UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

José Mauro da Silva Sandy

UAISharing - Interface Universal de Acesso a Recursos Compartilhados

São João del-Rei

2019

UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

José Mauro da Silva Sandy

UAISharing - Interface Universal de Acesso a Recursos Compartilhados

Dissertação de Mestrado apresentado ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São João del-Rei.

Orientador: Flávio Luiz Schiavoni

Coorientador: Elder José Reioli Cirilo

Universidade Federal de São João del-Rei – UFSJ

Mestrado em Ciência da Computação

São João del-Rei

2019

Ficha catalográfica elaborada pela Divisão de Biblioteca (DIBIB)
e Núcleo de Tecnologia da Informação (NTINF) da UFSJ,
com os dados fornecidos pelo(a) autor(a)

S222u

Sandy, José Mauro da Silva.
UAISharing - Interface Universal de Acesso a Recursos Compartilhados / José Mauro da Silva Sandy ; orientador Flávio Luiz Schiavoni; coorientador Elder José Reioli Cirilo. -- São João del-Rei, 2019.
72 p.

Dissertação (Mestrado - Ciência da Computação) -- Universidade Federal de São João del-Rei, 2019.

1. Empacotamento. 2. Distribuição. 3. Repositórios. 4. Compartilhamento. 5. Recursos. I. Schiavoni, Flávio Luiz, orient. II. Cirilo, Elder José Reioli, co-orient. III. Título.

José Mauro da Silva Sandy

UAISharing - Interface Universal de Acesso a Recursos Compartilhados

Dissertação apresentada como requisito para obtenção do título de mestre em Ciências no Curso de Mestrado do Programa de Pós Graduação em Ciência da Computação da UFSJ.

Trabalho aprovado. São João del-Rei, 29 de agosto de 2019:

Prof. Flávio Luiz Schiavoni
Orientador

Prof. Eduardo Bento Pereira
Universidade Federal de São João del-rei

Prof. Daniel Ludovico Guidoni
Universidade Federal de São João del-rei

São João del-Rei
29 de agosto de 2019

Dedico este trabalho à minha avó materna, "In Memorian", por ser essencial a minha formação como pessoa e estar sempre presente nos momentos mais difíceis sendo minha guia.

Agradecimentos

Por mais esforço que façamos para lembrar os momentos importantes em nossas vidas, sempre haverá alguns que não serão lembrados, não por serem irrelevantes, mas por estarem sobrepostos em nossas lembranças. Várias são as situações que enfrentamos em nosso dia a dia com a influência, direta ou indireta, de diversas pessoas, a estas: muito obrigado.

É inevitável citar algumas pessoas que tiveram uma participação especial nessa incrível jornada e participaram diretamente da minha formação como pessoa. Primeiramente, a minha avó materna, a eterna Dona Naná, que onde quer que esteja estará sempre olhando por mim, sendo minha inspiração e exemplo de superação: obrigado por me fazer ser quem eu sou. A minha namorada Flávia por sempre estar ao meu lado nos momentos mais difíceis da minha vida, não há palavras para agradecê-la por todo o apoio.

Também é impossível não agradecer ao meu orientador professor Flávio Luiz Schiavoni pelo aprendizado proporcionado ao longo destes semestres e da grande paciência que sempre teve comigo. Flávio, posso imaginar o desafio que foi aceitar um aluno fora dos padrões pré-estabelecidos, jamais me esquecerei disso: muitíssimo obrigado.

"Nós somos aquilo que fazemos repetidamente.
Por isso, a excelência não é um ato, mas sim um hábito."
(Aristóteles)

Resumo

A modernização das plataformas computacionais vem proporcionando o surgimento de novas características nos sistemas de informação que são pouco exploradas, e quando são, em boa parte, não apresentam padronização quanto à sua utilização. Neste cenário, um dos pontos a ser explorado é o compartilhamento de recursos criados pelos mais diversos tipos de usuário de sistemas computacionais. No entanto, é possível verificar que o compartilhamento de recursos entre usuários finais de sistemas computacionais costuma não possuir uma padronização para controlar seu empacotamento, distribuição e versionamento. Considerando as particularidades envolvidas em cada sistema de informação, torna-se necessário a definição de uma camada de abstração que propicie que este compartilhamento possa ocorrer de forma transparente entre os usuários dos mais diversos sistemas. Com isso, este trabalho traz o UAISharing - uma interface Universal de Acesso a Recursos Compartilhados, uma proposta que visa apresentar um modelo de empacotamento e distribuição de recursos produzidos por usuários finais.

Palavras-chave: empacotamento. distribuição. repositórios. compartilhamento. recursos.

Abstract

The modernization of computing platforms has provided the emergence of new features in information systems that are little explored, and when they are, for the most part, do not present standardization regarding their use. In this scenario, one of the points to be explored is the sharing of resources created by the most diverse types of computer systems users. However, it can be seen that sharing of resources among final users of computer systems does not usually have a standardization to control their packaging, distribution and versioning. Considering the particularities involved in each information system, it is necessary to define an abstraction layer that allows this sharing to occur transparently among users of the most diverse systems. Thus, this work brings UAISharing - a Universal Interface to access shared resources, a proposal that aims to present a model of packaging and distribution of resources produced by end users.

Keywords: packaging. distribution. repositories. sharing. resources.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CLI	Command-Line Interface - Interface de Linha de Comando
FTP	File Transfer Protocol - Protocolo de Transferência de Arquivos
HTTP	Hypertext Transfer Protocol - Protocolo de Transferência de Hipertexto
IoT	Internet of Things - Internet das Coisas
LAN	Local Area Network - Rede de Área Local
PEP	Python Enhancement Proposals
PyPI	Python Package Index
QoS	Quality Of Service - Qualidade de Serviço
RPM	Red Hat Package Manager
SCM	Source Code Management - Gerenciamento de Código-Fonte
SOHO	Small Office or Home Office - Pequeno Escritório ou Escritório Doméstico
SVN	Subversion
TCP	Transmission Control Protocol - Protocolo de Controle de Transmissão
UDP	User Datagram Protocol - Protocolo de Datagrama de Usuário
WAN	Wide Area Network - Rede de Longa Distância

Sumário

1 Introdução

A crescente oferta e utilização de recursos computacionais faz com que estes sejam distribuídos através das mais variadas formas de comunicação em rede. Quanto mais heterogênea for a distribuição destes recursos, maiores serão os desafios a serem enfrentados para que a sua utilização ocorra de maneira satisfatória para os usuários neles interessados. Neste cenário, o compartilhamento de recursos entre vários usuários pode se tornar uma tarefa dispendiosa caso este não apresente uma padronização que o oriente os usuários durante o processo de comunicação. Mesmo com os grandes desafios impostos pelo compartilhamento de recursos computacionais por meio de uma rede de computadores, este é um caminho cada vez mais adotado por empresas e profissionais mediante à utilização de ambientes de trabalhos denominados SOHO (Small Office or Home Office – Pequeno Escritório ou Escritório Doméstico). Uma abordagem sobre os detalhes que envolvem empresas SOHO, bem como aspectos de segurança envolvidos em sua aplicação, são apresentados por (1). Logo, o compartilhamento de recursos, envolvendo principalmente os usuários domésticos, pequenas e médias empresas, tende a ser um processo complexo ao qual não se encontram padrões estabelecidos que propiciem a máxima produtividade, confiabilidade e integridade para os procedimentos realizados ao longo da sua realização.

