Lucas Nascimento Oliveira

Atualização automática de software para o ambiente Pure Data

Brasil

2015

Lucas Nascimento Oliveira

Atualização automática de software para o ambiente Pure Data

Monografia apresentada como requisito da disciplina de Projeto Orientado em Computação I do Curso de Bacharelado em Ciência da Computação da UFSJ.

Universidade Federal de São João del Rei – UFSJ

Departamento de Computação

Programa de Graduação

Orientador: Flávio Luiz Schiavoni

Brasil

2015

Lucas Nascimento Oliveira
Atualização automática de software para o ambiente Pure Data/ Lucas Nascimento Oliveira. – Brasil, 2015-
43 p. : il. (algumas color.) ; 30 cm.

Orientador: Flávio Luiz Schiavoni

Monografia (Graduação) – Universidade Federal de São João del Rei – UFSJ
Departamento de Computação
Programa de Graduação, 2015.
1. Palavra-chave1. 2. Palavra-chave2. I. Orientador. II. Universidade xxx. III. Faculdade de xxx. IV. Título

CDU 02:141:005.7

Lucas Nascimento Oliveira

Atualização automática de software para o ambiente Pure Data

Monografia apresentada como requisito da disciplina de Projeto Orientado em Computação I do Curso de Bacharelado em Ciência da Computação da UFSJ.

Trabalho aprovado. Brasil, 24 de novembro de 2015:

Orientador
Flávio Luiz Schiavoni

Professor
Elder José Reioli Cirilo

Professor
Fábio Corrêa

Brasil
2015

Resumo

Este estudo apresenta uma ferramenta de gestão de atualizações para um ambiente de programação musical chamado Pure Date. A definição do sistema de gerenciamento inclui também a definição de pacote, um conjunto para distribuir novos plugins, e um repositório mínimo para armazenar os plugins. Neste estudo apresentamos as decisões de projeto para criar a ferramenta, tal como o nome padrão, a extensão do pacote, a estrutura, o descritor e afins. Apesar do fato de este estudo apresentar decisões de projeto a cerca do ambiente de desenvolvimento Pure Data, ele pode ser também usado para criar um ferramenta semelhante para qualquer outro software que aceite extensões.

Palavras-chaves: Pure Data, Distribuição de software, Plugins, Extensões, Atualização de Software.

Abstract

This work presents a update manager system to a Music programming environment called Pure Data. The Update management system's definition includes the definition of a package, a bundle to distribute new plugins for this environment, and a minimal repository to store plugins. In this paper we present the project decisions to create this system such as the package name's standard, package's extension, package's structure, package's descriptor and so on. Despite the fact of the present paper describes project's decision to Pure Data environment, it can be used by programmers to create a similar tool to any other software.

Key-words: Pure Data, Software distribution, Plugins, Software update.

Sumário

1 Introdução

O desenvolvimento de software, em sua grande expansão, tem permitido uma grande evolução para os processos de gerenciamento de software. Com essa evolução, aumenta também a demanda de melhorias e novas funcionalidades para um determinado software. Para facilitar essa atualização, temos atualmente ferramentas que permitem a gestão de atualizações em formas de pacotes, permitindo a instalação ou remoção de alguma atualização de forma prática. Infelizmente, não é todo software que possui uma ferramenta de gerenciamento para tal tarefa. Pensando nisso, é proposto o desenvolvimento de uma ferramenta de atualização automática de plugins e extensões para o ambiente Pure Data, responsável por gerenciar o repositório de extensões e atualizações para o sistema de desenvolvimento do Pure Data. O desenvolvimento de sistemas modulares é hoje uma prática comum utilizada em diversos aplicativos como: ambientes de desenvolvimento de software, como Netbeans (MYATT *et al.*, 2008) e Eclipse (ECLIPSE.ORG, 2003); ferramentas para produção musical, como Ardour e Pure Data (ZMÖLNIG, 2001); navegadores como o Firefox (EDUARDO, 2013) e Chrome (CHROME, 2015); ferramentas de edição de imagens como o GIMP; ferramentas de gerenciamento de conteúdo web como o Drupal; entre outras. Estas ferramentas modularizaram seu desenvolvimento de maneira que novas funcionalidades podem ser adicionadas ao sistema e mesmo atualizações do sistema possam ser feitas por meio de plugins.

O desenvolvimento baseado em plugins permite que um sistema seja composto de diversos módulos podendo estes módulos terem sido desenvolvidos por diferentes desenvolvedores. Esta possibilidade traz ao sistema uma cooperação intrínseca a esta arquitetura de sistema além de simplificar a interação entre desenvolvedor e usuários. Tal ferramenta permitiria ao usuário ter acesso a um repositório de atualizações e plugins, prontos para serem instalados no ambiente do Pure Data, resultando em praticidade para o gerenciamento de tais extensões para usuários que não estão habituados a realizar tais tipos de manuseio do ambiente manualmente, tendo que acessar e alterar arquivos de sistema sem dado suporte. É proposto que a ferramenta trabalhe distribuidamente, permitindo o acesso globalmente remoto à um repositório e seus respectivos pacotes. O usuário teria a opção de usar um repositório já conhecido, ou de criar o seu próprio. Adicionalmente, repositórios podem ser baixados em outros repositórios.

1.1 Importância

A atualização de ambientes de softwares para a área de desenvolvimento é indispensável, seja para correção de algum erro, melhoria de desempenho ou até adição/remoção

de ferramentas no ambiente. Têm-se que os mais famosos softwares possuem apoio de um ambiente totalmente dedicado a realizar o gerenciamento de atualizações. A não atualização de um ambiente o torna obsoleto, não permitir a instalação de plugins e extensões pode torna-lo limitado, e fazer com que essas mudanças sejam feitas manualmente tendem à impraticidade para o usuário tanto em ambientes locais como distribuídos.

Em termos de praticidade, um sistema de gerenciamento como o proposto pode possuir um repositório de atualizações, plugins e informações sobre o ambiente, isso auxilia o processo de atualização para o usuário geral. Ter de desenvolver um ambiente novo para tais mudanças é de certa forma inviável considerando a criticidade da mudança, por isso existem inúmeros suportes à atualização de diversos ambientes de software. Temos assim, a necessidade de um ambiente ter o suporte de um sistema de gerenciamento de configurações. Levando em conta o ambiente Pure Data, temos uma quantidade extensa de plugins e extensões desenvolvidas por usuários. A criação de um repositório para armazenar tais atualizações e plugins será de grande utilidade aos usuários terem acesso, além de poderem contribuir, as atualizações contidas no mesmo.

1.2 Situação Atual

Apesar de crescentes estudos e trabalhos realizados na área de atualização e gerenciamento de software, o ambiente Pure Data sofre com a falta de dada atenção. Mas, como existem, de fato, vários sistemas de gerenciamento de configuração de software temos o necessário para realizar um estudo acerca de um novo sistema de gerenciamento de software para o Pure Data. Sistemas como o APT(APT-GET, 2009) e o dpkg(DEBIAN, 2015) são exemplos claros e famosos de como funciona o gerenciamento de pacotes de software, além de tais exemplos funcionarem também de forma distribuída, tendo eles acesso a um repositório apto a conter suas atualizações e extensões. De tal forma, vemos que todos os conceitos relacionados à gerencia de atualização proposta nesse projeto se encontra em diversos formas e sistemas. Apesar de ainda existirem problemas de como devem ser feitas as atualizações automáticas e como devem funcionar as características dentre outro fatores presentes. Sistemas de grande utilidade e porte podem necessitar da gerência de atualizações e extensões, portanto o acesso à esses sistemas nos permite visualizar o funcionamento de tal gerencia, além do estudo de aspectos de cada um desses sistemas que se comportarem de maneira diferente. Em resumo, atualmente temos vários trabalhos e estudos sobre a gerência e uso da atualização automática de software feitos de diversas formas, essa falta de padronização nos da a chance de estudar sobre as mais famosas e definir a nossa própria forma de gestão, baseada em sistemas conhecidos e estudados.

1.3 Motivação

Dada a falta de suporte para o gerenciamento de atualizações para o ambiente Pure Data, somada à importância de um sistema e o estudo de sistemas semelhantes ao, é interesse deste trabalho desenvolver um sistema de gerenciamento para atualizações para o ambiente Pure Data.

Em adição, o Pure Data é um ambiente comumente usado por músicos em geral, o desenvolvimento de um sistema tal como o proposto pode ajudar a manutenção ao Pure Data de forma mais prática, visto que esse público em geral não possui o conhecimento sobre computação necessário para gerenciá-lo autonomamente. Dado o interesse somado com a necessidade, o desenvolvimento desse projeto é justificado. Além disso, o estudo dos conceitos que abrangem a todo o projeto, tais como o gerenciamento de pacotes, o funcionamento de repositórios e a distribuição remota, são outros fatores que tornam o desenvolvimento desse projeto interessante e necessário academicamente.

