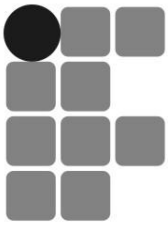


INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA FLUMINENSE
Campus Campos-Centro

Secretaria de Educação Profissional e Tecnológica

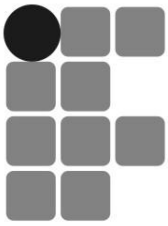
Ministério da Educação

BACHAREL EM SISTEMAS DE INFORMAÇÃO

PRISCILA MANHÃES DA SILVA

DESENVOLVIMENTO DE NOVAS FUNCIONALIDADES PARA A FERRAMENTA DE MANIPULAÇÃO DE ARQUIVOS EM NUVEM

Campos dos Goytacazes/RJ
2012



INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA FLUMINENSE
Campus Campos-Centro

Secretaria de Educação Profissional e Tecnológica

Ministério da Educação

BACHAREL EM SISTEMAS DE INFORMAÇÃO

PRISCILA MANHÃES DA SILVA

DESENVOLVIMENTO DE NOVAS FUNCIONALIDADES PARA A FERRAMENTA DE MANIPULAÇÃO DE ARQUIVOS EM NUVEM

Trabalho de conclusão de curso apresentado ao Instituto Federal Fluminense como requisito obrigatório para obtenção de grau em Bacharel de Sistemas de Informação.

Orientador: Prof. Rogério Atem Carvalho
Co-orientador: Prof. Fernando Carvalho

Campos dos Goytacazes/RJ
2012

PRISCILA MANHÃES DA SILVA

DESENVOLVIMENTO DE NOVAS FUNCIONALIDADES PARA A FERRAMENTA DE MANIPULAÇÃO DE ARQUIVOS EM NUVEM

Trabalho de conclusão de curso apresentado ao Instituto Federal Fluminense como requisito obrigatório para obtenção de grau em Bacharel de Sistemas de Informação.

Aprovada em 22 de novembro de 2012

Banca avaliadora:

Prof. Rogério Atem Carvalho (Orientador)
Doutor em Ciências de Engenharia / IFF Campus Campos
Instituto Federal de Educação, Ciência e Tecnologia Fluminense / Campus Campos Centro

---

Prof. Fernando Luiz de Carvalho Silva (Co-Orientador)
Mestre em Engenharia de Produção / UENF
Instituto Federal de Educação, Ciência e Tecnologia Fluminense / Campus Campos Centro

---

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

Dedico este trabalho à minha mãe e amigos por todo apoio, compreensão

e contribuição que me prestaram.

AGRADECIMENTOS

Primeiramente agradeço a minha mãe que sempre esteve ao meu lado nestes longos anos. Agradeço também a todos meus amigos e colegas de trabalho do NSI, que me apoiaram e incentivaram. Bem como aos professores Rogério Atem e Fernando Carvalho que contribuiram muito para que esse projeto fosse realizado.

O computador não é mais o centro do mundo digital.

Tim Cook

RESUMO

Este trabalho descreve sobre o desenvolvimento de novas funcionalidades para uma ferramenta livre de manipulação de documentos em larga escala, sendo o objetivo dessas funcionalidades permitir a ferramenta que se adapte a manipulação outros formatos de arquivos, de multimídia, por exemplo. Ainda é descrito no trabalho um pouco sobre as aplicações e/ou outras ferramentas que foram integradas a fim de tornar possível a existência destas funcionalidades. Através da parceria da empresa francesa Nexedi SA, com o Núcleo de Pesquisa em Sistemas de Informação(NSI-IFF), foi possível implementar as funcionalidades de manipulação de formatos correspondentes a arquivos de vídeo, áudio, imagens e PDF; cabendo a esta manipulação realizar a conversão, inserção e extração de dados destes arquivos. Encontram-se ainda, neste trabalho, a estrutura da ferramenta, conceitos básicos empregados para formação da ferramenta e sua integração com o núcleo linux, bem como conceitos da linguagem Python, a qual foi utilizada para seu desenvolvimento.

PALAVRAS-CHAVE: Serviço Web, Software livre, Cloud Computing

ABSTRACT

This work describes about the development of new functionalities to a free tool of document manipulation on a large scale, being this new functionalities responsible for allowing the tool to adapt in handling other file formats, such as multimedia ones, for example. It stills describes a little about the applications and/or other tools used for integrating its own with propose to handle with this new functionalities. Through this partnership between the French company Nexedi SA with the aid of the Nucleus of Research in Information Systems (NSI-IFF), was possible to implement the ability to manipulate shapes corresponding to video files, audio, images and PDF; being this manipulate as well responsible for converting, inserting and extracting data from this files. Still can find in this job, the structure of this tool, basic concepts used to create the tool and integrate it with the core Linux, as well as concepts of Python, which was used for its development.

KEYWORDS: Web Service, Free software ,Python, Cloud Computing

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Inicialmente quando a internet não passava de um experimento militar restrito para proteger o Governo de ataques inimigos (TESLA, 1994), o conceito de computação em nuvem não existia. Sob este cenário diversos usuários foram levados a se habituar com uma vasta diversidade de aplicações a nível *desktop* para suprir suas necessidades diárias.

Atualmente a internet sofreu tantas modificações que seu uso se tornou constate e praticamente dependente para qualquer tarefa, como por exemplo transformar um documento qualquer em um arquivo PDF para transporte. Isto se deu por sua evolução de comunicação e pela demonstração de um mundo de inúmeras possibilidades ainda não exploradas.

Assim, há aproximadamente 4 anos, a empresa francesa Nexedi SA e o Núcleo de Pesquisas em Sistemas de Informações(NSI) do Instituto Federal Fluminense(IFF), iniciaram o desenvolvimento de uma ferramenta Web de código aberto para substituir a ferramenta OpenOffice Daemon (OOOD), originalmente desenvolvida apenas pela Nexedi.

A OOOD, cujo desenvolvimento foi iniciado em 2006, tinha por objetivo a conversão de documentos do tipo Office, visando principalmente a realização de formatos privados para abertos. Com o uso freqüente desta ferramenta chamados ambientes de produção, isto é, em ambientes finais idealizados para demandas reais da ferramenta, esta demonstrou certa instabilidade. Está instabilidade deu-se em erros aparentemente não justificáveis e exceções inadequadamente tratadas.

Os problemas identificados variavam desde problemas com a preparação do servidor da ferramenta, como perdas de requisições e estouros de memória, até erros de aplicação, como *deadlock* no OpenOffice.Org e em processos.

Somados esses problemas as novas demandas da aplicação, provou-se inviável reescreve ou fazer sua manutenção, dessa forma foi dado o desenvolvimento de uma nova ferramenta que além de fazer estas conversões pudesse ser capaz de manipular estes documentos para demais atividades, tais como a "granularição", utilidade necessária para o projeto Biblioteca Digital, do NSI.

Em 2010, a OOOD 2.0 foi concluída e apresentada como uma versão extremamente mais estável e que implementava grande parte das tarefas necessárias aos principais projetos

que a utilizavam.

Nesta nova versão foram criados controladores de memória e processos para ponderar o uso do sistema pela ferramenta, dessa forma seria possível conter e mesmo reiniciar processos que pudessem vir a causar gargalos em suas operações. Os controles de memória trabalhavam com base em variáveis de ambiente definidas na instalação da ferramenta, enquanto os controladores de processos trabalham em cima de um tempo média de execução que um processo pode levar, pois caso contrário a ferramenta subentende que esta aplicação não está realizando a tarefa que a foi passada.

Além da estabilidade a nova versão da ferramenta possibilitava a inserção e extração de informações de documentos do tipo Office, conforme parte do proposto inicial, essas questões, bem como as conversões ficaram a critério do uso da aplicação OpenOffice.Org.

Dado o sucesso nesta transição da ferramenta, demais funcionalidades foram almejadas, entre elas a "granularição", anteriormente citada, e também a conversão de outros tipos de arquivos, entre os quais PDF que não era ainda bem atendido pela ferramenta OpenOffice.Org e que precisaria do uso de uma nova ferramenta.

Também eram desejáveis as conversões entre arquivos de vídeo, áudio e imagens, bem como a manipulação destes para inserção e extração de informações.

Assim este trabalho visa apresentar uma ferramenta nova, cujo inclusive o nome veio a ser modificado para CloudOoo, que fosse estável e capaz de converter e manipular, documentos Office, documentos PDF, áudio, vídeo e imagens e ainda "granularizar" documentos em geral.

1.1 Objetivos

Este trabalho pretende contribuir com novas funcionalidade para ferramenta de extração e inserção de dados e conversão de arquivos em nuvem chamada CloudOoo.

Além disso, visa criar uma forma de documentação técnica e manual para futuros usuários que venham a se interessar pelo uso e mesmo pela contribuição nesta ferramenta.

Sendo assim o projeto apresenta características das principais ferramentas envolvidas no desenvolvimento e atuação do mesmo, bem como sua estrutura através dessas.

Demonstrar as mudanças e melhorias no CloudOoo desde sua versão anterior, OOOD 2.0.

1.2 Justificativa

Optou-se por colaborar com a manutenção e expansão desta ferramenta de software livre e aberto visando gerar benefícios para sociedade.

Através deste trabalho a ferramenta teve ampliadas suas funcionalidades possibilitando o tratamento de arquivos diversos, entre eles áudios, vídeos, imagens e documentos PDF.

Além disso esta colaboração visou a manutenção da ferramenta a fim de garantir sua estabilidade, tendo em vista seus problemas detectados anteriormente.

1.3 Organização do Trabalho

Este trabalho se divide em seis capítulos, sendo este primeiro um capítulo introdutório e explicativo sobre o conteúdo do trabalho e de suas intenções.

O segundo capítulo deste trabalho cita e explica sobre o uso das principais tecnologias que estão por trás do funcionamento do CloudOoo, focando em seu uso específico para este, e portanto não abrangendo demais funcionalidades de cada uma dessas.

No terceiro capítulo é apresentada a estrutura sobre a qual o CloudOoo vem sendo desenvolvido desde de sua versão anterior, promovendo apenas atualizações.

Ao quarto capítulo são apresentadas tecnologias consideradas semelhantes ao CloudOoo, sendo estas no entanto demonstrações mais semelhantes a sua versão anterior e mais especificamente a documentos ou apenas a conversões de arquivos.

O quinto capítulo foca sobre um breve estudo de caso do desenvolvimento desta ferramenta, avaliando sua forma de instalação em suas plataformas mais comuns e sobre seu uso, levando em conta uma breve comparação com estudos anteriores do mesmo.

Por fim no sexto capítulo são apresentadas as conclusões deste trabalho com base no quanto cresceu e visando sobre propostas futuras de mudanças para a melhoria desta ferramenta.

2 TECNOLOGIAS EMPREGADAS

Para o desenvolvimento do CloudOoo foi essencial o uso de outras ferramentas as quais pudessem ser incorporadas neste e tornassem possível a criação de novas funcionalidades. Este capítulo apresenta um conceito simplificado de cada uma dessas integrações, desde a linguagem principal empregada até as ferramentas utilizadas.

2.1 Python

Python é uma linguagem de programação poderosa, fácil de aprender de código aberto e livre.

Ela possui estruturas eficientes e de alto nível de dados além de uma abordagem simples, entretanto eficaz, para programação orientada a objeto. Sua sintaxe elegante de tipagem dinâmica, juntamente com a sua natureza interpretável, tornam-na uma linguagem ideal para *scripts* e desenvolvimento rápido de aplicações em muitas áreas, em diversas plataformas (ROSSUM, 2003).

Por todas estas vantagens, hoje, Python é uma das linguagens mais utilizadas no mundo, assim muitas de suas bibliotecas e ferramentas são bem conhecidas. Abaixo serão descritas as ferramentas Buildout e Zope e a biblioteca Subprocess.

2.1.1 Buildout

Buildout é uma ferramenta desenvolvida em Python, capaz de reproduzir o desenvolvimento de aplicações em um ambiente dedicado. Ele fornece suporte para criação dessas aplicações, especialmente para as desenvolvidas Python.

Esta ferramenta se baseia na execução de procedimentos especificado em um arquivo de configuração conhecido por *recipe*

A partir de um conjunto de características e variáveis de ambiente previamente definidas nesse *recipe*, é gerada a estrutura da aplicação, sendo esta composta pelas dependências de sistema,por pacotes Python ou mesmo por outras aplicações.

Um único *recipe* de uma aplicação pode conter definições de configuração, processos e ainda outras aplicações, que por sua vez podem ser organizadas com seu próprio *recipe* (BRANDOM, 2008).

Levando em consideração o ideal do ambiente dedicado, também chamado de puro, a instalação do Buildout é extremamente simples podendo ser realizada a partir de um *script* Python simples e de fácil acesso on-line, (PYTHON SOFTWARE FOUNDATION, 2012). Após executado, este *script* gerará um segundo *script* conhecido por "bin/buildout".

Através do uso deste segundo *script* é possível a execução das definições de cada *recipe* a fim de obter a aplicação final. No exemplo 2.1 ocorre a construção da aplicação Xprompt.

```
1  [buildout]
2  develop = .
3  parts =
4    xprompt
5    test
6
7  [xprompt]
8  recipe = zc.recipe.egg:scripts
9  eggs = xanalogica.tumbler
10 interpreter = xprompt
11
12 [test]
13 recipe = zc.recipe.testrunner
14 eggs = xanalogica.tumbler
```

Código 2.1: Exemplo de um *script* Buildout

O *recipe* inicia-se com a declaração "[Buildout]", que é a única parte realmente obrigatória para identificá-lo como um *recipe* do Buildout.

Caso a aplicação em questão esteja em desenvolvimento e/ou em fase de testes, utiliza-se a variável *develop*, que deve informar a localização deste desenvolvimento.

Por *parts* subentende-se o percurso do *script*, isto é, a ordem de execução que o Buildout deve obedecer para instalação da aplicação independente da ordem que estejam descritas a aplicações ao longo do *recipe*.

Cada *part* possui uma descrição padrão que consiste de variáveis que envolvem cada componente responsável pela sua instalação. O *eggs*, por exemplo, se trata da opção referente as dependências necessárias para execução desta parte.

O *interpreter* é utilizado para gerar um interpretador Python contendo as dependências relacionadas em *eggs*. Para utilizá-lo basta citá-lo e nomeá-lo. Neste caso o interpretador da aplicação será chamado "xprompt".

O tamanho de cada *recipe* vária de acordo com a necessidade da aplicação. Neste caso com poucas especificações será possível gerar uma nova instância de uma aplicação Xprompt.

Ainda que incomum, é valido ressaltar que aplicações não necessariamente desenvolvidas em Python podem ter seu ambiente gerado pelo Buildout, com base na vasta disponibilidade de *recipes* disponíveis. Isso geralmente é usado para suporte de dependências de sistema, em sua maioria desenvolvidas em C. O próprio Buildout será responsável por organizar e informar o caminho dessas aplicações, se necessário sobrescrevendo a variável de ambiente PATH.

2.1.2 Subprocess

O Subprocess é uma biblioteca Python que permite a criação de processos no sistema, similar a biblioteca Thread porém mais eficiente no quesito escalabilidade, pois permite que seus processos sejam divididos e utilizem diferentes processadores.

Esta biblioteca foi criada na intenção de substituir outros módulos e funções obsoletas, e permitir também maior interação entre a aplicação e o sistema no qual se encontra.

```
1 >>> import subprocess
2 >>> subprocess.Popen('echo "Hello World!"', shell=True)
3 Hello World!
```

Código 2.2: Exemplo de uso do Subprocess

No código 2.2 existe um pequeno exemplo de como o Subprocess pode ser utilizado.

Neste exemplo ele reproduz um acesso ao *shell* do sistema a fim de demonstrar um *Hello World* através do comando *echo*, é possível observar o uso do *shell* no final da segunda linha em que cita-se “shell=True”.

Em aplicações que utilizam ambientes dedicados, como citado na sessão 2.1.1, é preferencial que a variável *shell* tenha valor *false*,a fim de usar as aplicações deste ambiente, entretanto este já é seu valor por padrão, assim não é preciso declará-lo, como é visto no código 2.3.

Nesses casos também é preferível o uso da variável “env” que representa variáveis de *environment*, isto é, variáveis de ambiente. Ao substituir esta variável o Buildout direciona o processo a utilizar os binários que estão no seu ambiente próprio ao invés do ambiente original, que neste caso é o próprio sistema.

O processo de sobrescrever a variável “env” é similar ao passo de redefinir a variável *PATH* do sistema.

```
1 command = ["convert", self.file, output]
2 stdout, stderr = Popen(command,
3                         stdout=PIPE,
4                         stderr=PIPE,
5                         env=self.environment).communicate()
```

Código 2.3: Exemplo de uso do Subprocess com *PIPE* extraído do CloudOoo

No exemplo, código 2.3, também é utilizada a opção *PIPE* que permite ao comando Popen retornar as mensagens de saída e erro respectivamente através das variáveis *stdout* e *stderr*, para que seu conteúdo possa ser após o uso do comando.

2.1.3 Zope

Zope (Z Object Publishing Environment) é um serviço web livre, de código aberto, desenvolvido em Python (ZOPE FOUNDATION, 2012), em outras palavras, um *framework* Python.

