Modelo de Empacotamento e Distribuição de Pacotes para o projeto *Mosaicode*

José Mauro da Silva Sandy[1]

[1]Universidade Federal de São João del-Rei (UFSJ)
Departamento de Computação

jmsandy@gmail.com

Abstract. *This article aims to propose a package packaging model for the Mosaicode project. The proposed architecture for distribution of the packages can be client-server or P2P, throughout the article the scenarios will be described in relation to the use of this architecture.*

Resumo. *Este artigo tem por objetivo propor um modelo de empacotamento e distribuição de pacotes para o projeto Mosaicode. A arquitetura proposta para distribuição dos pacotes poderá ser tanto cliente/servidor quanto P2P, ao longo do artigo serão descritos os cenários em relação à utilização dessa arquitetura.*

1. Introdução

O desenvolvimento de softwares é um processo complexo que passa por diversas fases ao longo do seu ciclo de vida. Para facilitar o processo de criação, manutenibilidade e utilização de softwares, componentes podem ser utilizados. Um componente de software é uma parte lógica e substituível de um sistema ao qual se adapta e fornece a realização de um conjunto de interface, sendo que que bons componentes tornam possível a substituição de componentes mais antigos por outros componentes mais novos [Booch et al. 2006].

A reutilização de artefatos de software traz a possibilidade de diferentes ganhos ao longo do processo de desenvolvimento, podemos citar:

1. **produtividade:** pode ser dado um foco maior na funcionalidade central do software;
2. **aprendizado:** possibilita aprender sobre a solução apresentada por um componente existente em funcionamento;
3. **cooperatividade:** componentes podem ser compartilhados entre projetos;

No contexto deste artigo, o processo de reutilização de componentes terá como base o projeto *Mosaicode*. [e Luan Luiz Gonçalves 2017] define o projeto *Mosaicode* como um ambiente de programação visual que utiliza o paradigma de encapsulamento de código em blocos, e cada bloco realiza uma tarefa específica. A visão geral do funcionamento do projeto *Mosaicode* é ilustrado na Figura 1, onde ao executar os blocos um aviso sonoro é emitido no navegador.

Como pode ser observado blocos podem ser combinados por meio de conexões para geração de novas aplicações. Portanto, tal combinação, pode dar origem a novos blocos complexos e estes se compartilhados com outros usuários podem dar origem a novas aplicações. Este artigo apresentará um modelo de empacotamento e distribuição de pacotes entre os usuários da aplicação. A seção 2 mostrará alguns modelos existentes de empacotamento e distribuição de pacotes, já a seção 3 apresentará um modelo para o projeto *Mosaicode* e, por fim, a seção 4 traz considerações sobre o modelo proposto.