O compartilhamento de recursos entre usuários é muito bem definido para alguns grupos específicos, como os usuários desenvolvedores, onde muitas ferramentas podem ser utilizadas para este fim. Para estes usuários, destacam-se dois tipos de repositórios: i) repositórios de recursos prontos para serem utilizados, como: programas e bibliotecas e ii) repositórios de códigos-fonte utilizados para criação de programas. O primeiro tipo de repositório consiste na distribuição de algo já pronto que possa ser utilizado para construção de outros programas, como as bibliotecas, ou como uma aplicação final, como os programas. Como principais representantes deste tipo de repositório, têm-se: NuGet, PyPi, Maven, Apt, dentre outros. Já o segundo, possibilita o compartilhamento dos códigos-fonte que dão origem aos recursos que são distribuídos pelo primeiro tipo de repositório. Tal compartilhamento, pode ser realizado tanto em repositórios centralizados quanto em repositórios distribuídos. Atualmente, o Git e o SVN são os principais meios de distribuição deste tipo de repositório. Estes dois tipos de repositórios permitem alcançar um nível bastante satisfatório de controle sobre as modificações dos recursos que compõem alguma solução de sistema e dos recursos desenvolvidos. Além disso, estas ferramentas permitem realizar todo o controle de versão dos recursos presentes nestes repositórios, permitindo assim, o compartilhamento adequado de novas versões dos produtos criados. Devido aos pontos apresentados, pode-se assumir que a distribuição de recursos de software está bastante clara para este grupo de usuários devido ao fato desta já contar com

uma infraestrutura computacional suficientemente madura para auxiliá-los no compartilhamento.

No entanto, para usuários que não se enquadram neste grupo, tais como: artistas, secretárias (os), advogados, médicos, contadores e muitos outros, o compartilhamento não possui um processo bem definido e unificado que permita uma experiência satisfatória na distribuição e utilização dos recursos desenvolvidos. Comumente, estes usuários compartilham documentos, planilhas, apresentações e, muitos outros recursos, entre si. Para estes usuários uma mescla dos dois tipos de repositórios apresentados pode ser uma opção viável e interessante para aumentar o controle sobre o compartilhamento dos recursos, pois desta forma uma única abstração é criada sobre os recursos, possibilitando assim, uma maior adaptação em relação ao processo de compartilhamento.

Pode-se ter como exemplo destes cenários, os contadores e administradores de empresas. Estes, em seus escritórios, podem utilizar planilhas para controlar a apuração de impostos dos clientes, sendo estas compartilhadas entre seus diferentes departamentos (contábil, fiscal, recursos humanos, etc.). Não havendo uma padronização adequada para o compartilhamento, o conteúdo destas planilhas pode ser afetado acarretando uma série de consequências, algumas com imensa significância: como o cálculo de um imposto utilizando parâmetros defasados ou a sobreposição de um mesmo orçamento de um fornecedor que acaba por ser gravado no mesmo diretório.

Embora o compartilhamento de recursos seja o objetivo final dos usuários, é necessária uma série de etapas de apoio que devem ser realizadas para que o mesmo seja alcançado de maneira simples e segura. Dentre estas etapas, destaca-se os controles de modificações e de versões, pois são de suma importância na manutenção da consistência dos recursos compartilhados. O controle de modificação tem por finalidade informar aos usuários quais recursos foram adicionados, alterados ou removidos, além de indicar quem foi o responsável por estas modificações e quando as mesmas ocorreram. Já o controle de versão permite o usuário acompanhar a evolução do recurso utilizado e, porventura, escolher uma versão que melhor se adéque às suas necessidades.

Existem diversas soluções que permitem o compartilhamento de planilhas, documentos de textos e diversos outros aplicativos, podendo estes serem acessados online através de plataformas de nuvem, utilizando recursos, como: Google Drive, Office 365, dentre outros. Porém, alguns documentos por questões de confidencialidade não podem estar em uma plataforma de nuvem, haja vista que não há como uma empresa que utiliza tais serviços garantir a localização exata dos recursos disponibilizados nesta plataforma. Embora existam serviços que garantam a região geográfica onde os recursos ficarão disponibilizados, não há como precisar a sua localização exata, pois não se tem o controle sobre a infraestrutura de compartilhamento contratada. Além disso, tais serviços muitas vezes se limitam a um conjunto de aplicativos, como os voltados para escritórios, e podem não

oferecer um suporte satisfatório aos recursos que não se enquadrem nestas características. Há também uma questão de segurança e privacidade nestes serviços que impedem empresas que possuem dados confidenciais de utilizá-los para resolver seus compartilhamentos. Por fim, a obrigatoriedade de conexão com a Internet para a utilização destes serviços é outro quesito impeditivo para a adoção ampla dos mesmos em alguns contextos, pois não há garantia de disponibilidade em todas as situações. Por estas razões, acreditamos que tais serviços não resolvem alguns problemas aqui apresentados.

Neste modelo é ainda possível de expansão ao tratar do mesmo como computação nas nuvens. A computação em nuvem é um modelo para habilitar o acesso ubíquo, conveniente e sob demanda, por rede, a um conjunto compartilhado de recursos de computação (e.g., redes, servidores, armazenamento, aplicações e serviços) que possam ser rapidamente provisionados e liberados com o mínimo de esforço de gerenciamento ou interação com o provedor de serviços (2). Vale ressaltar, que o advento da Internet das Coisas (IoT), um novo paradigma surgiu, computação em névoa. A computação em névoa é um paradigma que estende a computação em nuvem e traz os serviços para a borda da rede, suportando aplicações emergentes da Internet das Coisas (IoT) que exigem latência em tempo real/previsível (automação industrial, transporte, redes de sensores e atuadores) (3). Talvez uma solução em névoa fosse mais adequada para o problema aqui apresentado, pois envolveria cópias locais de dados disponibilizados remotamente.

Com base no exposto é possível perceber que para usuários desenvolvedores o compartilhamento de recursos é uma realidade consolidada e presente em seu dia a dia. No entanto, para a maioria dos outros grupos de usuários, o compartilhamento, mesmo que existente, não apresenta as características necessárias para garantir a melhor experiência de utilização.