Zope já foi considerado o principal responsável pela popularização do Python. Não existiu ferramenta em Perl que o levasse as pessoas como o Zope levou o Python (UDELL, 2000). Entretanto o Zope se tornou muito extenso e complexo, requerendo uma alta de taxa de aprendizagem e assim sendo ofuscado pela ferramenta Django em questão de popularidade.tanto

Dado seu nível de complexidade e todas as extensões disponíveis, muitos continuam utilizando o Zope, indiferentes a sua queda de popularidade, fazendo com que suas extensões continuem surgindo, sendo mantidas em processo de atualização continua por seus mantenedores.

2.1.3.1 Zope Interfaces

Como citado na sessão 2.1.3, diversas extensões Zope foram desenvolvidas com intuito de auxiliar na criação de outras aplicações Python, de forma a não deixá-las muito extensas ou pesadas. Entre estas a *zope.interface* é um exemplo de extensão independente escrita em Python e mantida pela equipe Zope.

A *zope.interface* foi criada com intuito de permitir a comunicação entre quaisquer componentes externos que possuíssem uma *Application Programming Interface* (API). A idéia de criar uma interface para os componentes de uma aplicação é uma forma elegante de resolver um antigo problema de tipagem dinâmica tratando as informações recebidas de forma genérica, a fim de renderizá-las para um tratamento mais específico.

2.2 XML

Extensible Markup Language (XML) é uma linguagem em formato de texto simples para representação de informações sobre estruturas, sejam elas de documentos, dados, configurações, entre outros.

Esta linguagem foi derivada de um formato antigo chamado SGML, e adaptada para ser mais flexível ao uso Web (QUIN, 2010).

Atualmente se tornou um dos formatos mais comuns para compartilhamento de informações em aplicações web, por ter uma escrita simples e corresponder a um padrão similar ao HTML, que por muitos anos já vem sendo utilizado.

Além disso esta linguagem dispõe de outras vantagens como: padrão de *markup*, cujo o nome é detalhado de forma redundante; a descrição da estrutura, que de forma geral também permite uma compreensão bem literal, como se fossem textos; todas suas versões são correspondentes, isto é, mesmo que algumas delas procurem por *markups* específicos, qualquer versão mais nova de XML funcionará em uma aplicação que utilizasse uma versão anterior ou mais antiga.

2.3 Formato Aberto

O formato aberto é uma especificação publicada para armazenar dados digitais, mantido geralmente por uma organização de padrões não proprietária, e livre de limitações legais no uso.

Um formato aberto deve ser implementável tanto em software proprietário quanto em livre, usando as licenças típicas de cada um. Em oposição a esta idéia o formato proprietário é controlado e defendido por interesses particulares de empresas proprietárias detentoras de seus direitos.

O objetivo principal dos formatos abertos é garantir o acesso a longo prazo aos dados sem incertezas atuais ou futuras no que diz respeito às direitos legais ou à especificação técnica. Um objetivo secundário dos formatos abertos é permitir a competição de mercado, em vez de deixar que o controle de um distribuidor sobre um formato proprietário iniba o uso de um produto de competição.

2.3.1 Formatos abertos de Documentos

O *.txt* é o formato aberto mais comum para arquivos de texto por ser pequeno e na maioria dos casos dispor de vários programas de edição em qualquer plataforma operacional, entretanto não é considerado o formato ideal para documentos, uma vez que não possui opções de formatação; tais como itálico e negrito.

Em 01 de maio de 2005, surgiu o *OpenDocument Format*(ODF), um conjunto de formatos para aplicações de escritório com o objetivo de padronizar os formatos abertos para documentos. Seu nome original era *Open Document Format for Office Application*, uma iniciativa da *Organization for the Advancement of Structured Information Standards*(OASIS), baseado em um XML criado por desenvolvedores do OpenOffice.org. Esta aplicação na época era uma das poucas capazes de utilizar sua estrutura.

Atualmente os formatos ODF existem a 7 anos e foram adotados por diversas aplicações. Mesmo em sotwares proprietário entre elas o Microsoft Office, mesmo sendo um software originalmente proprietário através de um *pluggin* ele permite a edição e manipulação destes formatos.

Através da figura 2.1, adaptada de (SILVA, 2010), é possível acompanhar o crescimento do uso do ODF até 2010, ano em que fez 5 anos.

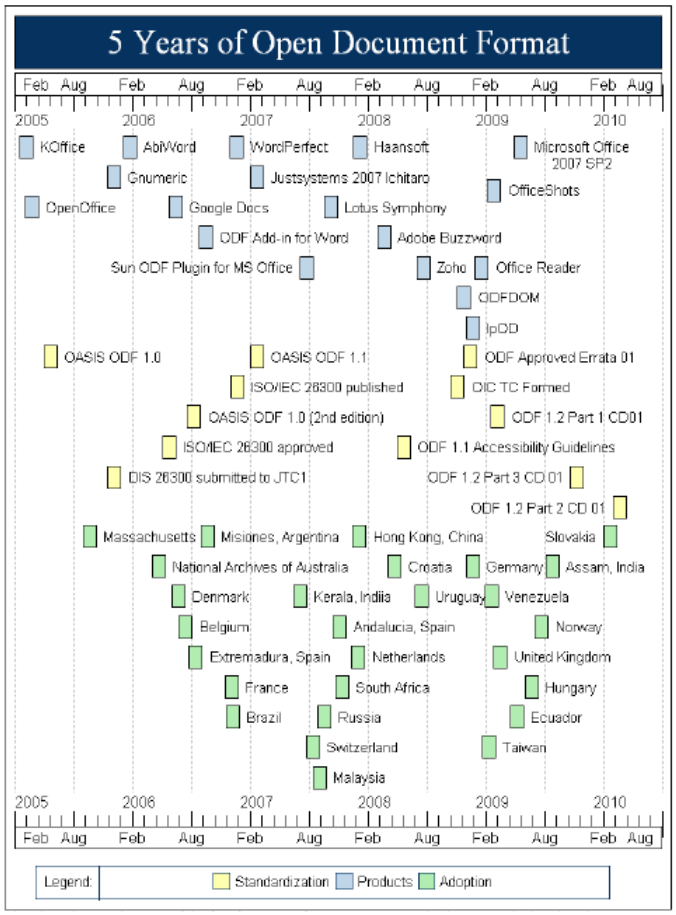


Figura 2.1: Crescimento do ODF nos primeiros cinco anos.

> O *OpenDocument Format* ainda é uma incógnita à grande maioria dos usuários comuns, mas sua adoção cresce em várias partes do mundo, especialmente nos meios corporativos e governamentais. No Brasil, por exemplo, o ODF já conta inclusive com aprovação da Associação Brasileira de Normas Técnicas (ABNT), que aconteceu em 2008 (norma NBR ISO/IEC 26300)(ALECRIM, 2011).

2.3.1.1 Formatos de documentos ODF

O embora o mesmo padrão seja aplicado de forma geral em documentos ODF, esse padrão possui variações de extensões, no que diz respeito a documentos são elas:

- Para documentos de texto: .odt, .fodt;
- Para planilhas eletrônicas: .ods, .fods;
- Para apresentações: .odp, .fodp;
- Para bancos de dados: .odb;
- Para desenhos: .odg;
- Para fórmulas: .odf.

E ainda existe um padrão a parte para modelos(*templates*), são eles:

- Para documentos de texto: .ott;
- Para planilhas eletrônicas: .ots;
- Para apresentações: .otp;
- Para bancos de dados: .otb.

Por guardar desde estrutura e dados textos até imagens presentes em seus arquivos a estrutura ODF é considerada um padrão comprimido assim como arquivo ZIP. É com base nesse padrão que diversas bibliotecas foram desenvolvidas para trabalhar com esses arquivos, entre elas a PyUno 2.4.1.1.

2.3.1.2 Estrutura de documentos ODF

Como citado na sessão 2.3.1 um documento ODF é uma estrutura de padrão aberto semelhante a arquivos comprimidos, como por exemplo arquivos ZIP.

Esta estrutura é composta principalmente por:

- mimetype: arquivo de linha única constituído pelo mimetype do documento;
- content.xml: arquivo que armazena o conteúdo criado pelo usuário do documento;
- meta.xml: arquivo responsável por armazenar os metadados do documento, ou seja, dados como autor, data de criação, data de modificação e outros;
- styles.xml: arquivo que contém os estilos do documento, tais como formatações de texto, parágrafos e outros;
- Pictures: pasta que armazena figuras existentes no documento.

Existem ainda diversos arquivos e pastas que podem compor um documento ODF, mas que não são muito referidos em uso pratico.

2.3.2 Formatos de Imagens

No que diz respeito a imagens os formatos abertos tratam sobre patentes dos formatos, ou inclusa neles.

Assim em 1996 surgiu o formato *Portable Network Graphics*(PNG) que tinha como principal objetivo substituir o formato GIF, portador de inúmeros algoritmos patenteados, que no entanto vieram a expirar em 2003.

Desde o princípio sua principal intenção foi sua utilização em qualquer aplicação sem necessidade de conflitos sobre patentes.

Além disso este formato permite comprimir imagens sem perda de qualidade e também a retirada do fundo de imagens através do canal alfa; possui suporte de milhões de cores, diferentemente do GIF, cujo suporte era de 256 cores; e ainda permite a criação de animações, cujas extensões podem variar em *.mng* e *.apng*, mas são igualmente livres.

Este formato é apoiado pela *World Wide Web Consortium*(W3C), e se tornou um padrão internacional.

2.3.2.1 Estrutura do PNG

Um arquivo PNG consiste de uma assinatura PNG e seguido de vários blocos em serie (ROELOFS, 1999).

Sua assinatura equivale aos primeiros oito *bytes*, consiste da serie “137 80 78 71 13 10 26 10”.

Cada bloco consiste de:

- length: inteiro correspondente ao numero de *bytes* dos dados da imagem;
- chunk type: código equivalente ao tipo do bloco;
- chunk data: dados da imagem;
- Cyclic Redundancy Check (CRC): campo que contém o valor total de *bytes* do bloco, o *chunk type* e o *chunk data*, mas sem o valor do *length*.

O inicio da serie de blocos deve conter um *IHDR chunk* e o ultimo bloco deve conter um *IEND chunk* como *chunk type* sinalizando que representam o inicio e fim da serie respectivamente.

2.3.2.2 Exif

O *Exchangeable Image File Format*(Exif) é um conjunto de metadados a respeito da imagem em questão, ou seja, são dados como autor, dia em que a foto foi tirada, câmera utilizada, entre outros listados conforme um padrão.

Todas essas informações ficam dentro da própria imagem, no entanto é preciso ter uma aplicação especifica para vê-lo, e em casos de informações mais abrangentes, as vezes é necessário possuir também uma aplicação para manipular a imagem e inserir nela metadados a respeito da imagem, aplicações estas como 2.5.

2.3.3 Formatos de Áudio

Os formatos abertos de áudio tratam sobre codecs disponíveis sem uma patente aplicada aos mesmos. Nesta categoria conta-se com os formatos Vorbis(.ogg) e *Free lossless Áudio Codec*(.flac).

O .flac é um formato livre de áudio comprimível e sem perda de qualidade e dados durante a o processo de compreensão. Seu algoritmo permite que o arquivo reduza em até 60% seu tamanho original, e também permite a manipulação de metadados.

O .ogg é um novo formato comprimível de áudio. É comparado de forma grosseira com outros formatos utilizados para guardar e reproduzir musicas, tais como MP3, VQF, AAC, e outros formatos de áudio digital. Mas é diferente de todos estes formatos porque é livre, aberto e sem patentes (XIPH.ORG FOUNDATION, 2012).

2.3.4 Formatos de Vídeo

Assim como os arquivos de áudio, a licença dos formatos de vídeo tratam sobre patentes de codecs, e neste caso as extensões mais comuns são conhecidas por Theora(.ogv) e Matroska(.mkv).

Theora é de propósito geral, um codec de vídeo com perda de dados. É baseado no codec de vídeo VP3 produzido pela *On2 Tecnologies* (FOUNDATION, 2011).

Matroska é o nome de uma iniciativa ousada para a criação de formatos universais de *containers* de áudio e vídeo digitais integrados, também chamados de formato de vídeo (WIKIPEDIA, 2012).

Assim o .mkv trata-se de um “contentor” de padrão aberto para vídeos, que pode conter vários dados de diferentes tipos de codificações.

2.4 LibreOffice

O LibreOffice é um pacote de software, ou seja, uma suíte para documentos de escritório, de produtividade compatível com a maioria dos pacotes semelhantes, e que esta disponível para várias plataformas. É um software de código aberto, livre para baixar, utilizar e distribuir (PARKER; JR, 2010).

Seu inicio foi marcado em 2000, quando a Sun Microsystems liberou o código de seu produto StarOffice. Neste momento se chamava OpenOffice.Org.

Em 2010 a comunidade que desenvolvia o projeto anunciou uma fundação independente, *The Document Foundation*, a fim de cumprir com a independência explicita da carta anunciada ao inicio do projeto.

Assim no inicio de 2011 foi lançado o LibreOffice 3.3, que bem como o OpenOffice.org 2.0, já suportava a edição da suíte *OpenDocument*.

2.4.1 UNO

O projeto LibreOffice possuía uma característica muito útil e pouco utilizada que era a capacidade de integrar seu funcionamento com outros aplicativos, isto é, um componente foi disponibilizada a fim de que aplicações em outras linguagens que estivessem fora do projeto pudessem interagir com o mesmo. Esse componente é conhecido por UNO (Universal Network Objects), que por sua vez é composto por um modelo dos componentes do LibreOffice.

O UNO oferece interoperabilidade entre diferentes linguagens de programação, diferentes modelos de objetos, diferentes arquiteturas e processos, em uma rede local ou mesmo através da internet. Seus componentes podem ser implementados e acessados através de *bindings* deste (PYTHON.ORG, 2008).

Atualmente existem *bindings* para as linguagens C, C++, Java e Python. Desde a versão 1.1 o LibreOffice dispõe do PyUNO em suas instalações por padrão.

2.4.1.1 PyUNO

O PyUNO representa uma “ponte” entre o LibreOffice e aplicações Python. Através dele é possível a manipulação do componente UNO, seção 2.4.1, para utilizar praticamente todas funcionalidades disponíveis no LibreOffice por *scripts* Python.

No entanto, segundo Thomas (2007), essa ferramenta ainda não atingiu seu uso absoluto podendo conter diversos *bugs*, erros, e assim dependendo em grande parte de colaboração por parte da comunidade que utiliza a mesma.

No código 2.4 é possível ver um exemplo pratico do uso do PyUNO em um código Python:

```
1  import sys
2  import os
3  import uno
4
5  # Get the uno component context from the PyUNO runtime
6  uno_context = uno.getComponentContext()
7  # Create the UnoUrlResolver on the Python side.
8  url_resolver = "com.sun.star.bridge.UnoUrlResolver"
9  resolver = uno_context.ServiceManager.createInstanceWithContext(
10   url_resolver,
11   uno_context)
12 # Connect to the running OpenOffice.org and get its context.
13 uno_connection = resolver.resolve("uno:socket,host=%s,port=%s;urp;
       StarOffice.ComponentContext" % (host, port))
14 # Get the ServiceManager object
15 return uno_connection.ServiceManager
```

Código 2.4: Exemplo de uso do Uno

Neste exemplo extraído diretamente do CloudOoo utiliza-se o contexto do processo ativo do PyUNO para retorna um serviço de gerenciamento do UNO, isto é, uma conexão é estabelecida entre o principal serviço de gerenciamento do UNO e o PyUNO utilizado pelo CloudOoo para que este possa por fim controlar as ações deste serviço.

2.5 ImageMagick

ImageMagick é uma suíte de aplicações para criar, editar,compor ou converter imagens *bitmap* (IMAGEMAGICK STUDIO, 2012). Na realidade o ImageMagick disponibiliza um conjunto de binários, compatíveis com varias plataformas, que no Linux são dados como comandos de níveis, separados para diversas funcionalidades.

Para Tesla (1994) é uma ferramenta, originalmente criada por John Cristy, para visualizar e manipular imagens, que esta amplamente disponível na Internet.

As funcionalidades que podem ser consideradas mais comuns e utilizadas são *convert* e *identify*, essas funcionalidades podem respectivamente: converter imagens para outros formatos ou formatação, como invertido ou de girar em 180 graus; e identificar os metadados disponíveis na imagem, como autor, ou data que foi criada ou tirada.

Como as demais ferramentas citadas por este trabalho, é um *software* livre, disponível para baixar e utilizar da forma desejada ao usuário.

2.6 XPDF

Xpdf é uma suíte de binários de código aberto para visualização e manipulação básica de FFMPEG arquivos *Portable Document Format*(PDF), criada por Glyph e Cog (GLYPH COG, 2011).

Além de permitir a visualização de arquivos PDF o XPDF é uma ferramenta que permite a extração de textos dentro de documentos PDF e a conversão dos mesmos para o formato *postscript*, formato este especialmente composto de informações e desenvolvido originalmente pela *Adobe System*.

Assim como a maioria das aplicações para Linux é utilizada através de comandos, neste caso o mais comum *pdftotext* o qual captura o texto disponível no documento PDF e retorna este texto através de um arquivo de texto simples.

2.6.1 Poppler

O poppler é uma biblioteca para renderizar PDF baseada no xpdf 3.0 (FREEDESKTOP.ORG, 2011).

