SOCIEDADE BRASILEIRA DE INSTRUÇÃO
UNIVERSIDADE CANDIDO MENDES
INSTITUTO UNIVERSITÁRIO CANDIDO MENDES-CAMPOS
Credenciada por Decreto de 24/11/1997
Rua Anita Peçanha, 100 - Parque São Caetano / Cep: 28040-320 / Campos dos Goytacazes - RJ
Telefone / Fax : (0xx22) 2733-4100 http:www.ucam-campos.br

GABRIEL MANHÃES MONNERAT

FERRAMENTA PARA MANIPULAÇÃO DE DOCUMENTOS EM LARGA ESCALA

Campos dos Goytacazes - RJ
Junho - 2010

GABRIEL MANHÃES MONNERAT

FERRAMENTA PARA MANIPULAÇÃO DE DOCUMENTOS EM LARGA ESCALA

Monografia apresentada à Universidade Cândido Mendes como requisito obrigatório para a obtenção do grau de Bacharel em Ciências da Computação.

ORIENTADOR: Prof. D.Sc. Dalessandro Soares Vianna

Campos dos Goytacazes - RJ
Junho - 2010

FERRAMENTA PARA MANIPULAÇÃO DE DOCUMENTOS EM LARGA ESCALA

Monografia apresentada à Universidade Cândido Mendes como requisito obrigatório para a obtenção do grau de Bacharel em Ciências da Computação.

Aprovada em ____ de _____ de 2010.

BANCA EXAMINADORA

Prof. Dalessandro Soares Vianna - Orientador
Doutor em Informática na Universidade Federal Fluminense

Prof. Fernando Carvalho
Especialista em Producão e Sistemas / IFF Campus Campos
Instituro Federal de Educacao, Ciência e Tecnologia Fluminense / Campus Bom Jesus

---

Prof Fábio Duncan de Souza
Mestre em Pesquisa Operacional e Inteligência Computacional / UCAM-Campos
Instituto Federal de Educacao, Ciência e Tecnologia Fluminense / Campus Campos

Dedico este trabalho a meus pais, irmão, namorada e amigos por todo apoio e compreensão prestado durante essa importante fase da minha vida.

Agradecimentos

Resumo

Este trabalho desenvolve uma ferramenta que tem o objetivo de substituir uma já existente que com o uso excessivo e prolongado mostrou problemas graves de escalabilidade e confiabilidade, sendo necessário a criação de uma nova ferramenta para além de correção dos problemas, realizar uma busca por padrões mais consolidados e ter uma estrutura mais flexível. Neste trabalho são apresentados conceitos básicos que foram utilizados para o desenvolvimento desta ferramenta e, além disso, junto com a descrição da ferramenta foram apresentados soluções para os problemas já conhecidos. Para validação da ferramenta desenvolvida, esta foi aplicada em um estudo de caso simples para desmonstrar o uso em um ambiente próximo do real.

PALAVRAS-CHAVE: Internet - Serviço Web - Desenvolvimento de Software - Escalabilidade

Lista de Figuras

2.1 Arquitetura do UNO. Adaptado de: [Ramme 2010]. 19
2.2 *Scripts* Python Utilizando PyUno. Adaptado de: [Thomas 2008]. 20
2.3 *Scripts* Python como componente UNO. Adaptado de: [Thomas 2008]. . . 20
2.4 Exemplo de Arquivo XML. Fonte: [Santos 2007]. 21
2.5 Serviço Web com sistemas internos de plataformas diferentes, Adaptado de: [Newcomer 2002]. 23
2.6 Troca de mensagens utilizando XML-RPC, Adaptado de Fonte: [Stervinou 2003]. 25
2.7 Arquivo de configuração do Buildout. 27

3.1 Utilização da API *IApplication* para controlar o processo Xvfb. 30
3.2 Utilização da API *IApplication* para controlar o processo OpenOffice.org. . 30
3.3 Exemplo de utilização do objeto *Manager*. 31
3.4 Exemplo de uso do *FileSystemDocument*. 32
3.5 Exemplo de uso da interface *ILockable* no objeto OpenOffice. 34
3.6 Exemplo de utilização do *OOHandler*. 35
3.7 Propriedades de um filtro. 36
3.8 Exemplo de um filtro do OpenOffice.org. 36
3.9 Importando todos filtros do OpenOffice.org para o *Mimemapper*. 37
3.10 Exemplo de criação de um objeto *Filter*. 38

4.1 Conexão do cliente com servidor. 40
4.2 *Download* do Buildout para instalação das dependências do ERP5. Fonte: [Honoré 2010]. 41
4.3 Comando para instalar dependências. Fonte: [Honoré 2010]. 41
4.4 Comando para instalar configurar o Buildout com as dependências instaladas no sistema operacional. Fonte: [Honoré 2010]. 42
4.5 *Download* do Buildout para instalação de uma instância do ERP5. Fonte: [Honoré 2010]. 42
4.6 Arquivo de configuração "my_config.cfg". Fonte: [Honoré 2010]. 42
4.7 Comandos que configuram as dependências no novo Buildout. Fonte: [Honoré 2010]. 43
4.8 Comando para iniciar todos aplicativos. Fonte: [Honoré 2010]. 43
4.9 Arquivo de configuração para criação da intância ERP5. Fonte: [Honoré 2010]. 43
4.10 Comando para criar instância ERP5. Fonte: [Honoré 2010]. 44

4.11 Comando para iniciar instância ERP5. Fonte: [Honoré 2010]. 44
4.12 Opção para acessar as preferências do ERP5. 44
4.13 Página de prefêrencias da instância ERP5. 45
4.14 Arquivo de configuração da instância. 45
4.15 Opção para acessar a página de configuração dos *Business Templates*. . . . 46
4.16 Botão para importar ou exportar *Business Templates*. 46
4.17 Página para definir o endereço do repositório a ser utilizado. 47
4.18 Lista de *Business Templates* instaláveis. Fonte: [Smets 2010]. 47
4.19 Lista de *Business Templates* instaláveis. Fonte: [Smets 2010]. 48
4.20 Trecho de código que representa um dos métodos de compatibilidade entre aplicações. 48
4.21 Página Principal do ERP5. 49
4.22 Opção para criar um objeto documento no ERP5. 49
4.23 *Upload* de um documento. 50
4.24 Pré-visualização de um documento. 50
4.25 Definição do metódo que irá prover o serviço. 53
4.26 Método que inicia a aplicação e retorno do objeto que o cliente recebe quando se conecta. 53
4.27 Amostra do arquivo de configuração. 54
4.28 Iniciando a aplicação OOOD com *Paste*. 54

Lista de Tabelas

2.1 *Tabelas de formatos OpenDocument* 16

Sumário

1 Introdução **12**

2 Conceitos básicos **15**
2.1 *OpenDocument Format* . . . 15
2.1.1 Estrutura de um *OpenDocument* . . . 16
2.1.2 Formatos *OpenDocument* . . . 16
2.2 OpenOffice.org . . . 17
2.3 UNO . . . 18
2.3.1 PyUNO . . . 19
2.4 XML . . . 20
2.5 WSGI . . . 21
2.6 *Web Service* . . . 22
2.7 XML-RPC . . . 24
2.8 ERP . . . 25
2.9 ERP5 . . . 26
2.9.1 *Business Templates* . . . 26
2.10 Buildout . . . 26
2.11 Subversion . . . 27

3 OOOD v2.0 **28**
3.1 *IApplication* . . . 29
3.1.1 OpenOffice e Xvfb . . . 29
3.2 *IManager* . . . 30
3.2.1 *Manager* . . . 31
3.3 *IDocument* . . . 31
3.3.1 *FileSystemDocument* . . . 31
3.4 *IMonitor* . . . 32
3.4.1 *MemoryMonitor* . . . 32
3.4.2 *TimeoutMonitor* . . . 33
3.4.3 *RequestMonitor* . . . 33
3.5 *ILockable* . . . 33
3.5.1 OpenOffice . . . 34
3.6 *IHandler* . . . 34

3.6.1 *OOHandler* . . . 34
3.7 *IMimemapper* . . . 35
3.7.1 Filtros do OpenOffice.org . . . 35
3.7.2 *Mimemapper* . . . 37
3.8 *IFilter* . . . 37
3.8.1 *Filter* . . . 38

4 Estudo de Casos **39**
4.1 Ambiente de desenvolvimento . . . 39
4.2 Instalação do OOOD 2.0 . . . 39
4.3 Todos os passos de uma resposta ao cliente . . . 40
4.4 Instalação do ERP5 . . . 41
4.4.1 Configurando ERP5 para utilizar o OOOD . . . 44
4.4.2 Instalação do *Business Template* erp5_dms . . . 45
4.5 Compatibilidade entre o ERP5 e o OOOD 2.0 . . . 47
4.6 Usando o ERP5 para converter documentos . . . 47
4.7 Problemas e Decisões de Projeto . . . 49
4.7.1 Vazamento de memória . . . 50
4.7.2 Armazenamento e busca dos filtros . . . 51
4.7.3 Arquivos compactados . . . 52
4.8 Explicando o funcionamento do prestador de serviço com *Paste* . . . 53
4.9 Comparando a *perfomance* entre a OOOD 2.0 e o OOOD 1.0 . . . 54
4.10 Resultados obtidos . . . 55

5 Conclusão **56**
5.1 Objetivos alcançados . . . 56
5.2 Trabalhos futuros . . . 56

Capítulo 1

Introdução

A Internet, uma das grandes revoluções tecnológicas da comunicação dos últimos dez anos, deve essa popularidade ao avanço na tecnologia Web, que introduziu uma interface gráfica intuitiva baseada em *hiperlinks*, possibilitando a fácil utilização dos serviços disponibilizados via internet por usuários domésticos e/ou corporativos. Este crescente volume de computadores conectados a rede altera profundamente o modo de desenvolvimento dos aplicativos, obrigando corporações a introduzirem novas tecnologias que suportem a grande demanda. Logo, com o crescimento tanto de usuários quanto de aplicações via rede de computadores, novas dificuldades surgiram com a extensão das aplicações, como maior complexidade de manutenção, a dificuldade de inserir novas funcionalidades e a curva de aprendizado acentuada dos desenvolvedores. Visando minimizar estas dificuldades, surge a necessidade de serviços descentralizados, mais simples e com uma confiança maior para desenvolvimento de softwares em larga escala.

Com a descentralização das informações, o desenvolvimento de sistemas simples e escaláveis fez com que surgissem cada vez mais serviços Web que tivessem somente uma responsabilidade e pudessem se comunicar com outros aplicativos via internet ou intranet. Portanto, serviços simples, de fácil manutenção e extensão, proporcionam o aumento da perfomace e escalabilidade, eliminando o uso de um serviço único com todo volume de informações para serem processadas constantemente.

A iniciativa entre a empresa francesa Nexedi SA(empresa situada em Lille, França) e o NSI(Núcleo de pesquisa em Sistema de Informação) do Instituto Federal Fluminense-Campos de desenvolverem uma ferramenta Web de código aberto que subsitui-se uma já existente, chamada OpenOffice.org Daemon (oood), desenvolvida originalmente pela Nexedi SA, é um exemplo da necessidade do uso de novas tecnologias para suportar novas demandas e requisitos.

A primeira versão do oood foi desenvolvida em 2006 pela Nexedi com o objetivo inicial de converter documentos do tipo Office. A partir do uso prolongado em ambientes de produção foram identificados erros ou tratamentos inadequados de exceções que ocasionaram problemas como: perda de requisições, *deadlock* no OpenOffice.org e em processos, *memory leak*, entre outros. Logo, surgiu a necessidade de corrigir o serviço para torná-lo mais estável com um bom desempenho e escalável em forma de cluster. A análise preliminar do oood determinou que seria inviável a manutenção do mesmo, pois havia demanda por novos requisitos e a necessidade de corrigir os problemas mencionados acima.