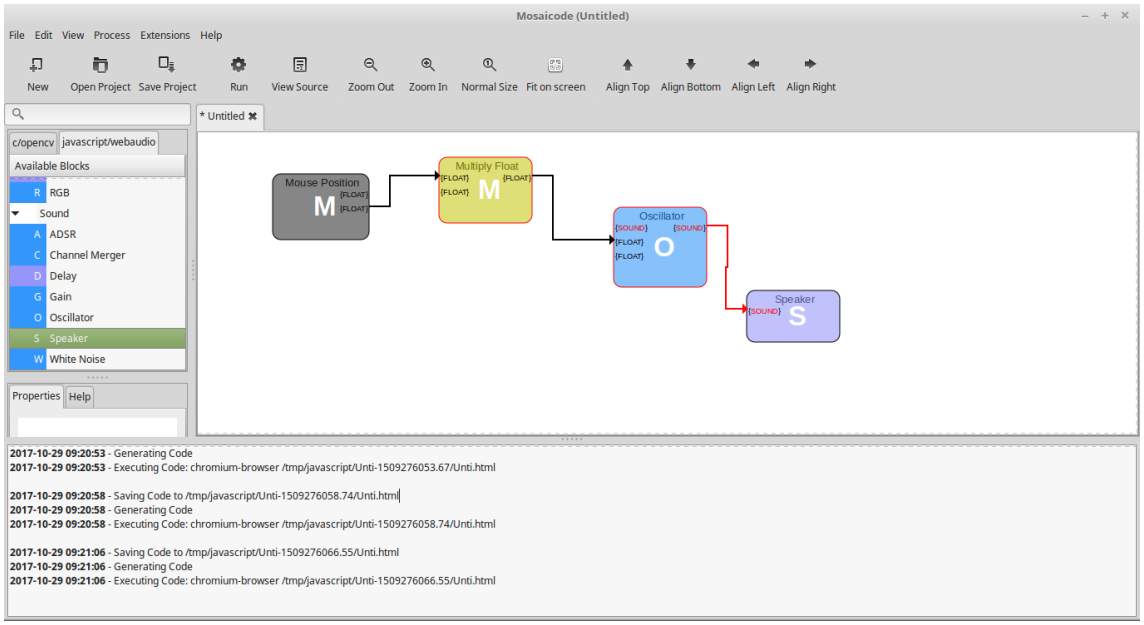

Figura 1. Ambiente de Programação Visual do *Mosaicode*

2. Modelos de Empacotamento e Distribuição

Nesta seção serão abordadas diferentes formas de empacotamento e distribuição de pacotes com base em modelos já adotados. A primeira parte apresentará as formas de empacotamento e em seguida as formas de distribuição.

2.1. Empacotamento

Um pacote de software consiste em um arquivo com um conjunto completo e documentado de programas que compõem a sua instalação, ou atualização, em um sistema. O mesmo contém todos os itens necessários de configuração, dependência, documentação e funcionalidade que fazem com que o funcionamento seja adequado com o especificado. Antes que um pacote de software possa ser distribuído é necessário que o mesmo seja empacotado sob uma padronização clara e objetiva para a correta distribuição e utilização. Os sistemas operacionais Linux, através de suas distribuições, trazem vários formatos de pacotes estabelecidos que são organizados e mantidos pelos sistemas gerenciadores de pacotes que possuem um conjunto de ferramentas para instalação, remoção, atualização e configuração de pacotes. Dois dos mais conhecidos gerenciadores de pacotes são os utilizados pelas distribuições baseadas em *Debian* e *Red Hat*, cujas suas extensões são *.deb* e *.rpm*, respectivamente. Em sua forma mais básica, a nomenclatura de um pacote contém o nome, versão, edição e plataforma, abaixo um exemplo desta taxonomia.

<nome>-<versão>-<edição>.<arquitetura>.<extensão>

Após a criação dos pacotes é necessário que os mesmos sejam distribuídos para uso. A distribuição se caracteriza por disponibilizar os pacotes em repositórios que serão lidos pelos requisitantes que procuram por um determinado software.

2.2. Distribuição

A distribuição de pacotes consiste em oferecer um conjunto de funcionalidades que são empacotadas e posteriormente disponibilizadas para utilização em algum ponto previa-

mente estabelecido. Ao distribuir um pacote espera-se que todas as dependências para seu funcionamento adequado sejam conhecidas e claramente documentadas. No contexto deste artigo, repositórios são considerados para distribuição dos pacotes. Repositório é um local onde os pacotes são publicados, ou disponibilizados, para pesquisa, instalação ou atualização quando solicitados.

Nas subseções que seguem serão apresentadas plataformas de distribuição de software que são utilizadas em algumas das principais linguagens utilizadas atualmente. A subseção 2.2.1 mostrará a distribuição dependências entre projetos *.NET*, já a subseção 2.2.2 apresentará os principais detalhes para projetos *Java*. Por fim, a subseção 2.2.3 discorrerá sobre a distribução de pacotes para a linguagem *Python*.

2.2.1. NuGet

NuGet é um gerenciador de pacotes para projetos *.NET*, suas ferramentes proveem a capacidade de criar, atualizar, consumir e compartilhar pacotes entre projetos. Atualmente, os pacotes disponibilizados em seu repositório central, atinge mais de noventa e cinco mil pacotes únicos e mais de um milhão se consideradas as versões disponibilizadas para cada um [NuGet 2017b].

Um pacote *NuGet* consiste em um arquivo compactado com a extensão *.nupkg* que contém os códigos compilados e os demais arquivos relacionados aos mesmos. Uma vez criado, um pacote deve ser publicado em um repositório para ser acessado. Além do repositório central, https://www.nuget.org, que se caracteriza por ser público e disponível a todos os usuários, é possível criar repositórios privados, em uma nuvem privada, rede local ou até mesmo, no sistema local de arquivos. Os repositórios privados auxiliam na distribuição de pacotes internos de uma organização que não devem ser externados para o público em geral. A figura 2 apresenta a distribuição dos pacotes mencionada deste a criação até o consumo dos pacotes em projetos finais.