É uma das bibliotecas de código livre mais utilizada pelos sistemas Linux para leitores PDF, seu desenvolvimento é idealizado pela FreeDesktop.Org.

2.7 PDFTk

Se um documento PDF é um trabalho eletrônico, então o PDFTk é utilizado de removedor de grampos, furador, pasta, entre outros utilitários de documentos. o PDFTk é uma ferramenta consideravelmente simples para fazer as tarefas diárias com documentos PDF (STEWARD, 2011).

Esta ferramenta esta sob licença GPL e utiliza bibliotecas que possuem suas próprias licenças de uso.

Na realidade de simples o PDFTk tem apenas sua forma de uso, pois realiza tarefas complicadas no contexto de documentos PDF. Sua confiabilidade é altamente elogiável dado que seu criador e principal mantenedor, Sid Steward, é também autor do livro “PDF Hacks”, que é considerado uma das referências do ramo.

2.8 FFMPEG

Para Zhang (2011), a melhoria constante do uso do processamento de multimídia, requerida e obtida pela expansão multi funcional e acelerada dos equipamentos de hardware, requer também aplicações eficientes e escaláveis a medida que este processo avança. E partindo deste principio uma das ferramentas ideais para projetos escaláveis é o FFMPEG.

FFMPEG é um rápido conversor de vídeo e áudio que também consegue tratar informações de ambos. Ele é capaz de converter faixas arbitrarias de amostras e redimensionar vídeos através de filtros polifásicos de alta qualidade (FFMPEG.ORG, 2012).

Mais do que isso, o FFMPEG é uma suíte de aplicações via linha de comando capaz de converter, extrair e inserir metadados em arquivos de áudio e vídeo de simples entendimento, de fácil uso.

2.9 SERVIÇOS WEB

Segundo Pirnau (2011), os serviços web representam a metodologia em que aplicações podem se comunicar através de mensagens assíncronas ou chamadas remotas. Assim pode se dizer que serviços web são aplicações acessáveis remotamente.

Toda empresa tem por objetivo prover serviços, sejam esses para própria empresa, ou para clientes que por sua podem ser outras empresas. A anos esses serviços têm sido automatizados, inicialmente aplicações *desktop* eram criadas quando a empresa era pequena e possuía poucos computadores, ou a comunicação entre elas não era tão necessária.

Quando a rede passou a estar presente no dia a dia de forma geral essas aplicações foram evoluindo e buscando a comunicação entre as mesmas.

Este conceito na verdade trata da iniciativa por parte dessas empresas de retirar suas aplicações dos computadores e passá-las para potente servidores que disponibilizarão esta na internet, assim basta ter acesso a internet e é possível utilizar esta aplicação.

2.10 XML-RPC

É um protocolo para chamadas remotas que utiliza HTTP para transporte e XML para codificação. O XML-RPC foi desenhado para ser o mais simples o possível, enquanto permite que uma estrutura de dados complexa seja transmitida, processada e retornada (SCRIPTING NEWS, 2011).

Foi originalmente criado por Dave Winer na *UserLand Frontier*, e inspirado por outros

dois protocolos, um deles também desenvolvido pelo próprio Dave Winer e outro que representava o começo do protocolo SOAP. Entretanto seu uso é bem mais simples de se utilizar e entender que o SOAP. Suas mensagens correspondem a uma requisição *HTTP-POST*, enquanto composição do corpo da mensagem é escrita em XML, bem como a resposta que a requisição recebe.

Dada sua simplicidade e eficiência se tornou um protocolo muito popular e utilizado hoje nos dias atuais.

2.11 WSGI

O *Web Service Gateway Interface*(WSGI) é uma interface de entrada de dados para serviços web. É também uma especificação para que serviços e aplicações web se comuniquem com outras aplicações web, embora possa ainda ser utilizada para outras funções. É um padrão Python, escrito sob a *PEP* 333 (BROWN, 2009).

Esta interface foi escrita com o objetivo de fornecer uma forma relativamente simples e compreensiva de comunicação entre aplicações e servidores, ou pelo menos com a maioria das aplicações web em Python, e que ainda pudesse suportar componentes *middleware*.

2.11.1 Paster

Paster se trata de um componente para serviços web, composto pelo Python *Paste*, que segue o padrão da interface Python WSGI.

Possui dois níveis de linha de comando composto inicialmente por *paster*, onde o segundo comando especifica o serviço desejado, como *serve* no caso de estabelecer o servidor, seguindo como parâmetros o restante das informações necessárias para estabelecer o serviço desejado.

É considerado um dos mais simples servidores web para Python, no entanto pode ser utilizado assincronamente e manter uma escalabilidade considerável até 2000*rps*.

2.12 Git

Git é um sistema de controle de versões distribuídas livre e de código aberto, projetado para lidar com qualquer projeto, desde o menor ao maior com rapidez e eficiência (CHACON, 2009).

A historia do Git está muito relacionada a criação do Linux e de Linus Torvalds, seu criador, bem como com toda comunidade de desenvolvimento Linux. Durante anos a comunidade

utilizou a ferramenta *BitKeeper* para guardar a modificações do projeto.

Em 2005, após um problema com a proprietária deste, a comunidade decidiu criar sua própria ferramenta a partir da experiência com a anterior, houve um novo foco em: velocidade, *design* simples, suporte para desenvolvimento paralelo, distribuição completa e a habilidade necessária para lidar com projetos grandes sem perda de velocidade e dados.

Assim, esse novo sistema de versionamento permite que qualquer repositório seja o centro do versionamento, deixando todo *log* das modificações guardados nele sem que para isso precise de uma conexão a rede ou servidor geral.

2.12.1 Git e Subversion

Diferentemente do Git, o Subversion é um sistema de controle de versões centralizado, entretanto muito utilizado atualmente, principalmente por projetos livres.

Embora seja consideravelmente rápido, é extremamente desaconselhável para projetos grandes e principalmente desenvolvidos paralelamente.

3 CLOUDOOO

Em função da necessidade da conversão de documentos a empresa francesa Nexedi SA originou a construção da ferramenta OpenOffice.Org Daemon (OOOD), entretanto seu uso prolongado apresentou erros que precisavam de tratamento e ainda com novas demandas crescendo era necessário a realização de mudanças e até de uma nova ferramenta mais estável. Em parceria com o NSI, foi idealizado o desenvolvimento desta nova ferramenta.

O resultado da continuação do desenvolvimento foi a ferramenta OOOD 2.0, posteriormente nomeada por CloudOoo, apresentada em 2010. Esta nova versão da ferramenta se provou bem mais estável no uso a longo prazo, embora fosse considerada mais lenta em processos individuais, devido aos tratamentos adicionados para os erros conhecidos, e suas modificações de novas funcionalidades.

Ao final do processo de melhoria da ferramenta, novas funcionalidades foram idealizadas para diferentes tipos de arquivos, tais como arquivos de áudio, vídeo, imagem e PDF. A princípio essas funcionalidades seriam as mesmas aplicadas a documentos, conversões e manipulações gerais, vindo a criação de funcionalidades especificas quando a ferramenta fosse considerada estável.

Assim o CloudOoo apresentado neste capítulo, é um serviço Web livre e de código aberto, sob a licença LGPL, que foi desenvolvido através parceria da Nexedi e do NSI, na linguagem de programação Python e que utiliza o protocolo XML-RPC para troca de mensagens, que pode ser utilizado inteiramente ou em partes separadas.

3.1 Estrutura

Desde de sua estrutura anterior, o CloudOoo foi desenvolvido para trabalhar de forma genérica prevendo futuras mudanças. Sua estrutura contém as interfaces:

- IApplication: representa os métodos de controles as aplicações externas do servidor;
- IFile: representa métodos para manipulação dos arquivos recebidos;
- IOdfDocument: representa métodos de manipulação específica de documentos ODF;

- IHandler: representa os objetos que irão realizar a requisição emitida pelo cliente;
- IMonitor: representa métodos de controle e manuseio dos processos estabelecidos no servidor;
- IMimemapper: representa métodos utilizados para trabalhar com filtros;
- IFilter: representa métodos de tratamento de filtros;
- ILockable: representa os métodos de controles para região crítica do servidor;
- ITableGranulator: representa métodos para extrair tabelas de documentos;
- IImageGranulator: representa métodos para extrair imagens de documentos;
- ITextGranulator: representa os métodos para extrair o conteúdo de um documento em capítulos e parágrafos;
- IERP5Compability: representa os métodos de compatibilidade com o ERP5;
- IManager: representa os métodos utilizáveis entre cliente e servidor.

As próximas subseções apresentam detalhadamente sobre cada interface e as principais classes a implementá-las.

3.1.1 ILockable

Quando se possui-se um recurso compartilhado, isto é, um recurso representado por outra aplicação rodando em paralelo à aplicação principal, admiti-se também existir uma região crítica.

Esta visão ocorre devido ao fato que determinadas aplicações não conseguem atender a mais de um processo simultaneamente, assim é necessário controlar essa região crítica prevendo travá-la caso um processo já esteja em usando a mesma.

Com este intuito a interface ILockable foi implementada.

3.1.2 IApplication

Por possuir a opção instalação em um ambiente dedicado onde tanto o CloudOoo, quanto suas ferramentas podem possuir instalação própria, a partir do uso do Buildout, foi preciso construir uma interface para controlar as funções dos processos utilizados pela aplicação, ou seja, uma classe que fosse capaz de carregar a configurações das aplicações, controlar a inicialização e finalização de cada processo, e que fosse capaz de verificar se continuavam rodando no sistema operacional, a partir de um identificador e/ou da porta que cada uma utilizasse.

3.1.2.1 Application

Esta classe implementa a interface IApplication e tem por objetivo controlar processos externos que estejam sendo utilizados dentro do CloudOoo, como o por exemplo os processos do LibreOffice que precisam estar iniciados para possibilitar a manipulação dos documentos.

Além dos métodos citados em 3.1.2 esta classe é capaz de apresentar erros ocorridos durante processos e também de retornar o *pid* utilizado pela aplicação. E ainda um método responsável pelo endereço dessa aplicação, ou seja, onde esta estabelecida e em qual porta.

No código 3.1 existe uma parte da implementação desta classe, em que é exibido como retornar o *pid* de cada processo. No código é possível notar o uso da biblioteca *implements* do Zope.Interface, utilizada para implementar a interface IApplication.

```
1  from zope.interface import implements
2  from cloudooo.interfaces.application import IApplication
3
4  class Application(object):
5
6    implements(IApplication)
7
8    name = "application"
9
10   def pid(self):
11     if not hasattr(self, 'process'):
12       return None
13     return self.process.pid
```

Código 3.1: Trecho de código referente a função de pid

3.1.2.2 OpenOffice

A classe OpenOffice foi exclusivamente criada visando controlar a aplicação LibreOffice, anteriormente conhecida por OpenOffice.Org, a qual foi responsável por maior parte dos erros obtidos no CloudOoo.

Esta classe estende o uso da interface IApplication, seção 3.1 e ainda implementa a interface ILockable, 3.1.1, para que assim seja possível controlar a aplicação, trancá-la e destrancá-la durante seu processo de uso.

A partir do método *isLocked*, 3.2, é possível verificar se esta aplicação está trancada ou disponível para uso, evitando assim erros, como o *deadlock* por exemplo. Este método realiza uma consulta por meio da função *locked* implementada pela classe Lock, nativa do Python, e própria para trabalhar com Threads.

```
1  from threading import Lock
2
```

```
3  class OpenOffice(Application):
4
5    implements(ILockable)
6
7    def __init__(self):
8      self._bin_soffice = 'soffice.bin'
9      self._lock = Lock()
10     self._cleanRequest()
11
12   def isLocked(self):
13     return self._lock.locked()
```

Código 3.2: Trecho isLocked da classe OpenOffice

3.1.3 IFile

Esta interface propõe um contrato de tratamento de arquivos, a fim de assegurar uma resposta eficiente e consistente ao cliente. Nela são contidos métodos para que o conteúdo do arquivo seja guardado durante sua instânciação de forma no acontecimento de erros não previstos este conteúdo pode ser recuperado, ou mesmo restaurado a forma original.

3.1.3.1 File

Com base na implementação da interface IFile, esta classe possui métodos para manter qualquer arquivo recebido do cliente no sistema apenas durante o uso do mesmo.

Ao receber um arquivo ela escreve o mesmo no disco, podendo assim recuperar seus dados, e obter informações do mesmo, como seu caminho, por exemplo. No código 3.3 existe uma representação de como cada File é recriado.

```
1  import tempfile
2  from zope.interface import implements
3  from cloudooo.interfaces.file import IFile
4
5  class File(object):
6
7    implements(IFile)
8
9    def __init__(self, base_folder_url, data, source_format):
10     self.base_folder_url = base_folder_url
11     self.directory_name = self._createDirectory()
12     self.original_data = data
13     self.source_format = source_format
14     self.url = self.load()
15
16   def _createDirectory(self):
17      return tempfile.mkdtemp(dir=self.base_folder_url)
```

Código 3.3: Trecho de criação da classe File

Para identificar o diretório do File, é utilizado o base folder url o qual se refere a pasta base informada pelo próprio usuário em questão, além do *mkdtemp* que cria um diretório temporário para esta nova instância.

Após o uso deste arquivo, o File é instituído a remover a instância do sistema, bem como qualquer arquivo criado a partir desta a fim de não esgotar o servidor com arquivos desnecessários.

3.1.4 IOdfDocument

Embora muito similar a interface IFile, seção 3.1.3, porém seu tratamento é especifico para documentos ODF, dada a complexidade de armazenamento e manipulação destes.

3.1.4.1 Document

Por ter sido inicialmente desenvolvido para documentos ODF a estrutura do CloudOoo é relativamente planejada para estes, que por possuírem uma estrutura complexa e compacta exigiram a criação de classes especificas.

Como no caso do Document, o qual tem seus métodos desenhados com base em estudos anteriores sobre estruturas XML e sobre a manipulação de documentos ODF.

Assim, no código 3.4, esta um trecho do código desta classe que visa a estruturação de um documento ODF com base no uso do ZipFile para ordernar seu XML através da obtenção do conteúdo de “content.xml”.

```
1  from zope.interface import implements
2  from zipfile import ZipFile
3  from StringIO import StringIO
4  from lxml import etree
5  from cloudooo.interfaces.file import IOdfDocument
6
7
8  class OdfDocument(object):
9
10   implements(IOdfDocument)
11
12   def __init__(self, data, source_format):
13     self._zipfile = ZipFile(StringIO(data))
14     self.source_format = source_format
15     self.parsed_content = etree.fromstring(self.getContentXml())
16
17   def getContentXml(self):
18     return self._zipfile.read('content.xml')
```

Código 3.4: Trecho de estruturação do content.xml

3.1.5 IMonitor

Esta interface foi desenvolvida principalmente com base nos erros anteriormente obtidos com o uso do LibreOffice, no entanto é importante para o sistema como um todo.

Seu uso estabelece controles sobre princípios básicos do sistema, como uso de memória, tempo de requisições, tempo de uso do processo, entre outros.

3.1.5.1 Monitor

Basicamente a classe Monitor funciona como uma simples implementação da interface IMonitor a fim de estabelecer seus atributos principais, como por exemplo o tempo mínimo entre a monitorações do sistema(*interval*), que se pode notar no código 3.5.

Além deste, outro atributo essencial é a instância de OpenOffice, 3.1.2.2, através da variável *openoffice*.

```
1  zope.interface import implements
2  from cloudooo.interfaces.monitor import IMonitor
3
4  class Monitor(object):
5
6    implements(IMonitor)
7
8    def __init__(self, openoffice, interval):
9      self.status_flag = False
10     self.openoffice = openoffice
11     self.interval = interval
```

Código 3.5: Inicialização da classe Monitor

As próximas subseções explicam detalhadamente sobre estes controles através da herança desta classe.

3.1.5.2 MonitorMemory

Nas configurações do CloudOoo existem definições que podem ser modificadas de acordo com o sistema em que vai ser instalado, entre elas existe uma variável responsável pelo uso máximo de memória pelo LibreOffice.

A partir dessa definição, dada em *megabytes*, a MonitorMemory monitora o uso da memória do sistema pela variável *limit*, vista em 3.6. Assim caso este limite máximo seja atingido, a aplicação é reinicia com intuito de limpar da memória mensagens trocadas e que não foram liberadas de uso da mesma, evitando assim o evento chamado *memory leak*, o qual consiste no uso de toda memória do sistema.

```
from monitor import Monitor
from multiprocessing import Process
from time import sleep

class MonitorMemory(Monitor, Process):

  def __init__(self, openoffice, interval, limit_memory_usage):
    Monitor.__init__(self, openoffice, interval)
    Process.__init__(self)
    self.limit = limit_memory_usage
```

Código 3.6: Trecho de criação da classe MonitorMemory

Além de Monitor, 3.1.5.1, estas subclasses também implementam a classe Process, nativa do Python, que tem por função ser o controle dos processos iniciados no sistema pela aplicação;

3.1.5.3 MonitorTimeout

Através da definição citada na subseção 3.1.5.2, o MonitorTimeout realiza uma comparação de tempo ativo da aplicação em função do *interval* para estabelecer o chamado *timeout*, isto é, o tempo limite de execução de um determinado processo.

