EMPACOTAMENTO DE SOFTWARE: O CASO DA FERRAMENTA HARPIA

Carlos Magno Geraldo Barbosa, discente em Ciência da Computação
Flávio Luiz Schiavoni, Departamento de Ciência da Computação

RESUMO

Empacotamento de software é uma etapa essencial no desenvolvimento e na distribuição de uma ferramenta. Portanto, esta pesquisa teve como seu principal objetivo realizar um estudo sobre empacotamento de software e aplicar este conhecimento no processo de empacotamento da ferramenta Harpia / Mosaicode. Este trabalho possui como objetivos secundários discutir questões iniciais relacionadas a distribuição e resolução de dependências de software.

1 INTRODUÇÃO

O Harpia foi desenvolvido originalmente pelo S2i[1]na UFSC, Universidade Federal de Santa Catarina, como uma ferramenta de visão computacional e geração de código orientada por blocos. O objetivo principal dessa ferramenta era de gerar uma solução de código aberto de controle de qualidade [2]. Esta ferramenta chegou a integrar repositórios oficiais Debian e de sistemas operacionais baseado no mesmo como por exemplo o Ubuntu, porém a mesma foi descontinuada em 2007.Com o objetivo de colaborar na recuperação desta ferramenta e a transformação da mesma em uma ferramenta multidisciplinar orientada por plugins, surge o presente projeto. Esta pesquisa também teve que avaliar as dependências do projeto, avaliar alternativas de empacotamento e

distribuição do software, criar ferramentas para distribuição do software e criar um pacote de instalação para a ferramenta. A nova ferramenta, chamada Mosaicode, tem como seu principal objetivo explorar as características de programação orientado por blocos e geração de código. Dentro deste projeto, o presente artigo, apresenta os esforços para garantir a distribuição da nova ferramenta. Todo software para ser utilizado em ampla escala precisa possuir um pacote de instalação rápido, simples e eficiente. Portanto um dos passos essências para garantir a usabilidade de um programa é fornecer este pacote de instalação, por este motivo este trabalho abordou este tema.

Tabela 1: Quadro comparativo entre as ferramentas Harpia e Mosaicode.

Característica	**Harpia**	**Mosaicode**
Geração de código em	Linguagem C	Llinguagens C e Javascript
Método de Programação	Orientada por blocos	Orientada a plugins
Objetivo	Controle de Qualidade	Arte Digital
Área	Visão Computacional	Multidisciplinar

2 AVALIAÇÃO INICIAL

A distribuição de um aplicativo é normalmente feita por um pacote de software.

Um **Pacote** é um arquivo normalmente compactado que contém arquivos com binário, shell scripts, documentação, imagens, xml e outros. Este arquivo contém toda a informação necessária para instalação, remoção, manutenção e atualização de um programa [3].

A instalação desse pacote pode ser realizada manualmente pelo usuário ou por um gerenciador de pacotes em sistemas Linux, este pacote pode ser um arquivo compactado

(formato .zip) ou em formato especifico de acordo com a distribuição Linux. Existe diversos formatos de empacotamento nas distribuições Linux, sendo que podemos destacar as seguintes:

• **Debian**: Extensão .deb , utilizado pelo gerenciador de pacotes dpkg e geralmente em distribuições baseada no Debian,

• **RPM**: Extensão .rpm, utilizado pelo gerenciador de pacotes RPM, Red Hat Package Manager,

• **TGZ**: Extensão . tgz, utilizada pelo gerenciador de pacotes pkgtool e geralmente utilizada na distribuição Slackware.

Este pacote facilita a instalação e a utilização para uma ampla gama de usuários, além disso o mesmo vai facilita a resolução de conflitos e instalação das dependências necessárias para o funcionamento do software. Estes arquivos também podem possuir assinaturas que são utilizados para garantir a sua integridade ou seja garantir que eles não estão corrompidos.

Normalmente, um pacote é manipulado por uma aplicação chamada **Gerenciador de Pacote**. O Gerenciador de Pacotes é um sistema que fornece um método [4] consistente de instalação, atualização e remoção de software. Além disso o mesmo deverá auxiliar no processo de resolução de dependências do software no momento da instalação.

As Dependências são as relações dos arquivos necessários para o funcionamento de um programa, esses arquivos são bibliotecas externas que foram utilizadas no desenvolvimento do produto. Portanto diversas vezes são necessários a instalação de diversos pacotes antes da instalação do pacote final. Estas dependências podem ser

resolvidas manualmente pelo usuário ou, como citado anteriormente, por meio do gerenciador de pacotes[5].