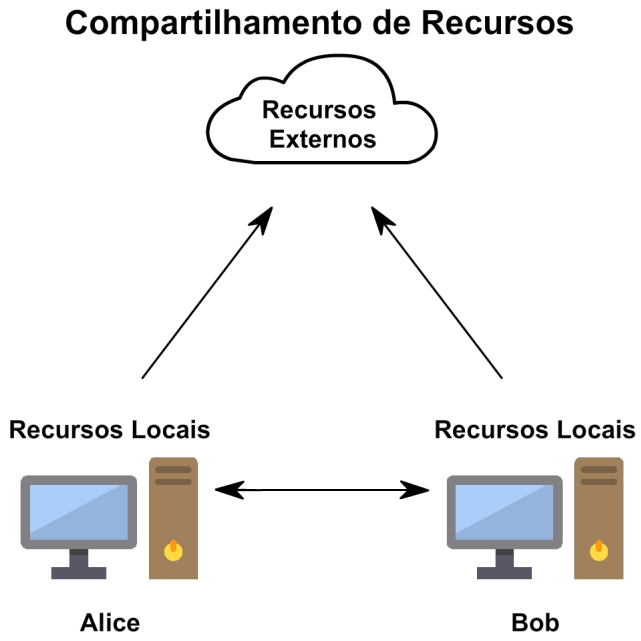
1.1 Justificativa

Atualmente, o compartilhamento de recursos entre usuários ocorre através das mais variadas formas, ocorrendo desde recursos de hardware até recursos de software. Para recursos de hardware há uma série de protocolos que padronizam este compartilhamento, como principais: Wi-Fi, Bluetooth ou infravermelho, empregando algum destes protocolos tem-se, por exemplo, o compartilhamento de impressoras. Já os recursos de software que são compartilhados possuem uma padronização bem estabelecida apenas para alguns grupos específicos de usuários, como os desenvolvedores, cuja existência de uma vasta variedade de ferramentas permite a reutilização dos recursos já desenvolvidos. Neste contexto, ao estabelecer uma padronização para facilitar o compartilhamento de recursos dentre os mais diferentes tipos de usuários e aplicações, torna-se possível a criação de uma grande rede de compartilhamento com alta disponibilidade e diversidade de

recursos, podendo assim, atender uma diversidade maior de usuários.

A Figura 1 ilustra um cenário fictício onde dois usuários, *Alice* e *Bob*, estão dedicando-se a trabalhos distintos, e realizam o compartilhamento dos recursos produzidos de duas maneiras distintas: i) intermediada por um servidor de recursos externos; e ii) diretamente entre os repositórios de recursos locais.

Figura 1 – Compartilhamento de Recursos Entre Usuários

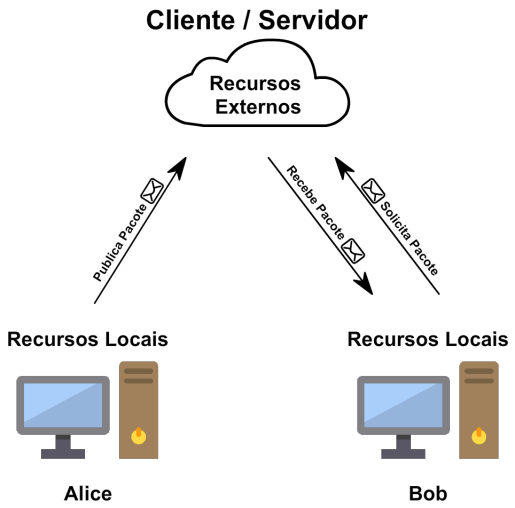


Fonte: o autor

No cenário ilustrado pela Figura 2, o compartilhamento ocorre sobre o intermédio de um computador central que estabelece uma rede cliente/servidor. Os usuários, representados na Figura como *Alice* e *Bob*, criam recursos, cada um em sua própria estação de trabalho, e os disponibilizam em um servidor de recursos externos que tem como função centralizar o compartilhamento dos recursos entre todos os usuários que porventura necessitem de algum recurso que nele esteja disponibilizado. No exemplo apresentado, *Alice* cria e compartilha um recurso em um servidor de recursos externo e, em seguida, *Bob* solicita o recurso disponibilizado por *Alice* junto ao servidor de recursos externos. Uma vez recebido o recurso solicitado, o mesmo passa a ser utilizado por *Bob*, o usuário solicitante.

O próximo cenário, apresentado pela Figura 3, mostra um usuário, *Alice*, que cria um recurso qualquer, em algum aplicativo, e o publica para compartilhamento em um repositório de recursos locais - em sua própria máquina - para que outros usuários possam utilizar sua criação. Para que outro usuário, *Bob*, possa fazer uso do recurso

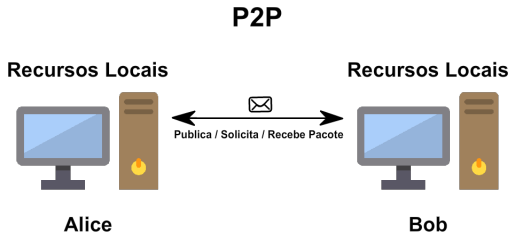
Figura 2 – Compartilhamento de Recursos Mediados Por Servidores de Recursos Externos (Cliente/Servidor)



Fonte: o autor

criado e disponibilizado por *Alice*, este obtém o repositório de *Alice* através de mensagens Multicast e, posteriormente, procura o recurso desejado criando uma rede P2P e, após a sua localização, o solicita para instalação. Após receber o recurso solicitado, o mesmo fica acessível em sua área de trabalho e passa integrar a sua relação de recursos utilizáveis e, por consequência, estará disponível para ser utilizado localmente.

Figura 3 – Compartilhamento Direto de Recursos Entre Usuários (Redes P2P)



Fonte: o autor

Com base nos cenários expostos é possível notar que o processo de compartilha-

mento de recursos entre os usuários deve ser fácil e transparente. Quando mais simples for este processo em um sistema é mais provável que este seja utilizado por novos usuários e, também, que ocorra a expansão da padronização por ele utilizada para outros sistemas, criando assim um grande ecossistema de compartilhamento de recursos.

1.2 Objetivos

Atualmente, os usuários de computadores geram uma quantidade significativa de conteúdo a cada dia. Embora boa parte deste conteúdo não possua valor relevante para outros usuários, ou até mesmo para o usuário que o criou, a parcela dos recursos que possuem alguma relevância deve ser gerida de forma a potencializar as características que os tornam importantes dentro do seu contexto de utilização.

Quando um recurso se torna importante para mais de um usuário é comum que este seja compartilhado. Ao realizar este compartilhamento, alguns problemas surgem devido a distribuição destes recursos, podendo destacar: **perda do controle de alterações** e **ausência do controle de versão**. Além disto, outras questões como o direito autoral e a propriedade intelectual podem ser levantadas quando se compartilha um determinado arquivo de computador. Tais problemas dificultam o controle do compartilhamento de forma consistente entre os usuários e acabam gerando retrabalho e, portanto, a criação de recursos já existentes, duplicação de recursos existentes ou a total falta de backup de recursos criados pelo usuário. Neste cenário, é necessário que exista um protocolo de comunicação que permita aos usuários compartilharem os recursos entre si de forma simples e transparente, de modo que as alterações efetuadas por outros usuários reflitam em novas versões disponíveis para utilização e, por conseguinte, facilite que todos os recursos compartilhados estejam controlados e em concordância com as necessidades destes usuários.