Com a necessidade da correção dos problemas identificados e o uso de resursos mais atuais e estáveis, surge a motivação para o desenvolvimento de uma nova ferramenta que tivesse como funcionalidades básicas converter documentos para a base ODF (*Open Document Format*), exportar documentos para outras extensões, adicionar e extrair metadados de um documento nos formatos suportados pelo OpenOffice.org. Outros fatores que também influenciaram na iniciativa de uma nova ferramenta foi a busca por uma estrutura mais flexível, de fácil manutenção, extensível e escalável.

Portanto, este trabalho tem como objetivo desenvolver uma ferramenta Web que utilize de recursos computacionais para manipulação de documentos, de forma que seja possível extrair e adicionar informações e fazer conversões destes documentos para qualquer formato suportado pelo Openoffice.org. Além disso, apresentar soluções para os problemas identificados, tais como *deadlock* no OpenOffice.org, perda de requisições e falta de escalabilidade.

No segundo capítulo são apresentados conceitos básicos para melhor entendimento das tecnologias e padrões utilizados durante o trabalho. No terceiro capítulo, é explicado sepa-

radamente cada componente desenvolvido com o intuito de apresentar no quarto capítulo estudos de casos descrevendo em detalhes como a junção destes componentes formam um serviço Web e o deste em um ambiente próximo do real.

Por fim, o quinto capítulo apresenta os resultados obtidos com o estudo de casos e a conclusão obtida a partir do desenvolvimento desta aplicação.

Capítulo 2

Conceitos básicos

Nas seções seguintes serão apresentados conceitos, tecnologias, padrões e ferramentas que são essenciais para o entendimento deste trabalho.

2.1 *OpenDocument Format*

O *OpenDocument Format*(ODF), forma abreviada de *OASIS OpenDocument Format for Office Applications*, é um conjunto de formatos de arquivos para aplicações de escritório(edição de texto, planilhas, apresentações de slides, banco de dados, manipulação de imagens, entre outros) desenvolvido com a proposta de oferecer um padrão aberto que pode ser adotado por qualquer pessoa ou instituição.

Esse padrão foi desenvolvido pelo consórcio OASIS e é baseado no formato XML. O acesso a este padrão é público, o que possibilita a implementação em qualquer sistema, seja ele de código aberto ou não. Segundo [Santos 2007], a intenção do formato ODF é prover uma alternativa aos formatos proprietários de documentação, para que não fiquem aprisionados a um único fornecedor.

Nas subseções seguintes será apresentada a estrutura deste formato e os formatos de documentos para cada tipo de documento.

2.1.1 Estrutura de um *OpenDocument*

Por ser em código aberto, é possível ter acesso a todos os dados de um documento, pois são arquivo compactados e utilizam arquivos XML para estruturar seus dados [Eisenberg 2005]. Então, quando um documento em ODF é criado, um conjunto de arquivos e pastas o constitui. Os principais são:

- mimetype - Arquivo de linha única com o *mimetype* do documento;
- content.xml - Arquivo que armazena o conteúdo do documento criado pelo usuário;
- meta.xml - Armazena os "metadados"do documento, ou seja, informações como autor, data de modificação, quantidade de palavras, etc;
- styles.xml - Contém estilos para o documento, por exemplo, formatações específicas para parágrafos e listas;
- Pictures - Pasta que armazena as imagens do documento;

2.1.2 Formatos *OpenDocument*

Para todo tipo de documento, como por exemplo texto, existe um formato base para este tipo. Os formatos para cada tipo são apresentados na Tabela 2.1.

Tabela 2.1: *Tabelas de formatos OpenDocument*

odt	Documento de Texto
ods	Panilha Eletrônica
odp	Documento de Apresentação
odg	Desenho Vetorial
odf	Equações
odb	Banco de Dados
odj	Documento Mestre

Na seção seguinte será apresentado um software de manipulação de documentos que têm como padrão o ODF.

2.2 OpenOffice.org

Inicialmente o OpenOffice.org era chamado de StarOffice. A Sun MicroSystems comprou este na versão 5.1, em 1999, da empresa alemã StarDivison. Em agosto de 1999, a Sun disponibilizou gratuitamente a versão 5.2 do StarOffice e em meados de 2000 anunciou a liberação do código fonte para *download* sob as licenças *Lesser General Public License*(LGPL) e *Sun Industry Standards Source License*(SISSL) com o intuito de criar uma alternativa de baixo custo, código aberto e de alta qualidade. LGPL e SISSL impõem que o código aberto desenvolvido esteja disponível em forma de biblioteca, mas não exige que o mesmo seja aplicado a outros softwares que empreguem seu código [Wikipedia 2010]. No mesmo ano da liberação da versão 5.2 do StarOffice, o novo projeto recebe o nome de OpenOffice.org e no Brasil de BrOffice.org.

Segundo [Santos 2007], OpenOffice.org é uma suíte de aplicativos multiplataforma, sendo distribuída para sistemas operacionais como Linux, Microsoft Windows, Solaris e Mac OS X. A suíte usa os formatos ODF e é compatível com formatos da Microsoff Office. No caso de documentos que não estão no formato ODF, estes arquivos são convertidos para uma extensão de mesmo tipo que seja ODF. O OpenOffice.org divide estas extensões em tipos, ou seja, cada documento tem o seu próprio tipo e só pode ser convertido para outra extensão que faça parte do seu grupo de tipos. As extensões nas suítes são dividas em 5 tipos onde cada tipo tem a sua extensão padrão. Por exemplo, um documento do tipo texto é convertido para o formato ODT quando carregado no aplicativo OpenOffice.org. Com essa padronização, documentos criados e suportados pelo aplicativo são compatíveis com instalações do mesmo software em outros sistemas operacionais.

A seção seguinte apresenta o UNO que é um mecanismo utilizado para comunicação entre linguagens de programação e o OpenOffice.org.

2.3 UNO

UNO ou *Universal Network Object* é um modelo de componentes do OpenOffice.org, que oferece interoperabilidade entre linguagens de programação e o modelo de objetos do OpenOffice.org, ou seja, componentes UNO podem ser implementados e acessados a partir de uma linguagem de programação [Ramme 2010]. Estes componentes têm recursos idênticos de uma aplicação OpenOffice.org aberta em um sistema operacional, como por exemplo abrir um documento em um formato específico e salvá-lo em outro formato suportável.

Para entender melhor como é o funcionamento do UNO, que tem uma idéia diferente de programação comparada à programação tradicional, é necessário distinguir entre especificação e implementação. Em aplicações normais, as especificações são declaradas em documentos na linguagem natural. Em contrapartida, UNO declara seus métodos e propriedades em uma *Application Programming Interface*(API) chamada IDL. IDL é uma linguagem de computador utilizada para descrever interfaces de componentes de softwares. Esta linguagem de programação é independente, com isso possibilita a comunicação de linguagens diferentes. Ou seja, na IDL não existe nenhuma implementação de código, existe somente a declaração e descrição dos métodos. A implementação é realizada na linguagem de programação que possui biblioteca para acessar os recursos do UNO, como demonstra a Figura 2.1 a separação da especifição e linguagens que possuem pontes para se comunicar com o UNO.

Com a IDL tem-se a fácil comunicação entre uma linguagem de programação e o UNO, seja intranet ou internet. Quando se está com um aplicativo da suíte OpenOffice.org aberto, tem-se por trás de todo os recursos uma utilização constante de componentes UNO de acordo com o uso do aplicativo. Lembrando que, mesmo o UNO sendo usado internamente, é através de uma linguagem de programação que é feita a comunicação entre OpenOffice.org e UNO.

A sessão seguinte apresenta uma biblioteca implementada em Python para utilizar o UNO nesta linguagem.

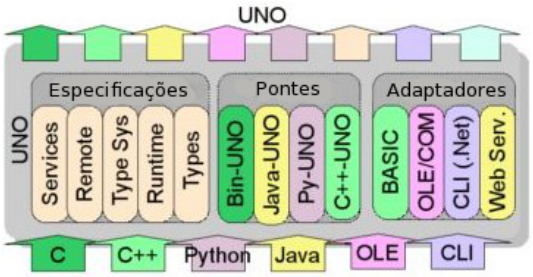

Figura 2.1: Arquitetura do UNO. Adaptado de: [Ramme 2010].

2.3.1 PyUNO

A biblioteca PyUNO permite que toda a API do OpenOffice.org seja utilizada na linguagem de programação Python de formas diferentes, como por exemplo utilizar *scripts* executáveis em Python e através do *framework* de *scripts* do OpenOffice.org. A biblioteca não contém métodos que acessam o UNO, visto que toda implementação do UNO é carregada no Python [Thomas e Budischewski 2008]. Isto faz com que todo uso dos componentes seja no Python, toda especificação esteja na IDL e toda implementação no próprio UNO.

A subseções seguintes apresentam formas de utilização desta biblioteca.

2.3.1.1 Python *Scripts*

Scripts em Python são executáveis que se conectam ao OpenOffice.org utilizando o UNO. De todas maneiras possíveis, a utilização de *scripts* é a forma mais lenta de uso, pois toda transferência é feita via conexão de rede. Como são implementados em Python, os *scripts* são independentes do OpenOffice.org. A Figura 2.2 mostra como é executado um Python *script*.

Analisando a Figura 2.2 percebe-se que parte do código em Python é executado dentro do ambiente do PyUNO. Estes códigos são chamadas à API do UNO para criação dos componentes e a outra parte do código faz uso ou não destes objetos. Um exemplo de uso dos componentes do UNO é a criação do objeto que faz conexão ao processo OpenOffice.org.

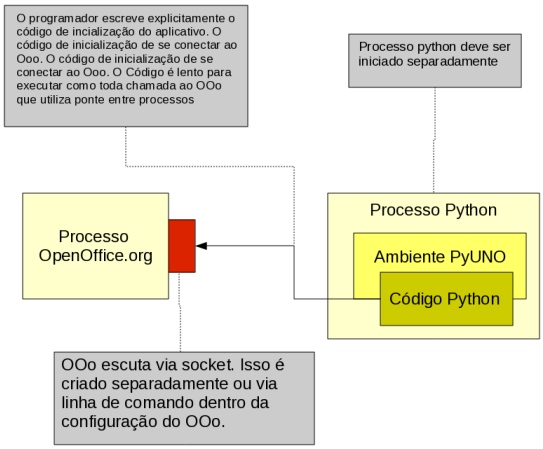

Figura 2.2: *Scripts* Python Utilizando PyUno. Adaptado de: [Thomas 2008].

2.3.1.2 *Script* Python como componente UNO

Desta forma, os Python *scripts* são implementados para serem inseridos dentro do OpenOffice.org, com isso há um ganho de desempenho pois não há necessidade de fazer conexão entre o processo OpenOffice.org e o Python *script*. A Figura 2.3 mostra como o componente trabalha dentro de um processo OpenOffice.org.

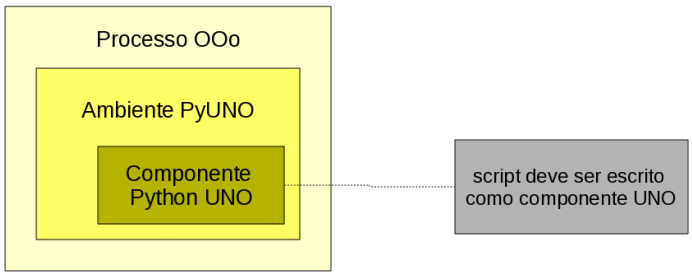

Figura 2.3: *Scripts* Python como componente UNO. Adaptado de: [Thomas 2008].