Caso este tempo seja excedido a aplicação é forçada a parar, sendo reiniciada posteriormente, 3.7.A utilidade desta limitação é dada pela idéia de que caso este tempo tenha excedido ocorreu algum erro durante o processo, provavelmente em função da resposta da aplicação, assim reiniciá-la pode resolvê-lo.

```
from monitor import Monitor
from multiprocessing import Process
from time import sleep

class MonitorTimeout(Monitor, Process):

  def run(self):
    sleep(self.interval)
    if self.openoffice.isLocked():
      self.openoffice.stop()
```

Código 3.7: Método run da classe MonitorTimeout

A utilidade desta limitação é dada pela idéia de que caso este tempo tenha excedido ocorreu algum erro durante o processo, provavelmente em função da resposta da aplicação, neste caso o OpenOffice. Assim reiniciá-la pode resolver tal erro.

3.1.5.4 MonitorSleepingTime

Com intuito de poupar uso do sistema em momentos desnecessários esta classe foi criada para observar o momentos de inutilização da aplicação e para parar a mesma nestes momentos.

A partir de uma definição inicial para o tempo de inutilização, definida na instalação do CloudOoo (sleeping time), o MonitorSleepingTime "trancar" a aplicação, linha 11, e para sua execução para enfim destrancá-lo caso algum processo queira utilizar a aplicação, 3.8.

Assim é possível economizar no uso de recursos, e disponibilizá-los para outras aplicações que possam vir a utilizar o mesmo.

```
1  from monitor import Monitor
2  from threading import Thread
3  from time import sleep, time
4
5  class MonitorSleepingTime(Monitor, Thread):
6
7    def run(self):
8      while self.status_flag:
9        current_time = time()
10       if self.openoffice.status() and\
11         (self._touched_at + self.sleeping_time) <= current_time:
12         self.openoffice.acquire()
13         self.openoffice.stop()
14         self.openoffice.release()
15       sleep(self.interval)
```

Código 3.8: Método run da classe MonitorSleepingTime

3.1.5.5 MonitorRequest

A fim de conservar a estabilidade do CloudOoo, o MonitorRequest implementa um controle em função do valor máximo de requisições (request limit), isto é, um número máximo de requisições do cliente que podem ser respondidas por cada instância da aplicação no servidor.

Caso o valor informado seja excedido a instância é encerrada, em seguido uma nova instância da mesma aplicação é iniciada, como visto em 3.9.

```
1  from monitor import Monitor
2  from threading import Thread
3  from time import sleep
4
5
6  class MonitorRequest(Monitor, Thread):
7
8    def run(self):
9      while self.status_flag:
10       if self.openoffice.request > self.request_limit:
11         self.openoffice.acquire()
```

```
12              self.openoffice.getAddress())
13            self.openoffice.restart()
14            self.openoffice.release()
15          sleep(self.interval)
```

Código 3.9: Método run da classe MonitorRequest

3.1.6 IMimemaper

Em casos de aplicações como o LibreOffice que representam uma suíte de menores utilitários é preciso reconhecer a extensão de arquivo específica para cada utilitário.

De forma geral estas extensões são explicitas no nome do arquivo. Entretanto, caso de que esta não sejam explicitas, é preciso reconhecer o tipo de arquivo de alguma outra forma.

Neste caso existem o *mimetypes*, que são identificadores presentes no conteúdo do arquivo, que permitem decidir sua extensão.

Esta interface propõe métodos ara lidar com a identificação desses *mimetypes*.

Seu exemplo de uso é a classe Mimemapper a qual será apresentada na próxima subseção.

3.1.6.1 Mimemapper

O CloudOoo possui seus próprio filtros para identificar e renderizar arquivos dentro de sua própria instância.

No entanto, dada a necessidade de suas aplicações internas, tornou-se necessário identificá-los de forma a torná-los igualmente reconhecível dentro de cada aplicação específica.

No caso do LibreOffice é possível, através do uso do UNO, extrair os *mimetypes* e demais informações como filtros próprios para esta aplicação. Um vez extraídos é importante definí-los numa inicialização, 3.10.

```
1  from zope.interface import implements
2  from cloudooo.interfaces.mimemapper import IMimemapper
3
4  class MimeMapper(object):
5
6    implements(IMimemapper)
7
8    def __init__(self):
9      self._loaded = False
10     self._filter_by_extension_dict = {}
11     self._extension_list_by_type = {}
12     self._doc_type_list_by_extension = {}
13     self._mimetype_by_filter_type = {}
```

```
14      self._document_type_dict = {}
```

Código 3.10: Trecho de criação da classe Mimemapper

Assim uma vez que tenham sido definidos os filtros não é necessário saber a extensão do arquivo por extenso.

3.1.7 IFilter

Citados na seção anterior, 3.1.6.1, os filtros podem ter demais propriedades que ao serem requisitadas precisam estar disponível de forma facilmente utilizável.

Com base neste principio de utilização, esta interface propõe um contrato para trabalhar com os filtros mais complexos da melhor forma possível e igualmente da forma mais simples.

3.1.7.1 Filter

Se comparada as outras classes e suas devidas interfaces, a classe Filter pode ser considerada a que representa um dos métodos mais simples.

Ao ser iniciada ela guarda todos os dados de cada filtro em diversos atributos que foram selecionados prevendo seu uso posterior na aplicação, 3.11.

```
1  from zope.interface import implements
2  from cloudooo.interfaces.filter import IFilter
3
4  class Filter(object):
5
6    implements(IFilter)
7
8    def __init__(self, extension, filter, mimetype, document_service, **
       kwargs):
9      self._extension = extension
10     self._filter = filter
11     self._mimetype = mimetype
12     self._document_service = document_service
13     self._preferred = kwargs.get('preferred')
14     self._sort_index = kwargs.get('sort_index')
15     self._label = kwargs.get("label")
16
17   def getName(self):
18     return self._filter
19
20   def getDocumentService(self):
21     return self._document_service
```

Código 3.11: Trecho de criação da classe File e método getDocumentService

Apesar de simples essa classe possui métodos de extrema importância, como *getDocumentService*, linha 10 3.11, que no caso do LibreOffice retornar qual aplicação será utilizado

para aquela instância.

3.1.8 IHandler

Esta interface foi especificamente criada para estabelecer o contrato entre as aplicações externas utilizadas pelo servidor em função dos pedidos do cliente no que desrespeito a manipulação direta do arquivo, como no caso de conversões por exemplo, ou então extrações e inserções de metadados, respectivamente os métodos de *convert, getMetadata* e *setMetadata*.

Para as classe que implementam esta interface é recomendado o igual uso de objetos do tipo File, seção 3.3, para manipulação dos tipos de arquivos de cada uma dessas.

3.1.8.1 OOHandler

Inicialmente nomeado em função do OpenOffice.Org, este *handler* é uma implementação específica de comunicação com o LibreOffice, que trata de requisições especificas a documentos a fim de manipulá-los.

```
1  from cloudooo.interfaces.handler import IHandler
2  from cloudooo.handler.ooo.mimemapper import mimemapper
3  from cloudooo.file import File
4  from cloudooo.handler.ooo.monitor.timeout import MonitorTimeout
5  from cloudooo.handler.ooo.monitor import monitor_sleeping_time
6  from psutil import pid_exists
7
8
9  class Handler(object):
10
11   implements(IHandler)
12
13   def __init__(self, base_folder_url, data, source_format, **kw):
14     self.document = File(base_folder_url, data, source_format)
15     self.zip = kw.get('zip', False)
16     self.uno_path = kw.get("uno_path", None)
17     self.office_binary_path = kw.get("office_binary_path", None)
18     self.timeout = kw.get("timeout", 600)
19     self.refresh = kw.get('refresh', False)
20     self.source_format = source_format
21     if not self.uno_path:
22       self.uno_path = environ.get("uno_path")
23     if not self.office_binary_path:
24       self.office_binary_path = environ.get("office_binary_path")
```

Código 3.12: Trecho de criação da classe OOHandler

No código 3.12 é possível notar a incorporação de outras classes como File 3.3, utilizada para salvar o arquivo recebido em sistema; MimeMapper 3.1.6.1 para controlar os filtros do LibreOffice; e MonitorTimeout 3.1.5.3 para controlar o tempo gasto pela aplicação para realizar

suas funcionalidades.

3.1.8.2 PDFHandler

Por utilizar de duas ferramentas, o Poppler e PDFTk, esta classe foi nomeada em função do tipo de arquivo que é responsável, ou seja, arquivos PDF.

```
1  from zope.interface import implements
2  from cloudooo.interfaces.handler import IHandler
3  from cloudooo.file import File
4
5  class Handler(object):
6
7    implements(IHandler)
8
9    def __init__(self, base_folder_url, data, source_format, **kw):
10     self.base_folder_url = base_folder_url
11     self.document = File(base_folder_url, data, source_format)
12     self.environment = kw.get("env", {})
```

Código 3.13: Trecho de criação da classe PDFHandler

No código 3.13 é possível notar que a implementação desta classe é bem mais simples que a OOHandler 3.12, por exemplo. Nela são especificados apenas o diretório (base folder url), os dados do arquivo (*document*) e o PATH deste sistema (*environment*), que apontará para os binários do Poppler e PDFTk.

3.1.8.3 IMAGEMAGICKHandler

No que se trata de ferramentas para imagens o ImageMagick é uma das melhores disponível a nível de comando, ela consegue inclusive manipular dados do tipo *Exif*, que são metadados referentes a imagens.

Assim com base na ferramenta que utiliza, esta classe é responsável pela conversão e extração de metadados de arquivos de imagem, visto em 3.14. Deseja-se ainda implementar uma funcionalidade de inserção destes, no entanto esta funcionalidade requer uma base de estudos maior.

```
1  from zope.interface import implements
2  from cloudooo.interfaces.handler import IHandler
3
4  class Handler(object):
5
6    def getMetadata(self, base_document=False):
7      """Returns a dictionary with all metadata of document.
8      along with the metadata.
9      """
10     command = ["identify", "-verbose", self.file.getUrl()]
```

```
11      stdout, stderr = Popen(command,
12                             stdout=PIPE,
13                             stderr=PIPE,
14                             close_fds=True,
15                             env=self.environment).communicate()
16      metadata_dict = {}
17      for std in stdout.split("\n"):
18        std = std.strip()
19        if re.search("^[a-zA-Z]", std):
20          if std.count(":") > 1:
21            key, value = re.compile(".*\:\ ").split(std)
22          else:
23            key, value = std.split(":")
24          metadata_dict[key] = value.strip()
25      self.file.trash()
26      return metadata_dict
```

Código 3.14: Método getMetadata da classe IMAGEMAGICKHandler

3.1.8.4 FFMPEGHandler

O FFMPEG é capaz de manipular arquivos de áudio e vídeo facilitando assim a criação de uma classe que pudesse ser responsável simultaneamente pela manipulação de ambos.

Assim como as demais classes que implementam o IHandler, esta classe é capaz de manipular os metadados desses arquivos, bem como convertê-los para determinadas extensões do mesmo tipo.

Sendo o FFMPEGHandler uma contribuição direta deste trabalho, é possível encontrá-lo detalhadamente em 3.18

3.1.9 ITableGranulator, IImageGranulator, ITextGranulator

Uma das principais funções desenvolvidas para documentos foi a "granularização", ela trata da extração de partes importantes de documentos que não sejam especificamente textos, como por exemplo tabelas e imagens.

Dada a complexidade dessa tarefa foi necessário a implantação das novas interfaces, para que estes processos fossem realizados.

A interface ITableGranulator é a interface responsável pelo processo de "granularização" específico de tabelas presentes nos documentos. Ela implementa funções respectivas a uma tabela comum, baseada nas linhas e colunas dessas tabelas.

Na interface IImageGranulator ocorre a "granularização" das imagens presentes nestes documentos, que são extraídas em seu formato original, ou em *PNG*.

Por fim através da interface ITextGranulator é possível partir o documento em textos menores dividindo o mesmo a partir de seus parágrafos, ou mesmo em capítulos.

Apesar dessas funcionalidades tratarem diretamente da "granularização" de documentos, podemos subdividi-las em dois tipos de documentos: os documentos livres de formato ODF, onde o processo ocorre a partir estudos realizados anteriormente dos mesmos; e documentos PDF, que são igualmente livres, porém tem um tratamento complexo dada sua falta de detalhamento e especificações.

3.1.9.1 OOGranulator

Esta classe foi desenvolvida especialmente para tratar do documentos ODF.

Suas funcionalidades são escritas com base nos *namespaces* encontrados por padrão no XML que compõe esses documentos, no código 3.15 é possível ver alguns exemplos desses.

```
TEXT_URI = 'urn:oasis:names:tc:opendocument:xmlns:text:1.0'
TABLE_URI = 'urn:oasis:names:tc:opendocument:xmlns:table:1.0'
DRAWING_URI = 'urn:oasis:names:tc:opendocument:xmlns:drawing:1.0'

TABLE_ATTRIB_NAME = '{%s}name' % TABLE_URI
TEXT_ATTRIB_STYLENAME = '{%s}style-name' % TEXT_URI
DRAWING_ATTRIB_STYLENAME = '{%s}style-name' % DRAWING_URI
DRAWING_ATTRIB_NAME = '{%s}name' % DRAWING_URI
```

Código 3.15: URI e ODF Namespaces

Até o momento essa classe é a única a implementar as exatas três interfaces da subseção anterior.

No trecho abaixo, 3.16, é possível notar como funciona a pesquisa desta classe por uma tabela especifica a fim de retorná-la ao cliente.

```
  def getTableMatrix(self, id):
    row_list = self.document.parsed_content.xpath(
                          '//table:table[@table:name="%s"]/table:table-row' %
                              id,
                          namespaces=self.document.parsed_content.nsmap)
    if len(row_list) == 0:
      return None

    matrix = []
    for row in row_list:
      matrix_row = []
      for cell in row.iterchildren():
        matrix_row.append(''.join(cell.itertext()))
      matrix.append(matrix_row)
    return matrix
```

Código 3.16: Método de getTableMatrix

3.1.9.2 PDFGranulator

Esta classe implementa apenas as interfaces ITableGranulator e IImageGranulator, **??**.

É específica para o tratamento de documentos PDF, que exigem a utilização de bibliotecas e aplicações para "desmembrá-lo" e tratar o resultado desta tarefa.

Por não possuir tantas especificações quanto documentos ODF, tratá-lo para "granularização" foi uma das contribuições mais complicadas deste trabalho, e no entanto com resultados não tão desejáveis.

3.1.10 IManager

A IManager trabalha como a interface entre o cliente e servidor a fim de estabelecer um protocolo entre ambos.

Ela detém um padrão genérico para troca de informações entre diferentes tipos de arquivos, inclusive vídeos.

E possui ainda os principais métodos para funcionalidades do CloudOoo, no que desrespeito a todos seus *handlers*, ou seja, módulos para tratar diversos tipos de arquivos.

3.1.11 IERP5Compability

Esta interface estabelece as funcionalidades entre cliente e servidor especificamente para o uso da aplicação ERP5.

Ela reescreve a chamada dos métodos da aplicação OOOD para utilizarem os novos métodos sem interferir nas requisições feitas pelo uso do ERP5, e outros possíveis clientes que utilizassem a antiga plataforma.

3.1.11.1 Manager

A classe Manager implementa as classes IManager, IERP5Compability, ITableGranulator, ITextGranulator e IImageGranulator com o propósito de interligar suas funcionalidades ao cliente que venha a requeri-las, em outras palavras esta classe é a principal responsável pela conectividade entre cliente, servidor, aplicação e funcionalidades.

Nela são absorvidos os dados importantes para o funcionamento do CloudOoo, como por exemplo a base de *mimetypes*, os *handlers* disponíveis neste, as pastas de trabalho deste, entre outros dados.

Assim a partir desta classe é possível iniciar o servidor do CloudOoo, 3.17, nela são

especificados a pasta de temporários do CloudOoo (path tmp dir), os *mimetypes* registrados (mimetype registry), a lista de *handlers* disponíveis nesta instância do CloudOoo (handler dict) e por fim a demais variáveis por um dicionário (*kw*).

```
1 class Manager(object):
2   implements(IManager, IERP5Compatibility, ITableGranulator,
      IImageGranulator,
3              ITextGranulator)
4
5   def __init__(self, path_tmp_dir, **kw):
6     self._path_tmp_dir = path_tmp_dir
7     self.kw = kw
8     self.mimetype_registry = self.kw.pop("mimetype_registry")
9     self.handler_dict = self.kw.pop("handler_dict")
```

Código 3.17: Inicialização do Manager

3.2 Novas Funcionalidades

Nesta seção encontram-se a funcionalidades diretamente desenvolvidas para o CloudOoo por este trabalho.

Abaixo são listadas todas as contribuições deste trabalho:

- Criação da classe FFMPEGHandler para manipulação de vídeos e áudio;
- Criação do módulo cloudoooTestCase para testes de serviço;
- Criação do módulo externo CloudOooTestnode;
- Criação do módulo externo cloudooo buildout;
- Manutenção de classes para fins de migração;
- Criação da classe PDFGranulator;
- Criação do método granulateFile para granularização de documentos;
- Manutenção e extensão de código e testes.

As próximas subseções apresentam detalhadamente cada contribuição citada:

3.2.1 Criação da classe FFMPEGHandler para manipulação de vídeos e áudio

Esta foi a primeira funcionalidade desenvolvida por este trabalho e que pode ser explicita desde o “nível zero". Assim pretende-se dizer que o CloudOoo não detinha qualquer parte escrita ou planejada para conversão e manipulação de vídeos e áudio antes desta funcionalidade.