Ao estabelecer este contexto, tem-se como principal objetivo deste trabalho propor uma Interface Universal de Acesso a Recursos Compartilhados denominada UAISharing, um modelo de compartilhamento que seja eficiente para as necessidades dos usuários finais. A princípio, o foco será a distribuição de recursos em repositórios locais onde usuários possam gerir seus próprios recursos e definir o que será ou não disponibilizado para os que tiverem interesse em seus recursos. O modelo que será proposto tem como característica fundamental a abrangência e não se limitar a algum tipo de recurso em específico. Esta característica faz com que os conflitos que porventura ocorrerem durante as alterações sejam de responsabilidade dos usuários, pois muitas vezes estes recursos serão de formatos binários o que impede a resolução automática destes conflitos. Uma vez estabelecida a padronização deste compartilhamento, os usuários poderão disponibilizar os recursos desejados para serem consumidos por outros usuários. Por padrão, este compar-

tilhamento ocorrerá em uma rede local através de troca de mensagens de descobertas utilizando Multicast, mas caso o usuário queira acessar recursos externos a rede local, novos repositórios externos podem ser adicionados para sincronização. Além disso, caso queiram disponibilizar seus recursos fora de uma rede local, basta que o repositório esteja acessível externamente e que os consumidores conheçam as informações de acesso.

2 Conceitos e Trabalhos Relacionados

A utilização de repositórios apresenta uma série de desafios a serem enfrentados para manter a integridade dos recursos que são compartilhados. Entre estes desafios encontram-se problemas que as vezes são específicos para os recursos de software utilizados na construção de aplicações, como a dependência de bibliotecas externas ou de outros recursos. Apesar de este problema de verificação de dependências ser computacionalmente bastante complexo, atualmente os desenvolvedores apenas declaram as dependências que desejam utilizar e através de ferramentas customizadas tornam-nas prontas para o uso em seu ambiente de desenvolvimento. Neste contexto, (4) propõe uma rede de dependências de baixa granularidade que visa criar um ecossistema de versões e como estas afetam um determinado recurso. Ao se determinar quais dependências são afetados por uma determinada versão de recurso, torna-se possível a prevenção de anomalias ocorridas ao atualizar para novas versões. Logo, o modelo visa a realizar predição de problemas que porventura possam ocorrer durante tais atualizações, e serve de exemplo para relatar o quão o desafio de compartilhar recursos já se encontra avançado entre os desenvolvedores. No entanto, tais soluções podem não ser coerentes para outros ecossistemas.

A criação de um ecossistema de recursos faz com que exista uma grande oferta de recursos similares que muitas vezes podem ser iguais, sendo apenas disponibilizados em repositórios distintos. A descoberta da localização destes recursos é uma tarefa que pode ser subjetiva ou baseadas em requisitos ineficientes. Desta forma, (5) define um modelo de descoberta baseados em metadados que visa definir a localização mais apropriada para a obtenção de um recurso a ser utilizado. Neste modelo, os requisitos dos clientes são levados em consideração para predizer qual o menor tempo, levando em conta as propriedades de QoS (Quality Of Service - Qualidade de Serviço). Logo, ao utilizar as premissas estabelecidas para esta medição, estas baseadas em metadados, é possível selecionar diferentes versões do mesmo recurso para o usuário e este definir qual a melhor para a solução.

Nas próximas seções, serão apresentados alguns dos principais conceitos associados compartilhamento de recursos de software.

2.1 Recursos

Um dos principais aspectos a ser definido para o compartilhamento entre usuários é a definição dos **recursos**. Segundo (6), **recurso** é um termo bastante abstrato, mas caracteriza bem o conjunto de coisas que podem ser compartilhadas de maneira útil em um sistema de computadores interligados por rede.

Cada recurso compartilhado possui algumas características que definem a sua utilização, desde o ambiente de sua utilização até a licença de cessão de uso. De uma maneira resumida, há dois aspectos importantes a serem observados: **descritor** e **licença**. O primeiro pode ser entendido como um arquivo que contém as informações necessárias para utilização dos recursos, tais como: dependências, autores, restrições, versão, dentre outras. Já a licença define os limites de uso que um usuário possui sobre algum recurso compartilhado. Há diferentes tipos de licenças, cada uma com suas peculiaridades, algumas mais restritivas não permitindo alteração ou distribuição, e outras mais permissivas que permitem uma vasta possibilidade de utilização e distribuição. Portanto, é de suma importância escolher a licença adequada para os recursos compartilhados, seja ele para consumo ou distribuição.

A distribuição e gerenciamento destes recursos por meio de uma rede de computadores pode ser vista como um sistema distribuído. Há inúmeras definições utilizadas ao longo dos anos para a definição de sistemas distribuídos, mas de maneira geral, um sistema distribuído é um conjunto de computadores independentes que se apresenta a seus usuários com um sistema único e coerente (7).

Ao se optar por adotar um sistema distribuído como parte da solução de um problema computacional, uma série de desafios devem ser enfrentados a fim de garantir as melhores experiências de utilização aos usuários deste sistema. Segundo (6), os principais desafios a serem enfrentados são:

1. **Heterogeneidade:** os usuários podem acessar os serviços e executar aplicativos por um conjunto heterogêneo de computadores e redes;

2. **Sistemas abertos:** determina, principalmente, o grau que novos serviços de compartilhamento de recursos podem ser adicionados e disponibilizados para uso por uma variedade de programas clientes;

3. **Segurança:** as informações mantidas possuem alto valor intrínseco para seus usuários. A segurança de recursos de informação tem três componentes: i) **confidencialidade:** proteção contra exposição para pessoas não autorizadas; ii) **integridade:** proteção contra alteração ou dano; e iii) **disponibilidade:** proteção contra interferência com os meios de acesso aos recursos;

4. **Escalabilidade:** indica que o sistema permanece eficiente quando há um aumento significativo ou diminuição no número de recursos e usuários;

5. **Tratamento de Falhas:** falhas ocorrem e os sistemas distribuídos devem fornecer um alto grau de disponibilidade permitindo que o mesmo se mantenha operante mesmo no caso de uma falha ocorrer;

6. **Concorrência:** os recursos disponibilizados podem ser acessados de forma simultânea por vários usuários;

7. **Transparência:** definida como a ocultação da separação dos componentes utilizados, de modo que o sistema seja percebido como um todo, em vez de uma coleção de componentes independentes;

8. **Qualidade de serviço:** o serviço ofertado deve oferecer qualidade satisfatória, oferecendo confiabilidade, segurança e desempenho.

Uma vez enfrentados os desafios citados, é possível apresentar uma estrutura robusta e confiável para compartilhamento de recursos entre os usuários localizados nas mais diversas localidades.