2.4 XML

Segundo, Dykes e Tittel 2005, XML é uma linguagem de marcação que utiliza *tags* para rotular, categorizar e organizar as informações de maneira específica. A marcação descreve o documento ou a organização do mesmo. O conteúdo, como textos, imagens e dados, são partes do código que as *tags* de marcação contêm. Além disso, o XML não está limitado a

um conjunto especifíco de marcações, estas podem ser criadas para atender às necessidades, como exemplo a Figura 2.4 demonstra a criação de novas *tags* de marcação para representar alunos.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<alunos>
        <aluno>
                <nome>João da Silva</nome>
                <email>joaodasilva@ygh.com</email>
                <nascimento>16/12/1989</nascimento>
                <turma>1M</turma>
                <serie>1 EM</serie>
        </aluno>
</alunos>
```

Figura 2.4: Exemplo de Arquivo XML. Fonte: [Santos 2007].

Pode ser notado na Figura 2.4, a própria estrutura e declaração das *tags* do XML declaram o conteúdo do documento. Isto facilita a leitura pelas aplicações na hora de abrirem os arquivos XML e converterem para linguagem de programação.

2.5 WSGI

Historicamente o *Web Service Gateway Interface*(WSGI) foi proposto em 2003 por Philip J. Eby e programadores Python. Esta proposta, segundo [Eby 2010], tinha objetivo de criar um caminho fácil de integrar diferentes aplicações Web e obter uma única aplicação final.

Python possui atualmente uma variedade de *frameworks* para construção de aplicações Web, tais como Zope, Django, Twisted Web, entre outros. Essa variedade tem um lado positivo onde é possível escolher um *framework* que atenda melhor as necessidades atuais do desenvolvedor. Mas por outro lado, a escolha do *framework Web* irá limitar a seleção de serviços Web utilizáveis, e vice-versa.

Em contrapartida, embora a linguagem de programação Java tenha também uma va-

riedade de *frameworks* disponíveis para construção de aplicações Web, esta tem a Java Servlet API que torna possível que aplicações desenvolvidas em qualquer *framework Web* Java rode em qualquer servidor Web que suporta a API do *servlet*.

Com a possibilidade de qualquer servidor Web rodar qualquer aplicação Web, tem-se o desenvolvimento independente da escolha do servidor Web. Além disso, deixa livre os desenvolvedores para escolherem o que melhor os atenderem e se concentrar em sua área de especialização, deixando para outro desenvolvedor a responsabilidade de escolher qual servidor Web utilizar.

Em suma, esta proposta tenta resgatar a independência de aplicações e servidores Web como o Java Servlet API, para que facilite a escolha da tecnologia a ser utilizada tanto no desenvolvimento da aplicação quanto na ferramenta servidora.

2.6 *Web Service*

Existem diferentes maneiras de definir um *Web Service*. Booth et al. 2004 define *Web Service* como:

> *Um software projetado para suportar interações interoperáveis entre máquinas sobre uma rede. Tem uma interface descrita em um formato processável por máquina(especificamente WSDL). Outros sistemas interagem com este serviço Web de uma maneira prescrita usando mensagens SOAP, tipicamente transmitida usando HTTP com serialização XML em conjunto com outras normas relacionadas à Web.*

Simple Object Acess Protocol(SOAP), citado por [Booth et al. 2004], é um protocolo baseado em XML para troca de informações entre computadores, tendo seu foco principal a chamada de procedimentos remotos transportados via HTTP [Cerami 2002]. Portanto, permitem que aplicativos cliente possam facilmente conectar-se a serviços remotos e invocar metódos.

Web Service Description Language(WSDL), também citado por [Booth et al. 2004], é um formato XML para descrever serviços de rede como um conjunto de parâmetros operacionais sobre as mensagens que contenham qualquer informação ou documento orientado para processo de orientação. As operações e mensagens são descritas abstratamente e estão ligado a um protocolo de rede concreto e formato de mensagem para definir um ponto de extremidade [Christensen et al. 2001].

Em outras palavras, é uma solução utilizada na integração de sistemas e na comunicação entre aplicações de diferentes arquiteturas [Alonso et al. 2004], como demonstra a Figura 2.5.

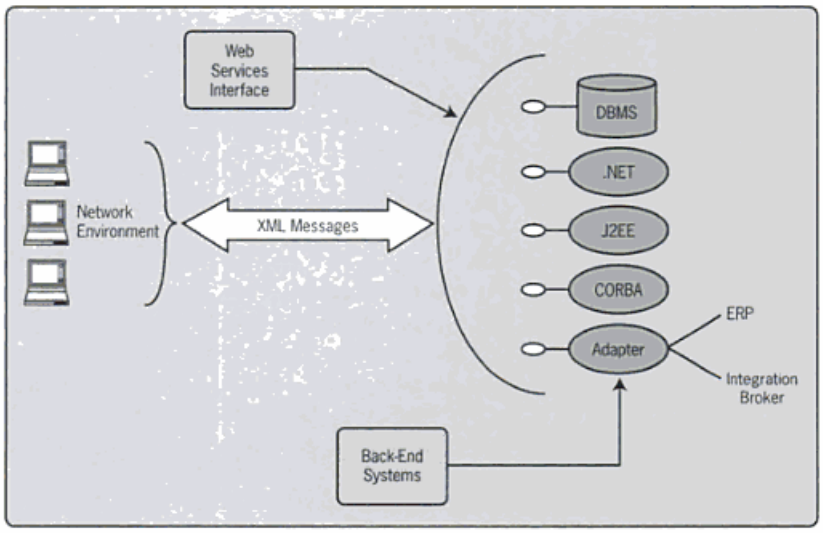

Figura 2.5: Serviço Web com sistemas internos de plataformas diferentes, Adaptado de: [Newcomer 2002].

Note que na Figura 2.5, os clientes enxergam a aplicação como um sistema único e utilizam o mesmo protocolo de comunicação. Com isso, há uma independência tanto do cliente quanto do servidor, permitindo que estes exerçam o seu papel da melhor maneira possível.

Segundo [Cerami 2002] e [Balani 2009], existem três papéis principais dentro da arquitetura de serviços Web:

- Prestador de Serviço - Fornece os serviços através da Web e responde os pedidos dos

consumidores;

- Consumidor de Serviço - Utiliza uma rede existente através da abertura de uma conexão de rede e envia uma solicitação em XML. Em serviços Web baseados em SOAP, o prestador de serviço publica o contrato(WSDL) do serviço através da internet e o consumidor pode acessá-lo diretamente ou buscar o registro de serviço;

- Registro de Serviço - Diretório lógico centralizado de serviços onde os desenvolvedores podem publicar ou encontrar novos serviços já existentes, portanto, serve como uma agência centralizada para as empresas e seus serviços.

Com esta arquitetura flexível, um *Web Service* pode ser um caminho para expansão dos negócios de uma empresa, com o objetivo de aumentar a eficiência do processo de negócio e sua experiência com o cliente, já que este não está limitado à automatização dos processos [Singh et al. 2004]. Além disso, pode simplificar as interações com serviços externos, tais como validação do cartão de crédito ou compra com o mesmo. Como resultado, são oferecidos aos clientes uma experiência enriquecida, com mais opções de escolha e mais flexibilidade.

Com isso, cada *Web service* é responsável por um processo, ou seja, cada aplicação tem a sua responsabilidade. Por exemplo um sistema que controla usuário e senha em um caixa eletrônico, é responsável somente autenticar usuários. Logo, tendo poucas atribuições ao serviço faz-se o uso do mesmo com mais rapidez e clareza.

2.7 XML-RPC

XML-RPC, segundo [Laurent, Dumbill e Johnston 2001], é um dos métodos mais simples de serviços Web, e que torna fácil computadores chamarem procedimentos de outros computadores. Este método permite que programas façam chamadas de função ou procedimento em uma rede utilizando o protocolo *Hypertext Transfer Protocol*(HTTP) para transferir informações de um computador cliente a um computador servidor. É utilizado a linguagem

XML tanto para fazer requisições pelo cliente quanto para responder a ele. A Figura 2.6 apresenta como é a comunicação e a transferência das informações entre cliente e servidor.

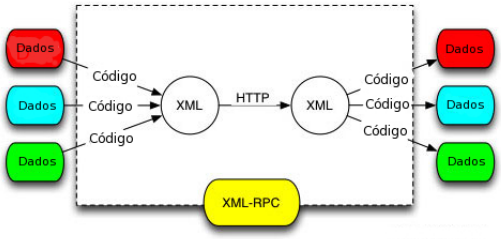

Figura 2.6: Troca de mensagens utilizando XML-RPC, Adaptado de Fonte: [Stervinou 2003].

Em uma arquitetura cliente-servidor, todos procedimentos são requisitados pelo cliente e executados pelo servidor. Para que isto seja possível utiliza-se uma das tecnologias de comunicação entre processos, a *Remote Procedure Call*(RPC). RPC permite a um programa de computador chamar um procedimento no mesmo computador ou em outro conectado por uma rede. No caso do XML-RPC todas as chamadas são feitas através de uma rede de computadores mas para o RPC não existe diferença entre o procedimento estar na mesma máquina ou em outro computador.

2.8 ERP

Enteprise Resource Planning(ERP) é um sistema de informação usado para gerenciar os recursos internos e externos de uma corporação [Wikipedia 2010]. Sua arquitetura tem como objetivo a facilidade de gestão do fluxo de informações, sendo este gerado pela integração dos dados e processos de vários ou mesmo da totalidade dos departamentos da organização num único sistema. Esta integração pode ser efetuada ao nível departamental ou funcional (sistemas finanças, marketing, comercial, pessoal, produção, etc.) ou ao nível processual(sistema de tratamento de encomendas, sistema de informação de gestão, sistema de apoio à decisão, etc.) [Norris, Hurley e Hartley 2000].

Resumidamente, segundo [Norris, Hurley e Hartley 2000], ERP é uma tecnologia de su-

porte de software que forma um núcleo de processamento de transações, tendo sua aplicação em várias áreas.

2.9 ERP5

Para [Carvalho e Campos 2007], o ERP5 é um ERP de código aberto que visa oferecer uma solução de alta tecnologia e baixo custo, sendo desenvolvida pela Nexedi, criadora do software, e instituições de pesquisa de países da Europa e do Brasil. Este software é desenvolvido na plataforma Zope, escrito na linguagem de programação Python. O Zope é um servidor de aplicações *Web Open Source* desenvolvido em Python [Lutz 2006] que oferece um banco de dados orientado a objeto(ZODB) e um máquina de estado(DC Workflow). Com isso, além de utilizar os recursos oferecidos pela plataforma, o ERP5 incorpora a sicronização entre diferentes bancos de dados orientado a objeto e um mecanismo de mapeamento de banco de dados relacional que indexa atributos de cada objeto em um banco de dados relacional [Ramachandran e Carvalho 2009].

2.9.1 *Business Templates*

Business Templates é a utilização de componentes existentes no ERP5 para adicionar novas funcionalidades ao núcleo do ERP5. Em outras palavras, a partir da reutilização dos recursos, campos e *scripts*, é possível criar modelos de negócio utilizando componentes já existentes [Oram e Wilson 2007].

2.10 Buildout

Buildout é uma ferramenta, desenvolvida em Python, que fornece suporte para criação de aplicações, seja ela desenvolvida em Python ou em outra linguagem de programação [Fulton 2010]. Entre vários mecanismos de criação e instalação de aplicações que o Buildout utiliza, os *recipes* são os mais utilizados. *Recipes* são *scripts* Python que executam algum

procedimento específico e possuem todas as informações do Buildout que está executando-o. A Figura 2.7 é um exemplo de um arquivo de configuração do Buildout.