3.2.1.1 Conversão

Como o desenvolvimento de todos os *handlers*, a primeira parte deve tratar da conversão direta do tipo de arquivo, que neste caso utiliza a ferramenta FFMPEG:

```
from zope.interface import implements
from cloudooo.interfaces.handler import IHandler
from cloudooo.file import File
from subprocess import Popen, PIPE
from tempfile import NamedTemporaryFile

class Handler(object):

  implements(IHandler)

  def __init__(self, base_folder_url, data, source_format, **kw):
    self.base_folder_url = base_folder_url
    self.input = File(base_folder_url, data, source_format)
    self.environment = kw.get("env", {})

  def convert(self, destination_format):
    output_url = NamedTemporaryFile(suffix=".%s" % destination_format,
                        dir=self.input.directory_name).name
    command = ["ffmpeg", "-i", self.input.getUrl(), "-y",output_url]
    if destination_format == "webm":
      command.insert(3, "32k")
      command.insert(3, "-ab")
    try:
      stdout, stderr = Popen(command,
                             stdout=PIPE,
                             stderr=PIPE,
                             close_fds=True,
                             env=self.environment).communicate()
      self.input.reload(output_url)
      if len(self.input.getContent()) == 0:
        logger.error(stderr.split("\n")[-2])
      return self.input.getContent()
    finally:
      self.input.trash()
```

Código 3.18: Conversão do FFMPEGHandler

Como por padrão, o FFMPEGHandler utiliza o File para gravar seus dados em sistema, e para obter o arquivo da pós conversão (output url). Para realizar a conversão propriamente

dita utiliza-se a biblioteca Subprocess para que um processo com a ferramenta externa seja realizado, este processo retornará duas mensagens *stdout* e *stderr*, caso o processo seja realizado com sucesso apenas o *stdout* retornará um valor, caso haja um erro o *stderr* poderá explicar qual o erro encontrado.

3.2.1.2 Extração de metadados

Para extraçao de metadados utilizasse um diferente binário, que também pertence a aplicação FFMPEG, neste caso o *ffprobe* retornará apenas os dados do arquivo em questão, como pode-se notar em 3.19:

```
1  def getMetadata(self, base_document=False):
2    command = ["ffprobe",self.input.getUrl()]
3    stdout, stderr =  Popen(command,
4                            stdout=PIPE,
5                            stderr=PIPE,
6                            close_fds=True,
7                            env=self.environment).communicate()
8    metadata = stderr.split('Metadata:')[1].split('\n')
9    metadata_dict = {}
10   for data in metadata:
11     if len(data) != 0:
12       key, value = data.split(':')
13       metadata_dict[key.strip().capitalize()] = value.strip()
14   self.input.trash()
15   return metadata_dict
```

Código 3.19: getMetadata do FFMPEGHandler

A lista retornada pelo processo na mensagem do *stderr* é tratada e transformada num dicionário para retornar ao cliente.

3.2.1.3 Inserção de metadados

Para a inserção de metadados utilizasse novamente a ferramenta FFMPEG e a biblioteca Subprocess, da mesma forma que para uma conversão, nesta funcionalidade entretanto não há a troca de extensões, apenas a inserção de informações no arquivo recebido no servidor, 3.20.

```
1  def setMetadata(self, base_document=False):
2    output_url = NamedTemporaryFile(suffix=".%s" % destination_format,
3                                    dir=self.input.directory_name).name
4    command = ["ffmpeg", "-i", self.input.getUrl(), "-y",output_url]
5    for metadata in metadata_dict:
6      command.insert(3, "%s=%s"%(metadata, metadata_dict[metadata]))
7      command.insert(3, "-metadata")
8    try:
9      stdout, stderr = Popen(command,
10                            stdout=PIPE,
```

```
                                    stderr=PIPE,
                                    close_fds=True,
                                    env=self.environment).communicate()
        self.input.reload(output_url)
        return self.input.getContent()
      finally:
        self.input.trash()
```

Código 3.20: setMetadata do FFMPEGHandler

O cliente receberá um arquivo idêntico ao enviado, porém com os dados desejados inseridos.

3.2.2 Criação do módulo cloudoooTestCase para testes de serviço

Com a criação de tantos *handlers* simultaneamente notou-se a necessidade de que fosse estabelecido um padrão para os testes destes a fim de que não houvessem repetições demasiadas neste processo. Para

```
class TestCase(backportUnittest.TestCase):

  def setUp(self):
    server_cloudooo_conf = environ.get("server_cloudooo_conf", None)
    if server_cloudooo_conf is not None:
      config.read(server_cloudooo_conf)
    self.hostname = config.get("server:main", "host")
    self.port = config.get("server:main", "port")
    self.env_path = config.get("app:main", "env-path")
    #create temporary path for some files
    self.working_path = config.get("app:main", "working_path")
    self.tmp_url = path.join(self.working_path, "tmp")
    self.proxy = ServerProxy(("http://%s:%s/RPC2" % (self.hostname, self.
        port)),\
                  allow_none=True)
    self.afterSetUp()
```

Código 3.21: setUp do cloudoooTestCase

Com a criação do método setUp, por exemplo, toda parte de inicialização do servidor para os testes pode ser "cortada", isto por que sempre que fosse realizar um teste seria necessário informar ao mesmo as variáveis de sistema, com este método entretanto, elas passam a ficar guardadas enquanto os testes precisarem das mesmas.

Outra parte bem reaproveitada se trata dos testes de conversões que antes precisariam ser repetidos a cada teste, com o método testConvertFile, visto em 3.22, basta passar um cenário em forma de lista, com a relação das conversões e este método realizará todas e em seguida informará caso falhem.

```
  def _testConvertFile(self, input_url, source_format, destination_format,
                       destination_mimetype, zip=False):
```

```
    fault_list = []
    try:
      output_data = self.proxy.convertFile(encodestring(open(input_url).
          read()),
                                                    source_format,
                                                    destination_format,
                                                    zip)
      file_type = self._getFileType(output_data)
      if destination_mimetype != None:
        self.assertEquals(file_type, destination_mimetype)
      else:
        if file_type.endswith(": empty"):
          fault_list.append((source_format, destination_format, file_type))
    except Fault, err:
      fault_list.append((source_format, destination_format, err.faultString
          ))
    if fault_list:
      template_message = 'input_format: %r\noutput_format: %r\n traceback:\
          n%s'
      message = '\n'.join([template_message % fault for fault in fault_list
          ])
      self.fail('Failed Conversions:\n' + message)
```

Código 3.22: testConvertFile do cloudoooTestCase

3.2.3 Criação do módulo externo CloudOooTestnode

Outro módulo criado especificamente para testes do CloudOoo foi o CloudOooTestnode.

Entretanto este módulo ultrapassa a simplicidade de apenas realizar os testes do CloudOoo, ele é responsável por criar uma instância do CloudOoo num ambiente dedicado, com base em passos próprios, e nesta nova instância rodar todos os testes deste em sequência e sem intervalos.

Seu processo foi considerado tão longo e complexo que foi considerado por si só um projeto externo e até mesmo uma nova ferramenta utilizada exclusivamente para literalmente testar o CloudOoo, sempre em sua versão mais recente extraída diretamente de seu repositório de desenvolvimento.

Atualmente o uso desta ferramenta encontra-se restrito através do uso do SlapOS 4.1.3.

3.2.4 Criação do módulo externo cloudooo buildout

O cloudooo buildout é um módulo externo do CloudOoo, estabelecido no GitHub do NSI, com o intuito de promover uma instalação mais simples e direta deste sem que fosse necessário o uso do SlapOS 4.1.3.

```
[buildout]
extends =
```

```
    profiles/libreoffice-bin.cfg
    profiles/libpng.cfg
    profiles/lxml-python.cfg
    profiles/python-2.6.cfg
    profiles/xorg.cfg
    profiles/fonts.cfg
    profiles/poppler.cfg
    profiles/imagemagick.cfg
    profiles/pdftk.cfg
    profiles/xpdf.cfg
    profiles/ffmpeg.cfg
    profiles/file.cfg
    profiles/rdiff-backup.cfg
    profiles/supervisor.cfg

parts =
    create-directories
    libreoffice-bin
    libXdmcp
    libXext
    libXau
    libSM
    libXrender
    liberation-fonts
    ipaex-fonts
    libpng12
    imagemagick
    file
    poppler
    xpdf
    pdftk
    ffmpeg
    bootstrap2.6
    rdiff-backup
    supervisor
    cloudooo-repository
    template
    cloudooo-instance

develop =
    ${:parts-directory}/cloudooo

var-directory = ${:directory}/var
etc-directory = ${:var-directory}/etc
log-directory = ${:var-directory}/log
run-directory = ${:var-directory}/run


[create-directories]
recipe = z3c.recipe.mkdir
paths =
    ${buildout:var-directory}
    ${buildout:etc-directory}
    ${buildout:log-directory}
    ${buildout:run-directory}

[bootstrap2.6]
python = python2.6
```

```
61
62
63 [template]
64 recipe = z3c.recipe.template
65 input = ${buildout:directory}/cloudooo.cfg.in
66 output = ${buildout:etc-directory}/cloudooo.cfg
67 working_path = ${buildout:run-directory}
68 uno_path = ${buildout:parts-directory}/libreoffice-bin/basis-link/program/
69 office_binary_path = ${buildout:parts-directory}/libreoffice-bin/program/
70 openoffice_port = 23060
71 ip = 0.0.0.0
72 port = 23000
73 PATH = ${buildout:parts-directory}/xpdf/bin:${buildout:parts-directory}/
       imagemagick/bin:${buildout:parts-directory}/ffmpeg/bin:${buildout:parts-
       directory}/pdftk/bin:${buildout:parts-directory}/poppler/bin:${buildout:
       parts-directory}/ghostscript/bin
74 LD_LIBRARY_PATH = ${buildout:parts-directory}/file/lib:${buildout:parts-
       directory}/zlib/lib:${buildout:parts-directory}/freetype/lib:${buildout:
       parts-directory}/libXext/lib:${buildout:parts-directory}/libXau/lib:${
       buildout:parts-directory}/libX11/lib:${buildout:parts-directory}/
       libXdmcp/lib:${buildout:parts-directory}/libxcb/lib
75
76
77 [cloudooo-repository]
78 recipe = git-recipe
79 repository = https://www.github.com/nsi-iff/cloudooo.git
80 newest = True
81
82 [cloudooo-instance]
83 recipe = zc.recipe.egg
84 python = python2.6
85 interpreter = pycloudooo
86 eggs =
87   ${lxml-python:egg}
88   PasteScript
89   python-magic
90   PIL
91   psutil
92   WSGIUtils
93   cloudooo
94 entry-points =
95   main=cloudooo.paster_application:application
96   cloudooo_tester=cloudooo.bin.cloudooo_tester:main
97   runCloudoooUnitTest=cloudooo.tests.runHandlerUnitTest:run
98   runCloudoooTestSuite=cloudooo.tests.runTestSuite:run
99 scripts =
100   paster=cloudooo_paster
101   runCloudoooUnitTest
102   runCloudoooTestSuite
```

Código 3.23: Arquivo de configuração do cloudooo buildout

A partir do uso do Buildout e do arquivo *buildout.cfg* 3.23, disponível junto ao download deste módulo, uma instância do CloudOoo é construída no computador desejado a parti de comandos básicos que podem ser vistos na seção 4.4.2.

Essa instância contará não só com o CloudOoo, como todas suas dependências instala-

das isoladamente do restante do sistema.

3.2.5 Manutenção de classes para fins de migração

Por ser uma aplicação em nuvem, o CloudOoo não precisa de manutenção direta para migrações das ferramentas atuais por ferramentas mais novas, entretanto uma vez que essas modificações podem influenciar em estabilidade e melhorias gerais elas são realizadas a medida do possível.

Neste trabalho foram realizadas pequenas modificações que poderiam influenciar diretamente no uso do python2.6 em função do python3.0, exemplo no código 3.24 e algumas modificações em função das constantes mudanças do LibreOffice.

No código 3.25 estão alguns exemplos dessas modificações, nele as linhas que começam com — representam a parte retirada, e as linhas iniciadas por +++ representam a parte nova reescrita.

```
---     output_url = mktemp(suffix=".%s" % self.input.source_format,
---                         dir=self.input.directory_name)

+++     output_url = NamedTemporaryFile(suffix=".%s" % self.input.
    source_format,
+++                         dir=self.input.directory_name).name
```

Código 3.24: Exemplo de modificação para Python 3

```
     self.command = [join(self.office_binary_path, self._bin_soffice),
+++             '--headless',
+++             '--invisible',
---             '--headless',
---             '-invisible',
            '-nocrashreport',
+++             '--nologo',
+++             '--nodefault',
+++             '--norestore',
+++             '--nofirststartwizard',
---             '-nologo',
---             '-nodefault',
---             '-norestore',
---             '-nofirststartwizard',
+++             '--accept=socket,host=%s,port=%d;urp;' % (self.hostname, self.
    port),
---             '-accept=socket,host=%s,port=%d;urp;' % (self.hostname, self.
    port),
            '-env:UserInstallation=file://%s' % self.path_user_installation,
+++             '--language=%s' % self.default_language,
---             '-language=%s' % self.default_language,
            ]
```

Código 3.25: Exemplo de modificação pra LibreOffice

3.2.6 Criação da classe PDFGranulator

Na versão 1.24 do CloudOoo já existia a "granularização" de documentos comuns, entretanto esta não atendia a documentos PDF, assim neste trabalho esta funcionalidade foi desenvolvida.

Para documentos a ferramenta utilizada era o libreoffice, a qual foi estudado o igual uso no começo deste desenvolvimento, mas por não atender de forma adequada foi trocada pela ferramenta Poppler que melhor representa uma suíte de menores ferramentas.

No que respeito a imagens o Poppler fornece a ferramenta *pdftoimages* e para o tratamento de textos existem *pdftotext* e *pdftohtml*.

```
class PDFGranulator(object):

  def __init__(self, base_folder_url, data, source_format, **kw):
    self.file = File(base_folder_url, data, source_format)
    self.environment = kw.get("env", {})
    self.grain_directory = mkdtemp(dir=self.file.directory_name)

  def getImageItemList(self):
    command = ["pdftohtml", self.file.getUrl(), "%s/"%self.grain_directory]
    stdout, stderr = Popen(command,
                           stdout=PIPE,
                           stderr=PIPE,
                           close_fds=True,
                           env=self.environment).communicate()
    # XXX - PDF can be protect
    if 'Erro' in stderr:
      return False
    else:
      removeEqualImages(self.grain_directory)
      images = glob("%s/*.*"%self.grain_directory)
      imagesList = getImages(images)
      return imagesList
```

Código 3.26: método getImageItemList do PDFGranulator

Após uma avaliação dessas ferramentas entretanto, apenas a ferramenta *pdftohtml* foi utilizada para ambos os processos. No caso da "granularização" de imagens esta ferramenta separava corretamente as imagens numa pasta onde essas eram avalizadas pelo método *removeEqualImages* e removidas caso fossem iguais, 3.26.

```
 def getTablesMatrix(self):
    output_url = NamedTemporaryFile(suffix=".xml",dir=self.file.
        directory_name).name
    command = ["pdftohtml", "-xml",  self.file.getUrl(), output_url]
    stdout, stderr = Popen(command,
                           stdout=PIPE,
                           stderr=PIPE,
                           close_fds=True,
                           env=self.environment).communicate()
```

```
    # XXX - PDF can be protect
    if 'Erro' in stderr:
      return False
    else:
      output = etree.fromstring(open(output_url).read())
      row_list = output.xpath('//text')
      name,previous,next = '', '', ''
      tables = {}
      element = []
      line = []
      matrix = []
      i,j,l,m = 0,0,0,0
      old_x_left = 600
      for x in row_list:
        base_line = x.attrib['top']
        base_column = x.attrib['left']
        i += 1
        for y in row_list[i:]:
          if base_line == y.attrib['top']:
            l += 1
            line.append(get_text(y))
            base_column = y.attrib['left']
            row_list.remove(y)
          elif base_column == y.attrib['left']:
            m = l
            if len(element) > 0:
              element.append(get_text(y))
            # In case name of the table is after table
            if len(line) == 0:
              next = get_text(x)
              if next != None and len(next.split(':')) == 2:
                name = next
                next = ''
            elif len(line) > 0:
              element.append(line.pop())
              element.append(get_text(y))
          else:
            if len(element) > 0:
              line.insert(m-1,element)
            l = 0
            element = []
            base_column = 0
            break

        if len(line)>0:
          # In case name of the table is before table
          previous = get_text(x.getprevious())
          if previous != None and len(previous.split(':')) == 2:
            name = previous
            previous = ''
          line.insert(0,get_text(x))
          if len(line) > 1:
            matrix.append(line)
        line = []
        if x.attrib['left'] < old_x_left and len(matrix)>0:
          if len(matrix)>0:
            j += 1
            if name == '':
```

```
67                  name = "Tabela %d" % j
68                name += " - pag %s" % x.getparent().attrib['number']
69                tables[name]= matrix
70              name = ''
71              matrix = []
72            old_x_left = x.attrib['left']
73        return tables
```

Código 3.27: método getTablesMatrix do PDFGranulator

É possível notar no código 3.27 que a "granularização" de tabelas em documentos PDF é consideravelmente mais complexa e trabalhosa. Em certos trechos o fim da extração do *pdftohtml* o resultado é tratado como uma matriz em que pesquisasse pelas linhas e colunas da tabela tentando de certa forma ordená-los corretamente.