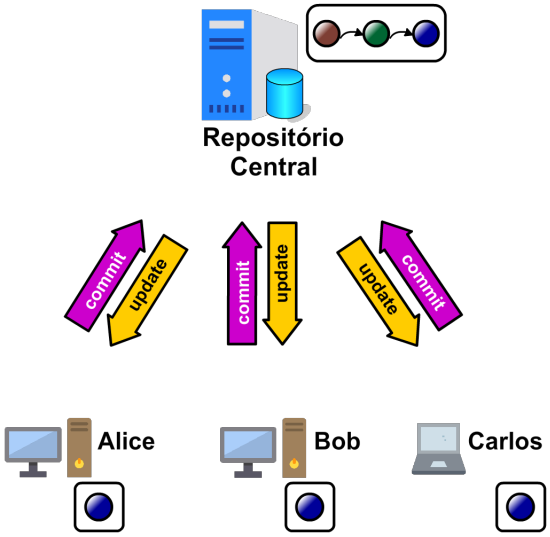
2.2 Repositórios

De maneira geral, repositórios podem ser definidos como um local onde se guarda, arquiva ou coleciona algo. Logo, pode-se associar esta definição ao cenário de compartilhamento, onde um repositório conterá todos os recursos que foram disponibilizados para utilização dos usuários. Para alguns grupos de usuários específicos, os recursos existentes nestes repositórios ganham uma nomenclatura especial, por exemplo: **pacotes** para usuários dos sistemas operacionais Linux ou **bibliotecas** para usuários desenvolvedores. Estas são apenas algumas, das muitas, definições que o conteúdo presente em um repositório pode assumir. Ao longo deste texto, ao se referir ao conteúdo de um repositório, o termo utilizado será **recurso**, pois o intuito é abranger de maneira ampla todos os tipos de usuários que podem se envolver no compartilhamento.

Os repositórios podem ser organizados em dois tipos: centralizado ou distribuído. O repositório centralizado pode ser associado a topologia em estrela, na qual um único repositório central coordena todas as cópias de recursos que são disponibilizadas aos usuários, ou seja, qualquer comunicação, obrigatoriamente, é intermediada pelo repositório central. A Figura 4 apresenta os usuários: Alice, Bob e Carlos, acessando o mesmo recurso através de um repositório central, sendo todas as operações de envio e recebimento são realizadas online.

Diferentemente da estrutura centralizada, os repositórios distribuídos não possuem uma topologia definida e são independentes entre si. Logo, cada usuário possui um repositório local onde suas próprias mudanças podem ser controladas sem presença de qualquer intermediador. Desta maneira, os usuários podem selecionar com quais repositórios desejam sincronizar seus recursos, não havendo, a princípio, uma hierarquia ou ordem de importância de um repositório em relação ao outro. A Figura 5 apresenta o usuário Alice realizando as operações: i) envio/recebimento de recursos entre a área de trabalho e o

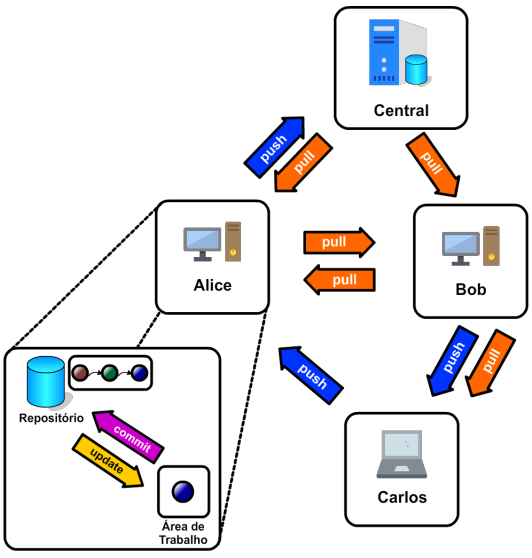
Figura 4 – Comunicação Envolvendo Repositório Centralizado



Fonte: o autor

repositório local; ii) envio/recebimento de recursos entre o repositório local e um repositório estabelecido como central; iii) recebe atualizações do repositório local de Bob. Já o usuário Bob, além da comunicação já mencionada com Alice, oferece atualizações para o repositório central e busca atualizações nos repositórios de Alice e Carlos. Além disso, Bob envia informações para o repositório local de Carlos. Carlos por sua vez, envia informações para o repositório de Alice. Como pode ser observado, não há uma hierarquia de comunicação entre os repositórios, o que propicia uma maior flexibilidade de utilização.

Figura 5 – Comunicação Envolvendo Repositórios Distribuídos



Fonte: o autor

A publicação e compartilhamento de novos recursos está diretamente relacionada ao tipo do repositório escolhido. Embora ambos possuam as operações de envio (commit) e atualização (update), há diferenças sutis entre cada um devido ao **espaço de trabalho** utilizado. Os repositórios centralizados coordenam todos os espaços de trabalho de seus usuários, posto que todas as operações são realizadas diretamente com o repositório central, logo qualquer envio ou atualização ocorre em um ponto comum a todos os usuários. Já para os repositórios distribuídos, as operações de envio e atualização ocorrem no espaço de trabalho local do usuário, podendo estas serem submetidas para qualquer outro repositório. Por não possuir uma estrutura centralizada, repositórios deste tipo podem ser sincronizados a outros através de comandos de envio remoto (push) e atualização remota (pull).

Uma outra forma de se caracterizar os repositórios é quanto a visibilidade dos recursos que são disponibilizados aos usuários, podendo ser privados ou públicos. De maneira geral, os recursos compartilhados em repositórios públicos são acessíveis a todos os usuários que os desejem utilizar, ao passo que os repositórios privados são acessíveis apenas para uma parcela específica de usuários, por exemplo, recursos empresariais exclusivos. No entanto, é preciso observar que mesmo para os repositórios públicos é comum a existência de políticas que norteiam o compartilhamento de recursos pois, sem o estabelecimento destas, não haveria uma padronização para que os recursos fossem evoluídos para gerar valor para os usuários. Nas próximas seções serão apresentados alguns dos principais repositórios que são utilizados por usuários desenvolvedores. Com objetivo organizacional, as seções serão divididas em: i) repositórios de aplicações e ii) repositórios de códigos. Embora ambos disponibilizem recursos, o primeiro é mais abrangente e no cenário apresentando refere-se à recursos prontos para serem utilizados, como uma aplicação ou biblioteca, sendo o segundo responsável pelo versionamento dos códigos-fonte que darão origem aos recursos disponibilizados no primeiro repositório.

2.2.1 Repositórios de Aplicações

Os desenvolvedores, através dos códigos-fonte (ver subseção 2.2.2) criados, realizam a geração de alguma aplicação que pode ser utilizada em sua totalidade, ou em parte, por algum outro desenvolvedor sem que este necessite de recriar a aplicação disponibilizada. Normalmente, os conteúdos destes repositórios são disponibilizados em servidores cuja comunicação é realizada através de protocolos conhecidos: HTTP ou FTP, mas vale ressaltar que este é apenas um rol exemplificativo, pois há muitos outros protocolos que podem ser utilizados. Para interagir com estes servidores é necessário satisfazer critérios mínimos de segurança, por exemplo, para enviar novos recursos, seja ele um novo recurso ou uma atualização, é necessário que o usuário possua permissão para tal. Já para obtenção do conteúdo compartilhado, o processo pode ser mais permissivo e não requerer

qualquer tipo de restrição.