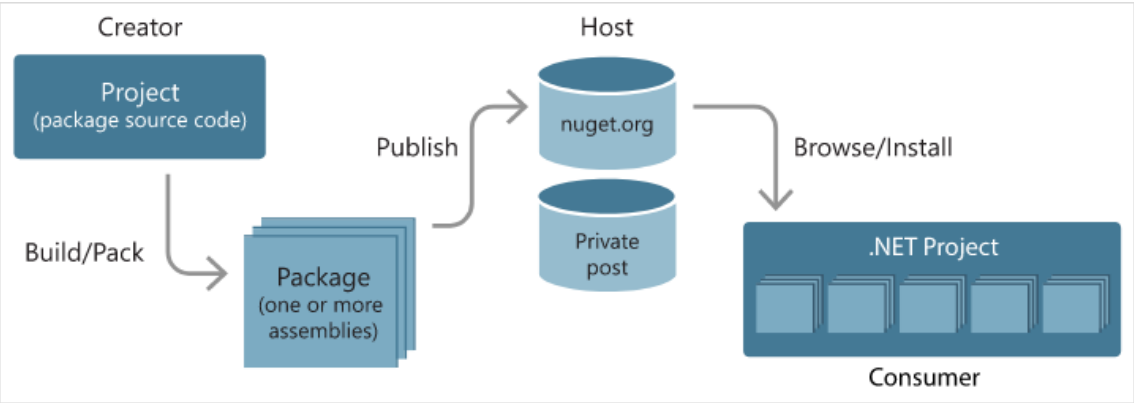

Figura 2. Distribuição de Pacotes *NuGet*. Fonte: NuGet [NuGet 2017a]

2.2.2. Maven

Maven é caracterizado por ser um sistema que gerencia e automatiza o processo de construção de um projeto de software. Utilizado em sistemas desenvolvidos em java,

utiliza um arquivo denominado *pom.xml* para especificar a estrutura, configuração, etapas de construção (por exemplo: compilação, empacotamento, testes, etc.) e dependências de um software. Tais dependências são obtidas automaticamente de repositórios, públicos ou privados, pelo *Maven* após a sua especificação em um arquivo *pom.xml* [Raemaekers et al. 2014].