1.4 Objetivos

De tal forma, o objetivo ao se desenvolver tal ferramenta, além do entendimento e estudo de conceitos relacionados e como eles se comportam de maneira distribuída, esse projeto visa permitir à usuários a possibilidade de alterar ou incrementar a experiência de desenvolvimento com o ambiente Pure Data, Tendo em vista as limitações do sistema em seu estado atual, além das limitações de usuários que não possuem de fato algum conhecimento sobre a alteração, instalação ou remoção de extensões e atualizações para o ambiente em questão, tanto como qualquer ambiente em si. Em suma, o objetivo é de fato criar uma ferramenta que permita à usuários a terem um controle prático e eficiente para a gerencia de plugins e extensões. Apesar de este trabalho ter como ambiente o Pure Data, o conceito de gerenciamento de atualizações e configurações de software pode, e é muito bem recomendado, para diversos outros ambientes que permitam o adicionamento de extensões, os conceitos aqui estudados e os diversos aspectos definidos poderão ser usados para estudo e aplicação em quaisquer outro trabalho relacionado.

2 Conceitos Envolvidos e Ferramentas Relacionadas

2.1 Conceitos Envolvidos

Esta seção apresentaŕa os conceitos estudados, usados de referencia e motivação para o desenvolvimento do trabalho em si. Como e o que é a atualização automática, quais conceitos são os mais adequados e o que precisamos entender para desenvolver o nosso próprio gerenciador.

2.1.1 Atualização Automática

Uma atualização à um software é a adição, remoção, melhora de algum elemento de um software, ao invés de se desenvolver e distribuir todo um sistema novo, a atualização permite que essas mudanças ocorram de forma prática, ou seja, o usuário não necessita de um conhecimento avançado sobre configuração de software para realizá-la. A automatização da atualização permite que ela seja instalada de forma trivial, permitindo o usuário escolher quais atualizações instalar ou até mesmo instalar sempre a mais recente(MENESES; HUZITA, 2009), de tal forma, o sistema verificaria as versões ou datas das últimas modificações ou atualizações em um software, caso se encontre um versão ou data inferior à de uma atualização existente, é instalada a mais nova versão.

2.1.2 Gerenciamento de Atualização de Software

Utilizar ferramentas para o Gerenciamento de Configuração de Software durante o desenvolvimento de um sistema de software é uma prática recomendada ao desenvolvedor. O gerenciamento de configuração de software pode ser feito utilizando um conjunto de ferramentas de apoio para o desenvolvimento e manutenção do mesmo. O controle de versão, o controle de mudança e a auditoria das configurações são alguns dos atributos dessa gestão. Essa gerência tem por objetivo analisar o que mudou e quando mudou, o porque dessa mudança, o que tal mudança fez e o que podemos fazer com essa mudança. Essas questões representam as atividades praticadas pelo gerenciamento de configuração de software, tal como o controle de versão, é capaz de dizer o que mudou e quando mudou, o controle de mudanças é capaz de atribuir os motivos a cada uma das mudanças, a auditoria por sua vez responde quem fez a mudança e o que podemos fazer com ela(MENESES; HUZITA, 2009).

Uma vez que houve mudança em um sistema, o próximo passo é atualizar as

instalações existentes de maneira automática e transparente ao usuário. Para isto, as mudanças auditadas pelo Gerenciamento de configuração de Software são refletidas em pacotes de software que poderão ser instalados por meio de ferramentas automatizadas que se utilizam de repositórios para proceder esta atualização.

2.1.3 Pure Data

O Pure Data(PUCKETTE, 2007) (Pd) é um ambiente de programação musical visual, focado em permitir que artistas, músicos e pesquisadores possam programar softwares graficamente, sem utilizar linhas de códigos. Este ambiente é utilizado para processar e gerar som e vídeos e foi projetado também para permitir a fácil colaboração em rede para músicos e colaboradores conectados via LAN ou mesmo distribuídos ao redor do mundo para criar música ou outros projetos em tempo real. Este ambiente é ainda utilizado para o ensino e aprendizado de processamento de mídia e programação visual assim como para o desenvolvimento de soluções de sistemas complexos de larga escala (BRINKMANN *et al.*, 2011) como jogos, sintetizadores ou outras aplicações que necessitam de recursos sonoros.

A programação em Pure Data é feita de maneira modular onde cada funcionalidade é fornecida por um objeto representado visualmente por uma "caixinha"(PUCKETTE *et al.*, 1996) (ver Figura 1). Vale lembrar aqui que a definição de um Objeto em Pure Data não segue a definição de Orientação a Objetos em linguagens de programação como C++ ou Java. Um objeto pode se conectar a outros objetos por meio de suas entradas e saídas, como um componente de software. Esta conexão é feita por meio de *inlets* (entradas) e *outlets* (saídas) dos objetos. A conexão de um objeto a outro é feita visualmente por meio de uma "cordinha".

Um programa em Pure Data é chamado de *patch* (remendo) e pode conter dezenas de objetos e conexões. A Figura 1 apresenta um *patch* com 3 objetos conectados entre si. Por esta figura é possível notar que nem todos objetos possuem entradas e saídas e que nem todas as entradas e saídas dos objetos são utilizadas para comporem um *patch*.

Este ambiente foi desenvolvido usando 2 linguagens de programação distintas sendo o *engine* feito em C e sua GUI feita em Tcl/tk sendo que *engine* e GUI se conectam por um socket. Há atualmente distribuições deste ambiente para diferentes sistemas operacionais como Linux, Unix, MacOS, Android e Windows. Além de conter vários objetos em sua distribuição oficial (chamada *Vanila*), o Pd pode ser estendido por meio de *plugins*.

2.1.4 Plugins

Um plugin é um programa de computador que serve como um extensor de funcionalidades, usado para adicionar, modificar ou até remover alguma funcionalidade específica para outro programa. Uma aplicação pode utilizar tal técnica por certos motivos, um dos

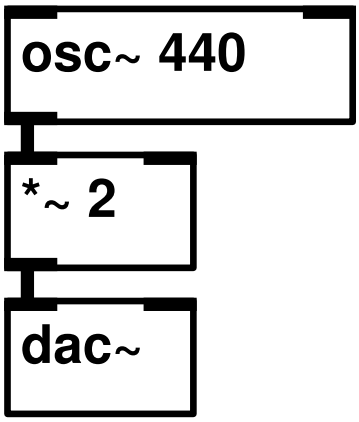

Figura 1 – Exemplo de um *patch* do Pure Data

principais é permitir que desenvolvedores de software externos estendam as funcionalidades do produto. Um produto que permite sua extensão por meio de plugins é chamado de host ou hospedeiro, ele provê serviços que o plugin pode usar, incluindo uma forma da extensão registrar a si mesma no hospedeiro e um protocolo para a troca de informações entre eles. Os plugins dependem de tais serviços, e geralmente não trabalham por conta própria. O hospedeiro por sua vez é independente, de forma que é possível adicionar e atualizar plugins dinamicamente, sem a necessidade de efetuar alterações no hospedeiro em si.

2.1.5 Plugins do Pure Data

O Pure Data, assim como outras aplicações de áudio, é um famoso hospedeiro, vários plugins são desenvolvidos por externos para diversas aplicações de áudio(SCHIAVONI ANTONIO JOSé HOMSI GOULART, 2012). Um plugin em Pure Data pode ser feito de três maneiras(ZMÖLNIG, 2001): a) no próprio ambiente Pure Data como um patch, sendo comumente chamado neste caso de abstrações, b) em Tcl/tk no caso de plugins que adicionam funcionalidades a GUI, também chamados de plugins de GUI e c) em C para funcionalidades que envolvem processamento de sinais, também chamados *externals*. É ainda possível desenvolver um plugin utilizando todas as possibilidades anteriores, como, por exemplo, desenvolver um plugin que possue uma GUI em TCL/Tk, a ajuda em Pure Data e processamentos em C.

Os plugins podem ser instalados no mesmo diretório do Pure Data, em um diretório do sistema compartilhado com todos os usuários ou em um diretório local do próprio usuário. Plugins do tipo abstrações possuem uma extensão **.pd** e são independentes do sistema operacional. Plugins de GUI, por serem feitos em TCL/Tk, são independentes do Sistema operacional e possuem obrigatoriamente um nome terminado com **-plugin** e extensão **.tcl**. Plugins externos dependem do Sistema operacional onde eles foram compilados e possuem extensões diferentes para cada Sistema, como, por exemplo, **.pd_linux**,

.pd_darwin (MacOS) e **.dll** (Windows).

Um plugin pode ainda conter um arquivo de ajuda que explica seu funcionamento. Arquivos de ajuda são feitos em Pure Data e possuem o mesmo nome do plugin mas com terminação **-help.pd**. Mais sobre escrita de plugins para o Pure Data pode ser encontrado em(ZMÖLNIG, 2001).

2.1.6 Pacotes

Um pacote de software é um aquivo único de certo formato constituído de um conjunto completo e documentado de programas, tais como sua documentação, manual de instruções e o guia, tais conjuntos e documentações estão, como se refere o nome, empacotado dentro desse arquivo. O pacote contém todos os arquivos necessários, sejam eles scripts, configurações, dados, documentação e outros, para a instalação do software, além de todas as informações necessárias para a instalação em si, a sua desinstalação e configuração. O pacote se encontra em um formato de arquivo definido, próprio para ser instalado por um sistema de gestão de pacotes ou instalado autonomamente.