A complexidade da tabela se dá também em função de não haver um padrão próprio e corretamente adotavel para quantidade de linhas por colunas e vice versa, somado a falta de detalhe que a própria ferramenta consegue extrair.

O resultado deste processo é a soma de todas as tabelas do documento PDF, ao contrário de documentos normais que tratados com LibreOffice podem apresentar apenas os títulos das tabelas deixando que o cliente faça a requisição apenas da tabela desejada que parece mais viavel a todos os projetos ligados ao CloudOoo, exceto a Biblioteca Digital, do NSI.

3.2.7 Criação do método granulateFile para granularização de documentos

Por ser o principal arquivo de conexão e inicialização do CloudOoo, poucas modificações foram realizadas no Manager 3.1.11.1, entre elas a mais atual foi o método granulateFile que é diretamente responsável por retornar os grãos do processo de "granularização" para o cliente.

```
1   def granulateFile(self, data, source_format="odt"):
2     """This function allows BD NSI's project to completely granulate
3     document file"""
4     if source_format.lower() == "pdf":
5       pdfgranulator = PDFGranulator(self._path_tmp_dir, decodestring(data),
            'pdf',
6                                     **self.kw)
7       table_list = pdfgranulator.getTableItemList()
8       grains = []
9       if table_list != 'PDF Protect or have no Table Item List':
10        tables = []
11        for item in table_list:
12          table = pdfgranulator.getTable(item)
13          tables.append(table)
14        grains = map(encodestring, tables)
15      images = pdfgranulator.getImageItemList()
16      if images != False:
```

```
17          # XXX - encodestring cant convert list
18          grains += map(encodestring, str(images))
19
20        # XXX - if has no grains
21        if grains == []:
22          return "This PDF is protect or has no grains"
23        return grains
```

Código 3.28: método granulateFile do Manager

No código 3.28 esta representado como o CloudOoo trata a requisição de granulateFile do cliente para o servidor, verificando o tipo de documento com base na sua extensão (source format) e a reenderiza para o Handler responsável, que de forma geral retornará uma lista com os grãos de tabelas e imagens encontrados no documento.

3.2.8 Manutenção e extensão de código e testes

Este capítulo visa ressaltar algumas menores modificações que foram importantes para continuidade deste projeto.

Entre essas modificações destacam-se:

- Modificações de reescrita em pequenos erros não reparados anteriormente;
- Partes reescritas em função de melhorias em funcionalidades pré-existentes;
- Pequenas funcionalidades criadas com intuito de reduzir código;
- Pequenas funcionalidades criadas com intuito de reduzir tempo de execução dos processos;
- Erros corrigidos em função do novo versionamento de alguma ferramenta interna;
- Testes incrementados para garantir maior segurança às funcionalidades existentes;
- Funcionalidades que começaram a ser escritas mas não foram concluídas.

Essas modificações se dão pelo fato do CloudOoo continuar em desenvolvimento simultaneamente por dois grupos, que primeiramente visam seu interesse direto nesse produto, mas sem necessáriamente afastar-se do interesse de outros grupos nesta ferramenta.

3.2.9 Tabela de Novas funcionalidades

Na tabela 3.1 estão representadas algumas das funcionalidades do CloudOoo em função do que foi desenvolvido neste trabalho:

Tabela 3.1: Comparação de funcionalidades do CloudOoo.

Funcionalidades	OOOD 1.0	OOOD 2.0	CloudOoo 1.24	Neste trabalho
Conversão de documentos	X	X	X	-
Manipulação de metadados dos documentos	-	X	X	-
“Granularização” de documentos	-	-	X	-
“Granularização” de documentos PDF	-	-	X	X
Controle de problemas com sistema	-	X	X	X
Controle de problemas com LibreOffice	-	X	X	-
Conversão de PDF	-	-	X	X
Conversão de Imagens	-	-	X	-
Conversão de Áudio	-	-	X	X
Conversão de Vídeos	-	-	X	X
Manipulação de metadados áudio e vídeo	-	-	X	X

4 ESTUDO DE CASO

Este capítulo apresenta um estudo de caso do CloudOoo e sua implantação num ambiente dedicado.

Para este estudo foram escolhidas duas formas de instalação: a primeira a partir do uso do SlapOS, subseção 4.1.3; a segunda a partir do uso direto do *buildout*, forma que é principalmente utilizada para desenvolvimento e para serviços do NSI.

Não existe praticamente diferença entre ambas as instalações, exceto pelo tempo que levam, e configuração final.

Além disso este capítulo também apresentará uma avaliação quanto a forma de desenvolvimento do CloudOoo em comparação a sua versão anterior e do uso de técnicas ágeis.

4.1 Aplicações relacionadas ao CloudOoo

Com seus quase três anos de produzido o CloudOoo já vem sendo utilizado a muito por três principais aplicações, sendo uma delas diretamente responsável por sua instalação, como citado na seção 4.4.1.1 .

A subseções a seguir apresentam sobre essas aplicações.

4.1.1 Biblioteca Digital

A Biblioteca Digital da Rede Nacional de Pesquisa e Inovação (RENAPI), é um projeto que visa disponibilizar um acervo bibliográfico digital para contribuir com a disseminação de material científico e tecnológico produzido na rede de Educação Profissional Científica e Tecnológica (EPCT), sendo esse material periódicos, teses, monografias, artigos entre outros. Assim esse disseminação visa colaborar na qualificação do material humano digitalizado e na disseminação de conhecimento.

Este projeto é um dos principais desenvolvidos no Núcleo de Pesquisa em Sistemas de Informação (NSI), que conta atualmente com 20 bolsistas e 20 pesquisadores.

4.1.2 ERP5

Um ERP é capaz de integrar processos e dados de uma organização, através de recursos tecnológicos que padronizam e automatizam os mesmos.

Muito embora seja um sistema propriamente dito, ele foca mais em processos do que em funcionalidades, ele mascará informações em funcionalidades transparentes (DESAI; PITRE, 2010).

No entanto, apesar de trazer muitas vantagens a organização, esse processo de automatização, de forma geral, é longo, de alto custo e complexidade, e até mesmo difícil de implementar.

De acordo com Smets-Solanes e Carvalho (2003), esta situação que motivou a criação do ERP5, cujas ferramentas são de código aberto, permitindo que a organização modifique-o a fim de torná-lo mais flexível aos seus processos.

Ele também incorpora conceitos avançados como o de banco de dados orientados a objetos, um sistema de gerenciamento, de sincronização, variação, *workflows*, e possibilita a implementação de *Business Templates*.

Compreende-se assim o ERP5, como sendo um ERP de baixo custo de implantação e alta tecnologia para pequenas e médias empresas.

4.1.3 SlapOS

O SlapOS é um sistema operacional de código aberto para o uso de redes distribuídas em computação em nuvem, que se baseia em que tudo se trata de processos.

Para Smets-Solanes, Cerin e Courteaud (2011) a computação em nuvem é dividida em três camadas, infraestrutura como serviço (IaaS), Plataforma como Serviço (PaaS) e *Software* como Serviço (SaaS). Na IaaS esta o funcionamento virtual do computador e seu armazenamento, sob o ele é construído o Paas, que funciona como coração dos serviços, como servidor e bancos de dados. Finalmente sobre Paas estão as aplicações de uso do usuário.

Através de uma API unificada e simples, que requer poucos minutos para aprendizagem, este sistema combina computação em grade e o conceito de ERP para fornecer estas categorias previstas na compução em nuvem.

Dada sua abordagem unificada e arquitetura modular, ele tem sido usado como uma ferramenta de testes para *benchmark* de bancos de dados NoSQL e para otimização do processo de alocação em nuvem.

4.2 Ambiente de desenvolvimento

O estudo apresentado neste trabalho foi desenvolvido no NSI e contou com a disponibilidade de um computador com processador Intel Core I5 CPU 650 3.20 x4, 4GB de memória RAM, 320GB de HD; bem como de um notebook com processador Intel Core 2 Duo CPU 2.13 Hz, 8GB de memória 50GB de HD disponíveis para o sistema; além de dois servidores do NSI um de processador Xeon CPU E5335 2.00 x8, 14 GB de memória e 20GB de HD e um segundo com processador Xeon CPU E5335 2.00 x4, 4GB de memória e 20GB de HD, em uma máquina virtual.

Os dois primeiros computadores contavam com o sistema operacional Ubuntu 12.04 Lt, enquanto os servidores utilizavam o sistema operacional Debian 6.0.

4.3 Processo de Desenvolvimento

Para Pressman (1995), a garantia da qualidade de software esta diretamente ligada ao emprego de testes sobre o mesmo, onde esses testes representam a expectativa do usuário sobre a aplicação, e podem ser empregados de forma automatizada ou manual. Embora testes automatizados tendam a gastar mais tempo em relação a programação dos mesmos, sua cobertura sobre o produto garante menor porcentagem de erros quando executada a aplicação.

A técnica TDD (*Test-Driven Development*), ou desenvolvimento orientado a testes, defende o desenvolvimento dos testes antes do desenvolvimento da parte funcional da aplicação, dado que os testes também fazem parte da mesma. A eficácia dela garanti que todas as expectativas da aplicação sejam testadas e portanto garantidas ao usuário.

No inicio da parceria do NSI e Nexedi no desenvolvimento do CloudOoo quase não se utilizavam testes, entretanto com seu crescimento como produto e apresentada a dificuldade em mantê-lo estável foi dado o começo ao desenvolvimento de teste, a técnica TDD ainda não era empregada neste primeiro momento em função da porcentagem ja desenvolvida do produto, atualmente porém vêm sendo empregada diretamente.

Segundo Astels (2003), projetos que utilizam TDD devem possuir uma suíte exaustiva de testes que por sua vez determinam o código que deve ser escrito. No entanto para uma aplicação de serviço web, existe certo grau de dificuldade de começar do zero apenas com testes, tendo por justificativa suas dependências como outras aplicações bem como a utilização de rede por este.

Para a realização de testes unitários no CloudOoo foi preciso antes o desenvolvimento de *scripts* que pudessem interligar todas as bibliotecas envolvidas para o funcionamento do mesmo. Também foram desenvolvido *scripts* que “ligam” a aplicação e realizam testes na rede

local, a fim de garantir que as respostas estejam corretas quando este serviço estive ativo e responder a conexões distantes.

Além dos testes, outra ferramenta que garante o desenvolvimento do CloudOoo é o uso de um sistema de controle de versão, como o *subversion*, que era utilizado na versão 2.0, e que atualmente foi trocado pelo *git* em função de vantagens como, por exemplo, o controle de versões distribuído. A importância desta ferramenta esta no controle do crescimento da aplicação, que por momentos contou com uma equipe de desenvolvimento sem contato constante e em diferentes períodos.

Assim por meio de ferramentas que podiam acompanhar as modificações da aplicação, bem como reverter determinadas alterações em casos de erros posteriores na mesma e dar um detalhamento de quando essas modificações ocorreram, foi possível seguir com este desenvolvimento.

Sob certo ponto de vista o uso destas práticas sugere sobre o desenvolvimento do CloudOoo severas semelhanças com os métodos ágeis, muito embora não exista a definição propriamente dita de nenhuma delas aplicadas diretamente sobre o projeto.

Em seu artigo Silva, Monnerat e Carvalho (2010), comprovam o uso do TDD no CloudOoo, e de suas complicações de emprego, uma vez que inicialmente não existia total proficiência por parte dos desenvolvedores. Além disso foi verificado sobre o estabilidade do desenvolvimento do projeto, tomando por base a lista de mudanças armazenadas pelo controle de versão. Observou-se através deste artigo que embora no incio do projeto nenhum teste falhasse, conforme o mesmo foi acrescido de mudanças e novas funcionalidades testes que antes passavam passaram a apresentar erros antes não conhecidos, precisando assim de correções.

Estas implicações trazem ao meio de software uma compreensão muito similar ao do ramo de indústria, no que se desrespeita a aplicação de novas técnicas para garantir que quando um produto chega ao meio de produção possa continuar estável ao uso.

4.4 Processo de instalação

Para instalação via SlapOS, foi definido como pré-requisito a instalação do próprio SlapOS, sendo este dependente apenas da existência da instalação do Python, em qualquer versão uma vez que o SlapOS instala todos seus requisitos de funcionamento, inclusive o próprio Python. Variando de acordo com o sistema escolhido podem ser necessários demais pacotes de dependências de sistema.

Da mesma forma, para instalação via *buildout*, forma um pouco mais simples, é necessário igualmente possuir a instalação do Python e do Git como pré requisitos, uma vez que através deles e do uso da biblioteca *bootstrap*, disponível em (PYTHON SOFTWARE FOUNDATION,

2012), é possível gerar o *script bin/buildout*, bem como utilizá-lo para viabilizar a instalação do CloudOoo por seus arquivos de configuração, ou receitas.

Nas próximas subseções serão apresentadas as devidas instalações.

4.4.1 Instalação via SlapOS

Esta instalação foi realizada a partir da modificação do tutorial de instalação do ERP5 no SlapOS (NEXEDI SA, 2012), o qual passa por modificações ocasionalmente em função das atualizações frequentes desta ferramenta.

Nestas subseções será apresentado o tutorial já modificado em sequência de instalação:

4.4.1.1 Instalação do SlapOS

Está primeira subseção é a instalação padrão para qualquer ferramenta que utilize o SlapOS.

É aconselhado que seja realizada a instalação na pasta raiz do sistema, assim requerendo privilégios para instalação. É possível para o usuário optar por instalá-lo em sua pasta de usuário com algumas modificações no tutorial, entretanto em determinados momentos será podem ocorrer erros inesperados, e a exigência do uso de privilégio de qualquer forma.

Como é possível notar no código 4.1, cria-se uma pasta para instalação em */opt/slapos*, e também um arquivo de configuração *buildout*.

Após a criação deste arquivo, através do uso do Python, é realizada a execução de um *bootstrap* próprio da Nexedi.

```
1  $ sudo mkdir /opt/slapos
2
3  $ cd /opt/slapos
4  $ touch buildout.cfg
5  $ vi buildout.cfg
6
7  [buildout]
8  extends =
9    http://git.erp5.org/gitweb/slapos.git/blob_plain/refs/tags/slapos-0.57:/
       component/slapos/buildout.cfg
10
11 $ sudo python -S -c 'import urllib2; print urllib2.urlopen("http://www.
     nexedi.org/static/\
12 packages/source/slapos.buildout/bootstrap-1.5.3-dev-SlapOS-002.py").read()'
       | python -S -
13
14 $ sudo bin/buildout -v
```

Código 4.1: Primeira parte da instalação do SlapOS

Com o termino do *bootstrap* é executado o *script bin/buildout*, que realiza a instalação dos componentes necessários ao SlapOS.

Após a instalação dos componentes e dependências é preciso configurar o mesmo, no código 4.2, existe um exemplo de arquivo de configuração e logo em seguida na figura 4.3, existe a configuração de rede para uso do SlapOS.

É necessário que o *slapproxy* seja iniciado e mantido rodando em *background* sempre que esta máquina estiver ligada e/ou em uso, ele representa funcionalidades de conectividade do SlapOS *master*.

```
$ touch slapos.cfg

[slapos]
software_root = /opt/slapgrid
instance_root = /srv/slapgrid
master_url = http://127.0.0.1:5000/
computer_id = vifibnode

[slapformat]
computer_xml = /opt/slapos/slapos.xml
log_file = /opt/slapos/slapformat.log
partition_amount = 5
bridge_name = br1331
partition_base_name = slappart
user_base_name = slapuser
tap_base_name = slaptap
ipv4_local_network = 10.0.0.0/16

[slapproxy]
host = 127.0.0.1
port = 5000
database_uri = /opt/slapos/proxy.db
```

Código 4.2: Arquivo de configuração do SlapOS

```
$ sudo bin/slapproxy -vc /opt/slapos/slapos.cfg
$ sudo brctl addbr br1331
$ sudo ip l s dev br1331 up
$ sudo ip a l dev br1331 fd00::1/64
$ sudo vi /etc/network/interfaces

  auto br1331
  iface br1331 inet6 static
    adress fd00::1
    netmask 64
    bridge_ports none

$ sudo bin/slapformat -c /opt/slapos/slapos.cfg
$ sudo bin/slapgrid -c /opt/slapos/slapos.cfg
```

Código 4.3: Configurações de rede do SlapOS

É possível reparar que o SlapOS é uma ferramenta adaptada ao IPV6, e necessariamente

é preciso configurá-lo através da interface *br1331*.

O *script /etc/network/interfaces* do sistema também é configurado para habilitar as funcionalidades de IPV6, a fim de que não seja preciso iniciá-las sempre que reinicie o sistema.

Por fim nas duas ultimas linhas do código 4.3 são executados os *scripts slapformat* e *slapgrid*, eles são responsáveis por registrar seu computador na nuvem e seus devidos *slots* além de habilitar o cliente SlapOS em seu computador, respectivamente.

4.4.1.2 Instalação do CloudOoo

Após realizada a instalação do SlapOS, o passo de instalação de ferramentas através do mesmo é consideravelmente simples.