Para que uma aplicação e/ou biblioteca possa ser disponibilizada para outro usuário, esta deve seguir um padrão que consiste em um conjunto de características que permita a distribuição de recursos, a este conjunto dá-se o nome de **pacote**. Um pacote deve conter toda a informação necessária para sua utilização. Normalmente, informações contendo as dependências, licença, códigos-fonte, metadados e configurações são disponibilizadas juntamente com a aplicação e/ou biblioteca para o seu adequado funcionamento e distribuição. Uma vez criado o pacote, a distribuição pode ser realizada em um repositório para utilização, sendo que as informações anteriormente apresentadas são de extrema importância para que os pacotes possam ser utilizados corretamente. Comumente, um pacote possuirá relacionamento com outros pacotes, nomeia-se esta relação como **dependências**. Ao se instalar um novo pacote, é desejável que todas as dependências necessárias para o seu funcionamento estejam, ou sejam, instaladas em conjunto por um gerenciador de pacotes. É importante observar, que mesmo com todas as dependências satisfeitas e o pacote pronto para uso, a licença deve ser analisada para não violar qualquer aspecto de legalidade, por exemplo, alguns pacotes podem ser utilizados somente para fins públicos, não sendo permitida a sua utilização em um ambiente privado sem que os códigos-fonte do pacote que o incluiu também sejam disponibilizados, a violação desta disponibilização afeta diretamente a utilização devido a restrições impostas pela licença.

Todo pacote disponibilizado em um repositório apresenta dois diferentes níveis de visão para seus usuários. Em uma primeira visão têm-se os usuários que são responsáveis pela manutenção dos pacotes, ou seja, estão envolvidos desde a criação até a publicação, podendo, muitas vezes, também utilizá-los em outros pacotes ou em novas aplicações. A segunda visão de usuários presente nos repositórios consiste, exclusivamente, na utilização dos pacotes publicados pelos usuários desenvolvedores. Desta forma, os papéis dos usuários envolvidos em cada tipo de visão é bem claro, desenvolvedores são responsáveis pela criação, atualização, publicação e remoção, isto é, de todo o processo de manutenção do pacote. Já os usuários comuns apenas utilizam os pacotes publicados para seus interesses, mesmo que estes sejam a criação de outros pacotes, quando isso ocorrer, ele visualizará de forma distintas os pacotes criados e utilizados.

Nas próximas seções serão apresentados alguns exemplos de repositórios que são utilizados por desenvolvedores para disponibilizar aplicações que estão prontas para serem utilizadas.

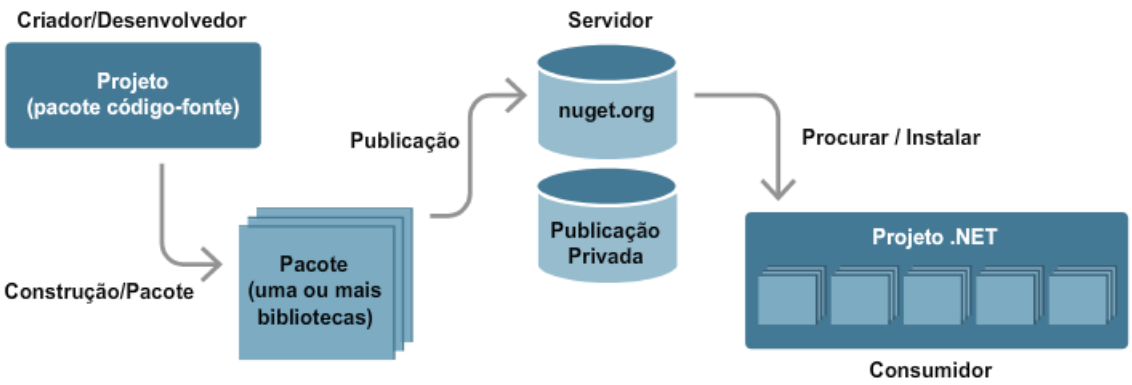
2.2.1.1 NuGet

NuGet é um gerenciador que permite a adição de novos componentes a projetos .NET, suas ferramentas proveem a capacidade de criar, atualizar, consumir e compartilhar componentes entre projetos. Atualmente, os componentes 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 (8).

Um componente 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 componente deve ser publicado em um repositório para ser acessado. Além do repositório central[1], 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 6 apresenta a distribuição dos pacotes mencionada, desde a criação até o consumo dos pacotes em projetos finais.

Figura 6 – Distribuição de Pacotes NuGet



Fonte: NuGet 9

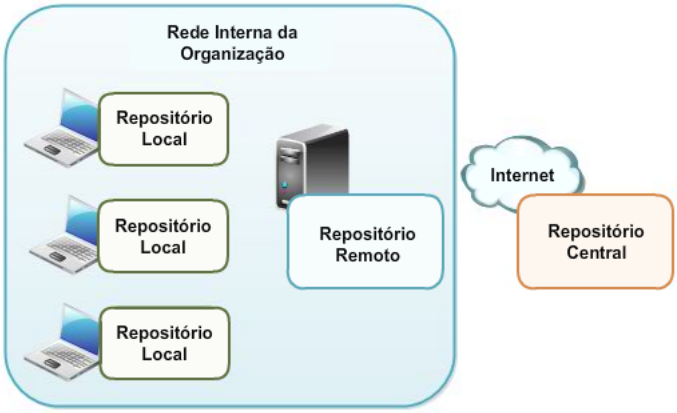
Em resumo, um pacote é construído com todas as dependências necessárias (creator) e, posteriormente, empacotado (build/pack) e publicado (publish) no servidor (host) para utilização. Após estas etapas, o pacote encontra-se disponível para utilização para os desenvolvedores.

2.2.1.2 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, faz uso de um arquivo de configuração 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 no arquivo de configuração (10).

[1] Repositório central disponível em: <https://www.nuget.org>

Figura 7 – Distribuição de Pacotes Maven



Fonte: o autor

Ao analisar a Figura 7 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 usuário e possui uma cache dos componentes utilizados em seus projetos, o mesmo tem como principal objetivo melhorar a performance da utilização dos componentes vinculados. Os repositórios remoto e central possuem os componentes que serão recebidos e armazenados, quando necessários, pelo repositório local. A principal diferença entre eles é 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 os componentes criados por uma organização e todos os componentes comuns e públicos são obtidos do repositório central. Ao se configurar um novo projeto Maven o arquivo de configuração possuirá os endereços de acesso de ambos os repositórios para obtenção dos componentes necessários.

2.2.1.3 PyPI

A linguagem de programação Python possui um repositório de distribuição de componentes denominado PyPI (Python Package Index) (11). Um componente para ser distribuído neste repositório possui alguns arquivos que definem toda a sua estrutura. Através destes arquivos, torna-se possível configurar os aspectos dos projetos (12). A estruturação e organização destes 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 componentes públicos disponibilizados. Assim como os gerenciadores apresentados nas seções anteriores, sub-

seção 2.2.1.1 e subseção 2.2.1.2, existe a possibilidade de criação de repositórios privados e disponibilizar as bibliotecas apenas dentro do escopo desejado.