Em outros contextos, um pacote de software pode ser considerado por ser um conjunto de classes e interfaces. Um grande exemplo de utilização de pacotes de software é o das distribuições Linux, que utiliza softwares em pacotes, manipulados por um gerenciador de pacotes. Essa gestão de pacotes torna mais fácil o processo de instalar e desinstalar softwares(FERREIRA, 2006). Um aspecto importante do pacote é o seu nome, é o principal referenciamento ao mesmo. Normalmente os nomes se dão ao nome do órgão desenvolvedor, como o RPM e o Debian, ou também especificamente à sua utilidade. Pacotes são distribuições de software e metadados, assim, é necessário que o mesmo possua informações essenciais ao uso como o nome completo, descrição, utilização, versão/revisão, fabricante, website, a lista de dependências caso necessária, arquitetura e licença. Esses metadados auxiliam à gerência tanto manual como automática, o nome do arquivo geralmente é formulado pelo nome e versão, é possível entender melhor o software lendo-se a descrição e utilização além de ser possível ter contato com o desenvolvedor tendo informações como o fabricante e o website fonte. Essas informações compõe o descritor do pacote, que faz parte da estrutura do mesmo.

2.1.7 Gerenciamento de Pacotes

Um sistema de gerenciamento de software consiste em um conjunto de ferramentas para executar a instalação, remoção, configuração e atualização de pacotes de software. Esse sistema é chamado de gestor de pacotes,e são comumente usado em sistemas Linux. O gerenciador de pacotes, obtém os pacotes de software de repositórios, resolve as dependências e os instala no sistema. O gerenciador de pacotes também facilita a re-

moção de pacotes ou a atualização dos mesmos. O número de pacotes disponíveis para instalação depende de quais repositórios existem adicionados aos sistema(FERREIRA, 2006)(NOVELL, 2011).

2.1.8 Repositórios

Um repositório de software é um local, geralmente um ou mais servidores, onde ficam armazenados pacotes de software para determinada aplicação. Estes servidores permitem que os pacotes possam ser recuperados e instalados na aplicação de maneira automática. A utilização de um servidor como repositório de pacote é um recurso muito utilizado por editores de software e outras organizações, mantendo um servidor na internet disponível para conexões HTTP ou FTP entre outras para este fim. Repositório normalmente fornecem também um sistema de indexação de pacotes que permite o gerenciamento e consulta dos pacotes pertencentes ao repositório. A utilização de repositórios públicos para pacotes de atualização de software pode transformar o gerenciamento de pacotes e atualizações em uma atividade transparente ao usuário e mais gerenciável ao desenvolvedor por evitar que o desenvolvedor envie uma atualização para cada cliente que possui sua aplicação instalada e evitar que o cliente tenha que lidar diretamente com atualizações.

2.1.9 Dependências

Um aspecto importante do repositório de pacotes são as relações contidas. Efetivamente, os pacotes também se relacionam com arquivos de outros pacotes, como os pacotes de aplicativos precisam de um ambiente de execução para realmente executar o aplicativo empacotado. As dependências do pacote são utilizadas para expressar tais relações. Sendo assim, um pacote necessita da instalação de outro pacote previamente para que o mesmo funcione corretamente. Dependências de pacote são transitivas, sendo por isso ter de instalar uma quantidade relativamente grande de pacotes antes de instalar o pacote desejado. Dependências em bibliotecas são comuns e boa parte de seus aplicativos individuais dependem de um conjunto de pacotes de bibliotecas. Os pacotes e as dependências do pacote são aspectos muito importantes das distribuições Linux, pois fornecem uma maneira modular para criar e gerir um sistema operacional e seus aplicativos. É também uma maneira eficiente de manter um sistema estável e seguro. Assim, quando uma falha afeta uma biblioteca usada por um ou vários aplicativos, atualizar um pacote singular atualizará o sistema para todos aplicativos.(NOVELL, 2011).

3 Trabalhos Relacionados

Dada a alta gama de softwares que utilizam conceitos similares aos julgados necessários para o desenvolvimento deste trabalho, foi feito um estudo a cerca desses softwares, sistemas e plugins a fim de que, de acordo com as definições justificadas por tais trabalhos sejam estudadas e possivelmente usadas no trabalho proposto.

3.1 Trabalhos Relacionados

Existem poucos trabalhos especificamente relacionados sobre a atualização automática para o ambiente Pure Data, porém, podemos estudar muito sobre trabalhos acerca de gerenciamento de pacotes e extensões do Pure Data.

Atualização Automática

O DiSEN (Distributed Software Engineering Environment)(HUZITA E. H. M.; SILVA, 2008) é um ambiente de desenvolvimento de software que, objetiva fornecer uma infra estrutura de desenvolvimento de software para equipes geograficamente distantes. É necessário portanto, que seja possível que os membros das equipes tenham em seus ambientes, as ferramentas sempre atualizadas para construção dos artefatos produzidos durante o processo de desenvolvimento de software. Além disso, é necessário que cada indivíduo seja notificado quando alguma alteração nas ferramentas associadas ao seu perfil for efetuada. E para realizar e gerenciar as atualizações do DiSEN, existe a ferramentaDiSEN-Updater(MENESES; HUZITA, 2009) que é responsável por gerenciar o processo de atualização do ambiente no qual esteja inserido. Essa ferramenta oferece tanto um conjunto de ferramentas utilitárias que auxiliam o usuário na gerencia dos módulos disponíveis no ambiente, quanto uma infra estrutura para garantir que os usuários utilizem apenas os módulos que foram autorizados, pelo seu gerenciador. Sobre o DiSEN-Updater temos diversos conceitos relacionados ao trabalho em si, como o uso de pacotes e repositórios, que serve de exemplo para a ferramenta proposta nesse estudo que utiliza dos mesmo conceitos.

Temos vários exemplos famosos de gerenciadores de pacotes de software, alguns deles serão apresentados aqui. Esses são vários trabalhos conhecidos nos sistemas Linux onde são aplicados os conceitos de gerenciamento de pacotes e atualizações. Segundo Cosmo, Zacchiroli e Trezentos (COSMO STEFANO ZACCHIROLI, 2008), os gerenciadores de pacotes podem ser classificados em duas categorias: instaladores, que implantam pacotes individuais no sistema de arquivos e meta instaladores, que atuam a nível interno

do pacote.

RPM (Red Hat Package Manager) (RPM.ORG, 2015), um gerenciador de pacote de sistemas, usado principalmente por sistemas Linux, capaz de instalar, verificar e atualizar pacotes de software, cada pacote .rpm contêm arquivos necessários além das informações sobre o mesmo (descrição, versão, tipo). Desde junho de 2010, existem 2 modelos do RPM, um deles, o RPM.org, é o modelo padrão da distribuição Red Hat do Linux e também do projeto Fedora.

Urmpi (URPMI.ORG, 2015), uma ferramenta de gerenciamento de pacotes, utilizada pelo Mandriva do Linux, para instalar remover e atualizar pacotes de software localmente ou remotamente, acoberta o gerenciador de rpm com a premissa de que o usuário não deve "sofrer" com depêndencias geralmente encontradas.

Apt-get (Advanced Package Tool) (APT-GET, 2009), uma interface gratuita para usuários que funciona com um núcleo de bibliotecas para lidar com a instalação e remoção de software nas distribuições Linux e variantes. o APT-GET simplifica o processo de gerenciar softwares em sistemas tipo Unix automatizando a recuperação, configuração e instalação de pacotes de softwares, tanto de arquivos pré compilados ou compilando o código fonte.

Synaptic (SYNAPTIC, 2015), um software com uma interface gráfica amigável para o sistema de gerenciamento de pacotes APT, tem sido um forte aliado para alguns novos usuários, facilitando à sua migração para várias distribuições do sistema operacional Linux, ele facilita a vida do usuário que precisa de mais tempo em produção de uso dos aplicativos e crê que não precisa aprender a instala los, uma vez que um software faz todo o trabalho bruto por eles, deixando mais tempo direcionado na produtividade que o usuário necessita.

APT-RPM (APT-RPM, 2009), uma versão do APT modificada para funcionar junto ao RPM.

Emerge (EMERGE, 2009), utilizado pelo Portage (sistema gerenciador de software da distribuição Gentoo Linux), é um gerenciador de pacote de sistemas baseado no conceito de coleções de portas.

Yum (YELLOWDOG, 2015), o atualizador do Yellowdog, é um gerenciador de linahs de comandos para Linux que utiliza o gerenciador de pacotes RPM, o Yum permite o gerenciamento de atualizações automáticas, pacotes e dependências em distribuiçoes baseadas em RPM e assim como o APT o Yum trabalha com repositórios que permitem acesso tanto localmente como remotamente.

dpkg (DEBIAN, 2015), um sistema de gerenciamento e instalação de pacotes utilizada pelo Debian e seus inúmeros derivados, usado para gerenciar pacotes .deb. Permite ao Debian acesso à diversos outros programas necessário para o sistema de empacotamento.

Up2Date (UP2DATE, 2015), é uma ferramenta que baixa e instala novos softwares e atualiza sistema operacionais, funciona como a etapa inicial para o gerenciador RPM e adiciona características avançadas como a solução automatica de dependencias.

YaST (Yet another Setup Tool) (OPENSUSE, 2015), é uma ferramenta que permite a configuração de vários aspectos do sistema, utilizado pela distribuição SUSE e suas distribuições comerciais derivadas.