```
[buildout]
parts = executar_comando

[executar_comando]
recipe = plone.recipe.command
command = echo "Imprime na Tela"
```

Figura 2.7: Arquivo de configuração do Buildout.

Na organização do arquivo de configuração, existem dois atributos importantes no buildout que são: *parts*, *recipe*. O *parts* é utilizado para declarar qual procedimento será chamado. Dentro da declaração do procedimento tem o atributo *recipe* que por sua vez é o caminho do script que executará todo procedimento. Por fim, na Figura 2.7 temos além da declaração dos atributos essenciais, temos um atributo *command*. Este atributo será utilizado pelo *recipe* que foi implementado para executar o que estiver dentro do *command*.

Em suma, o Buildout é extremamente flexível e simples para criação de ambientes de desenvolvimento específicos e enxutos.

2.11 Subversion

Segundo [Pilato 2004], Subversion é um sistema de controle de versão *open-source*. Sistemas de controle de versão são utilizados em ambientes de desenvolvimento de software para controlar diferentes versões do código-fonte e possibilitar o trabalho em equipe. Para que isto ocorra, é fornecido pelo sistema um local centralizado para armazenamentos dos arquivos do projeto e sua revisões [Shane e Shore 2007]. Com isso, cada desenvolvedor pode utilizar uma revisão específica sem que outros desenvolvedores sejam afetados.

Capítulo 3

OOOD v2.0

O OOOD 2.0 é um serviço Web desenvolvido neste trabalho que utiliza o protocolo XML-RPC para troca de mensagens. Esta aplicação, desenvolvida em Python, provê a extração e inserção de metadados em um documento e conversão do mesmo para qualquer formato suportado pelo OpenOffice.org. É um projeto *Open Source* que se encontra sob a licença LGPL.

Cada parte da aplicação pode ser utilizada separadamente ou em outra aplicação. Estas partes são dividas em oito interfaces:

- *IManager* é a declaração de todos os métodos que o usuário pode utilizar;
- *IDocument* representa cada documento que o servidor recebe;
- *IHandler* representa objetos que irão fazer todo trabalho que é pedido pela requisição de um cliente;
- *ILockable* define métodos para controle de uma determinada região que não pode ser acessada por mais de um cliente. Isto é utilizado em casos de serviços com grande demanda que necessitam ter controle do uso dos recursos disponibilizados;
- *IMonitor* define métodos para controle e manuseio dos processos;

- *IMimemapper* fornece métodos para manusear todos filtros existentes no OpenOffice.org;
- *IApplication* define métodos para controle de aplicativos externos ao serviço Web;
- *IFilter* representa cada filtro do *software* OpenOffice.org;

Cada uma dessas interfaces e suas implementações são explicadas nas subseções seguintes.

3.1 *IApplication*

Para um melhor controle dos processos externos ao OOOD 2.0 é necessário que métodos possam não só iniciar e parar o processo, mas também saber a qualquer momento se o mesmo está rodando no sistema operacional, o seu identificador e qual porta do sistema ele está usando. No caso do OOOD 2.0 também é necessário que a configuração do processo seja dinâmica. Para isso, além dos métodos básicos de controle de um processo, foi inserido o método *loadSettings* para que seja possível configurar o processo da melhor maneira.

3.1.1 OpenOffice e Xvfb

Com a necessidade de um aplicativo *desktop* em um ambiente de produção, sendo na maioria dos casos servidores que não possuem interface gráfica instalada, é necessário que nesses ambientes o aplicativo fique sempre aberto e funcional. Para que isso seja possível, estes ambientes de produção exigem uma interface gráfica para que o OpenOffice.org fique aberto e possibilite que a partir somente do identificador da interface, seja possível abrir qualquer aplicativo nesta interface gráfica em memória.

Portanto, a classe Xvfb é responsável por controlar a interface gráfica em memória e a classe OpenOffice utiliza este para abrir o aplicativo. As Figuras 3.1 e 3.2 apresentam exemplos de uso de ambos os objetos.

```
1  # -*- coding: utf-8 -*-
2  from oood.application.xvfb import xvfb
3  xvfb.loadSettings(hostname='localhost', port=6099,
4  »       »       path_run_dir='/tmp',
5                  display_id='99')
6  xvfb.start()
7  xvfb.status() # Retorna True
8  xvfb.stop()
```

Figura 3.1: Utilização da API *IApplication* para controlar o processo Xvfb.

```
1  # -*- coding: utf-8 -*-
2  from oood.application.openoffice import openoffice
3  openoffice.loadSettings(hostname='localhost', port=4060,
4  »       »       path_run_dir='/tmp',
5                  display_id='99',
6                  office_bin_path='/opt/openoffice.org3/program',
7                  uno_path='/opt/openoffice.org3/basis-link/program')
8  openoffice.start()
9  openoffice.status() # Retorna True
10 openoffice.stop()
```

Figura 3.2: Utilização da API *IApplication* para controlar o processo OpenOffice.org.

Nota-se que, na Figura 3.2 existem dois parâmetros de configuração declarados na linha 6 e 7, que não constam na Figura 3.1. O identificador da interface gráfica, com nome de *display_id*, em ambos os exemplos são iguais. Essas diferenças de controle entre o processo Xvfb e OpenOffice.org, são justificadas pelo fato de que o OpenOffice precisa do identificador da interface gráfica para iniciar o aplicativo OpenOffice.org e além disso, o caminho do binário do mesmo e da bibliteca UNO.

3.2 *IManager*

A interface *IManager* é o contrato entre cliente e servidor. Com o objetivo de um contrato mais genérico, foram declarados métodos que futuramente pudessem trabalhar com outros tipos de dados além de documentos. Então, caso a aplicação Web seja estendida para trabalhar, por exemplo, com conversão de vídeos e imagens, o cliente não é afetado.

3.2.1 *Manager*

O classe *Manager* é responsável por criar tarefas enviadas pelo cliente. A Figura 3.3 apresenta um exemplo da criação de uma tarefa simulando um cliente.

```
1  # -*- coding: utf-8 -*-
2  from oood.manager import Manager
3  from base64 import encodestring
4  arquivo = open("/tmp/test.odt").read()
5  cliente = Manager("/tmp")
6  cliente.convertFile(encodestring(arquivo), 'odt', 'pdf')
```

Figura 3.3: Exemplo de utilização do objeto *Manager*.

Note que nesta mesma figura, uma biblioteca de codificação é utilizada, chamada *base64*. Esta biblioteca tem como objetivo codificar os dados para transferência pela internet, pois a rede de computadores não suporta o tráfego de documentos em seu estado original.

Como os documentos recebidos pelo objeto *Manager* estão codificados em *base64*, antes de serem salvos no sistema de arquivo, são descodificados para retornar ao estado original. Além disso, os documentos retornados para o usuário também são codificados para serem enviados via rede.

3.3 *IDocument*

No serviço de aplicação OOOD, a prioridade é uma resposta eficiente e consistente ao cliente. Para que isso seja possível é necessário que haja integridade total nos dados enviados para o servidor. Com estes requisitos, a *IDocument* propõe um contrato em que é possível saber o contéudo do documento, atualizá-lo e em casos de problemas em tempo real, recuperar o documento original.

3.3.1 *FileSystemDocument*

A classe *FileSystemDocument* utiliza a própria estrutura de pastas do sistema operacional como forma de armazenar os dados. Os arquivos são postos em pastas temporárias que são

deletadas quando o objeto não é mais utilizado. Desta forma, cada pasta representa uma requisição do cliente. A Figura 3.4 demonstra como utilizar este objeto para manipular documentos no sistema de arquivo.

```
1  # -*- coding: utf-8 -*-
2  >>> from oood.document import FileSystemDocument
3  >>> arquivo = open("/tmp/teste.odt").read()
4  >>> documento = FileSystemDocument('/tmp', arquivo, 'odt')
5  >>> documento.getContent()
6  >>> documento.getUrl()
7  >>> documento.restoreOriginal()
```

Figura 3.4: Exemplo de uso do *FileSystemDocument*.

3.4 *IMonitor*

Com a dificuldade de controle dos processos e os constantes erros que afetavam toda a aplicação, a interface *IMonitor* definiu um contrato que, com atributos mínimos, torna possível monitorar o processo e obter informações à partir disso. Com isso, é possível criar monitores que fiscalizam uma parte da aplicação e caso haja algum resultado não esperado, o problema é corrigido.

A subseções seguintes apresentam monitores implementados para controlar um tipo de problema. Portanto, em cada subseção será mostrado o problema e a solução para o mesmo.

3.4.1 *MemoryMonitor*

Com o uso excessivo da conexão entre o OpenOffice.org e o UNO, em muitos casos, a memória utilizada para troca de mensagens não era liberada. Esta falha é a causa de um fenômeno, chamado de vazamento de memória ou *memory leak*, que ocorre em sistemas computacionais quando uma porção de memória alocada para uma determinada operação, não é liberada quando não é mais necessária [Gookin 2005]. A consequência disto é o uso de toda memória do computador.

Para o controle deste problema, a classe *MemoryMonitor* é responsável por fiscalizar a memória utilizada pelo OpenOffice.org e caso esta memória aumente de forma descontrolada o processo é parado imediatamente e reiniciado. Em suma, esta solução foi adicionada pois o sistema operacional não é programado para detectar essas falhas, sendo este erro acusado somente quando toda memória do computador é utilizada.

3.4.2 *TimeoutMonitor*

Com o uso excessivo do OpenOffice.org foi detectado que existem momentos que o mesmo não consegue abrir um documento, pois está travado. Então, a classe *TimeoutMonitor* tem como objetivo limitar o tempo de uso do OpenOffice.org para não permitir que a requisição do cliente fique esperando a resposta por tempo indeterminado. Caso o uso do OpenOffice.org extrapole o tempo máximo, o OpenOffice.org é reiniciado e a requisição é enviada novamente.

3.4.3 *RequestMonitor*

Com o comprometimento da estabilidade do aplicativo à partir de muitas requisições respondidas pelo OpenOffice.org, foi implementado a classe *RequestMonitor*, que reinicia o OpenOffice.org quando o número de requisições respondidas chega ao limite.

3.5 *ILockable*

A exclusão mútua é uma técnica de programação chamada programação concorrente. Esta técnica tem como objetivo impedir que processos ou *threads* tenham acesso a um recurso compartilhado, garantindo a consistência e integridade dos dados [Chandra et al. 2001]. Este trecho ou recurso compartilhado é chamado de região crítica. Portanto, a interface *ILockable* propõe métodos para controlar a região crítica.

3.5.1 OpenOffice

Além de implementar a interface *IApplication*, descrita na sessão 3.1, a classe OpenOffice necessita controlar o uso do aplicativo OpenOffice.org, pois este não pode atender mais de uma requisição ao mesmo tempo. Sendo assim, para que haja controle no uso do aplicativo, a classe OpenOffice implementa esta interface, como demonstra a Figura 3.5 o uso de métodos da *ILockable* para controlar o objeto.

```
1  # -*- coding: utf-8 -*-
2  >>> from oood.application.openoffice import openoffice
3  >>> openoffice.isLocked()
4  False
5  >>> openoffice.acquire()
6  >>> openoffice.isLocked()
7  True
8  >>> openoffice.release()
9  >>> openoffice.isLocked()
10 False
```

Figura 3.5: Exemplo de uso da interface *ILockable* no objeto OpenOffice.

3.6 *IHandler*

Com a necessidade de uma arquitetura flexível, além de fazer com que cada objeto tenha somente uma responsabilidade [Pilone e Miles 2007], foi criada uma interface que é o contrato para objetos responsáveis pela conversão de dados, como por exemplo documentos, vídeos e imagens.

3.6.1 *OOHandler*

A classe *OOHandler* é implementada para comunicar-se com o OpenOffice.org. Quando o serviço OOOD recebe uma requisição, ele automaticamente delega estes dados para o *OOHandler* e requista ao mesmo objeto algum procedimento, por exemplo extrair metadados. Além disso, para armazenar e manusear os dados enviados pelo cliente, o *OOHandler* possui um objeto *FileSystemDocument*, pois caso ocorra falhas, estes dados podem ser res-

taurados. A Figura 3.6 demonstra um exemplo de utilização do *OOHandler* e o documento com um objeto *FileSystemDocument*.