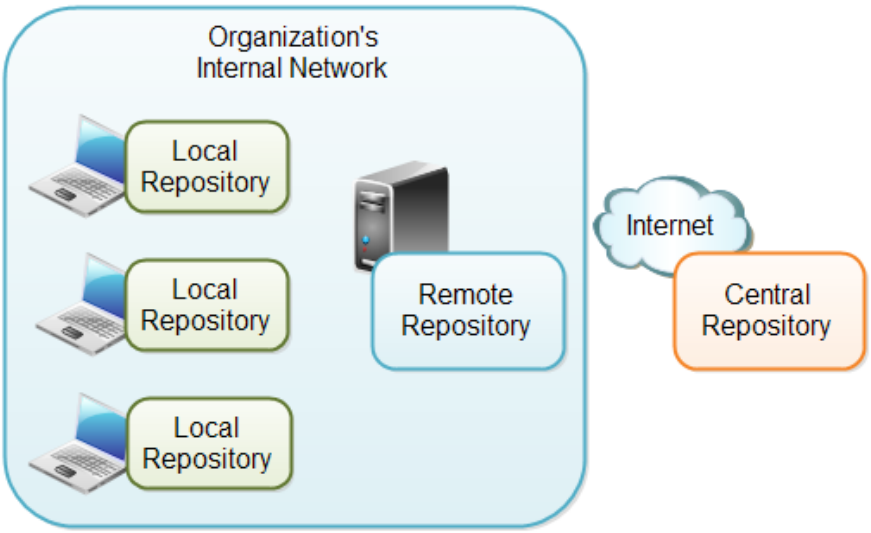

Figura 3. Distribuição de Pacotes *Maven*

Ao analisar a figura 3 é possível observar que existem três tipos básicos de repositórios que são utilizados pelo *Maven*: repositório local, repositório remoto e repositório central. Os dois primeiros tipos estão sob controle da organização e o repositório central é mantido pela comunidade *Maven*. O repositório local é um diretório presente no computador do desenvolvedor e possui um cache das dependências utilizadas em seus projetos, o mesmo tem como principal objetivo melhorar a performance da utilização das dependências vinculadas. Os repositórios remoto e central possuem as dependências que serão baixadas para o repositório local, a principal diferença entre um e outro e quanto a publicidade das informações, enquanto o repositório local está disponível de forma privada para alguma organização o repositório central é público e disponível para qualquer projeto que queira utilizá-lo. Normalmente, no repositório local encontra-se as dependências criadas por uma organização e todas as dependências comum e publicadas são obtidas do repositório central, ou seja, ao se configurar um novo projeto *Maven*, o arquivo *pom.xml* possuirá os endereços de acesso de ambos os repositórios para obtenção das dependências necessárias.

2.2.3. PyPI

A linguagem de programação *Python* possui um repositório de distribuição de pacotes denominado *PyPI - Python Package Index*, [PyPI 2017b]. Um pacote para ser distribuído neste repositório possui em sua estrutura alguns arquivos em sua definição, como: *setup.cfg*, *README.rst*, *MANIFEST.in*, *LICENSE.txt* e *setup.py*. O arquivo *setup.py*, o mais importante dentre eles, possui como principal funcionalidade configurar os aspectos

do projeto, [PyPI 2017a]. A estruturação dos arquivos são baseados em *PEP's - Python Enhancement Proposals*, que apresentam como a versão da biblioteca deve ser disponibilidade.

PyPI possui um repositório central com um grande número de bibliotecas públicas disponibilizadas. Assim como os gerenciadores apresentados nas seções anteriores, 2.2.1 e 2.2.2, existe a possibilidade de criação de repositórios privados e disponibilizar as bibliotecas apenas dentro do escopo desejado.

3. Empacotamento e Distribuição no *Mosaicode*

Um modelo de componente pode permitir a construção e descrição de sistemas e tornar possível o compartilhamento, [Zhijuan 2011]. Com este objetivo um modelo de empacotamento e distribuição será proposto para permitir que o compartilhamento entre os usuários possa ser alcançado.

3.1. Proposta de Empacotamento

O primeiro passo a ser adotado para a distribuição de pacotes é definir a forma de empacotamento que os pacotes devem seguir desde o momento da sua criação até a sua utilização por outro usuário. De uma forma geral, os pacotes criados pelo *Mosaicode*, será um único arquivo compactado com a extensão *mcpkg* e este por sua vez conterá todos os arquivos necessários para sua utilização, como: arquivos para execução, descritor, arquivo de ajuda e licença. O arquivo compactado com o conteúdo a ser publicado deverá seguir o padrão de versionamento definido em [SEMVER 2017] onde três incrementos (**MAJOR.MINOR.PATCH**) de versão podem ser utilizados para o pacote, que são:

1. **MAJOR**: quando fizer mudanças incompatíveis na API,
2. **MINOR**: quando adicionar funcionalidades mantendo compatibilidade, e
3. **PATCH**: quando corrigir falhas mantendo compatibilidade.

Desta forma, a nomenclatura do arquivo compactado gerado para distribuição entre os usuários será:

<nome>-<MAJOR.MINOR.PATCH>-<edição>.<arquitetura>.mcpkg

A estrutura do arquivo compactado contém todos os dados necessários para o funcionamento adequado do pacote. O arquivo descritor contém as informações pertinentes a especificação, funcionamento e dependência do pacote a ser distribuído. O mesmo será um arquivo com o mesmo nome do pacote, porém com a extensão *.json*. O formato *json* consiste em um arquivo formatado utilizado para troca de informações, o mesmo foi escolhido pela simplicidade de utilização e entendimento. O código 1 apresenta todos os campos presentes no descritor do pacote. A linha 2 contém o nome do pacote conforme apresentado anteriormente, as linhas 2, 3 e 4 correspondem as informações de versão, edição e arquitetura presentes no nome do pacote. Entre as linhas 5 e 20, são apresentados os dados de: tamanho (em bytes), data de lançamento, descrição, licença, grupo, palavras chaves, endereço de publicação, dependências e autores. Note que nas linhas 11, 16 e 20 é possível informar várias palavras chaves, dependências e autores, respectivamente.

```
{
  "name": "meupacote-1.0.0-release.x64.mcpkg",
  "version": "1.0.0",
  "edition": "release",
  "architecture": "x64",
  "size": 5000,
  "released_date": "2017-11-03T19:53:00Z",
  "description": "Package Description",
  "license": "MIT",
  "group": "audio-plugin",
  "keywords": [
    "audio",
    "oscilador"
  ],
  "url": "https://github.com/Mosaicode/mosaicode-javascript
     -webaudio",
  "authors": [
    "Autor 1",
    "Autor 2"
  ],
  "dependencies": [
    "mosaicode-components"
  ],
}
```