Para que as dependências possam ser resolvidas pelo Gerenciador de pacotes, as mesmas deverão ser explicitadas pelo desenvolvedor no pacote de software no momento do empacotamento. O **empacotamento** é uma parte essencial a ser considerada nas etapas de desenvolvimento de um projeto, a criação de um pacote de software é um processo de geração de um arquivo de instalação que irá conter toda a informação necessária para a instalação, remoção e configuração de um programa[3].

2.1 INTEGRIDADE DE PACOTES

Um ponto importante na distribuição de um pacote é a garantia da integridade do mesmo e que o pacote não foi corrompido durante a sua transferência. Diversos métodos de verificação de integridade realizam uma comparação da assinatura do arquivo original com a assinatura gerada no arquivo recebido, sendo que estas assinaturas deverão ser iguais para garantir a integridade desse pacote. Dentre os diversos métodos que podem ser utilizados para garantir essa integridade vamos enumerar os seguintes:

• Md5Sum - realiza uma comparação da hash md5 gerada no arquivo original e no recebido [6]

• sha224sum, sha256sum, sha384sum e sha512sum.

Um método seguro para garantir a integridade de um pacote é utilizar a ferramenta GNUPG[7]. Esta ferramenta utiliza o princípio de chaves assimétricas, em que são geradas duas chaves uma pública e uma privada, sendo que uma como o próprio nome já

diz, a chave pública será disponibilizada para os usuários e a privada será mantida em sigilo pelo gerador da assinatura.

Este processo consiste em obter a chave pública do gerador do pacote e comparar com chave que é distribuída juntamente com o pacote, por meio da ferramenta GNUPG.

2.2 GERENCIAMENTO DE VERSÃO

O controle de versão é um dos passos essências no desenvolvimento de um projeto, desde a sua versão inicial até a sua versão final toda modificação no código do software deve ser monitorada e registrada. Este controle fica por conta de um sistema de Controle de Versão. Um sistema de controle de versão [8] é uma maneira genérica de descrever a junção de sistemas que tem como objetivo manter registros cada alteração no código do projeto. No projeto foi utilizado a ferramenta Git e o site Gitgub[9] para fazer esse controle.

Além de controlar as versões de desenvolvimento, é necessário mapear a maturidade de cada versão no momento da distribuição do software para o usuário. A maturidade de um software é outro tópico importante nas decisões de desenvolvimento de um programa.

Outro fato que deve ser discutido é os passos necessários para uma mudança de versão de 1.0 para 2.0 . Programas comercias geralmente atualizam a sua major release anualmente, para garantir revenda de licenças. Softwares Livres geralmente realizam uma série de pequenas minor release antes de realizarem uma grande mudança e atualizar para 2.0. Portanto normalmente a mudança de um programa da versão 2.0 para 3.0 são mudanças significativas que podem no final gerar um software “novo”, cuja compatibilidade com versões anteriores não será garantida ou seja ele pode não garantir a retro compatibilidade [10].

Além disso um acompanhamento de mudanças de versão é importante para resolver problemas de dependências[11]. A medida que um sistema cresce e o número de dependências aumenta, um controle de versões é essencial para garantir compatibilidade do programa com as suas dependências, sendo que uma atualização major release desses códigos de terceiros pode resultar em uma instabilidade e mal funcionamento do programa principal. Portanto o controle de versão de um software é de grande importância para o programador e o usuário final.

2.3 REPOSITÓRIOS

Repositórios são um ou mais servidores onde ficam armazenados os pacotes para instalação ou atualização de um programa. Estes servidores podem normalmente ser acessados via internet. Geralmente os repositórios fornecem um sistema de indexação que permite gerenciamento e consulta de pacotes pertencentes ao repositório[5]. A utilização de um repositório público para pacotes de atualização pode deixar o processo mais transparente e gerenciável pelo usuário.

Por este motivo um dos grandes objetivos desse projeto é incluir o Mosaicode nos repositórios oficiais das principais distribuições Linux, uma vez que o Harpia já fez parte desse espaço.

Consideramos como repositórios oficiais os repositórios assinados pela distribuição Debian que contém apenas pacotes testados e seguros para serem instalados. Normalmente somente pacotes com versão estáveis e amplamente testados são adicionados nesses repositórios. Há ainda a possibilidade de criarmos um repositório a parte para a ferramenta, como um repositório **PPA,Personal Package Archives**, estes

repositórios são não oficiais e podem conter diversos pacotes que não foram garantidos ou liberados em repositórios oficiais.

Além dos repositórios para pacotes Debian, há ainda, os repositórios de aplicações python, como o Pypi Live e Pypi Test.

• Pypi Live - Destinado para os pacotes prontos para distribuição.

• Pypi Test - Contém pacotes que ainda não possuem uma versão estável para ser utilizado.