Para requerer a instalação do CloudOoo é necessário utilizar o *slapconsole* citando o arquivo de configuração citado na subseção 4.4.1.1, assim com o uso de uma instância do SlapOS(*slap*), referenciada na terceira linha da figura 4.4, informa-se a esta instância qual será sua função, isto é, seu produto específico.

A partir do uso do *slapgrid* a parte referente aos componentes deste produto será instalado de forma compilada pelo *Buildout*.

```
1  $ bin/slapconsole /opt/slapos/slapos.cfg
2
3  import slapos.slap.slap
4  slap = slapos.slap.slap()
5  slap.initializeConnection('http://127.0.0.1:5000/')
6
7  slap.registerSupply.supply(
8      "http://git.rp5.org/gitweb/slapos.git/blob_plain/cloudooo:/software/
           cloudooo/software.cfg",
9      computer_guid="vifibnode")
10 #teste que retorna id da instancia
11 computer_particion.getId()
12
13 $ sudo bin/slapgrid-sr -c /opt/slapos/slapos.cfg
```

Código 4.4: Requisição de instalação do CloudOoo no SlapOS

Nesta referência em particular o ponteiro esta voltado para o *branch* do CloudOoo, em função de suas configurações próprias que não foram incorporadas ao projeto do SlapOS.

Após a instalação dos componentes é necessário requerer uma instância, como em 4.5.

Este processo também requer o uso do *slapconsole*, de forma semelhante a instalação de componentes, no entanto é necessário fornecer um título a sua instância e é possível verificar se a mesma foi criada requirindo, por exemplo, seu *id*.

```
1  $ bin/slapconsole /opt/slapos/slapos.cfg
```

```
2
3  import slapos.slap.slap
4  slap = slapos.slap.slap()
5  slap.initializeConnection('http://127.0.0.1:5000/')
6
7  computer_partition = slap.registerOpenOrder(
8      "http://git.rp5.org/gitweb/slapos.git/blob_plain/cloudooo:/software/
           cloudooo/software.cfg",
9      "Meu CloudOoo")
10 #teste que retorna id da instancia
11 computer_partition.getId()
12
13 $ sudo bin/slapgrid-cp -c /opt/slapos/slapos.cfg
```

Código 4.5: Requisição de uma instância do CloudOoo via SlapOS

Por fim é utilizado novamente o *slapgrid*, mas desta vez com novos parâmetros para que a criação da instância seja finalizada.

Por se tratar de um serviço web em um sistema de nuvens, esta instalação compreende que após terminada deverá iniciar um servidor do CloudOoo pelas configurações estabelecidas no software.

4.4.2 Instalação do CloudOoo.git

Esta segunda instalação foi criada para ser mais flexível aos projetos do NSI que também utilizam o CloudOoo, bem como aos que tenham intenção de colaborar com esta ferramenta.

Esta instalação esta disponível no Github do NSI, (NSI, 2012a), e possui um *README* com instruções para instalação.

Além disso esta instalação esta orientada a fazer download do repositório igualmente disponível no Github do NSI, em (NSI, 2012b), que se mantém sempre na última versão de desenvolvimento, a mais atual.

Após o download do (NSI, 2012a), são necessárias apenas duas instruções para esta instalação, 4.6:

```
1
2  $ python -S -c 'import urllib2; exec urllib2.urlopen("http://python-
       distribute.org/boostrap.py").read()'
3
4  $ bin/buildout -vv
```

Código 4.6: Modo de instalação do cloudOoo (NSI, 2012a)

Na primeira etapa utiliza-se o Python para fazer *download* e rodar o *script* do *boostrap*, este irá fazer o download e disponibilizar o *Buildout*.

Ao termino desta fase, com o *script buildout* acrescido de argumentos no intuito de

torná-lo verboso e do arquivo padrão de configuração (*buildout.cfg*), ocorre a instalação do CloudOoo e todos os componentes necessários, entre eles o LibreOffice, FFMPEG, ImageMagick e demais.

Diferentemente da instalação via SlapOS o servidor do CloudOoo não fica disponível automaticamente para uso, é preciso ainda utilizar o *script bin/supervisord* para dar inicio ao servidor.

Ainda caso o usuário desejar é possível trocar a porta de funcionamento do servidor no arquivo *cloudooo.cfg*, bem como outras configurações padrão, como o limite do uso de memória.

4.5 Processos de requisições

Para estabelecer conexão entre um cliente qualquer e o CloudOoo é necessário o uso de uma biblioteca compatível ao XMLRPC. No código 4.7 foi realizado um exemplo de cada requisição básica disponível para todos *handlers*, através de uma conexão estabelecida pela biblioteca *xmlrpclib*:

```
1  from base64 import encodestring
2  from xmlrpclib import Server
3
4  conexao = Server("http://localhost:23000")
5
6  arquivo = open("test.ogv").read()
7
8  novo-arquivo = conexao.convertFile(encodestring(arquivo), 'ogv', 'mpeg')
9
10 info = conexao.getFileMetadataItemList(encodestring(arquivo), 'ogv')
11
12 nova-info = conexao.updateFileMetadata(encodestring(arquivo), 'ogv', dict(
       Titulo="Arquivo teste"))
```

Código 4.7: Exemplo prático de uso do CloudOoo

Na linha 6 do código a variável *arquivo* recebe o conteúdo do arquivo *test.ogv*.

Para conversão deste, na linha 8, é preciso codificá-lo para que durante sua passagem do cliente para o servidor esse arquivo não seja danificado, esta codificação realizada pelo uso da biblioteca *base64*.

No momento da conversão o servidor vai receber os dados do cliente, vai decodificar o arquivo e identificá-lo como um arquivo de vídeo, sendo assim o mesmo será encaminhado para o FFMPEGHandler, que por sua vez irá convertê-lo para o formato *mpeg*.

O FFMPEGHandler também será o responsável pelos métodos de *getFileMetadataItemList*, linha 10, e *updateFileMetadata*, linha 12, nos quais serão realizados respectivamente a

extração e inserção de metadados no arquivo.

Na linha 12 é possível notar que para inserção de metadados, os novos dados a serem passados devem estar em um dicionário.

Observe também que nesta inserção de metadados, foi utilizado o *nova-info*, como resposta da requisição de inserção de metadados. Isto porque esta requisição do CloudOoo retorna um arquivo com os dados inseridos no mesmo, e que já se encontra codificado para transporte.

Além destas requisições comuns a todos os *handlers* existem também requisições que foram previamente implantadas apenas para documentos:

- getAllowedExtensionList: retorna extensões permitidas para determinado arquivo;
- getChapterItem: retorna o capítulo selecionado do documento;
- getChapterItemList: retorna todos capítulos do documento;
- getColumnItemList: retorna colunas da tabela selecionada;
- getImage: retorna imagem selecionada;
- getImageItemList: retorna lista de imagens em documento;
- getLineItemList: retorna lista de linhas da tabela selecionada;
- getParagraph: retorna parágrafos selecionado;
- getParagraphItemList: retorna lista de parágrafos de um documento;
- getTable: retorna tabela selecionada;
- getTableItemList: retorna lista de tabelas no documento;
- system.listMethod: retorna lista de métodos disponíveis no servidor.

4.6 Uso do CloudOoo na Biblioteca Digital

Para disponibilizar seu acervo, a Biblioteca Digital tem como regra converter os arquivos recebidos seu formato livre compatível, para isso utiliza de requisições ao CloudOoo. Na figura 4.1, existe um exemplo de submissão de arquivos, do tipo Relatório, o qual passará pela conversão de DOC para ODT através de uma requisição ao CloudOoo, após a aprovação 4.2:

Figura 4.1: Pagina de submissão de documentos da Biblioteca Digital

Nas figuras não é implícito o uso da ferramenta, entretanto, uma vez que essa funcionalidade requer o uso de documentos em formato livre para realização de suas tarefas, é requerido o uso do CloudOoo para converter o arquivo em questão para um padrão livre e posteriormente realizar a funcionalidade de “granularização”.

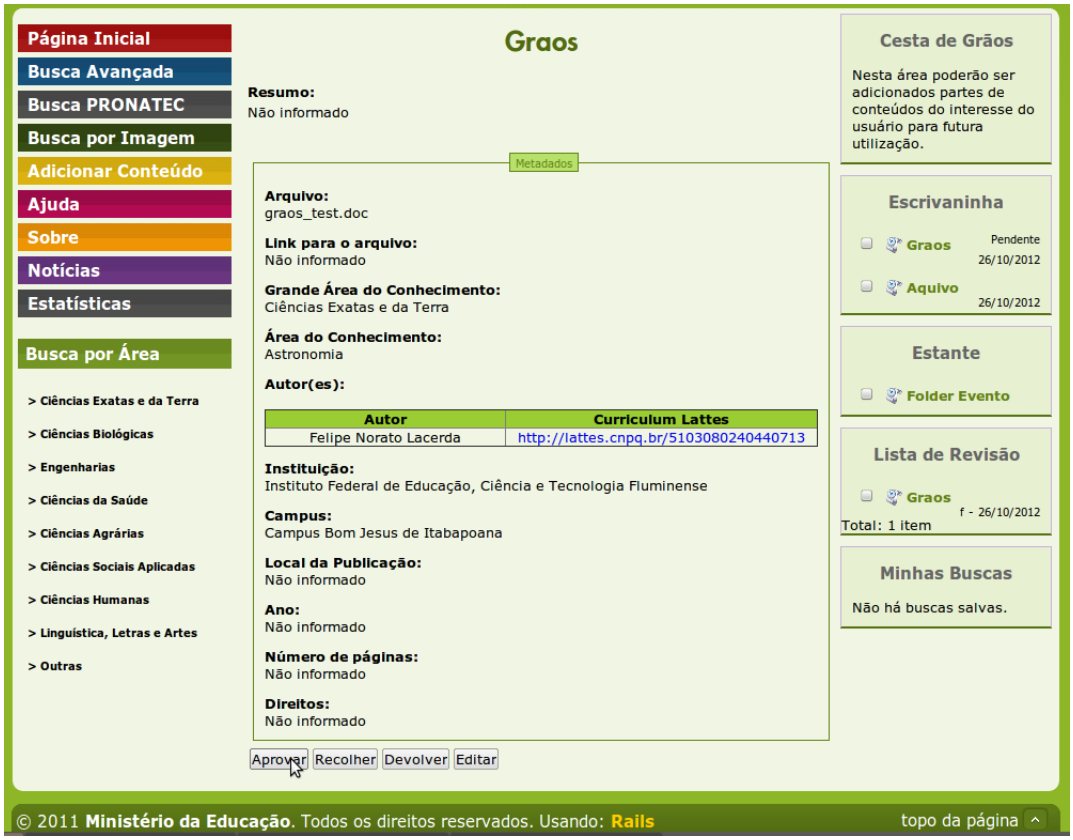


Figura 4.2: Pagina de aprovação de documentos da Biblioteca Digital

Na figura 4.3 há uma demonstração da funcionalidade de extração de metadados, enquanto os grãos encontram-se nas figuras 4.4:

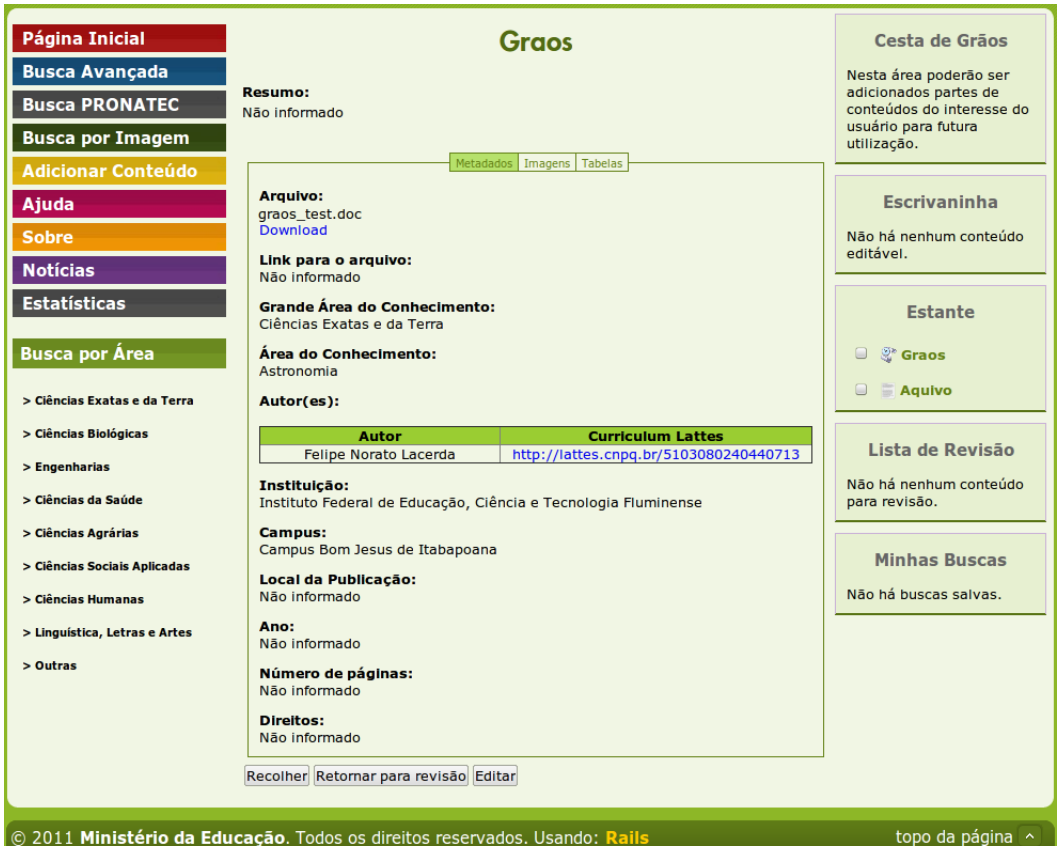


Figura 4.3: Pagina de exibição de metadados de documentos da Biblioteca Digital

Os grãos extraídos pelo processo de “granularização” são separados em imagens(4.4) e tabelas, que não existem nesse caso, e dispostos para visualização do usuário.

Figura 4.4: Pagina de exibição de grãos do tipo imagem de documentos da Biblioteca Digital

4.7 Performance

Para estabelecer diferença de performance em relação as versões anteriores do CloudOoo foram considerados dois testes.

O primeiro deles realizado em um trabalho anterior, por Monnerat (2010), contava 3699 documentos no formato ODT que foram selecionados previamente, entre eles alguns documentos inválidos e outros em formatos desconhecidos. Por ser a conversão mais utilizada, foi decido que neste teste estes documentos seriam convertidos para PDF. Este teste foi realizado por meio do uso de um *script*, neles além de prover a conversão também era provido o armazenamento do tempo de cada conversão, bem como o tempo total que todas as conversões levaram em cada versão do CloudOoo.

No resultado do primeiro teste notou-se que o OOOD 2.0 levará mais tempo para realizar a conversão de cada documento,entretanto por ser mais estável levou 10 horas para realizar o teste e apresentou 12 erros, enquanto o OOOD 1.0 apresentou 531 erros e levou 11 horas para realizar o teste.

No segundo teste foram escolhidos arquivos distintos, representado no código 4.8, este teste selecionou 3 diferentes documentos DOC para serem convertidos para ODT; 3 documentos com PDF para serem convertidos para TXT, 1 arquivo de vídeo AVI de aproximadamente 100 MB que seria convertido para THEORA; um arquivo de áudio MP3 de aproximadamente 3 MB

para ser convertido para VORBIS; e por fim uma imagem PNG de aproximadamente 30 KB para ser convertida para JPG.

```
from random import randint
from base64 import encodestring, decodestring
from datetime import datetime
from xmlrpclib import ServerProxy
import magic


mime_decoder = magic.Magic(mime=True)

documents = ['ISSCWR6PaperTemplate.doc','SiteLeaderAppWinter2012.doc', '
    Estonia.doc']
pdfs = ['Bankruptcies.pdf', 'WIA-IM_tfw_studentguide.pdf', '2012
    FinancialRpt.pdf']

type_convert_choices = { 0: [documents, "doc", "odt", 'application/vnd.
    oasis.opendocument.text'],
                  1: ["test.png", "png", "jpg", 'image/jpeg'],
                  2: ["test.mp3", "mp3", "ogg", 'application/ogg'],
                  3: ["test.avi", "avi", "ogv", 'application/ogg'],
                  4: [pdfs, "pdf", "txt", 'text/plain']
                }


proxy = ServerProxy("http://localhost:23000/RPC2")
id = 0
process = []

while True:

  name = 'test'
  file, source_format, destination_format, mime = type_convert_choices[
      randint(0,4)]
  data = ''
  if type(file) == str:
    data = open(file).read()
  else:
    name =file[randint(0,2)]
    data =  open(name).read()
  done, mimetype = False, False
  try:
    file = proxy.convertFile(encodestring(data), source_format,
        destination_format)
    mimetype = mime_decoder.from_buffer(decodestring(file))
    if mimetype == mime:
      done = True
  except:
    done = "Error"
  process.append([id, name, source_format, mime, destination_format,
      mimetype, done, '\n'])
  id += 1

content = "  id     |          file        |     source          |   dest
              |    done          \n" + str(process)
log = open('log.txt', 'w')
log.write(content)
log.close()
```

Código 4.8: *Script* de teste

O objetivo deste segundo teste era focar nas conversões de arquivos para formatos abertos, que são mais utilizados nos projetos que o CloudOoo atende.