```
1  # -*- coding: utf-8 -*-
2  >>> data = open("test.doc").read()
3  >>> handler = OOHandler("/tmp",
4                          data,
5                         'doc')
6  >>> handler.document
7  <oood.document.FileSystemDocument object at 0xb6e4a54c>
8  >>> doc_exportado = handler.convert("odt")
```

Figura 3.6: Exemplo de utilização do *OOHandler*.

3.7 *IMimemapper*

Antes do documento ser aberto pelo OpenOffice.org, existe um pré-processamento para saber o formato do arquivo. Este pré-processamento é necessário, pois para cada extensão existe um filtro específico.

Em um sistema *desktop* comum, os nomes dos arquivos incluem uma extensão que indica o formato do arquivo, por exemplo "document.doc". Antes de abrí-lo na suíte de aplicativos do OpenOffice.org, a suíte detecta a extensão pelo nome do arquivo e através disto abre o arquivo com o filtro referente ao formato. Em casos que o nome do arquivo não define a extensão, o OpenOffice.org tenta descobrir o formato, caso contrário ocorre uma falha dizendo que o arquivo não pode ser aberto pelo aplicativo.

Portanto, com o objetivo de criar um ambiente parecido com um sistema *desktop*, o *IMimemapper* provê métodos para obter todas informações necessárias para manipular um documento antes mesmo de carregá-lo no OpenOffice.org.

3.7.1 Filtros do OpenOffice.org

O aplicativo OpenOffice.org utiliza a linguagem XML para armazenar filtros, extensões e tipos de arquivos. Numa instalação normal, o OpenOffice.org armazena todos esses dados

dentro da pasta *TypeDetection* que localiza-se dentro da estrutura de pastas da instalação. Por exemplo, ao utilizar o pacote oficial de instalação, o caminho completo desta pasta seria "/opt/openoffice.org/basis3.1/share/registry/modules/org/openoffice/TypeDetection".

Dentro da pasta *TypeDetection* existem duas pastas, que são *Filter* e *Types*. A pasta *Filter* armazena os filtros e o serviço utilizado para exportar um documento utilizando este filtro e *Types* armazena as propriedades de um filtro. As Figuras 3.7 e 3.8, demonstram, respectivamente, um exemplo de filtro e tipo declarado no OpenOffice.org. Estes exemplos serão utilizados para explicação dos dados relevantes usados para o OOOD 2.0.

```
 5 »       <node oor:name="writer_web_HTML" oor:op="replace" >
 6 »       »       <prop oor:name="DetectService"><value>com.sun.star.text.FormatDetector</value></prop>
 7         <prop oor:name="URLPattern"><value>private:factory/swriter/web*</value></prop>
 8 »       »       <prop oor:name="Extensions"><value>html htm</value></prop>
 9 »       »       <prop oor:name="MediaType"><value>text/html</value></prop>
10 »       »       <prop oor:name="Preferred"><value>false</value></prop>
11 »       »       <prop oor:name="PreferredFilter"><value>HTML</value></prop>
12 »       »       <prop oor:name="UIName">
13 »       »       »       <value>HTML Document</value>
14 »       »       </prop>
15 »       »       <prop oor:name="ClipboardFormat"/>
16 »       </node>
```

Figura 3.7: Propriedades de um filtro.

```
 5 »       <node oor:name="HTML" oor:op="replace">
 6 »       »       <prop oor:name="Flags"><value>IMPORT EXPORT ASYNCHRON PREFERRED</value></prop>
 7 »       »       <prop oor:name="UIComponent"/>
 8 »       »       <prop oor:name="FilterService"/>
 9 »       »       <prop oor:name="UserData"><value>HTML</value></prop>
10 »       »       <prop oor:name="FileFormatVersion"><value>0</value></prop>
11 »       »       <prop oor:name="Type"><value>writer_web_HTML</value></prop>
12 »       »       <prop oor:name="TemplateName"/>
13 »       »       <prop oor:name="DocumentService"><value>com.sun.star.text.WebDocument</value></prop>
14 »       </node>
15
```

Figura 3.8: Exemplo de um filtro do OpenOffice.org.

Os dados apresentados nas figuras 3.10 e 3.8, são dados que analisando-os é possível saber qual extensão terá o arquivo caso seja exportado utilizando este filtro.

A ligação entre filtro e tipo é feita pela propriedade *Type* declarado na linha 11 da Figura 3.8 e o nome do nó, declarado na linha 5 da Figura 3.10. Então, com esta associção,

utilizando o filtro HTML o documento terá formato “htm” ou “html”, pois estas extensões estão declarados na propriedade *Extensions*, situado na linha 8 da Figura 3.10.

3.7.2 *Mimemapper*

O UNO possui metódos que possibilitam extrair qualquer tipo de informação do OpenOffice.org. A partir disto, a classe *Mimemapper* é responsável por utilizar estes metódos para obter todos os dados necessários para importar e/ou exportar documentos. A Figura 3.9 apresenta como os filtros são carregados para o *Mimemapper*.

Com os filtros carregados, estes são armazenados em objetos do tipo *Filter*, descrito na sessão 3.8.1, com o objetivo de tornar fácil o acesso as informações e oferecer uma maior flexibilidade na busca. Por exemplo, na linha 5 da Figura 3.9, a partir de informações sobre um determinado formato de arquivo, é possível saber qual nome do filtro é utilizado para importar ou exportar um documento no mesmo formato.

```
1  # -*- coding: utf-8 -*-
2  >>> from oood.mimemapper import mimemapper
3  >>> mimemapper.loadFilterList("localhost", 4060) # Carrega todos filtros
4
5  >>> mimemapper.getFilterName("pdf", 'com.sun.star.text.TextDocument')
6  "writer_pdf_Export"
7  >>> mimemapper.getMimetypeByFilterType("writer8")
8  "application/vnd.oasis.opendocument.text"
9  >>> mimemapper.getAllowedExtensionList(document_type='web')
```

Figura 3.9: Importando todos filtros do OpenOffice.org para o *Mimemapper*.

3.8 *IFilter*

Quando o *Mimemapper*, descrito na sessão 3.7.2, extrai os filtros do OpenOffice.org, cada um contêm informações que serão utilizadas durante todo o uso da aplicação, mas esses dados precisam ser armanazenados de forma segura e de fácil manuseio. Para isso criou-se

um contrato para que cada filtro seja um objeto e tenha métodos que possam extrair suas informações da melhor forma possível.

3.8.1 *Filter*

O *Filter* é a classe implementada mais simples de toda aplicação. Seu propósito é armazenar todas informações que um filtro do OpenOffice.org possui. A Figura 3.10 é um exemplo de como criar um filtro e extrair informações do mesmo através do métodos declarados na interface *IFilter*.

```
Python 2.6.4 (r264:75706, Jan  8 2010, 18:50:31)
>>> extensao = 'pdf'
>>> nome = 'writer_pdf_Export'
>>> mimetype = 'application/pdf'
>>> tipo_de_documento = 'text'
>>> preferido = True
>>> relevancia = 1000
>>> filter = Filter(extensao, nome,
                    mimetype,
                    tipo_de_documento,
                    preferred=preferido,
                    sort_index=relevancia)
>>> filter.isPreferred()
True
>>> filter.getName()
'writer_pdf_Export'
```

Figura 3.10: Exemplo de criação de um objeto *Filter*.

Capítulo 4

Estudo de Casos

Este capítulo apresenta um estudo de caso do OOOD 2.0 e sua implantação no ERP5. O estudo de caso foi idealizado para mostrar a instalação, configuração e uso do OOOD no ERP5. Além disso, foi realizado uma comparação entre o OOOD 2.0 e a ferramenta anterior.

4.1 Ambiente de desenvolvimento

Para desenvolvimento foi utilizado uma máquina com processador AMD Turion X2 1.6, 2 GB de RAM e 160 GB de HD. O sistema operacional utilizado foi o Mandriva na versão 2010.0, o mesmo que o ERP5 é desenvolvido.

Como pré-requisito de instalação foi necessário ter instalado no sistema operacional o Xvfb e o Python na versão 2.6, pois o OpenOffice.org é instalado junto com a aplicação.

4.2 Instalação do OOOD 2.0

Para instalação de toda estrutura, é utilizado o Buildout. Este mecanismo de instalação criará um ambiente com todos os pré-requisitos instalados dentro da própria estrutura do Buildout, exceto bibliotecas e aplicativos do sistema operacional. No sistema operacional é necessário que tenha instalado o Xvfb e um Python na versão 2.6 e o pacote *Subversion*.

O procedimento de instalação, após a instalação de todas dependências do sistema operacional, foi fazer download através do repositório SVN `https://svn.erp5.org/repos/experimental/oood.buildout`. Este Buildout, foi programado para fazer download do OOOD, que se encontra no repositório SVN `http://svn.erp5.org/repos/experimental/oood`. Após isto, acessou-se a pasta que foi criada em seu sistema operacional e executar o arquivo bootstrap.py dentro da pasta bootstrap com o Python 2.6. Com a execução do bootstrap.py, toda estrutura daquela pasta estará ligada diretamente ao python que foi executado. Para ser feita a instalação de todos pacotes necessários e o OpenOffice.org é preciso executar o comando "bin/buildout" dentro da mesma pasta.

Na execução deste último comando, além de todas bibliotecas, foi instalado também um OpenOffice.org, versão 3.2, dentro da mesma estrutura. Esta solução foi criada para facilitar a instalação, além de unificar e organizar toda estrutura em um só lugar.

4.3 Todos os passos de uma resposta ao cliente

Para um cliente se conectar ao serviço OOOD, a biblioteca utilizada para comunicação entre o cliente e o servidor é a *xmlrpclib*. A Figura 4.1 demonstra um exemplo de como se conectar ao serviço utilizando a linguagem de programação Python, realizando o envio de uma requisição para o servidor.

```
1 >>> import xmlrpclib
2 >>> conexao_com_servidor = xmlrpclib.ServerProxy("http://localhost:8008")
3
4 >>> conexao_com_servidor.convertFile(documento, 'odt', 'pdf')
```

Figura 4.1: Conexão do cliente com servidor.

As etapas para conversão de um documento são:

- O servidor recebe os dados do cliente e estes são inseridos no objeto instanciado *OOHandler*;

- Quando o *OOHandler* recebe os dados, um objeto *FileSystemDocument* é criado para salvar estes dados;
- Para conversão dos dados, o *OOHandler* bloqueia o objeto OpenOffice para uso, caso este esteja liberado. Caso contrário a requisição espera a liberação do objeto OpenOffice;
- Quando a conversão do documento é finalizada, o objeto OpenOffice é desbloqueado;
- Por fim, os dados convertidos são enviados para o cliente e a requisição é toda excluída inclusive os dados armazenados pelo *FileSystemDocument*;

4.4 Instalação do ERP5

Utilizando o pacote *Subversion*, foi realizado *download* do Buildout para instalação das dependências do ERP5. A Figura 4.2 demonstra como fazer *download* do Buildout.

```
$> svn co https://svn.erp5.org/repos/public/erp5/trunk/buildout/ erp5-software
```

Figura 4.2: *Download* do Buildout para instalação das dependências do ERP5. Fonte: [Honoré 2010].

Antes do *download*, foi necessário aceitar um certificado para que os arquivos sejam copiados para a máquina local. Para aceitar este certificado, foi selecionado a opção "permanente" para que isto não seja perguntado novamente.

Com os arquivo copiados na máquina local, o próximo passo é instalar todas dependências através de um *script* que está dentro da pasta "erp5-software", criada pelo passo anterior. O comando de execução deste *script* pode ver visto na Figura 4.3.

```
$> sudo erp5-software/helpers/mandriva2010.0.sh
```

Figura 4.3: Comando para instalar dependências. Fonte: [Honoré 2010].

Devidamente instaladas todas dependências, o Buildout é executado para instalação dos *softwares*, como por exemplo MySQL, Flare, OOOD. A Figura 4.4 demonstra como executar este comando.

