM SERVER

Descompactar o pacote do Soplum Commons em uma pasta;

Em seguida instalar na máquina virtual 2.5

```
#python2.5 install.py
```

Por padrão o instalador irá copiar os arquivos para a pasta:

`/opt/Soplum/SoplumServer`

A configuração padrão do servidor irá utilizar como pasta de instalação dos serviços a pasta:

`/opt/Soplum/Services`

Todos os pacotes de serviços deverão ser descompactados nesta pasta.

D.1.6 CONFIGURAÇÃO DOS SERVIÇOS NO SERVER

Para todos os serviços instalados devem-se configurar as instâncias ativas para os mesmos.

As instâncias de um serviço podem ser configuradas alterando o arquivo instances.cfg dentro do pacote do mesmo. Este arquivo possui uma seção denominada s*erver_instances* onde cada entrada (chave=valor) define uma instância do serviço. A chave é ignorada, devendo apenas ser única, no entanto o valor é utilizado pelo serviço para identificar o arquivo de configuração daquela instância. Por exemplo, para uma instância denominada "NISAccount", deve ser criado um arquivo de configuração denominado "NISAccout.cfg". Cada pacote de serviço inclui um arquivo de exemplo de configuração que pode ser utilizado como base.

As diretrizes de configuração de cada um dos serviços foram detalhadas no Capítulo V. O arquivo de configuração do servidor deve ser configurado se necessário, como demonstrado no Capítulo IV, o arquivo se encontra em:

`/opt/Soplum/SoplumServer/conf/SoplumServer.cfg`

D.1.7 CONFIGURAÇÃO DOS SERVIÇOS NO PLONE

Antes de configurar os serviços no Plone é preciso adicionar o servidor. Isto pode ser feito em qualquer lugar dentro de um site Plone, porém é recomendada a criação de uma pasta de administração da rede, dentro desta uma pasta para servidores, outra para serviços, etc. A estrutura sugerida pode ser vista na Figura D.1

Após acessar a subpasta "Servidores" pode-se adicionar um Servidor, os dados do servidor devem ser preenchidos no formulário de criação do conteúdo do Plone como demonstrado na Figura D.2.

Figura D.1 – Pasta "Administração da Rede"

Figura D.2 – Formulário de cadastro de um Servidor

A seguir, adicionamos um serviço ao servidor recém criado acionando o comando do "adicionar item" do menu como ilustrado na Figura D.3:

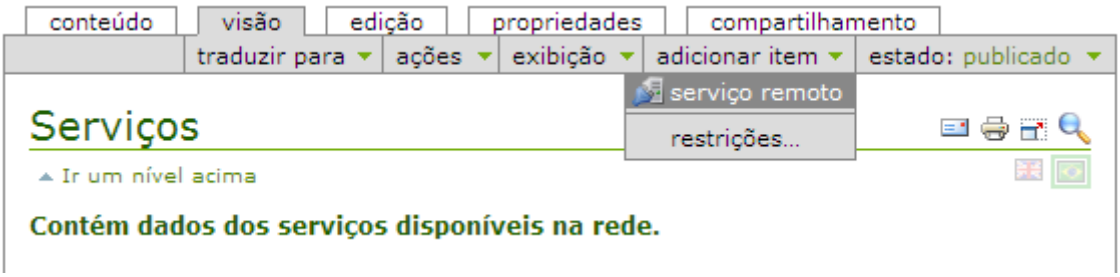


Figura D.3 – Comando do menu para cadastro de Serviço remote

Preenche-se o formulário mostrado na Figura D.4, com os dados do serviço, o nome da instância corresponde ao nome da instância configurada no serviço do tipo selecionado:

Figura D.4 – Formulário de cadastro de Serviço remoto

Para verificar se um serviço foi devidamente configurado, pode-se testar o funcionamento do mesmo através do botão “Verificar disponibilidade”, exibido na tela de visualização do conteúdo recém criado, essa ação pode ser observada na Figura D.5