Código 1. Exemplo Arquivo Descritor

Além do descritor e arquivos para execução é necessário informar os arquivos de ajuda e licença, ambos não possuem um formato padronizado mas devem ser condizentes com o conteúdo do pacote. O arquivo de ajuda deve facilitar a interação do usuário com o pacote e o arquivo de licença deve deixar claro as regras de distribuição, caso não informado pelo usuário assumirá a licença padrão *GNU GPLv3*, [Foundation 2017], cujas as principais características são:

1. executar o sistema;
2. estudar e modificar seu código-fonte;
3. compartilhar livremente o programa com a comunidade.

3.2. Proposta de Distribuição

Tão logo tenha realizado o empacotamento, a distribuição do pacote pode ser realizada. Inicialmente, pode-se pensar em duas arquiteturas para distribuir os pacotes entre os usuários do *Mosaicode*, cliente/servidor e *P2P*, cada uma com características específicas que podem afetar diretamente a forma de distribuição dos pacotes. No modelo cliente/-servidor, cada computador ou processo na rede desempenha um papel claro, ou seja, se tornam clientes ou servidores. Um cliente é o local onde o usuário acessa em primeiro lugar para solicitar a informação e os clientes se comunicam entre si através de servidores. Já um servidor é um nó central que recebe as solicitações dos clientes e podem

comunicar diretamente entre si quando necessário. A principal vantagem deste sistema é que os clientes ficam livre das responsabilidades de processamento e armazenamento, [Zhu et al. 2016]. Uma das desvantagens desse modelo é a necessidade de um servidor para intermediar a troca de pacotes entre os clientes. Para o modelo *P2P* não existem regras globais que definam a topologia da rede, sendo assim, sistemas desta natureza empregam redes de sobreposição não estruturada, [Lehmann et al. 2013]. Neste modelo, os usuários do *Mosaicode* podem trocar os pacotes de forma descentralizada assumindo que cada instância da aplicação pode vir a ser tanto um servidor quanto um cliente, esta abordagem possui como principal dificultador as redes locais que ficam atrás de *NAT's - Network Address Translation* que tem como objetivo fazer com que computadores presentes dentro de uma rede local possam obter acesso externo ou a rede mundial de computadores, [James F. Kurose 2013].

Independentemente da conexão com o provedor de recursos ser utilizado, modelo cliente/servidor ou *P2P*, é necessário um protocolo bem definido para que os pacotes possam ser publicados, atualizados e recuperados. Para instalar um pacote, o usuário deve ter uma lista de repositórios conhecidos ou se estiver em uma rede local, chamadas *multicast* podem ser utilizadas para obtenção dos pacotes disponíveis.

Os repositórios podem ou não exigir alguma forma de autenticação, para estes casos um arquivo de configuração **mcpkg.config** será utilizado para manter a relação de repositórios e dados de conexão, as informações a princípio serão mantidas em texto plano neste arquivo. O código 2 apresenta exemplo de configuração para casos que necessite ou não de autenticação, onde: 1) linhas 4 e 8 apresentam os endereços de dois repositórios; 2) linhas 5 e 9, mostram o tipo de autenticação com os repositórios, sem autenticação e com autenticação, respectivamente; 3) linhas 10 e 11, representam o nome de usuário e senha para o repositório que requer autenticação.

```
1  {
2    "repositories": [
3      {
4        "address": "http://repository.mosaicode.com.br",
5        "authentication-type": "none"
6      },
7      {
8        "address": "http://repository2.mosaicode.com.br",
9        "authentication-type": "required",
10       "username": "username",
11       "password": "password"
12     }
13   ]
14 }
```

Código 2. Exemplo Arquivo Configuração

Neste artigo, é considerado que apenas as operações de publicação, atualização e remoção de pacotes necessitam de atualização e as informações inseridas no arquivo de configuração será manual mediante a intervenção dos usuários.