2.2.1.4 Linux: RPM e DEB

Os usuários Linux possuem grande familiaridade com o conceito de repositórios e pacotes. Embora existam outras formas de empacotamento, as mais populares são RPM (Red Hat Package Manager) e DEB, amplamente utilizadas em sistemas da linha Red Hat e Debian, respectivamente. Ambos os formatos de pacote possuem suas características e são manipulados por utilitários específicos, mas mesmo com as diferenças apresentadas em cada formato, há muitos pontos em comum. Logo, em regra, apresentam uma estrutura bem definida que rege as regras de empacotamento e distribuição cujas principais são: descritores do pacote, dependências, versão, licença e autoria.

2.2.2 Repositórios de Códigos

Uma das premissas para se controlar a evolução das aplicações geradas é que estas possam ser rastreadas para definir quais eram as funcionalidades disponibilizadas e como estas funcionavam em um determinado momento.

Tais repositórios são conhecidos como repositórios de SCM (Source Code Management - Gerenciamento de Código-Fonte). Segundo (13), um repositório de SCM é um conjunto de mecanismos e estrutura de dados que permitem a uma equipe de software gerenciar alterações de maneira eficaz. Dentre suas principais características, encontram-se: versionamento, acompanhamento de dependências e gestão de alterações, controle de requisitos, gestão de configuração e pistas de auditoria. Há vários repositórios que satisfazem as características apresentadas, tais como: Git, SVN (Subversion), Mercurial, ClearCase, dentre outros.

O versionamento dos recursos publicados em um repositório de código implica no estado que estes recursos se encontram em um determinado momento, podendo existir o mesmo recurso, mas em versões diferentes, em repositórios e/ou ramificações distintas. A existência de várias versões traz consigo uma dificuldade a ser enfrentada: o espaço de armazenamento. Com o intuito de minimizar o espaço de armazenamento utilizado pelas versões utiliza-se o conceito de delta, que consiste em armazenar uma versão completa e as diferenças entre estas versões (14). Existem duas variações deste conceito: delta positivo e delta negativo. A verificação das diferenças utilizando delta negativo armazena a versão mais recente e as diferenças (deltas) ocorridas até esta versão. Já o delta positivo, diferentemente da outra abordagem, armazena a versão mais antiga e para montar as versões mais recente, realiza o processamento das diferenças (deltas) armazenadas (15). Um aspecto importante a se observar é que mesmo com o conceito do armazenamento das diferenças para otimizar o espaço de armazenamento, o versionamento de arquivos binários

continua a consumir um grande espaço se comparados aos arquivos textuais, haja vista que qualquer alteração em sua estrutura faz com que uma cópia inteira seja armazenada. Tal comportamento, faz com que a área de trabalho (workspace) do usuário apresente um aumento significativo em seu espaço de armazenamento quando arquivos binários são versionados, pois mesmo com a presença dos repositórios, sejam eles centralizados ou distribuídos, todas as alterações realizadas são efetuadas primeiramente em suas respectivas áreas de trabalho, antes que estas sejam sincronizadas em algum repositório.

O controle das alterações, bem como sua identificação, ajuda não somente no processo de manutenção da ordem, mas, principalmente, para garantir que qualquer alteração implementada esteja condizente com a solução proposta. Logo, está diretamente associada a questões de auditoria, que possuem o intuito de verificar se as modificações realizadas estão condizentes com as especificações exigidas pela solução.

Para que todo o processo de comunicação para a utilização dos recursos funcione é necessário estabelecer um protocolo de comunicação entre os envolvidos. Segundo (16), um protocolo define o formato e a ordem das mensagens trocadas entre dois ou mais envolvidos, bem como as ações realizadas na transmissão e/ou no recebimento de uma mensagem ou outro evento. Os principais repositórios SCM existentes atualmente fornecem diferentes protocolos para o estabelecimento desta comunicação. Alguns destes são criados especificamente para otimizar as necessidades inerentes aos repositórios, tais como: protocolo Git[2] e protocolo SVN[3], ao passo que outros são de uso comum e utilizados por diferentes aplicações, como o HTTP, FTP e o SSH.

2.2.2.1 Git

O Git é um sistema de controle de versão distribuído gratuito e de código aberto projetado para lidar com tudo, de projetos pequenos a muito grandes, com velocidade e eficiência (17). Tal repositório ganhou destaque por ser o responsável por hospedar os códigos-fonte do kernel do Linux e desde então vem ganhando vários adeptos, inclusive em empresas de grande porte, tais como: Google, Facebook, Microsoft, NetFlix, Gnome, e várias outras. Um aspecto interessante sobre os repositórios Git é que existem alguns locais que armazenam um grande número de recursos criados pelos usuários, tais como: github, gitlab e bitbucket. No entanto, vale observar que mesmo que os usuários centralizem o código-fonte nestes repositórios, esta centralização ocorre pelo desejo dos usuários e não por qualquer imposição do sistema de controle de versão, que em sua essência é distribuído.

[2] Protocolos utilizados pelo Git, disponível em: <https://git-scm.com/book/en/v2/Git-on-the-Server-The-Protocols>.

[3] Procotolos utilizados pelo SVN, disponível em: <http://help.collab.net/index.jsp?topic=/org.tigris.subclipse.doc/topics/protocol.html>.

2.2.2.2 SVN (Subversion)

Segundo (18), Subversion é um sistema de controle de versão de código aberto, sendo que tanto o projeto quanto o software Subversion obtiveram um sucesso incrível na década de 2000, sendo adotando tanto pela comunidade de código livre quanto pelo mundo corporativo.

Ao contrário dos repositórios Git (subseção 2.2.2.1) que visam a descentralização dos recursos, o Subversion centraliza os recursos gerenciados em um nó central que deve estar ativo para que os recursos possam ser acessados. Embora ambos sirvam para o mesmo propósito, esta diferença deve ser considerada ao escolher qual a se adotar, pois cada abordagem possui prós e contras que devem ser considerados para o contexto de utilização.

2.2.2.3 Git versus SVN

Para a definição de qual tipo de repositório a ser adotado é primordial que os aspectos relevantes ao contexto de utilização sejam avaliados. Ao se considerar Git e SVN como repositórios de códigos é necessário entender quais são as principais características de cada um e como elas se adéquam as necessidades avaliadas. Dentre as principais características do SVN têm-se: repositório centralizado, necessidade de se estar online e há um único repositório para todos os usuários. Tais características podem facilitar o controle sobre os recursos nele presentes, pois por possuir os recursos centralizados faz com que todos os usuários trabalhem, em geral, em uma única versão do código. Ao contrário do SVN, repositórios Git possuem como principal característica a descentralização, podendo existir vários repositórios, um para cada usuário, do mesmo código. Além disso, permite que usuários trabalhem de maneira offline em seus repositórios locais, podendo mais tarde, sincronizar com algum outro repositório. A Tabela 1 apresenta um resumo sobre os pontos discutidos nesta subseção.

Tabela 1 – Comparação entre Repositórios de Códigos.