Temos ainda, alguns trabalhos referente à utilização desses conceitos em um software, fazendo uma gerencia de processos e atualização própria. Entre eles, temos:

GATI (Gerência de Ambiente de Tecnologia da Informação) (RUZ F. W.; SANTOS, 2004), um ambiente integrado para administração de diretórios distribuídos, desenvolvido para gerenciar ambientes de tecnologia da informação. Utiliza um repositório de pacotes de software, no qual são depositados pacotes no formato RPM.

SOFA 2.0 (SOFtware Appliances) (BUREŠ *et al.*, 2006), um modelo de componentes sucessor direto do modelo de componentes SOFA(PLÁŠIL *et al.*, 1998) onde suas principais características destacam-se a distribuição de aplicações e, um ambiente de desenvolvimento distribuído com atualização dinâmica de componentes.

Eclipse (ECLIPSE.ORG, 2003), um ambiente de desenvolvimento integrado, uma IDE (Integrated Development Environment) desenvolvida na linguagem Java, que pode ser executada independentemente de plataforma. Pode ser descrito como um ambiente de programação empacotado como uma aplicação além de oferecer um ambiente para desenvolvimento de programas escritos na linguagem Java, este ambiente também fornece plug-ins para apoiar o desenvolvimento em outras linguagens, como C/C++ e PHP. O plugin funciona como um módulo de software que pode ser integrado na IDE. A arquitetura de plug-ins faz do Eclipse uma IDE extensível para outras linguagens de programação. Qualquer novo ambiente de desenvolvimento pode ser escrito como um plug-in e ser incorporado na plataforma.

NetBeans (NETBEANS.ORG, 2015), semelhante ao Eclipse é uma IDE de código aberto para Java. A plataforma pode ser utilizada para o desenvolvimento de aplicações. O Netbeans utiliza um conjunto de APIs (Appliacion Programming Interface) para conectar editores de código, compiladores, depuradores (debuggers) e outras ferramentas compatíveis ao ambiente. Uma das principais características do NetBeans é a sua modularidade, que possibilita o reuso em grande escala das aplicações.

Estes, são alguns famosos trabalhos relacionados ao gerenciamento de pacotes,que fazem uso de módulos de software, utilizam repositórios de pacotes de software, tratam do processo de atualização. Assim podemos ver a importância e motivação, além de ter conhecimento básico, sobre a utilização de um sistema de gerenciamento para um ambiente dedicado.

4 Desenvolvimento

4.1 Definição do Pacote para a ferramenta proposta

Estudados os principais conceitos relacionados ao trabalho proposto, é possível dar início ao desenvolvimento do mesmo. O primeiro passo feito neste desenvolvimento foi definir um formato de pacote a ser utilizado no sistema. A definição de pacote inclui a definição do nome e extensão do pacote, estrutura do pacote e descritor do pacote, conforme apresentada a seguir.

4.1.1 Extensão

A principal referência de um pacote é o nome do seu arquivo. Apesar de a maioria dos trabalhos estudados utilizarem como pacote um arquivo compactado (como zip ou rar), vários destes trabalhos utilizam uma extensão do nome de um pacote que se refere à sua aplicação. Exemplos encontrados de extensões de nome de arquivo são: deb para pacote da distribuição Debian, RPM é o nome de pacote da Distribuição Red Hat, nbm para plugins do NetBeans(MYATT *et al.*, 2008), xpi para plugins do Firefox(EDUARDO, 2013), crx para plugins do Chrome(CHROME, 2015), entre outras. Assim, é conveniente atribuir ao pacote proposto um nome relacionado a sua distribuição em vez de utilizar a extensão relacionada ao tipo da compactação do arquivo. Por esta razão, definimos o nome do pacote proposto para PDPK (Pure Data Package) em português: Pacote do Pure Data. Desta forma definimos a principal referência ao pacote a ser trabalhado, PDPK constará em nomes de arquivos, descritores e etc. . . .

4.1.2 Nome

O nome do arquivo do pacote deverá conter os atributos necessários para reconhecê-lo de maneira simples. Mais do que o plugin contido no pacote, é bastante útil ao usuário e ao desenvolvedor que o nome do pacote traga outras informações. Seguindo novamente exemplos de trabalhos relacionados, foi escolhido que o nome do pacote será composto do nome do produto juntamente com a sua versão, responsável por identificar entre versões novas e antigas do mesmo plugin, e a sua arquitetura, um aspecto importante para a verificação de compatibilidade entre sistemas. Definidos os mais importantes aspectos que irão compor o nome do arquivo do pacote de maneira legível, um exemplo de como seria tal composição é:

```
name-version-supported_architecture.pdpk
```

É importante salientar que o caractere que separará os aspectos no nome do arquivo será o hífen, e que tal caractere juntamente com o ponto, responsável por identificar a extensão do arquivo, não deverão ser utilizados em nomes de plugins. De forma alternativa, os espaços contidos serão substituídos pelo caractere subscrito, novamente por fins de legibilidade. Com o objetivo de prover um nome de arquivo fácil de identificar, manusear e ler, definimos assim o nome do arquivo do pacote.

4.1.3 Estrutura

A estrutura de um pacote é definida pelos arquivos que compõe o pacote e como eles são distribuídos dentro do pacote. Famosos pacotes como RPM(APT-RPM, 2009) e DEB(DEBIAN, 2014) possuem uma estrutura similar, o que permite o estudo de um certo padrão quanto a estrutura de um pacote proposto. De acordo com os pacotes estudados e também com a Norma NBR 12119(TÉCNICAS, 1998) referente à qualidade de pacotes de software, a estrutura de um pacote consiste na composição de: arquivos que formam o produto e seus dados, descrição do produto que contém todos os metadados relacionados ao produto e seu desenvolvimento, opcionalmente uma documentação do usuário, responsável por descrever o uso, instalação e manuseio do software e também opcionalmente a licença atribuída ao pacote. É necessário que todos esse arquivos componham um único pacote. Adicionalmente, a fim de padronizar descritores e documentações, é definido que os mesmos sejam em língua inglesa.

Estes são os elementos que contribuem à composição do pacote proposto. Os arquivos de documentação e descrição estarão contidos juntos aos arquivos da extensão e seus respectivos dados na própria fonte do pacote. Pensando nessa estrutura e novamente a fim de fornecer mais facilidade de manuseio e compatibilidade, o formato do pacote será um arquivo compressado .zip, um formato que permite uma fácil extração, permitindo até que a mesma seja feita manualmente por usuários. Um exemplo da composição do pacote pode ser vista no Código 4.1.

```
1  name_of_plugin-version-architecture.pdpk
2      +- name_of_plugin.pd_linux
3      +- name_of_plugin-help.pd
4      +- name_of_plugin-plugin.tcl
5      +- Descriptor.txt
6      +- README.txt
```

Código 4.1 – Exemplo da composição do pacote

4.1.4 Definição do descritor

O descritor do pacote é um conjunto de dados consistindo de toda informação necessária ao gerenciamento do produto. Assim a partir dos descritores estudados dos

pacotes RPM, Debian e APT-Get, foi possível definir os dados essenciais a serem usados pelo descritor proposto, sendo eles:

- Nome: Referenciará o nome do produto contido no pacote. Assim como descrito no nome do arquivo, o nome do pacote não pode conter hifens ou pontos.
- Sumário: Breve descrição ao funcionamento e uso do produto, com um limite de 144 caracteres.
- Autor(es): Nome do(s) desenvolvedor(es) ou órgão desenvolvedor do produto.
- Versão: Em qual versão de desenvolvimento o produto se encontra, versões alpha ou beta de um produto devem conter junto ao número da versão um "a"ou um "b"respectivamente para ser possível tal indicação.
- Arquitetura: A qual arquitetura o produto está destinado a funcionar, podendo ser mais de uma ou até mesmo todas. Este elemento que irá compor o nome do pacote em si, portanto é requerida a legibilidade de mesmo, bons exemplos são: (Linux x86, Windows 64,...)
- Data de Lançamento: A data em que o pacote será distribuído, em formato DD/MM/AAAA.
- Data de instalação: Referente a data em que o produto foi instalado, também em formato DD/MM/AAAA. Esta data será gerada automaticamente no processo de instalação.
- Tamanho: O tamanho em Bytes do produto, gerado automaticamente no processo de empacotamento.
- Licensa/Assinatura: Identificação da licensa.
- Grupo/Tipo: A qual grupo ou tipo de software o produto pertence. Especifica sua principal característica ou objetivo, auxiliando na sua identificação.
- Fonte: Referência ao código fonte do produto.
- Descrição: Documentação explicando o funcionamento, o propósito e todo outro aspectos importante ao produto em si.

*Além de opcionalmente termos:

- Dependências: Lista das dependências do produto.
- Conflitos: Lista de programas ou extensões que entram em conflito com o produto.

Quanto aos aspectos opcionais do descritor, caso algum ou todos não forem requeridos para especificação, o campo que os indica deverá ser deixado em branco. Além disso é necessário frisar que é requerido que a descrição seja o último campo do descritor para facilitar a escrita e leitura do mesmo.

A fim de alcançar uma melhor acessibilidade e compatibilidade, o formato do descritor será o .txt, sendo assim possível para usuários de qualquer arquitetura acessar o descritor, tanto como desenvolver o mesmo. O nome, novamente a fim da legibilidade, será "Descriptor.txt".