Repositórios de pacotes podem ser adicionados a qualquer momento e servem como ponto central para obtenção dos pacotes publicados pelos usuários do *Mosaicode*. Para adicionar um novo pacote é preciso informar o seu endereço de acesso no parâmetro *add-repository* para o aplicativo **mcpkg**, conforme abaixo:

$ mcpkg add-repository "http://repository.mosaicode.central.com.br"

Além da adição de novos repositórios, os mesmos podem ser removidos quando não forem mais necessários. Para tal, similar a adição de repositórios, o aplicativo **mcpkg** deve ser acionado informado o repositório a ser removido no parâmetro *remove-repository*.

$ mcpkg remove-repository "http://repository.mosaicode.central.com.br"

Uma vez adicionados os repositórios, todos os pacotes disponíveis em sua estrutura passa ser acessível para consulta e instalação.

3.2.1. Publicando Pacotes

A publicação de novos pacotes leva em consideração os itens empacotados pela especificação apresentada na seção 3.1. Não há necessidade de publicar pacotes em repositórios cujo acesso seja disponível pela internet, nestes casos a publicação será em um repositório local, mas estes serão disponíveis apenas para usuários que estejam na mesma sub-rede e não precisam ser adicionados na lista de repositórios dos clientes, pois a localização dos pacotes assim publicados dar-se-á por meio de *multicast*.

Os pacotes são publicados através dos aplicativo **mcpkg** sendo necessário fornecer o nome do pacote a ser publicado e o seu local de publicação. Para tal, os parâmetros *publish-package*, *name* e *repository* devem ser informados. Onde:

1. **publish-package:** indica a operação a ser realizada, neste caso publicação;
2. **name:** nome do pacote a ser publicado;
3. **repository:** endereço do repositório para publicação do pacote, é opcional e se omitido será publicado localmente.

Abaixo pode ser observado um exemplo de publicação de um novo pacote fornecido todos os parâmetros, inclusive o opcional.

$ mcpkg publish-package name="meupacote-1.0.0-release.x64.mcpkg"repository="http://repository.mosaicode.central.com.br"

Para a atualização de pacotes, a única alteração que se tem em relação à publicação de pacotes é o parâmetro relativo à operação a ser executada. Portanto deve ser substituído o parâmetro *publish-package* por *update-package*, conforme exemplo abaixo:

$ mcpkg update-package name="meupacote-1.0.0-release.x64.mcpkg"repository="http://repository.mosaicode.central.com.br"

3.2.2. Removendo Pacotes

Os pacotes podem ser removidos dos repositórios quando não são mais úteis para uma determinada situação. Logo, ao informar o parâmetro *remove-package* no aplicativo **mcpkg**, juntamente com o nome do pacote (*name*) e o seu endereço de repositório (*repository*), o mesmo será removido e não estará mais disponível para futuras instalações. Um exemplo de remoção de pacotes pode ser observado abaixo:

$ mcpkg remove-package name="meupacote-1.0.0-release.x64.mcpkg"repository="http://repository.mosaicode.central.com.br"

3.2.3. Procurando Pacotes

O ponto inicial para qualquer instalação é obter a lista de pacotes que estão disponíveis com base em alguma informação fornecida pelo usuário na busca do pacote. Desta forma, o usuário deve informar as informações referentes ao pacote para sua obtenção e posterior instalação.

$ mcpkg search-package "meupacote-1.0.0-release.x64.mcpkg"

Como pode ser observado acima, o aplicativo **mcpkg** recebe como parâmetro *search-package* o nome do pacote a ser procurado. A procura é realizada de duas formas: 1) o pacote é procurado em cada repositório registrado pela aplicação; 2) uma mensagem *multicast* é disparada para obtenção dos pacotes disponíveis na rede local.

3.2.4. Instalando Pacotes

Uma vez localizado o pacote desejado é possível instalá-lo mediante a utilização do aplicativo **mcpkg**. Para tal, deve informar os seguintes parâmetros:

1. **install-package:** indica a operação a ser realizada, neste caso instalação;
2. **name:** nome do pacote a ser instalado;
3. **repository:** endereço do repositório para obtenção do pacote, é opcional caso não haja duplicidade de nome dos pacotes publicados. Caso contrário, o mesmo deverá ser informado para resolver a duplicidade.