2.4 INSTALAÇÃO

O processo de instalação de um software exige que a resolução das dependências e conflitos entre pacotes sejam resolvidos nessa etapa, [12] este problema pertence à classe NP-completo, portanto pode ser difícil solucionar o mesmo. Por este motivo a resolução de dependências deve ficar por conta do programador, não é recomendado se exigir que essa solução fique por conta do usuário.

A medida que cresce o número e versões dessas dependências a complexidade também aumenta [12], tornando difícil encontrar uma solução eficiente para este problema. Portanto um sistema com inúmeras bibliotecas externas é suscetível a possuir uma instalação lenta e com elevado risco de conflitos.

Para facilitar a vida do programador e do usuário existe gerenciadores de pacotes que podem facilitar a instalação das dependências, porém devido à alta complexidade da relação entre esses pacotes eles podem não retorna resultados ou devolver resultados errados [12].

Além das dependências diretas, existe interdependência entre bibliotecas como pode ser observado na Figura 1 na dependência 2, nesse caso diferentes bibliotecas

precisam de um mesmo componente. Uma atualização dessas bibliotecas compartilhadas pode resultar em problemas de instabilidade. Um componente que possui ligação com inúmeros componentes pode ser definido como uma dependência forte [13], uma alteração nessas dependências resulta em um impacto direto em várias bibliotecas.

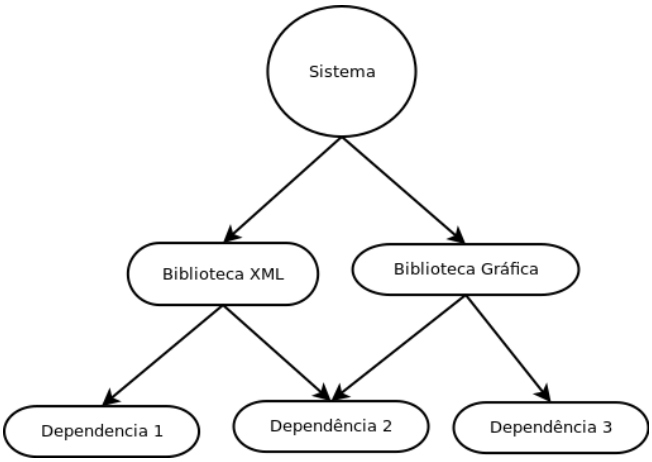

Figura 1: Exemplo de dependência e interdependência de pacotes de software

3 RESULTADOS

A ferramenta Harpia já esteve em repositórios oficiais Debian e devido a sua falta de manutenção, o projeto foi considerado descontinuado e por isto a mesma foi removida de repositórios oficiais. Por esta razão, no momento do início deste projeto o Harpia não contava com um pacote funcional de distribuição e instalação, sendo que a instalação do mesmo era executada por meio de um script, um código para resolver determinado problema, a resolução de todas as dependências necessárias deveria ser realizada manualmente pelos usuários.

Com o intuito de entregar uma solução funcional para o problema da instalação foram implementadas as seguintes medidas:

- **Criação de um pacote debian**: Foi criado um pacote debian com a listagem das dependências necessárias para o seu funcionamento, portanto esse pacote pode auxiliar na resolução do problema de dependências quando a sua instalação é realizada por meio de um gerenciador de pacotes.
- **Atualização do script de instalação**: O script de instalação do Harpia foi atualizado, o mesmo foi adaptado para gerar um pacote compatível com o repositório Pypi e foi adicionado uma verificação inicial de requisitos.
- **Criação de um script de resolução de dependências**: Para resolver o problema de dependências foi criado um script que automatiza a instalação dos mesmos.
- **Criação de um pacote funcional para o Pypi**: Foi disponibilizado para o novo Harpia um canal de distribuição no repositório de pacotes em python, Pypi, o pacote foi enviado para o servidor de teste desse repositório. O mesmo já pode ser instalado diretamente desse repositório.
- **Manual**: Foi criado um arquivo de manual que é adicionado durante a instalação na pasta de manuais da distribuição.
- **Questionamentos sobre o empacotamento**: Além dos resultados práticos apresentados, essa pesquisa levantou questionamentos e algumas respostas relacionadas a resolução de dependências, empacotamento e distribuição.

Atualmente, a instalação do Harpia pode ser feita com o seguinte comando:

sudo pip install -i https://testpypi.python.org/pypi harpia harpia.sh

No final deste trabalho foi disponibilizado uma versão de teste no repositório test pypi.

3.1 PACOTES DEBIAN

Pacotes com extensão .deb são pacotes utilizados na distribuição linux Debian[14] e em outras distribuições baseadas no mesmo. Considerando que o foco inicial de distribuição do Mosaicode seria nos sistemas baseados em Debian se explorou a criação de um pacote compatível com essa estrutura.