```
$> cd erp5-software
$> make
```

Figura 4.4: Comando para instalar configurar o Buildout com as dependências instaladas no sistema operacional. Fonte: [Honoré 2010].

Terminada a execução do Buildout, outra estrutura será utilizada para instalação de uma instância do ERP5. Esta separação é aconselhada, pois caso seja necessário criar uma outra instância, já exite um ambiente com todas as dependências instaladas. Então, para instalar a instância, utiliza-se novamente o Buildout, como demonstra a Figura 4.5.

```
$> cd ~
$> svn co https://svn.erp5.org/repos/public/erp5/trunk/buildout/ erp5-data
```

Figura 4.5: *Download* do Buildout para instalação de uma instância do ERP5. Fonte: [Honoré 2010].

Novamente após o *download* do Buildout, uma pasta será criada localmente mas com o nome de "erp5-data". Nesta pasta são utilizados arquivos de configurações já predefinidos para instalação e configuração da instância do ERP5. Para iniciar a configuração do Buildout foi criado um arquivo com a configuração apresentada na Figura 4.6, nome "my_config.cfg" e salvo dentro da pasta "erp5-data".

```
[buildout]
extends = profiles/deployment.cfg

[software_definition]
software_home = /home/USER/erp5-software
```

Figura 4.6: Arquivo de configuração "my_config.cfg". Fonte: [Honoré 2010].

Note que na Figura 4.6, possui o nome *USER* no campo "caminho da pasta". Este nome deve ser substituido pelo nome do usuário da máquina.

Com este arquivo de configuração, esta instância foi configurada para utilizar todas dependências instaladas no Buildout criado anteriormente. Com isso, não será necessário instalar todos *softwares* novamente, pois esta instância estará utilizando todos recursos do Buildout instalado na pasta "erp5-software". A Figura 4.7 apresenta os comandos para configuração desta instância.

```
$> cd erp5-data
$> /home/USER/erp5-software/bin/python2.4 bootstrap/bootstrap.py
$> bin/buildout -c my_config.cfg
```

Figura 4.7: Comandos que configuram as dependências no novo Buildout. Fonte: [Honoré 2010].

Após isto, a instância já está configurada com os *softwares*, que são utilizados pelo ERP5, como por exemplo o OOOD. Com o comando apresentado na Figura 4.8, todos estes aplicativos são iniciados.

```
$> var/bin/supervisord
```

Figura 4.8: Comando para iniciar todos aplicativos. Fonte: [Honoré 2010].

Com as dependências todas iniciadas, como por exemplo MySQL e OOOD, o próximo passo é a criação da configuração com nome de "development.cfg" para criação da instância do ERP5 instalada, desmonstrado na Figura 4.9, e executar o Buildout com este arquivo, apresentado na Figura 4.10.

```
[buildout]
extends = my_config.cfg profiles/development.cfg
parts += development-site
[software_definition]
software_home = /home/USER/erp5-software
```

Figura 4.9: Arquivo de configuração para criação da intância ERP5. Fonte: [Honoré 2010].

Por fim, o comando apresentado na Figura 4.11 inicia a instância. Como padrão dos arquivos de configuração, a *url* para acessá-la o ERP5 é `http://localhost:18080/erp5` com usuário "zope" e senha "zope".

```
$> bin/buildout -c development.cfg
```

Figura 4.10: Comando para criar instância ERP5. Fonte: [Honoré 2010].

```
$> var/development-site/bin/zopectl start
```

Figura 4.11: Comando para iniciar instância ERP5. Fonte: [Honoré 2010].

4.4.1 Configurando ERP5 para utilizar o OOOD

Após a instalação de toda estrutura do ERP5, foi necessário configurar o OOOD na instância instalada. Acessando a página principal do ERP5, a criação do arquivo de configuração foi realizada pela aba esquerda da mesma página, selecionando a opção "Preferences", como mostra a Figura 4.12. Quando esta opção foi selecionada, a página de preferências do ERP5 foi aberta.

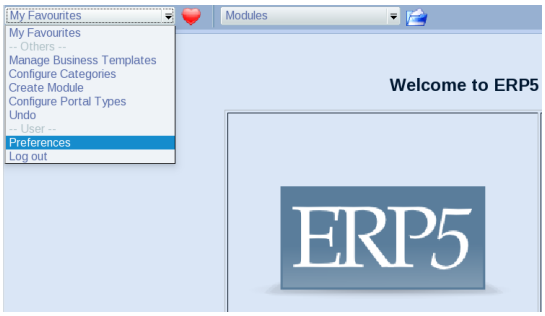

Figura 4.12: Opção para acessar as preferências do ERP5.

Após isto, selecionou-se a aba com nome de "Action" na página de prefências e a opção "Add System Preference", como demonstra a Figura 4.13. Um arquivo de configuração foi criado para que as configurações sejam definidas. A Figura 4.14 apresenta o arquivo de configuração criado.

Com o arquivo de configuração criado, o próximo passo foi definir o endereço do serviço OOOD nos campos "Conversion Server Address" para o endereço via internet e "Conversion Server Port" para declarar qual porta de acesso na máquina. Por fim, foi salvo as alterações

selecionando a ícone igual a um disquete no canto superior direito da página e para utiliza-la, selecionou-se a opção “Enable Preference” no campo “Action”.

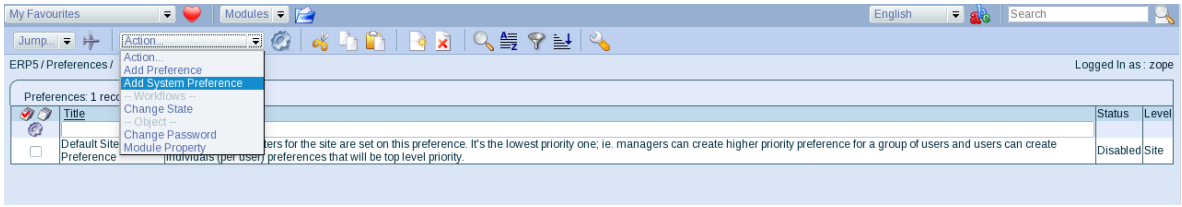

Figura 4.13: Página de preferências da instância ERP5.

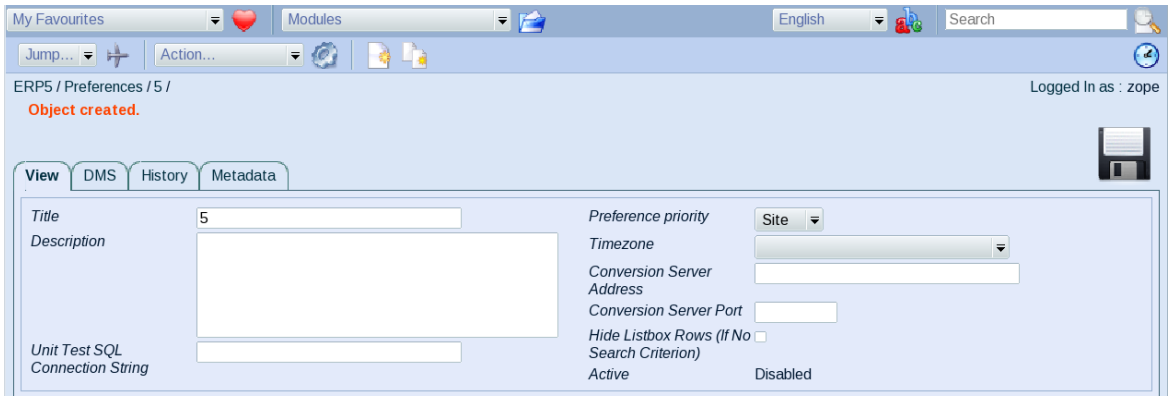

Figura 4.14: Arquivo de configuração da instância.

4.4.2 Instalação do *Business Template* erp5_dms

Para que fosse possível inserir documentos no ERP5 foi necessário utilizar o *Business Template*(BT5) erp5_dms. Para instalação deste pacote acessou-se a principal do ERP e selecionou-se a opção “Manage Business Templates” no campo de nome “My Favourites”, situado no canto esquerdo da mesma página, demonstrado na Figura 4.15.

Após isto, foi selecionado o botão “Import Export”, indicado pela seta na Figura 4.16, para que a página de atualização dos *Business Templates* fosse aberta. No campo ”Repositories”definiu-se o repositório `http://www.erp5.org/dists/snapshot/bt5`. A Figura 4.17 demonstra a página de atualização e o campo a ser preenchido.

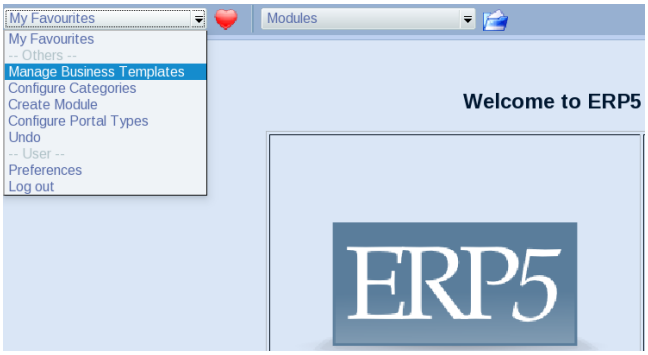

Figura 4.15: Opção para acessar a página de configuração dos *Business Templates*.

Figura 4.16: Botão para importar ou exportar *Business Templates*.

Definindo o repositório a ser utilizado, clicou-se em "Update Repository Information" para salvar a *url* adicionada e, após isto, selecionou-se a opção "Install Business Templates from Repositories" no campo "Select Exchange" para que a lista de BT5 fosse exibida como demonstra a Figura 4.18.

Na página de listagem dos *Business Templates*, selecionou-se os seguintes pacotes para instalação do erp5_dms:

- erp5_base
- erp5_dms;
- erp5_web;
- erp5_ingestion;
- erp5_ingestion_mysql_innodb_catalog;

Feito a seleção dos pacotes, clicou-se em "Install Business Templates from Repositories" no final da página e uma janela de validação de todos arquivos instaladas foi exibida, como demonstra a Figura 4.19. Para validar a instalação clicou-se em "Validate Installation" para que todos os pacotes fosse instalados.

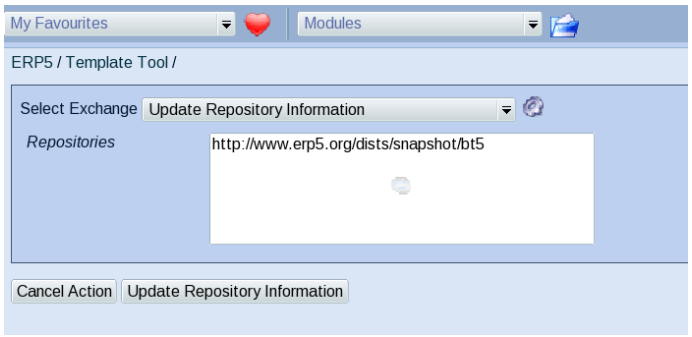

Figura 4.17: Página para definir o endereço do repositório a ser utilizado.

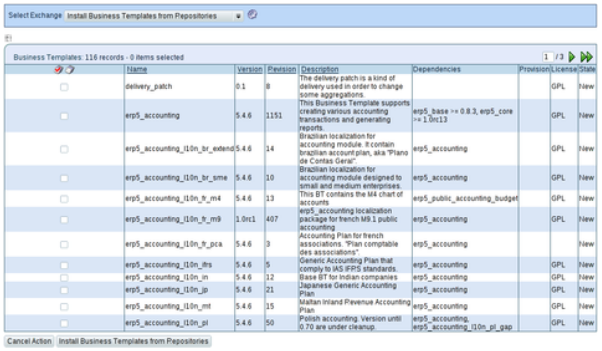

Figura 4.18: Lista de *Business Templates* instaláveis. Fonte: [Smets 2010].

4.5 Compatibilidade entre o ERP5 e o OOOD 2.0

Como o ERP5 possui muitos ambientes em produção, o custo para migrar todos estes ambientes para uma nova API seria muito grande. De tal forma, foi implementado uma interface de compatibilidade entre a aplicação Web e o ERP para redução de custo de migração para a nova API. Com isso, haverá a diminuição de falhas em ambientes de produção. Por exemplo, o metódo apresentado na Figura 4.20 recebe os parâmetros da mesma forma que são enviados pelo ERP5, chama o metódo da nova API e retorna os dados como é esperado pelo ERP5.

4.6 Usando o ERP5 para converter documentos

O OOOD é utilizado dentro do ERP5 para converter e extrair metadados de documentos. Para enviar um documento para o ERP5 clicou-se no *link* “Documents” na página principal

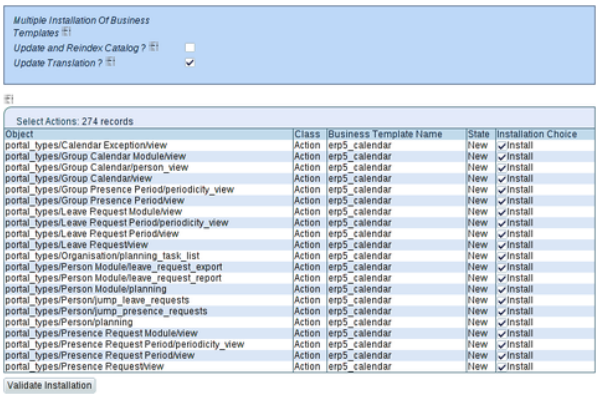

Figura 4.19: Lista de *Business Templates* instaláveis. Fonte: [Smets 2010].