Um exemplo de descritor pode ser visto no Código 4.2.

```
1 name: example_plugin
2 summary: An example plugin
3 author: Lucas Nascimento Oliveira
4 version: 0.0.1
5 supported architectures: Linux
6 date of creation: 23/05/2015
7 size: 3000 Kb
8 license: GPL
9 type: examples
10 source: http://www.nononon.com.br
11 dependency:
12 conflicts:
13 description: This is an exemplary plugin, made for better understanding
     of the functionality of this descriptor file.
```

Código 4.2 – Exemplo de descritor

4.1.5 Documentação do usuário

A documentação do usuário é um arquivo opcional ao pacote no qual cabe ao usuário prover e adicionar ao pacote, ela funcionará como um arquivo de ajuda, será um arquivo contendo toda instrução sobre o uso, instalação e manuseio geral do produto. Qual o propósito do produto, suas aplicações, como melhor aproveitar suas funções, motivação de uso, restrições e recomendações ao sistema, como instalar ou desinstalar manualmente, como mudar certos aspectos do produto caso possível. Todas instruções de uso e suas recomendações devem ser especificado de forma clara e inteligíveis ao olhos do usuário. Também é bastante recomendado uma lista de instruções de como se comportar diante de possíveis erros, seja no uso, na instalação, ou em qualquer outro aspecto relacionado ao produto. Recomenda-se também a adição das mudanças, caso o pacote seja uma versão superior, que ocorreram ao plugin, o que foi modificado, adicionado ou removido pela nova atualização.

O Código 4.3 exemplifica tal documentação.

```
1 Installation: There should be at least two ways of instalation, one
      manually and another automatically, these are clear instructions to
      install both ways...
2 HOW-TO: For the proper use of this plugin after installing it...
3 About: This plugin was made for..., it should/must work with...
4 Modding: The user can change these settings of the plugin for these
      purposes...
5 HELP-ME: In case of troubles, the user should take these actions...
6 CHANGE LOG: What changed and what is new compared to the previous
      version
```

Código 4.3 – Exemplo de documentação

4.2 Definição de repositório

Como foi explicado e estudado anteriormente, a ferramenta proposta trabalhará com dois tipos de repositórios, um local e outro remoto. Um repositório será reconhecido caso o mesmo possua um descritor de repositórios, um arquivo .txt com propósito semelhante ao descritor de pacotes. Através deste descritor de repositórios será possível fazer os reconhecimento dos pacotes pertencentes, permitindo o compartilhamento de descritores ou uma busca refinada por algum pacote. Tais funções e definições serão explicadas a seguir.

4.2.1 Descritor de repositório

Para ser reconhecido como um, o repositório deve conter o seu descritor, um arquivo de formato .txt chamado "RepositoryDescriptor"composto de dados que permitam a gerência do mesmo. O descritor de um repositório será gerado localmente, vasculhará o endereço escolhido por todos os pacotes contidos, obtendo toda informação necessária para usos propostos. os dados usados para auxiliar na identificação de repositórios são:

- Proprietário: Nome ou identificação do criador, dono do repositório
- Data de criação: a data em que foi criado o descritor em formato (dd/MM/AAAA).
- Data da última modificação: a data em que o descritor foi atualizado pela última vez em formato (dd/MM/AAAA)

Estes são os dados que ajudam a identificar e gerenciar o repositório. Adicionalmente, teremos os dados de pacotes previamente vistos extraídos dos descritores de cada pacote contido no repositório. Tais dados irão auxiliar em buscas, onde o usuário pode procurar por um pacote através de seu tipo ou das arquiteturas suportadas. Sendo eles:

- Nome do arquivo
- Nome do pacote
- Versão
- Arquiteturas suportadas
- Sumário
- Tipo/Grupo

Com os dados do repositório e dos pacotes nele contidos, o descritor de repositório está pronto para reconhecimento e uso, a partir dessa etapa o repositório está pronto para ser usado remotamente, cabendo ao usuário a tarefa de armazená-lo no servidor de preferência

Um exemplo de descritor pode ser visto no Código 4.4.

```
1 Owner: Lukaz
2 Date Created: 02/11/2015
3 Last Modified: 06/11/2015
4
5
6 Plugin1-0.3-Linux.pdpk /Name /Version /Supported Architectures /
    Summary /Type/Group
7 Plugin2-0.5-Windows.pdpk /Plugin2 /0.5 /Windows /A plugin /
    Examples
8 Plugin3-1.7-MacOS.pdpk /Plugin3 /1.7 /MacOS /A plugin /
    Examples
```

Código 4.4 – Exemplo de descritor de repositório

4.3 Desenvolvimento da Ferramenta

Além da definição do pacote para plugins do Pure Data, o trabalho desenvolvido até o momento incluiu a criação de uma ferramentas com diversas funções, auxiliar o desenvolvedor a criar e remover novos pacotes, auxiliar o usuário a instalar e desinstalar novos pacotes, além de permitir que usuários mais avançados criem repositórios locais e remotos, permitindo que usuários possam baixar pacotes além de verificar por novas versões de pacotes instalados. As funcionalidades propostas foram todas implementadas em TCL/tk, uma vez que é a mesma linguagem utilizada pelo Pure Data, além de ser leve e rápida e possuir um suporte agradável à interfaces gráficas. O plugin, que fica integrado ao Pure Data, estará na aba do Pure Data onde o usuário tem acesso às várias funções fornecidas como mostra a Figura 2.

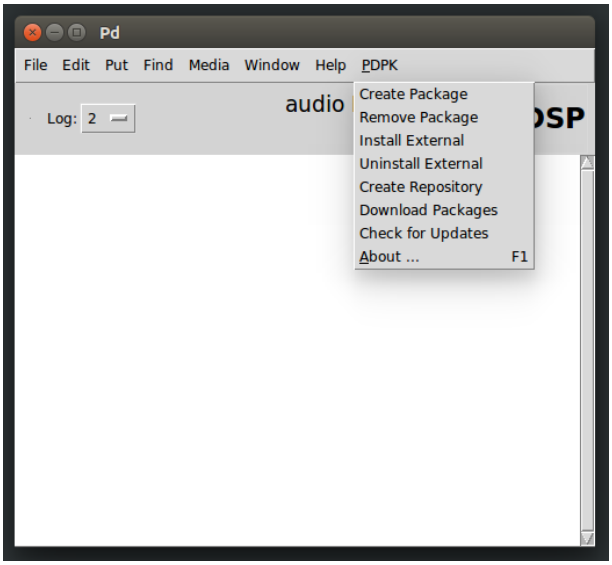

Figura 2 – pdpk integrado ao Pure Data e suas funções

4.3.1 Criação de Pacotes

A ferramenta empacotadora, responsável por criar o pacote, consiste em um plugin TCL/tk que auxilia o desenvolvedor a selecionar os arquivos que compõe o pacote e também a escrever seu descritor. Esta ferramenta cria o arquivo zip com extensão pdpk e define seu nome a partir dos meta-dados informados pelo desenvolvedor, conforme apresentado na Figura 3. O usuário pode escolher os arquivos que compõe o plugin, junto à uma documentação caso haja uma, através dos botões "Ad"na tela de criação, também existe o botão "Remove Selected"que permite o usuário reitrar algum arquivo que ele tenha previamente colocado pela função "Add". Adicionalmente, cabe ao usuário inserir os dados descritos na tela que irão compor o descritor, todos os dados são de preenchimento obrigatório, com exceção aos dados opcionais previamente descritos. Ao selecionar a opção "Create Package"a ferramenta comprime os arquivos selecionados em um pacote, no processo de empacotamento são adicionados as informações dos dados não atribuídos ao usuário como o tamanho do arquivo em bytes, além de serem feitas as renomeações necessárias a atender o padrão do pacote. Atendido os requisitos, o pacote estará pronto para uso, ou em uma melhor colocação, pronto para ser distribuído.

Tendo sido criado o pacote, o mesmo é armazenado em uma uma pasta específica para pacotes pdpk chamada de "pdpk-packages". Nesta pasta estará contido todos os pacotes criados e baixados pelo usuário

4.3.2 Instalação de pacotes

A ferramenta desempacotadora, responsável por extrair os arquivos do pacote e proceder a instalação do mesmo, é um ferramenta semelhante à empacotadora que permite ao usuário selecionar, através da mesma função "Add"apresentada previamente, o pacote

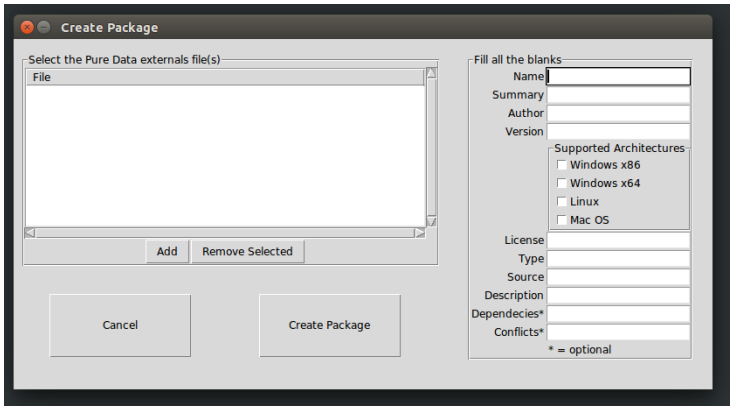

Figura 3 – Ferramenta Empacotadora: Seleção de arquivos e formulário para preenchimento do descritor

a ser instalado e a partir do mesmo extrair o(s) arquivo(s) que compõe o plugin em uma pasta de nome semelhante ao do pacote, na pasta de plugins do Pure Data (/pd-externals) juntamente com o seu descritor e documentação e licenças caso existentes. De tal forma, o Pure Data irá reconhecer a pasta e seus arquivos como um plugin e o mesmo já estará pronto pra uso.