Se optou por iniciar a distribuição com pacotes .deb devido a ampla gama de distribuições Linux que oferecem suporte para o mesmo, além disso diversas distribuições linux populares aceitam os pacotes debian, dentre as quais podemos destacar as seguintes:

- Debian,
- Ubuntu,
- Mint,
- Elementary OS.

3.2 PACOTES PYPI

Um forma popular e prática de distribuir aplicações desenvolvidas em python é utilizar o repositório de aplicações python Pypi, Python Package Index. A instalação dos pacotes nesse repositório é feita por meio do gerenciador de pacotes pip. Considerando que o Harpia foi desenvolvido nessa linguagem, assumiremos que tal formato de distribuição é pertinente para esta aplicação.

3.3 CONTROLE DE VERSÃO E DISTRIBUIÇÃO

Neste trabalho adotamos a estrutura Major.Minor.Patch[11] , na figura 2 destacamos a utilização desse modelo destacando o major e o minor release.

Agora explicando este modelo de maneira detalhada podemos destacar as seguintes características:

• Major Release: Utilizado para indicar grandes mudanças, indicado no primeiro digito da versão.

• Minor Release: Utilizado para indicar pequenas correções e atualizações de segurança, indicado no segundo digito da versão.

• Patch: Utilizado para correção de bugs, problemas e defeitos de funcionamento.

Figura 2: Exemplo Major e Minor Release

4 CONCLUSÃO

Através deste trabalho foi possível implementar um método funcional de distribuição da ferramenta Harpia / Mosaicode e realizar uma discussão das áreas correlacionadas ao empacotamento de software.

Ainda destacamos a importância do empacotamento para distribuição de um programa e que a resolução de dependências deve ser resolvida pelo programador, uma vez que está tarefa pode ser complexa.

O empacotamento está diretamente entrelaçado a questão de dependências e distribuição de um programa, sendo que o mesmo é uma parte essencial do desenvolvimento de um software. Portanto, esse trabalho se baseou em auxiliar as questões de empacotamento e as questões correlacionadas, sendo que o mesmo foi capaz de levantar alguns questionamentos e resolver parcialmente a questão da distribuição da mesma.

No final do projeto, como destacado no tópico de resultados foi entregue um pacote de teste funcional que já facilita uma etapa de testes e distribuição do Harpia / Mosaicode.

Dentre os problemas encontrados podemos destacar a dificuldade de leitura de artigos em outro idioma e uma dificuldade inicial com as etapas de desenvolvimento de uma pesquisa.

Além do conhecimento prático e teórico obtido com o desenvolvimento dessa pesquisa foi possível desenvolver o pensamento cientifico e habilidade de resolução de problemas, o que no final resultou em um crescimento acadêmico e pessoal.

5 AGRADECIMENTOS

Agradeço a UFSJ, pela oportunidade de desenvolver um projeto de iniciação cientifica durante a graduação e ao meu orientador pelas inúmeras revisões, paciência e suporte durante a execução deste trabalho.

REFERENCIAS

[1] s2i. S2i - sistemas industriais inteligentes, feb 2017.

[2] D.C.L Junges and R.M Stemmer. Desenvolvimento de uma ferramenta para auxílio na educação, treinamento, implementação e gerenciamento de sistemas de visão. 2007.

[3] Rubem E Ferreira. Gerenciamento de pacotes de software no linux. São Paulo. Novatec Editora, 2006.

[4] opensuse. Gerenciamento de pacotes, March 2017.

[5] Oliveira N. Lucas. Atualização automática de software para o ambiente pure data. POC, 2015. [6] Gnu. Gnu, feb 2017.

[7] GnuPG. The gnu privacy guard, feb 2017.

[8] Jon Loeliger and Matthew McCullough. Version Control with Git: Powerful tools and techniques for collaborative software development. "O'Reilly Media, Inc.", 2012.

[9] GitHub. Github, feb 2017.

[10] Guia do Hardware. MD5SUM, 2011 (Acessado em Abril, 2016). http://www.hardware.com.br/termos/md5sum.

[11] Tom Preston-Werner. Semantic versioning 2.0. 0. línea]. Available: http://semver. org, 2013.

[12] Paulo Trezentos, Inês Lynce, and Arlindo L Oliveira. Apt-pbo: solving the software dependency problem using pseudo-boolean optimization. In Proceedings of the IEEE/ACM international conference on Automated software engineering, pages 427–436. ACM, 2010.

[13] Pietro Abate, Roberto Di Cosmo, Jaap Boender, and Stefano Zacchiroli. Strong dependencies between software components. In Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, pages 89–99. IEEE Computer Society, 2009.

[14] Debian. Debian the universal operating system, feb 2017.