```
136   def run_convert(self, filename, file):
137     """Method to support the old API. Wrapper getFileMetadataItemList but
142     orig_format = filename.split('.')[-1]
143     try:
144       response_dict = {}
145       response_dict['meta'] = self.getFileMetadataItemList(file,
146                                                            orig_format,
147                                                            True)
148       response_dict['data'] = response_dict['meta']['Data']
149       del response_dict['meta']['Data']
150       return (200, response_dict, "")
151     except Exception, e:
152       logger.error(e)
153       return (402, {}, e.args[0])
```

Figura 4.20: Trecho de código que representa um dos métodos de compatibilidade entre aplicações.

do ERP, como demonstra a Figura 4.21.

Após isto, na aba “Action” selecionou-se a opção “Add Text”, demonstrado na Figura 4.22. Na página seguinte foi inserido o documento e o nome para este, como mostra a Figura 4.23. Note que nesta mesma figura, quando o arquivo foi inserido e salvo, o estado deste arquivo é alterado de “Empty”, depois para “Converting” e por fim “Converted”. Isto significa que o documento foi enviado para extrair os metados e convertido para a base ODF.

Para este documento ser vizualizado através do ERP5, clicou-se na aba “Preview”, como demonstra a Figura 4.24. Quando a opção de vizualização foi selecionada, o documento foi convertido para o formato html para que seja possível vizualizá-lo no ERP.

Além disso, em documentos em formatos para apresentação é possível através da aba

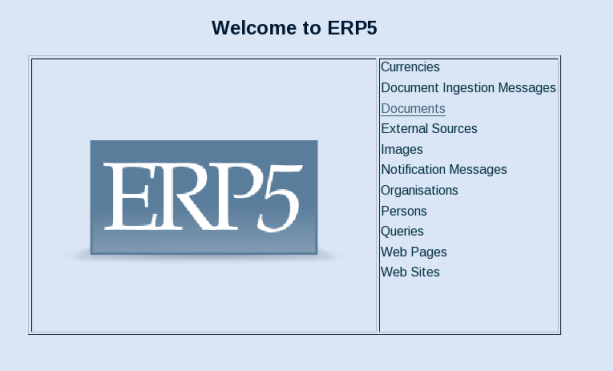

Figura 4.21: Página Principal do ERP5.

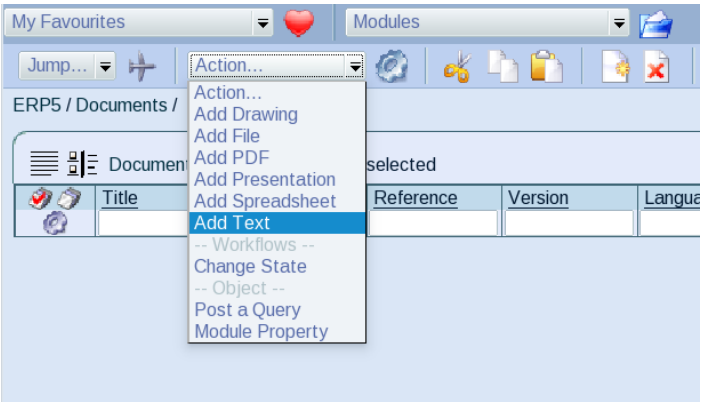

Figura 4.22: Opção para criar um objeto documento no ERP5.

Preview fazer *download* do documento completo no formato pdf. Para isto, clicou-se em "Download as PDF" no final da página de pré-vizualização.

4.7 Problemas e Decisões de Projeto

Com o desenvolvimento e testes excessivos na aplicação, algumas decisões de projeto foram tomadas para obter resultados mais eficientes ao clientes e correção de erros graves. Nesta sessão é explicado decisões que, em alguns casos, mudaram a arquitetura do projeto.

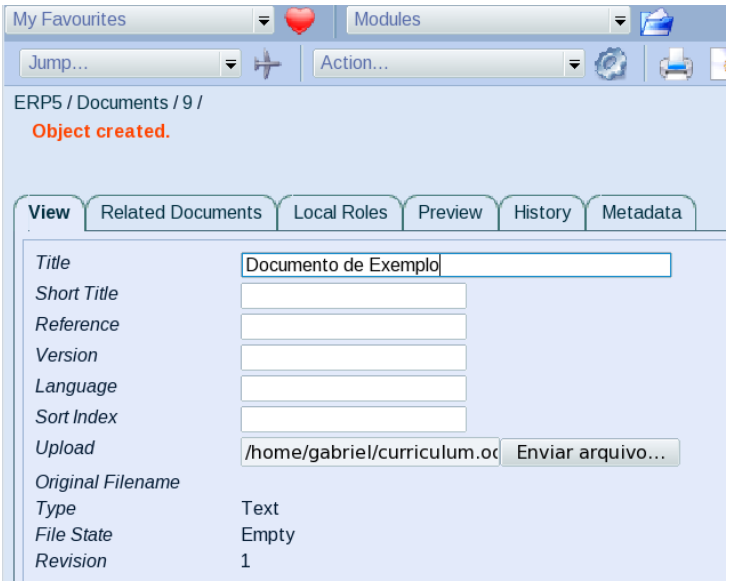

Figura 4.23: *Upload* de um documento.

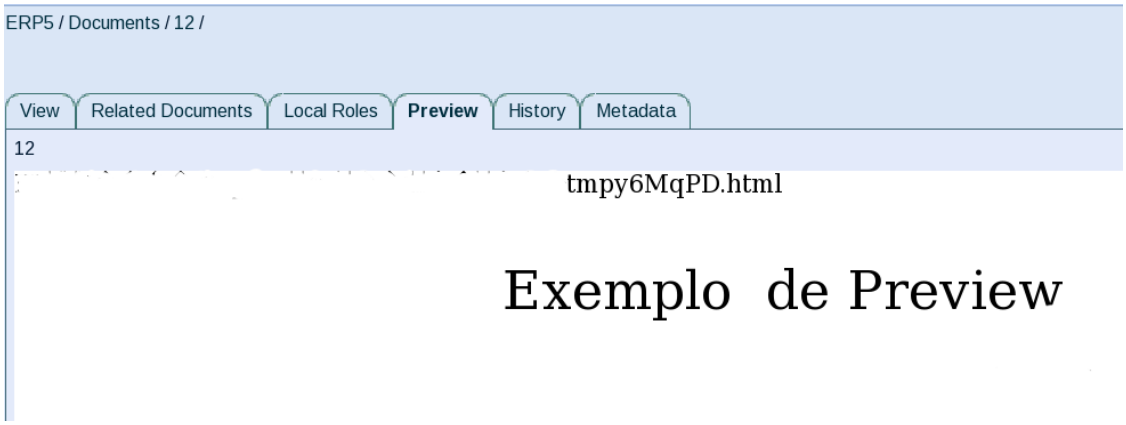

Figura 4.24: Pré-visualização de um documento.

4.7.1 Vazamento de memória

O uso de excessivo da aplicação OOOD mostrou que em algumas conexões entre o UNO e o OpenOffice.org, a memória utilizada não era completamente liberada. Este aumento gradativo do uso da memória, ocasiona um erro no sistema operacional, chamado *memory leak*, que ocorre quando não há mais memória livre para ser utilizada.

Com os resultados dos testes e a análise do uso de memória, foi possível detectar que o erro está no UNO que não libera a memória corretamente quando a conexão com o OpenOffice.org termina. Além disso, foi detectado também que a memória somente é

liberada corretamente quando todo o processo é finalizado, ou seja, quando o serviço é parado. Para que isso fosse resolvido era necessário remover a biblioteca UNO da memória quando esta não estivesse sendo utilizada.

A solução para este problema foi a implementação de dois *scripts* que fazem toda comunicação entre o OOOD e o OpenOffice.org. Desta forma, quando os *scripts* terminarem os procedimentos, toda memória utilizada será liberada e o serviço não sofrerá alteração, caso ocorram falhas.

Estes *scripts* foram implementados com objetivos diferentes, logo, as sessões abaixo explicam cada *script* e seus objetivos.

4.7.1.1 UnoConverter

O *script UnoConverter* foi implementado para conter as mesmas funcionalidades que o *OOHandler*. Esta implementação tem como objetivo retirar a comunicação entre OOOD e o OpenOffice.org. Com isso, o *OOHandler* utiliza o *script UnoConverter* para se comunicar com o OpenOffice.org.

Desta forma, a memória utilizada pela comunicação entre o *script UnoConverter* e o OpenOffice.org é liberada assim que o *script* é finalizado.

4.7.1.2 UnoMimeMapper

Além do acesso constante para conversão de documentos, uma outra necessidade de acesso ao OpenOffice.org é a extração de filtros feito pelo *MimeMapper*. Como este é um componente da aplicação OOOD, o *UnoMimeMapper* é utilizado para extrair os filtros do OpenOffice.org e retorná-los para o *MimeMapper*.

4.7.2 Armazenamento e busca dos filtros

A partir da extensão de um documento, como por exemplo “doc”, é possível saber para quais formatos este documento pode ser exportado. Com o objetivo de fazer com que esta funcionalidade tenha um desempenho melhor, foram criadas três estruturas que armazenam

as informações de maneiras diferentes para minimizar o tempo de busca pelos dados. As três maneiras são:

- Filtro por Extensão - armazena os filtros a partir da extensão do arquivo. Utilizada para selecionar o filtro adequado para exportar o documento;
- Extensões por Tipo de documento - armazena extensões de acordo com o seu tipo. Utilizado para buscar extensões de acordo com o tipo do documento;
- Tipo de Documento por extensão - armazena os tipos dos documentos suportados pela extensão. Com esta estrutura é possível verificar se a extensão de destino do documento suporta o mesmo tipo que a extensão do arquivo original;

Em suma, durante a importação dos filtros, os dados extraídos são organizados para eliminar qualquer processamento na hora de buscar por informações.

4.7.3 Arquivos compactados

Arquivos no formato "html" e "htm" não suportam imagens dentro do corpo do arquivo. Para que a imagem seja visualizada junto com o conteúdo em páginas Web, o caminho da imagem é inserido dentro código do documento. Como a aplicação OOOD suporta conversões de documentos, tanto o arquivo enviado ou recebido pelo o cliente em formatos "html" ou "htm", é necessário que as imagens sejam enviadas junto com o documento, caso contrário o documento gerado será inválido. Para que isto seja possível, o *FileSystemDocument* suporta compactação e descompactação de arquivos.

O caso de descompactação ocorre quando o cliente envia um arquivo e este está compactado. Neste caso, todos os dados são extraídos, salvos na mesma pasta, o arquivo original é removido e o arquivo com nome *index.html* é posto como documento principal.

Em contrapartida, a compactação é requisitada pelo usuário e, caso seja, o *FileSystemDocument* compacta todos arquivos que estão dentro da pasta e envia para o cliente.

4.8 Explicando o funcionamento do prestador de serviço com *Paste*

Para provêr os serviços do OOOD, foi utilizado a ferramenta *Paste*, que é um conjunto de ferramentas para construção de aplicativos e, além disso, provê serviços que implementam a interface WSGI.

Para configurar a aplicação OOOD com o *Paste*, foi definido no arquivo *setup.py* da aplicação, qual metódo será utilizado para iniciar o serviço, como demonstra a Figura 4.25.

Como isso, com a definição apresentada pela Figura 4.25, o método de nome *application* será responsável por iniciar o serviço. Além disso, este mesmo método, de acordo com o WSGI, deve retornar no final de seu procedimento qual método o cliente recebe ao se conectar com o serviço.