A ferramenta irá escanear a pasta de pacotes específica à pacotes .pdpk e listará ao usuário todos os pacotes nela contidos como mostra a Figura 4. Ao usuário cabe escolher quais pacotes deseja instalar, caso algum pacote já esteja instalado uma mensagem informará o usuário.

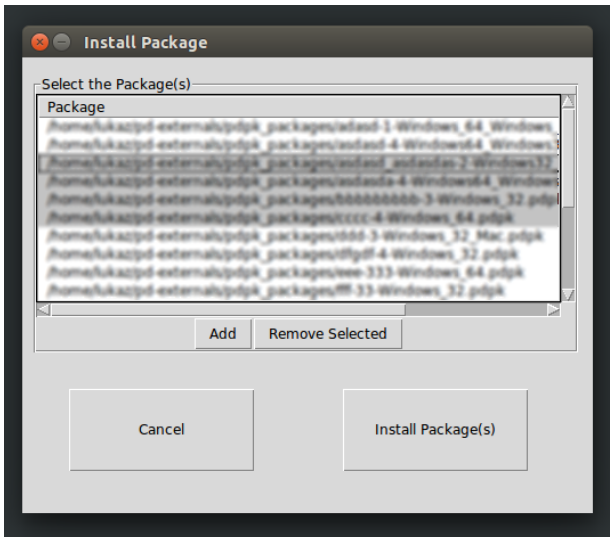

Figura 4 – Ferramenta Desempacotadora: Seleção de pacotes à serem instalados

4.3.3 Gerenciamento de Pacotes

Da mesma forma que a ferramenta permite a criação e instalação de pacotes, ela permite também a remoção e desinstalação de pacotes. Sendo um processo trivial,

consistindo apenas de exclusão de pastas e arquivos.

A ferramenta responsável à desinstalar pacotes irá vasculhar a pasta de plugins do Pure Data por todos os plugins instalados (Desde que os mesmos sejam originados de pacotes pdpk) e irá listar ao usuário quais dos plugins instalados ele deseja desinstalar como acontece na Figura 5. A ferramenta então, apaga as pastas escolhidas definitivamente do sistema, desta forma o Pure Data não reconhecerá o plugin mais, apesar de que o pacote do plugin desinstalado ainda estará armazenado em sua pasta padrão.

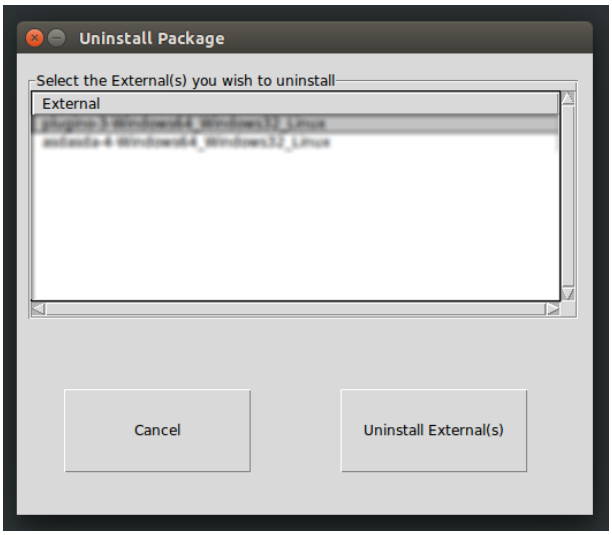

Figura 5 – Ferramenta Desinstaladora: Seleção de plugins a serem desinstalados

A ferramenta responsável por remover pacotes agirá de forma semelhante à ferramenta desinstaladora, ela vasculhará a pasta dedicada ao pacotes pdpk e listará todos os pacotes nela contidos, ao contrário da desinstaladora, a ferramenta que remove vai excluir o pacote definitivamente da pasta dedicada à pacotes. Ao usuário novamente, cabe selecionar quais pacotes ele deseja remover, conforme mostra a Figura 6. A ferramenta então, apaga os pacotes escolhidos definitivamente do sistema.

4.3.4 Gerenciamento de Repositórios

A ferramenta que faz a gestão de repositórios permite ao usuário criar dois tipos de repositórios: o principal e o personalizado como pode ser visto na Figura 7. O repositório principal é o que contém todo pacote pdpk que estiver instalado no momento da sua criação ou atualização. O repositório personalizado permite ao usuário escolher quais pacotes ele deseja adicionar ao mesmo. Ambos repositórios podem ser utilizados como repositórios remotos mas apenas o repositório principal será utilizado para verificar atualizações.

A ferramenta que gera o repositório principal irá vasculhar entre os pacotes instalados da mesma forma que faz a ferramenta desinstaladora e adicionará tais pacotes ao descritor de repositório, contendo juntamente as informações de proprietário, data criado

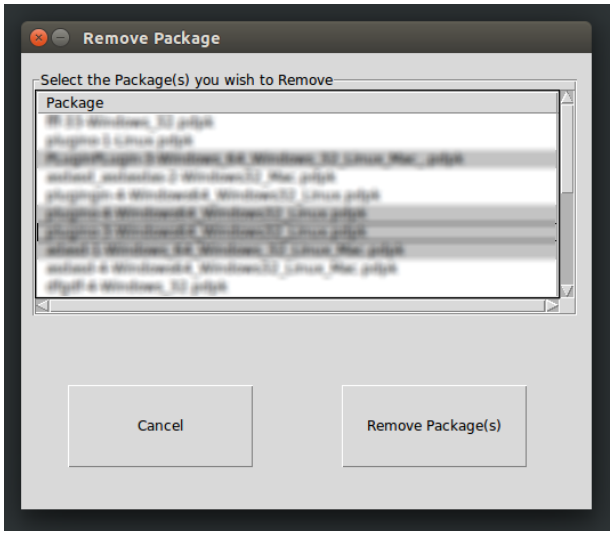

Figura 6 – Ferramenta Removedora: Seleção de pacotes à serem removidos

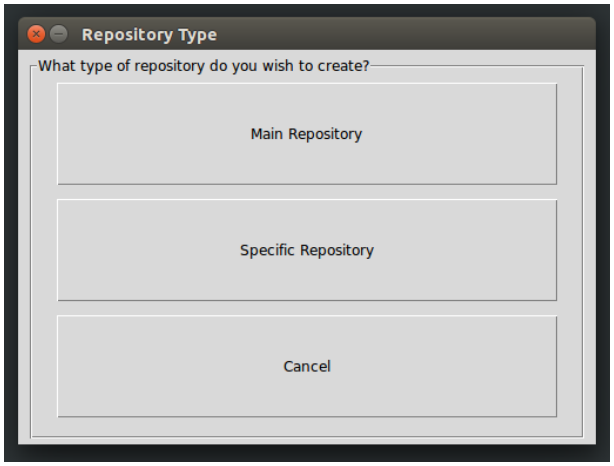

Figura 7 – Escolha do tipo do repositório a ser criado

e data da última modificação. Caso o descritor já exista, será dada a opção de atualizar ao usuário, sendo feito uma nova varredura por pacotes instalados, mas mudando apenas os pacotes e a data da última modificação. O repositório principal ficará em uma pasta denominada "pdpk-mainrepo"contendo seu descritor de repositório e seus pacotes.

A ferramenta que gera o repositório personalizado irá listar todos os pacotes existentes da mesma forma que a ferramenta removedora faz, assim o usuário poderá escolher quais pacotes ele deseja que façam parte do repositório como mostra a Figura 8, após isso a ferramenta cria o descritor do repositório de forma igual ao do repositório principal, o usuário tem a função de escolher em qual lugar ele deseja manter aquele repositório localmente, então uma pasta é criada com o nome de "pdpk-repo"contendo o descritor e os pacotes previamente escolhidos.

Após a geração dos repositórios, o usuário deve armazená-los em servidores de sua preferência para que o mesmo se torne um repositório remoto e assim seja possível o

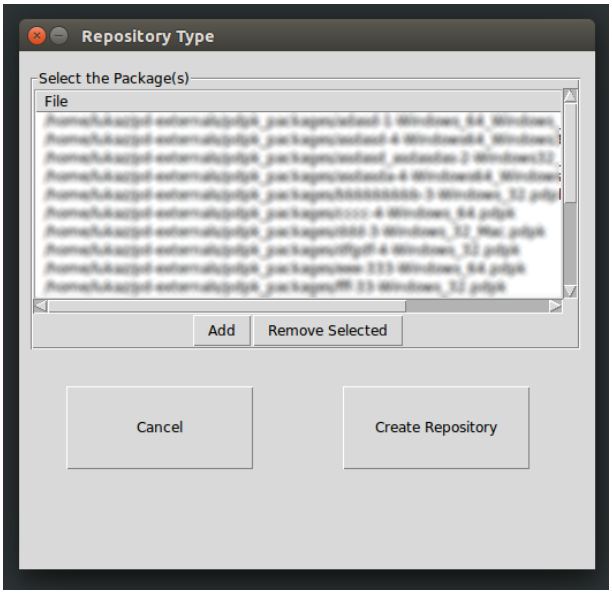

Figura 8 – Criação do pacote específico

download e a verificação de atualizações à outros usuários, tendo estes acesso ao endereço em que o repositório está armazenado remotamente.