Desta forma um pacote pode ser instalado com base no comando abaixo:

$ mcpkg install-package name="meupacote-1.0.0-release.x64.mcpkg"repository="http://repository.mosaicode.central.com.br"

Caso um pacote já exista associado a aplicação o mesmo pode ser atualizado para uma versão mais nova. Assim sendo, os pacotes podem ser atualizados utilizando o parâmetro *upgrade-package*. O comando para atualização é similar ao de instalação, exceto pela alteração da operação em execução, conforme segue:

$ mcpkg upgrade-package name="meupacote-1.0.0-release.x64.mcpkg"repository="http://repository.mosaicode.central.com.br"

Um ponto importante a ser observado é que um cache relacionado aos repositórios e pacotes é realizado localmente para evitar que seja realizada comunicação em rede de forma desnecessária. Portanto, ao instalar, sempre é verificado se há uma cópia do pacote desejado localmente ou em algum repositório presente na sub-rede. Está sincronização entre repositórios presentes na mesma sub-rede se dá através da troca mensagens *multicast* que são trocadas para armazenar os possíveis pacotes que estão disponíveis próximo ao usuário.

4. Conclusão e Trabalhos Futuros

A correta padronização para empacotamento e distribuição de pacotes no projeto *Mosaicode* tem por objetivo facilitar a sua adesão e tornar a sua utilização mais ampla do ponto de vista de compartilhamento de recursos. A arquitetura com repositórios centralizados é mais simples do que a arquitetura com repositórios em sub-redes locais haja vista que os mesmos podem ser localizados por um nome válido (endereço válido de rede) e não estão escondidos atrás de redes *NAT's*. O processo de descoberta de repositórios associados às sub-redes é um processo que pode ser estendido utilizando técnicas de perfuração de *NAT* e assim permitir que uma verdadeira rede *P2P* seja seja criada para compartilhamento de pacotes.

Como trabalho futuro a implementação do modelo proposto deve ser realizada, afim de validar os conceitos de empacotamento e distribuição apresentados ao longo deste artigo. Além disso, técnicas de perfuração de *NAT* podem ser acrescidas ao modelo para permitir que nós que não estejam na borda da rede sejam descobertos por outros nós e assim compartilhar seu repositório local para outros usuários que não sejam aqueles presentes em sua sub-rede.

Referências

Booch, G., Rumbaugh, J., and Jacobson, I. (2006). *UML: guia do usuário*. Elsevier Brasil.

e Luan Luiz Gonçalves, F. L. S. (2017). Programação musical para a web com o mosaicode. *XXVII Congresso da Associação Nacional de Pesquisa e Pós-Graduação em Música – Campinas - 2017*.

Foundation, F. S. (2017). Gnu general public license. Disponível em: <https://www.gnu.org/licenses/gpl-3.0.en.html>. Acesso em: 03 nov. 2017.

James F. Kurose, K. W. R. (2013). *Redes de Computadores e a Internet: uma abordagem top-down*. Pearson Education do Brasil, sexta edition.

Lehmann, M. B., Antunes, R. S., and Barcellos, M. P. (2013). Exploring your neighborhood: Comparing connection strategies in swarming networks through evolving graphs. In *IEEE P2P 2013 Proceedings*, pages 1–10.

NuGet (2017a). An introduction to nuget. Disponível em: <https://docs.microsoft.com/en-us/nuget/what-is-nuget>. Acesso em: 31 out. 2017.

NuGet (2017b). What is nuget?. Disponível em: <https://www.nuget.org/>. Acesso em: 31 out. 2017.

PyPI (2017a). Packaging and distributing projects. Disponível em: <https://packaging.python.org/tutorials/distributing-packages/>. Acesso em: 31 out. 2017.

PyPI (2017b). Pypi - the python package index. Disponível em: <https://pypi.python.org/pypi>. Acesso em: 31 out. 2017.

Raemaekers, S., van Deursen, A., and Visser, J. (2014). Semantic versioning versus breaking changes: A study of the maven repository. In *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*, pages 215–224.

SEMVER (2017). Semantic versioning 2.0.0. Disponível em: <http://semver.org>. Acesso em: 01 nov. 2017.

Zhijuan, W. (2011). A software component model with sharing. In *2011 International Conference on Business Computing and Global Informatization*, pages 505–508.

Zhu, Y., Wu, W., and Li, D. (2016). Efficient client assignment for client-server systems. *IEEE Transactions on Network and Service Management*, 13(4):835–847.