```
36      entry_points="""
37      # -*- Entry points: -*-
38      [paste.app_factory]
39      main = oood.oood:application
40      """,
41      )
```

Figura 4.25: Definição do metódo que irá prover o serviço.

```
44  def application(global_config, **local_config):
45
46    return WSGIXMLRPCApplication(instance=OOManager())
```

Figura 4.26: Método que inicia a aplicação e retorno do objeto que o cliente recebe quando se conecta.

Por fim, para iniciar uma aplicação utilizando a ferramenta *Paste*, esta última necessita de um arquivo de configuração que possui informações como, o pacote do *Paste* a ser utilizado e a porta que o servidor deve ser iniciado. A Figura 4.27 demonstra um trecho do arquivo de configuração para iniciar a aplicação OOOD e a Figura 4.28 demonstra como a aplicação é iniciada utilizando esta aplicação.

```
[app:main]
use = egg:oood

openoffice_hostname = localhost
# OpenOffice Port
openoffice_port = 4060

virtual_display_port = 6099
# ID of the virtual display where OOo instances are launched
virtual_display_id = 99
virtual_screen = 0

[server:main]
use = egg:PasteScript#wsgiutils
host = 0.0.0.0
port = 8008

```

Figura 4.27: Amostra do arquivo de configuração.

```
[gabriel@localhost ~]$
[gabriel@localhost ~]$ paster serve samples/oood.conf
[gabriel@localhost ~]$
```

Figura 4.28: Iniciando a aplicação OOOD com *Paste*.

4.9 Comparando a *perfomance* entre a OOOD 2.0 e o OOOD 1.0

Para comparar melhor o desenvolvimento da nova aplicação, foram desenvolvido *scripts* que submetiam documentos tanto para nova aplicação quanto para a antiga. Nestes scripts, foram registrados o tempo de conversão de cada documento e o tempo total de todas as conversões.

Para realização destes testes foram selecionados 3699 documentos no formato odt que alguns destes eram inválidos e outros em formatos desconhecidos. Além disso, para realização destes testes foi definido que o formato destino dos documentos fossem pdf, pois este formato é o mais utilizado pelo ERP5 para exportar documentos.

Com o resultado dos testes, notou-se que comparando a conversão do primeiro documento, a aplicação antiga é mais rápida que o OOOD 2.0. Mas, com o uso excessivo das aplicações e com os erros, o OOOD 2.0 se manteve estável enquanto a aplicação antiga apresentou-se instável em relação ao tempo de conversão. Os erros apresentados pelo OOOD 2.0 foram 12 de arquivos inválidos e não suportados, enquanto a aplicação antiga

apresentou 531 erros.

Apesar do OOOD 2.0 individualmente ser mais lento, o tempo gasto para conversão de todos foi de 10 horas. Enquanto a aplicação antiga gastou 11 horas para finalizar as conversões.

4.10 Resultados obtidos

No estudo de caso foi apresentado a utilização da aplicação OOOD junto com a ferramenta *Paste*, que provê o serviço. Além disso, foi instalado e configurado o ERP5 e o *Business Templates* para conversão e extração de metadados de documentos utilizando o OOOD.

Vale ressaltar que, a ferramenta que provê o serviço utilizada neste estudo de caso, pode ser substituída por qualquer outra ferramenta que forneça serviços que utilizem o padrão WSGI. Isto demonstra que o serviço pode ser modificado de acordo com a necessidade e demanda dos clientes.

Além disso, foram apresentados dados comprovando que com a utilização de uma arquitetura mais atual e flexível, trouxeram resultados satisfátorios.

Capítulo 5

Conclusão

5.1 Objetivos alcançados

Neste trabalho foi apresentado o *Web Service* OOOD 2.0, uma ferramenta *Open Source* para conversão de documentos em larga escala. Sua implementação de forma genérica, fez com que cada parte da aplicação possa ser utilizada separamente como uma biblioteca comum, tornando possível adicionar uma nova implementação caso almeje-se extender para atender requisitos específicos.

Foi visto também que, com o Python e seus recursos é possível desenvolver ferramentas que suportam grandes demandas sem que a legibilidade e flexibilidade do código sejam esquecidas.

5.2 Trabalhos futuros

Implementar manipuladores de imagens e vídeos para que seja possível converter e extrair metadados destes tipos de arquivos. Além disso, diminuir os dados que são tranferidos entre os componentes da aplicação e os *scripts* *UnoConverter* e *UnoMimeMapper* pois, diminuindo o tamanho dos dados, há um redução direta no tempo de resposta para o cliente.

É necessário também melhorar o ambiente de testes para que não seja necessário iniciar e parar o OpenOffice.org a cada teste, ou seja, para que os testes executem iniciando e parando o OpenOffice.org somente um vez, diminuindo o tempo gasto para executá-los.

Referências Bibliográficas

Alonso et al. 2004 ALONSO, G. et al. *Web Services: Concepts, Architecture and Applications*. [S.l.]: Springer Verlag, 2004. 123–125 p. ISBN 3-540-44008-9.

Balani 2009 BALANI, R. H. N. *Apache CXF Web Service Development: Develop and deploy SOAP and RESTful Web Services*. [S.l.]: Packt Publishing Ltd., 2009. ISBN 978-1-847195-40-1.

Booth et al. 2004 BOOTH, D. et al. *Web Services Architecture*. 2004. Disponível em: <http://www.w3.org/TR/ws-arch/>.

Carvalho e Campos 2007 CARVALHO, R. A. de; CAMPOS, R. de. Propostas para o processo de desenvolvimento do sistema erp5. 2007.

Cerami 2002 CERAMI, E. *Web Services Essentials*. [S.l.]: O'Reilly, 2002. ISBN 0596002246.

Chandra et al. 2001 CHANDRA, R. et al. *Parallel programming in OpenMP*. [S.l.: s.n.], 2001. 147–149 p. ISBN 1-55860-671-8.

Christensen et al. 2001 CHRISTENSEN, E. et al. *Web Services Description Language (WSDL) 1.1*. 2001. Disponível em: <http://www.w3.org/TR/wsdls>.

Dykes e Tittel 2005 DYKES, L.; TITTEL, E. *XML For Dummies; 4th Edition*. [S.l.]: Wiley Publising Inc., 2005. ISBN 0764588451.

Eby 2010 EBY, P. J. *Python Web Server Gateway Interface v1.0*. 2010. Disponível em: <http://www.python.org/dev/peps/pep-0333, acessado 15/05/2010>.

Eisenberg 2005 EISENBERG, J. D. *OASIS OpenDocument Essentials: Using OASIS OpenDocument XML*. [S.l.]: Friends of OpenDocument Inc., 2005.

Fulton 2010 FULTON, J. *zc.buildout: System for managing development buildouts*. 2010. Disponível em: <http://pypi.python.org/pypi/zc.buildout, acessado 11/06/2010>.

Gookin 2005 GOOKIN, D. *Troubleshooting Your PC For Dummies (For Dummies (Computer/Tech))*. [S.l.: s.n.], 2005. 146–147 p.

Honoré 2010 HONORé, J. *How to install ERP5 from Buildout*. 2010. Disponível em: <http://www.erp5.com/download/linux/Howto-Install.erp5.From.Buildout, acessado em 22/06/2010>.

Laurent, Dumbill e Johnston 2001 LAURENT, S. S.; DUMBILL, E.; JOHNSTON, J. *Programming Web Services with XML-RPC*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2001. ISBN 0596001193.

Lutz 2006 LUTZ, M. *Programming Python, Third Edition*. [S.l.: s.n.], 2006. 1130 p.

Newcomer 2002 NEWCOMER, E. *Understanding Web services: XML, WSDL, SOAP, and UDDI*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. 3 p. ISBN 0201750813.

Norris, Hurley e Hartley 2000 NORRIS, G.; HURLEY, J. R.; HARTLEY, K. M. *E-Business and ERP: Transforming the Enterprise*. New York, NY, USA: John Wiley & Sons, Inc., 2000. ISBN 0471392081.

Oram e Wilson 2007 ORAM, A.; WILSON, G. *Beautiful Code*. [S.l.: s.n.], 2007. 339–350 p.

Pilato 2004 PILATO, M. *Version Control With Subversion*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2004. ISBN 0596004486.

Pilone e Miles 2007 PILONE, D.; MILES, R. *Head first software development*. [S.l.]: O'Reilly, 2007. ISBN 9780596527358.

Ramachandran e Carvalho 2009 RAMACHANDRAN, M.; CARVALHO, R. A. de. *Handbook of Research on Software Engineering and Productivity Technologies: Implications of Globalization*. [S.l.: s.n.], 2009. 423–427 p. ISBN 978-1-60566-732-4.

Ramme 2010 RAMME, K. *This is the Uno Project*. 2010. Disponível em: <http://udk-.openoffice.org/, acessado em 28/06/2010>.

Santos 2007 SANTOS, R. A. *GRANULATEO3TOOL - UMA FERRAMENTA PARA GRANULARIZAÇÃO DE DOCUMENTOS*. 49 p. Monografia (Monografia) — Centro Federal de Educação Tecnológica - Campos - RJ, Campos dos Goytacazes/RJ, 2007.

Shane e Shore 2007 SHANE, S. W.; SHORE, J. *The Art of Agile Development: With Extreme Programming*. [S.l.]: O'Reilly Media, Inc., 2007. ISBN 978-0-596-15290-1.

Singh et al. 2004 SINGH, I. et al. *Designing Web Services with the J2EE 1.4 Platform: JAX-RPC, XML Services, and Clients*. [S.l.]: Pearson Education, 2004. ISBN 0321205219.

Smets 2010 SMETS, J.-P. *ERP5 Business Templates Download*. 2010. Disponível em: <http://www.erp5.com/web_page_module/1916/, acessado em 22/06/2010>.

Stervinou 2003 STERVINOU, J. *XML-RPC Home Page*. 2003. Disponível em: <http://www.xmlrpc.com/, acessado 10/06/2010>.

Thomas 2008 THOMAS, R. *Python-UNO bridge*. 2008. Disponível em: <http://udk-.openoffice.org/python/python-bridge.html, acessado 10/06/2010>.

Thomas e Budischewski 2008 THOMAS, R.; BUDISCHEWSKI, J. *Python-UNO bridge*. 2008. Disponível em: <http://udk.openoffice.org/python/python-bridge.html, acessado em 15/07/2010>.

Wikipedia 2010 WIKIPEDIA. *Enterprise Resource Planning*. 2010. Disponível em: <http://en.wikipedia.org/wiki/ERP, acessado 05/06/2010>.

Wikipedia 2010 WIKIPEDIA. *OpenOffice.org*. 2010. Disponível em: <http://pt-.wikipedia.org/wiki/OpenOffice.org, acessado 15/07/2010>.