4.3.5 Download de pacotes

A ferramenta que baixa pacotes de outros usuários armazenados remotamente, necessita, antes de tudo, de um endereço ao qual os pacotes desejados estejam armazenados. Dado o endereço, a ferramenta procura pelo descritor do repositório para que haja confirmação do endereço dado ser um repositório de pacotes pdpk, caso não ela não ache o descritor de repositório o processo de download fica inviável. Tendo achado o descritor do repositório no endereço dado, a ferramenta então permite ao usuário a listar os pacotes contidos no repositório, podendo ser utilizados uma ferramenta de busca para a refinação de resultados, o usuário poderá procurar por termos nos nomes, nas arquiteturas suportadas, nos tipos e nos sumários dos pacotes contidos no repositório, caso nenhum termo seja oferecido serão listados todos os pacotes contidos.

Tendo a lista de pacotes disposta ao usuário como mostra a Figura 9 ele poderá escolher quais ele deseja baixar. A ferramenta então baixa os pacotes escolhidos e os armazena na pasta dedicada ao pacotes pdpk, permitindo ao usuário instalar o pacote baixado logo após.

4.3.6 Verificação de Atualizações

Tal ferramenta tem a função de verificar os pacotes instalados e compará-los à pacotes em um dado repositório local a fim de procurar por versões mais novas de plugins. Ela utilizará dois descritores de repositórios para fazer uma comparação de pacotes. Os

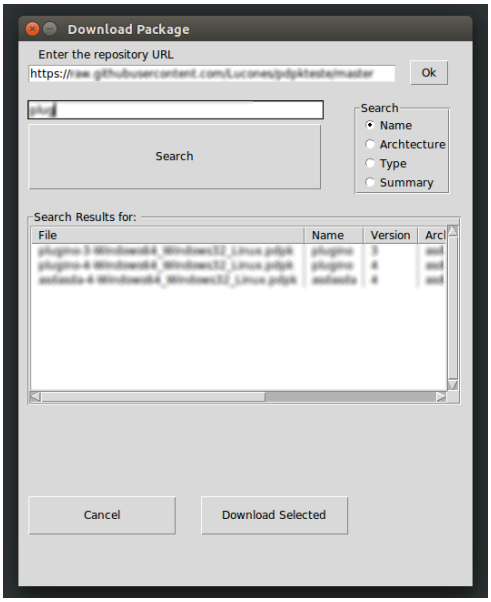

Figura 9 – Download de pacotes: Endereço, ferramenta de busca e lista de pacotes contidos no repositório

descritores de repositório utilizados serão o descritor do repositório principal, que contém os dados de todos os pacotes instalados, e o descritor do repositório remoto ao qual foi dado o endereço. A ferramenta age de forma semelhante à ferramenta que faz o download, dado um endereço remoto ela verifica se o mesmo é um repositório de pacotes pdpk, caso seja a ferramenta já analisará os descritores à procura de novas versões, os pacotes que possuírem uma nova versão no repositório remoto será listado ao usuário conforme a Figura 10, este deverá escolher quais atualizações deseja fazer, feito isso a ferramenta fará o download destes pacotes, cabendo ao usuário desinstalar a versão antiga caso queira e instalar a nova versão.

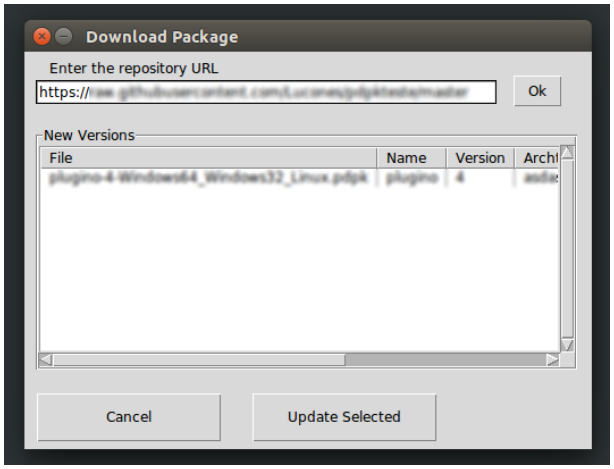

Figura 10 – Verificação de atualizações: Lista com os pacotes que possuem uma nova versão

5 Resultados Obtidos

5.1 Tipos de usuários

Com as definições e ferramenta desenvolvidas, o trabalho foca agora nos usuários. Foram definidos tipos de usuários que representam pra qual tipo de utilizador cada ferramenta se dispõe, os tipos não são exclusivos, uma vez que um usuário pode ser considerado fazer parte de todos os tipos definidos. Tais tipos de usuários são:

5.1.1 Usuário Comum

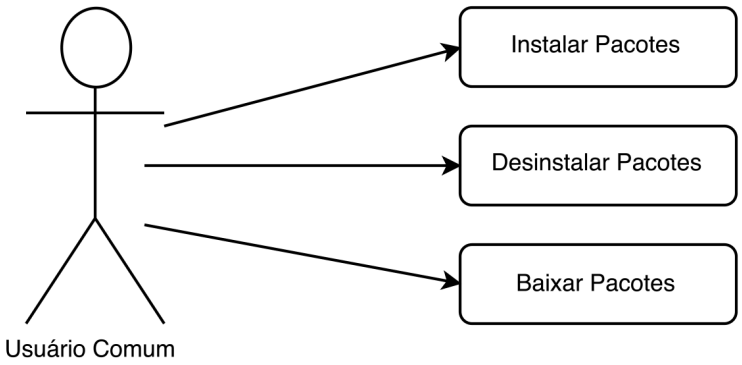

Figura 11 – Usuário Comum

Podemos considerar o usuário comum como o tipo mais básico de usuário mais básico, tal usuário seria algum utilizador que não tenha conhecido amplo sobre desenvolvimento de plugins ou relativo. A ele é destinado a instalação e desinstalação de pacotes além de fazer o download e atualização de pacotes, conforme descrito na Figura 11

5.1.2 Desenvolvedor

Tendo um conhecimento mais abrangente sobre plugins e criação dos mesmos, o desenvolvedor é o usuário que faz o plugin e o empacota para o mesmo ser distribuído, é destinado ao desenvolvedor o gerenciamento de pacotes, sua criação e remoção conforme mostra a Figura 12.

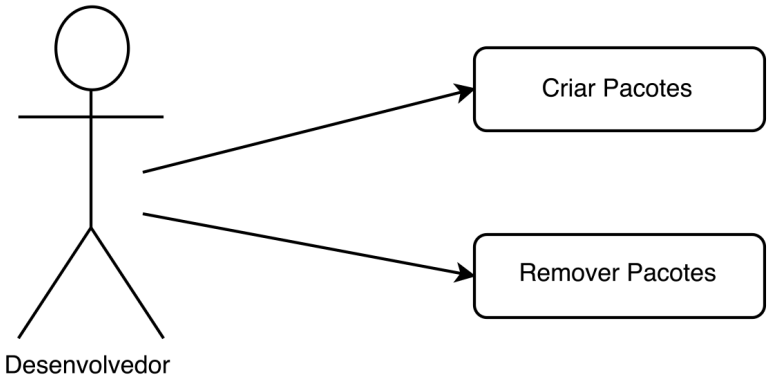

Figura 12 – Desenvolvedor

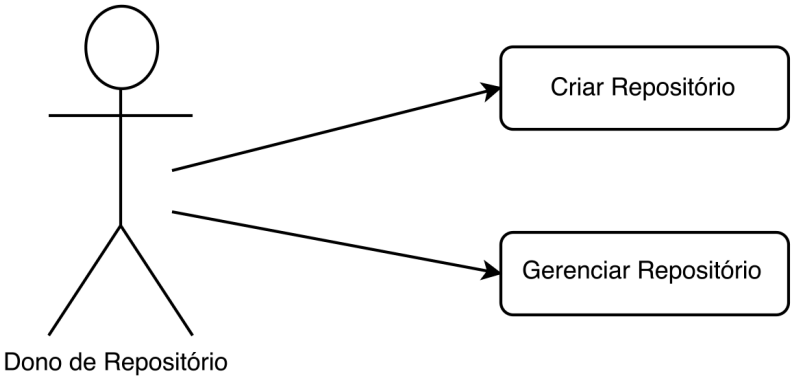

Figura 13 – Dono de Repositório

5.1.3 Dono de Repositório

O dono de repositório tem a função de disponibilizar a distribuição de pacotes, criando repositório remotos e armazenando pacotes nos mesmos, o dono de repositório é um usuário que tem um acesso abrangente a diversos repositórios, um usuário que possui um conhecimento sobre distribuição e disponibilização remotas. Suas tarefas são ilustradas na Figura 13

6 Conclusão e Trabalhos Futuros

Esta seção apresenta o conceito de conclusão do trabalho e o que podemos tirar do desenvolvimento da ferramenta, além de apresentar a ferramenta. E também dos próximos objetivos a serem alcançados na ferramenta, como pretendemos proceder em relação à desenvolvimento e quais os próximos passos a serem tomados.