Além disso a perspectiva era que um menor número de erros aparecesse em função dos inúmeros tratamento utilizados nesta ferramenta. Também haveria um menor número de conversões que a versão anterior em função do tamanho e tempo que arquivos de vídeo, imagem e áudio consomem comparados a documentos.

Para saber se as conversões foram realizadas com sucesso eram verificados os *mimetypes* de cada conversão de acordo com o esperado.

Infelizmente não foi possível a utilização de diversos arquivos aleatoriamente como seria o caso de um ambiente de produção, em função da dificuldade para adquirir estes arquivos por meio da internet que tem sido cada vez mais restrita a downloads.

Neste segundo teste foram utilizados dois computadores diferentes de desenvolvimento, citados em 4.2, no notebook e no servidor de 4GB de memória.

No notebook a performance foi consideravelmente estável, em 10 horas o CloudOoo realizou 2034 conversões, entre elas 405 conversões foram de DOC para ODT e apenas 3 retornaram erro; 260 foram de PDF para TXT e nenhuma retornou erro; 307 foram de AVI para OGV Theora e nenhuma retornou erro; 384 foram de MP3 para OGG *Vorbis*; e 678 foram de PNG para JPG sem retorno de erro.

Resultando assim em apenas 3 erros em 10 horas.

No servidor, entretanto o resultado foi menos promissor, o teste durou apenas 3 horas em função de um erro de estouro de memória no servidor, que detinha de aproximadamente 3 GB livres para o teste, mas que não se encontrava disposto apenas para o teste uma vez que estava em uso em tempo real.

Durante essas 3 horas foram realizadas 611 conversões, entre elas 122 conversões foram de DOC para ODT e apenas 1 retornou erro; 77 foram de PDF para TXT e nenhuma retornou erro; 91 foram de AVI para OGV Theora e nenhuma retornou erro; 114 foram de MP3 para OGG Vorbis; e 207 foram de PNG para JPG sem retorno de erro.

De forma positiva o CloudOoo se manteve estável durante essas horas tanto no notebook quanto no servidor, pois mesmo com o estoura de memória no segundo, ele se manteve ativo. Entretanto com os resultados dos teste foi constatado que com as novas funcionalidades adicionadas precisam ser revisadas em função de escalabilidade e uso em tempo real a fim de que não volte a ocorrer estouros de memória entre outros possíveis erros.

No que respeito aos erros de documentos, dada a falta de variedade, foram todos erros relativos aos documentos PDF com proteção contra conversão.

Na tabela 4.1 mostrando os resultados desses testes:

Tabela 4.1: Comparação entre versões do CloudOoo por meio de testes.

Versões	Documentos	Imagens	Áudio	Vídeo	Total de erros	Total de Conversões	Total de horas
OOOD 1.0	3699	0	0	0	531	3699	11
OOOD 2.0	3699	0	0	0	12	3699	10
CloudOoo	405+260	307	384	678	3	2034	10

5 TECNOLOGIAS SIMILARES

Este capítulo visa apresentar um estudo a respeito de outras aplicações que se assemelham ao tipo de aplicação que representa o CloudOoo, isto é, aplicações web capazes de realizar conversões de documentos e arquivos de multimídia, e que ainda pudessem manipular informações destes arquivos.

Este estudo foi baseado especialmente em aplicações livres e de código aberto que se encontram disponíveis para uso e colaboração tal como o CloudOoo, dado o pequeno número dessas aplicações também foram estudadas páginas que ofereciam o serviço de conversão online.

Entretanto cabe observar que dentre todas aplicações estudados não foi encontrada em nenhuma delas a funcionalidade de extração ou inserção de dados disponíveis no CloudOoo.

5.1 inout.io

O InOut é o serviço web que mais se aproximou de ser uma tecnologia similar ao CloudOoo. Ele foi desenvolvido exclusivamente para lidar com arquivos e suas conversões.

Este serviço é capaz de converter arquivos dos formatos:

- Documentos: PDF, Microsoft Office, OpenDocument, Open Office, Rich Text Format, Comma separated file;
- Vídeos: MPEG, Quicktime, Microsoft AVI, Windows Media, Microsoft Advanced Streamin, Theora, Flash, Matroska, Blue-Ray tracks, FLIC, SGI, DL Animation;
- Áudio: Wave, MP3, MPEG, MIDI, Raw, Windows Media, MPEG 4 audiostream, Dolby Digital, Apple Audio Interchange, Free Lossless audio, Real, Vorbis, Matroska;
- Imagens: JPEG, GIF, PNG, TIFF, Windows Bitmap, Windows Metafile, Adobe Illustrator vector, Encapsulated post script vector, Post script, Adobe Photoshop, Targa, Zsoft paintbrush, Mac Pict.

Para os formatos de:

- Documentos: PDF, Microsoft World, Open Document Textfile, Open Office Writer, Rich Text Format;
- Vídeos: MPEG, Flash;
- Áudio: Wave, MP3.
- Imagens: JPEG, PNG, GIF.

Através de uma breve analise às listas de conversões é possível notar que embora reconheça muitos padrões, o InOut tem preferência pela saída de arquivos em formatos livres de suas conversões.

Para comunicação entre seu serviço e qualquer aplicação que queira requisitá-lo, existe uma extensa documentação na página `https://api.inout.io/InOut-APIDocumentation.html`, esta documentação é especialmente desenvolvida para outros desenvolvedores.

Nessa documentação é recomendado o uso de RESTful e JSON para realizar a comunicação com o serviço, entretanto a linguagem a se utilizar fica por conta de cliente.

No código 5.1 é possível notar um exemplo de uso do InOut em que é requerido todos os padrão de conversão para seu uso através do JSON:

```
1
2  $ curl -u key:secret https://api.inout.io/profiles/
3
4
5  {"ok": true, "result": [
6      {"id": "IMAGE_TO_JPEG"},
7      {"id": "IMAGE_TO_PNG"},
8      {"id": "IMAGE_TO_GIF"},
9      // ...
10 ]}
```

Código 5.1: Requisição de perfis do InOut

Demais exemplos de uso do InOut podem ser encontrados na página da documentação citada anteriormente.

5.2 ServPDF

É um serviço web capaz de converter documentos, sejam eles de formatos abertos ou proprietários pela Microsoft.

Ele permite que qualquer documento possa ser convertido para PDF, e que qualquer PDF possa ser convertido em Postscript ou imagens.

É uma ferramenta extremamente simples baseada no Microsoft Office e modificada para atender ao LibreOffice.

5.3 Conversões Online

Fora esses serviços a gama de projetos livres para conversão de arquivos e documentos em geral é extremamente escassa. Existem muitas versões *desktop* e também versões de serviço on-line, neste caso os serviços são hospedados em páginas que ficam disponíveis gratuitamente para usuários.

A desvantagem desses serviços em relação ao CloudOoo é de que normalmente seus arquivos são limitados a um tamanho específico pelos proprietários da página, enquanto para um usuário que hospeda um CloudOoo é possível modificar esse limite de acordo com a máquina qual o mesmo será instalado, podendo este limite ser bem maior, entretanto dependendo do usuário para instalá-lo.

Assim é possível concluir que para um usuário final que não possua conhecimentos avançados de serviços web pode ser bem mais agradável utilizar essas páginas. As próximas subseções mostram alguns exemplos interessantes para seu uso.

5.3.1 youconvertit.com

O You Convert It é uma das páginas disponíveis na web para conversão de documentos, imagens, vídeos e áudio on-line. É uma das páginas mais aconselháveis pois possui o limite de arquivos no tamanha de até 1GB.

5.3.2 convertfiles.com

Assim como o You Convert It, o Convert.Files é capaz de converter desde documentos a arquivos de multimídia, entretanto com um limite bem menor de apenas 150MB por arquivo. Nele existe a vantagem de que o usuário possa mandar os arquivos convertidos para seus celulares ou aparelhos moveis em geral.

5.3.3 zamzar.com

Similar aos exemplos anteriores o Zamzar é capaz de realizar a conversão entre qualquer arquivo, entretanto seu tamanho limite para arquivos é de 100MB e até 5 conversões diárias, apesar disso é um dos serviços mais populares para conversão on-line.

5.4 Comparação entres CloudOoo e seus similares

Na tabela 5.1 com uma comparação entre as funcionalidades dos similares com o próprio CloudOoo:

Tabela 5.1: Comparação entre funcionalidades de ferramentas similares ao CloudOoo.

Funcionalidade	CloudOoo	InOut	ServPDF	YouConvertIt	Convert. Files	Zamzar
Servidor de serviços	X	X	X	-	-	-
Converte documentos	X	X	X	X	X	X
Converte Imagens	X	X	-	X	X	X
Converte Áudio	X	X	-	X	X	X
Converte Vídeos	X	X	-	X	X	X
Apresenta metadados	X	-	X	-	-	-
Modifica metadados	X	-	-	-	-	-

Após essa comparação é possível ressaltar a importância do CloudOoo como única ferramenta livre capaz de converter e manipular dados de diferentes tipos de arquivos, além de dispor de uma interface cliente-servidor.

6 CONCLUSÕES

6.1 Objetivos alcançados

Neste trabalho foi possível apresentar as contribuições realizadas a ferramenta de conversão e manipulação de arquivos em nuvem, CloudOoo.

Sendo esta apresentação uma simples descrição sobre todos os processos por trás dessas novas funcionalidades, entre elas a conversão e manipulação de áudio e vídeo, e a granularização de documentos PDF.

Além disso este trabalho apresentou detalhadamente a nova estrutura do CloudOoo, descrevendo um pouco sobre o uso desta estrutura dentro do mesmo.

Foram apresentadas também as tecnologias empregadas no CloudOoo, sendo essas aplicações livres e de código aberto, que se encontram disponível para acesso; e ainda outras ferramentas similares a esta aplicação.

Houve também uma breve descrição sobre como instalar e usar esta aplicação como ferramenta de conversão e manipulação de arquivos comuns aos tipos de documentos, imagens, vídeos, áudio e PDF.

Por fim, foi possível afirmar sobre as modificações e melhorias dessas funcionalidades através de um estudo de caso em cima do uso real em ferramentas utilizadas por empresas ao redor do mundo; e também através da realização de testes de escalabilidade comparados entre si e entre testes anteriores.

6.2 Trabalhos futuros

Apesar deste trabalho representar em grande parte a realização de objetivos propostos por um trabalho anterior com a aplicação CloudOoo, nota-se a necessidade de modificações futuras visando a melhoria continua da ferramenta.

Entre essas melhorias, visar maior estabilidade do projeto não só por meio dos testes implementados e seus acréscimos, bem como pela revisão da escrita do projeto em função das novas funcionalidades adquiridas, que se apresentaram poucos estáveis podendo causar

problemas com o uso excessivo de memória e do sistema como um todo.

Umas vez que os tipos de arquivos atendidos pelo mesmo foram expandidos também há o interesse de estender funcionalidades mais complexas já aplicadas aos documentos, como por exemplo a “granularização” de arquivos de vídeo, que é um processo já disponível e implantado no projeto da Biblioteca Digital.

Além dessas funcionalidades pretende-se que arquivos de multimídia também tenham seu dados tratados.

Por fim é de interesse do projeto que esta ferramenta seja capaz de trabalhar como um serviço RESTful para respostas mais simples e realizadas em diversas aplicações por uma interface JSON.

REFERÊNCIAS BIBLIOGRÁFICAS

ALECRIM, E. *OpenDocument Format*. 2011. Disponível em: <http://www.infowester.com-/odf.php>. Acesso em: 15/05/2012.

ASTELS, D. *Test-Driven Development: A Practical Guide*. New Jersey: Prentice Hall, 2003.

BRANDOM, R. *buildout tutorial. buildout howto. buildout review.* 2008. Disponível em: <http://renesd.blogspot.com.br/2008/05/buildout-tutorial-buildout-howto.html>. Acesso em: 15/05/2012.

BROWN, T. *An Introduction to the Python Web Server Gateway Interface (WSGI)*. 2009. Disponível em: <http://ivory.idyll.org/articles/wsgi-intro/what-is-wsgi.html>. Acesso em: 15/05/2012.

CHACON, S. *Pro Git*. São Paulo: Apress, 2009.

DESAI, M. S.; PITRE, R. Developing a curriculum for on-line internacional business degree: an integrated approach using systems and erp concepts. *Education*, 2010.

FFMPEG.ORG. *Ffmpeg Documetation*. 2012. Disponível em: <http://ffmpeg.org/ffmpeg-.html>. Acesso em: 15/05/2012.

FOUNDATION, X. *Theora Specification*. [S.l.]: Xiph.Org Foundation, 2011.

FREEDESKTOP.ORG. *Poppler*. 2011. Disponível em: <http://poppler.freedesktop.org/>. Acesso em: 15/05/2012.

GLYPH COG. *XPdf*. 2011. Disponível em: <http://www.glyphandcog.com/Xpdf.html>. Acesso em: 15/05/2012.

IMAGEMAGICK STUDIO. *ImageMagick: Convert, Edit or Compose Bitmap Images*. 2012. Disponível em: <http://www.imagemagick.org/script/index.php>. Acesso em: 15/05/2012.

MONNERAT, G. *Trabalho de Conclusão de Curso*. 2010. Disponível em: <https://github-.com/gmonnerat/monografia/blob/master/Monografia.pdf>. Acesso em: 23/09/2012.

NEXEDI SA. *User install ERP5 with SlapOS*. 2012. Disponível em: <http://www-.erp5.com/user-Install.ERP5.With.SlapOS/user-Install.ERP5.With.SlapOS>. Acesso em: 20/08/2012.

NSI. *Buildout for Cloudooo*. 2012. Disponível em: <http://github.com/nsi-iff-/cloudooo_buildout>. Acesso em: 20/08/2012.

NSI. *Cloudooo*. 2012. Disponível em: <http://github.com/nsi-iff/cloudooo>. Acesso em: 20/08/2012.

PARKER, H.; JR, R. F. *Getting Started with OpenOffice.org*. São Paulo: LibreOffice, 2010.

PIRNAU, M. P. C. A. M. B. G. The soap protocol used for building and testing web services. *Proceedings of the World Congress on Engineering*, v. 1, p. 475–480, july 2011.

PRESSMAN, R. S. *Engenharia de Software*. São Paulo: MAKRON Books, 1995.

PYTHON SOFTWARE FOUNDATION. *buildout.bootstrap*. 2012. Disponível em: <http://pypi.python.org/pypi/buildout.bootstrap/1.4.1>. Acesso em: 23/09/2012.

PYTHON.ORG. *Python Uno*. 2008. Disponível em: <http://www.python.org.br/wiki-/PythonUno>. Acesso em: 15/05/2012.

QUIN, L. R. *What is XML?* 2010. Disponível em: <http://www.w3.org/standards/xml/core>. Acesso em: 15/05/2012.

ROELOFS, G. *PNG The Definitive Guide*. United States of America: OREILLY, 1999.

ROSSUM, G. V. *An introduction to Python*. United Kingdom: Network Theory Limited, 2003.

SCRIPTING NEWS. *What is XML-RPC?* 2011. Disponível em: <http://xmlrpc.scripting.com-/>. Acesso em: 15/05/2012.

SILVA, F. L. de Carvalho e; MONNERAT, G. M.; CARVALHO, R. A. de. AutonomaÇÃo na produÇÃo de software. *ENEGEP*, 2010.

SILVA, J. *5 anos de ODF*. 2010. Disponível em: <http://homembit.com/2010/05/5-anos-de-odf-2.html>. Acesso em: 15/05/2012.

SMETS-SOLANES, J.; CARVALHO, R. de. Erp5: A next-generation, open-source erp architecture. *IT Professional*, august 2003.

SMETS-SOLANES, J.; CERIN, C.; COURTEAUD, R. Slapos: a multi-purpose distributed cloud operating system based on an erp billing model. *IEEE*, 2011.

STEWARD, S. *PDFtk the pdf toolkit*. 2011. Disponível em: <http://www.pdflabs.com/tools-/pdftk-the-pdf-toolkit/>. Acesso em: 15/05/2012.

TESLA, B. M. Graphical treasures on the internet. *Computer Graphics World*, n. 34, november 1994.

THOMAS, R. *Python-Uno Bridge*. 2007. Disponível em: <http://www.openoffice.org/udk-/python/python-bridge.html>. Acesso em: 15/05/2012.

UDELL, J. *Zope Is Python's Killer App'*. 2000. Disponível em: <http://web.archive.org-/web/20000302033606% -/http://www.byte.com/feature/BYT20000201S0004>. Acesso em: 15/05/2012.

WIKIPEDIA. *Matroska*. 2012. Disponível em: <http://pt.wikipedia.org/wiki/Matroska>. Acesso em: 15/05/2012.

XIPH.ORG FOUNDATION. *What is Vorbis?* 2012. Disponível em: <http://www.vorbis.com-/faq/>. Acesso em: 15/05/2012.

ZHANG, B. Design and implementation of a scalable system architecture for embedded multimedia terminal. *International Conference on Electrical and Control Engineering*, n. 6057581, p. 618–621, 2011.

ZOPE FOUNDATION. *Zope2*. 2012. Disponível em: <http://pypi.python.org/pypi/Zope2>. Acesso em: 15/05/2012.