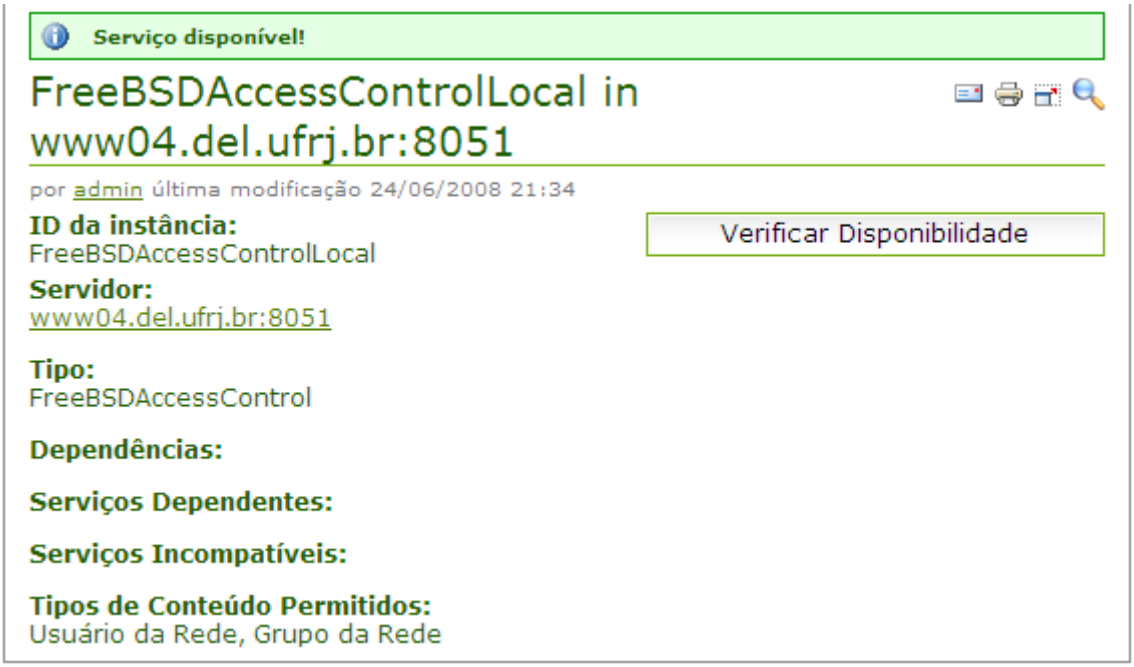


Figura D.5 – Mensagem de sucesso de disponibilidade do Serviço remoto

Quando um serviço não está configurado de maneira adequada, uma mensagem de erro amigável será exibida como mostra a figura a seguir, os detalhes do erro com mais informações de depuração podem ser encontrados no arquivo de log do servidor. Um exemplo pode ser observado na Figura D.6. As configurações de todos os serviços seguem este fluxo de criação, sendo um processo simples para um administrador de rede.

Figura D.6 – Mensagem indicando um Serviço indisponível

Apêndice E – Manual do Usuário

Por se tratar de um software em contínuo desenvolvimento, melhor mídia para a manutenção de um manual para um software desta natureza um hipertexto Web. Os roteiros guiados com as funcionalidades básicas do Soplum estão disponíveis na página do projeto, na seção de manual:

http://soplum.sourceforge.net/manual/index.html

Referências Bibliográficas

[1] Pedro da Fonseca Vieira. Sistema Integrado de Serviços e Informações do DEL (SID). 2002

[2] IEEE Standards Association. Recommended Practice for Software Requirements Specifications. IEEE Standards Association. http://standards.ieee.org/reading/ieee/std_public/description/se/830-1998_desc.html – último acesso 04/09/2008

[3] IEEE Standards Association . Recommended Practice for Software Design Descriptions. IEEE Standards Association. http://standards.ieee.org/reading/ieee/std_public/description/se/1016-1998_desc.html – último acesso 04/09/2008

[4] *Agile Alliance. Manifesto for Agile Software Development* – http://agilemanifesto.org

[5] *The Scrum Development Process* – http://www.mountaingoatsoftware.com/scrum/ – último acesso 04/09/2008

[6] Extreme Programming – http://www.extremeprogramming.org/ – último acesso 04/09/2008

[7] Kruchten, Philippe. What Is the Rational Unified Process. http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/jan01/WhatIstheRationalUnifiedProcessJan01.pdf – último acesso 04/09/2008

[8] Winer, Dave. XML-RPC Specification. XML-RPC Home Page. http://www.xmlrpc.com/spec/ – último acesso 04/09/2008

[9] XML-RPC Implementation list. XML-RPC Home Page http://www.xmlrpc.com/directory/1568/implementations – último acesso 04/09/2008

[10] *OpenSSL Project* – http://www.openssl.org/ – último acesso 04/09/2008

[11] *CMS Matrix* – http://www.cmsmatrix.org – último acesso 04/09/2008

[12] *Open Source Alternative for Enterprise Content Management* – http://www.alfresco.com/ – último acesso 04/09/2008

[13] Plone Documentation – http://plone.org/documentation – último acesso 04/09/2008

[14] Eclipse IDE – http://www.eclipse.org/ – último acesso 04/09/2008

[15] Subversion – http://subversion.tigris.org/ – último acesso 04/09/2008