6.1 Conclusão

Este trabalho apresentou os passos de uma pesquisa cujo foco é o desenvolvimento de um gerenciador de plugins para o ambiente Pure Data. Os passos iniciais foram dados para definir o formato e a estrutura do pacote a ser instalado no ambiente. Para auxiliar a tomada desta decisão, vários formatos de pacotes foram estudados como os pacotes do Debian, Red Hat, NetBeans, Firefox e outros.

O formato de pacote apresentado neste trabalho inclui tomada de decisões de alguns aspectos como: nome do pacote, formato, extensão, descritor e a estrutura interna do pacote. Tais decisões permitiram o desenvolvimento de uma ferramenta para auxiliar o desenvolvedor a empacotar seus plugins e uma ferramenta para auxiliar o usuário a instalar e gerenciar seus plugins. A criação de uma ferramenta para este ambiente propicia uma maneira mais simples de desenvolver projetos no Pure Data de maneira colaborativa, permitindo a desenvolvedores e usuários uma maneira simples de compartilhar códigos e plugins.

A ferramenta pdpk encontra-se disponível em um repositório público do Git-Hub mantido pelo criador, em sua página além da disponibilização da ferramenta instruções de uso e instalação estarão a mostra. tal repositório encontra se disponível no endereço https://github.com/Lucones/pdpk .

Apesar de este trabalho ter sido desenvolvido para o ambiente de programação Pure Data, as definições aqui presente podem auxiliar outros desenvolvedores a criar ferramentas de atualização similares para outra ferramenta.

6.2 Trabalhos Futuros

O primeiro passo em particular, se trata da distribuição da ferramenta para a comunidade do Pure Data, esperando algum feedback de usuários para eventuais mudanças e talvez expandir as funcionalidades da ferramenta conforme discussões forem feitas pela comunidade. Além disso têm se em mente a utilização de um repositório de grande

porte para funcionar como padrão, dando referência à usuários que não tiverem acessos a maioria dos repositórios que em primeira versão espera-se que sejam particulares em sua grande maioria. A ferramenta assim ganhará maturidade e eventualmente ser reconhecida para toda comunidade que a necessite.

Paralelamente ao projeto proposto e indo ao encontro da necessidade de tal iniciativa, foi iniciado um projeto de propósito semelhante já apresentado à comunidade do Pure Data intitulado de deken[1]. Este projeto visa facilitar o acesso do usuário aos externals do Pure Data através do próprio ambiente de desenvolvimento. Dada a semelhança da natureza do deken ao projeto pdpk aqui proposto, a fusão deste projeto junto ao deken é considerada, aliando o código desenvolvido com os conceitos aqui apresentados. Além disto, tal fusão pode trazer os benefícios do apoio da comunidade de desenvolvimento do Pure Data, a discussão e troca de ideias e a melhoria do ambiente como um todo.

[1] https://github.com/pure-data/deken

Referências

APT-GET. In: . [s.n.], 2009. Disponível em: <http://www.aptget.org>. Citado 2 vezes nas páginas 12 e 22.

APT-RPM. In: . [s.n.], 2009. Disponível em: <http://apt-rpm.org/docs.shtml>. Citado 2 vezes nas páginas 22 e 26.

BRINKMANN, P. *et al.* Embedding pure data with libpd. In: *Proceedings of the Pure Data Convention*. [S.l.: s.n.], 2011. v. 291. Citado na página 16.

BUREŠ, T.; HNTYNKA, P.; PLÁŠIL, F. Sofa 2.0: Balancing advanced features in a hierarchical component model. In: *Software Engineering Research, Management and Applications*. [s.n.], 2006. p. 40–48. Disponível em: <http://apt-rpm.org/docs.shtml>. Citado na página 23.

CHROME. *What are extensions?* 2015. https://developer.chrome.com/extensions. Citado 2 vezes nas páginas 11 e 25.

COSMO STEFANO ZACCHIROLI, P. T. R. D. Package upgrades in foss distributions: Details and challenges. In: *Proceedings of the 1st International Workshop on Hot Topics in Software Upgrades*. [S.l.: s.n.], 2008. Citado na página 21.

DEBIAN. 2014. http://manpages.debian.org/cgi-bin/man.cgi?query=deb&manpath=unstable. Citado na página 26.

DEBIAN. *DPKG*. 2015. https://alioth.debian.org/projects/dpkg. Citado 2 vezes nas páginas 12 e 22.

ECLIPSE.ORG. Eclipse. In: *Eclipse Plataform Technology Overview*. [s.n.], 2003. Disponível em: <http://www.eclipse.org/whitepapers/eclipse-overview.pdf>. Citado 2 vezes nas páginas 11 e 23.

EDUARDO, K. *Começando com as extensões do Firefox*. 2013. https://developer.mozilla.org/pt-BR/docs/XUL/School_tutorial/Comecando_com_as_Extensoes_do_Firefox. Citado 2 vezes nas páginas 11 e 25.

EMERGE. In: A Portage Introduction. [s.n.], 2009. Disponível em: <http://www-.gentoo.org/doc/en/handbook/handbook-x86.xml?part=2chap=1>. Citado na página 22.

FERREIRA, R. E. Gerenciamento de pacotes de software no linux. In: *Novatec Editora*. [S.l.: s.n.], 2006. p. 13–15. Citado 2 vezes nas páginas 18 e 19.

HUZITA E. H. M.; SILVA, C. A. W. I. S. T. T. F. C. Q. M. S. F. L. Um conjunto de soluções para apoiar o desenvolvimento distribuído de software. In: *Simpósio Brasileiro de Engenharia de Software. II Workshop de Desenvolvimento Distribuído de Software*. [S.l.: s.n.], 2008. Citado na página 21.

MENESES, R. C.; HUZITA, E. H. M. Disen-updater: Um mecanismo de atualização dinâmica de ferramentas para um ambiente de desenvolvimento distribuído de software. In: *Simpósio Brasileiro de Engenharia de Software. III Workshop de Desenvolvimento Distribuído de Software*. [S.l.: s.n.], 2009. Citado 2 vezes nas páginas 15 e 21.

MYATT, A.; LEONARD, B.; WIELENGA, G. Downloading, installing, and customizing netbeans. *Pro NetBeans^{TM} IDE 6 Rich Client Platform Edition*, Springer, p. 1–24, 2008. Citado 2 vezes nas páginas 11 e 25.

NETBEANS.ORG. *NetBeans*. 2015. http://www.netbeans.org. Citado na página 23.

NOVELL, I. Gerenciamento de pacotes. In: *Novell, Inc.* [S.l.: s.n.], 2011. Citado na página 19.

OPENSUSE. *Yast*. 2015. http://en.opensuse.org/YaST. Citado na página 23.

PLÁŠIL, F.; BÁLEK, D.; JANECEK, R. Sofa/dcup: Architecture for component trading and dynamic updating. In: *Software Engineering Research, Management and Applications*. [S.l.: s.n.], 1998. p. 43–52. Citado na página 23.

PUCKETTE, M. *The Theory and Technique of Electronic Music*. Hackensack, N.J: World Scientific Publishing Company, Incorporated, 2007. ISBN 9789812700773. Disponível em: <http://books.google.ca/books?id=TCtnWBfyhbwC>. Citado na página 16.

PUCKETTE, M. *et al.* Pure data: another integrated computer music environment. *Proceedings of the Second Intercollege Computer Music Concerts*, p. 37–41, 1996. Citado na página 16.

RPM.ORG. *RPM*. 2015. http://www.rpm.org. Citado na página 22.

RUZ F. W.; SANTOS, G. A. M. R. D. F. P. L. A. B. M. C. D. R. Uma ferramenta para a administração de serviços de diretório distribuídos baseados no openldap. *5º Fórum Internacional Software Livre*, 2004. Citado na página 23.

SCHIAVONI ANTONIO JOSé HOMSI GOULART, M. Q. F. L. Apis para o desenvolvimento de aplicações de áudio. In: *Instituto de Matemática e Estastística, Universidade de São Paulo*. [S.l.: s.n.], 2012. p. 209–216. Citado na página 17.

SYNAPTIC. *Synaptic Package Manager*. 2015. http://www.nongnu.org/synaptic. Citado na página 22.

TÉCNICAS, A. B. D. N. *NBR ISO/IEC 12119: Tecnologia de informação - Pacotes de software - teste e requisitos de qualidade*. [s.n.], 1998. Disponível em: <https://books.google.com.br/books?id=3Py6ZwEACAAJ>. Citado na página 26.

UP2DATE. *Using the Update Agent (up2date) from the command line*. 2015. http://www.redhat.com/advice/tips/up2date.html. Citado na página 23.

URPMI.ORG. *URPMI*. 2015. http://www.urpmi.org. Citado na página 22.

YELLOWDOG. *Yum – yellowdog updater modified*. 2015. http://yum.baseurl.org. Citado na página 22.

ZMÖLNIG, J. How to write an external for pure-data. *Institute for electronic music and acoustics*, 2001. Citado 3 vezes nas páginas 11, 17 e 18.