	GIT	**SVN (Subversion)**
Tipo	Distribuído	Centralizado
Offline	Sim	Não
Repositório	Um por usuário	Um para TODOS os usuários

Fonte: o autor

3 Definição da Proposta

Como apresentado na seção 2.2 há tipos específicos de repositórios para cada propósito e cada tipo possui características que os tornam mais indicados para determinados cenários. No entanto, mesmo existindo aspectos que possam indicar qual o tipo de repositório mais adequado para determinado cenário, não há nada que impeça a utilização de um tipo que, a princípio, não seja o mais indicado, mas esta decisão pode tornar sua utilização mais complicada. Por exemplo, uma aplicação após terminada é comumente disponibilizada através de um repositório de aplicação, mas também pode ser disponibilizada em um repositório de código para usuários que queiram compilá-la a partir de seu código-fonte. Da mesma forma, há pacotes de código-fonte disponibilizados em repositórios de aplicação, como os pacotes src em repositórios Linux. Apesar destas maneiras de realizar a distribuição não ser convencional e nem acessível a todos os perfis de usuários de uma determinada aplicação, isto ilustra a complexidade que pode ocorrer nos cenários aqui apresentados.

No entanto, apesar de haver sobreposições sobre a funcionalidade e aplicação de ambos os repositórios, é importante notar que estes são bastante conhecidos por desenvolvedores e, por consequência, são ferramentas amplamente utilizadas durante suas atividades. Porém, os demais usuários muitas vezes não possuem familiaridade, ou o conhecimento necessário, para utilização destes repositórios. Neste contexto, uma nova estrutura que permita que estes usuários controlem e distribuam seus recursos de uma maneira tão fácil quanto os desenvolvedores se faz necessária. Como dito, a princípio, cada tipo de repositório atende bem um determinado conjunto de necessidades e quando utilizados para outras, pode vir a tornar o processo trabalhoso e ineficiente, o que dificulta a sua adoção para usuários não desenvolvedores. Devido a isto, um único repositório que contemple todas as necessidades destes usuários deve unir as melhores características que cada tipo de repositório utilizado pelos desenvolvedores fornece.

A proposta aqui apresentada, denominada **UAISharing**, visa trazer uma estrutura de repositórios que, além de ser de fácil utilização, permita tanto o versionamento quanto a distribuição de recursos por usuários não desenvolvedores. Desta forma, qualquer usuário poderá versionar e distribuir os recursos criados com outros usuários de forma simples sem a preocupação de utilizar repositórios diferentes, pois a estrutura de compartilhamento possibilitará que todas atividades sejam unificadas.

3.1 Aplicabilidade da UAISharing

Para auxiliar o entendimento do escopo presente na UAISharing, serão apresentados alguns exemplos onde a ferramenta poderia ser aplicada para auxiliar este perfil de usuários.

3.1.1 Versionamento local

Sua principal característica é realizar o versionamento dos recursos de forma a permitir que o usuário responsável pelo repositório possa escolher qual versão do recurso utilizar em detrimento de outra. Este repositório pode estar ou não acessível a outros usuários, mas sozinho já permite que o versionamento de recursos seja realizado. Pode-se ter como exemplo um usuário chamado Alice que versionou localmente um recurso denominado *RecursoAlice.pdf* com duas versões, 1.0 e 1.1, a qualquer momento Alice pode escolher qual versão é mais adequada para sua atual necessidade e obtê-la do repositório para uso. Este cenário é bastante similar a um repositório git local.

3.1.2 Troca de recursos na mesma rede

Uma situação comumente encontrada em diversos ambientes é o compartilhamento de recursos entre usuários que estão localizados na mesma rede, tais como: escritórios, consultórios, laboratórios de pesquisa, dentre outros. Em um cenário exemplificativo, Alice e Bob trabalham em uma mesma empresa de projetos e necessitam compartilhar recursos de seus clientes entre si. Tanto Alice quanto Bob, possuem um repositório local e uma área de trabalho próprios que recebem os recursos que cada um disponibiliza, por estarem em uma mesma rede todos os recursos compartilhados podem ser acessados por ambos. Desta forma, Alice e Bob, podem obter os recursos compartilhados um do outro.

3.1.3 Backup de recursos na mesma rede

Quando há mais de um repositório presente em uma mesma rede pode-se configurar para que estes sejam replicados e desta forma os recursos compartilhados podem ser acessíveis através de vários locais aumentando a disponibilidade de acesso aos recursos. Em um cenário simples, Alice mantém backup dos arquivos de Bob localmente em sua máquina e vice-versa. Toda vez que Bob fizer uma alteração em seu repositório, Alice receberá a replicação desta atualização para fins de backup, o mesmo ocorre quando Alice fizer alguma alteração, mantendo assim, ambos os repositórios replicados. Um repositório pode ser elegido como nó central e ficar responsável por armazenar as réplicas dos demais repositórios da rede, tal processo é realizado em background sem que qualquer ação do usuário seja necessária.

3.1.4 Publicação de recursos

Neste cenário, têm-se os usuários Alice e Bob, cada um possuidor de um repositório local no qual controlam seus respectivos recursos. Estes repositórios não são acessíveis a outros usuários, pois não estão liberados para acesso externo. Tanto Alice quanto Bob, por critérios de segurança, preferem manter seus arquivos em seus próprios repositórios e dar publicidade apenas à recursos que são de interesse coletivo. Assim sendo, estes publicam os recursos que querem tornam públicos de seus repositórios em um novo repositório externo, este acessível para outros usuários. Desta forma, qualquer usuário que tenha acesso a este repositório externo pode acessar o conteúdo compartilhado por Alice e/ou Bob, e assim acompanhar a evolução do trabalho de ambos.

3.1.5 Modificação de recursos pelos clientes

Como complemento ao cenário apresentado na subseção 3.1.4, tem-se o usuário Carlos, que obtém um dos recursos disponibilizados por Alice ou Bob e realiza algumas modificações. Por não possuir permissão para enviá-las ao repositório externo, local ao qual obteve o recurso original, este as publica em um novo repositório com acesso externo ao qual seja possuidor e, caso Alice ou Bob queiram atualizar o conteúdo do recurso original compartilhado, eles obtêm a nova versão do repositório de Carlos e a aplica em seus repositórios.

3.1.6 Escritórios distribuídos

Um cenário mais complexo consiste em uma organização que possui diversos escritórios e deseja compartilhar recursos entre seus funcionários, bem como, com seus clientes. Neste caso, a organização disponibilizará tanto repositórios públicos quanto repositórios privados, para que a confidencialidade dos recursos seja mantida. A estrutura dos repositórios consiste em: i) cada funcionário possui seu próprio repositório; ii) cada escritório possui um repositório de backup para centralizar os recursos dos funcionários; iii) um repositório central da organização que recebe a replicação de todos os repositórios centrais de cada escritório; iv) por fim, um repositório público para que seus clientes possam obter os recursos compartilhados que sejam de interesse comum. Com exceção do repositório público, todos os demais são de acesso exclusivo da organização e não são acessíveis a usuários externos. Observa-se que mesmo sendo um repositório público, os usuários externos não podem modificar qualquer recurso nele compartilhado.

3.2 Elementos da UAISharing

O primeiro ponto a ser definido ao se realizar algum tipo de compartilhamento é definir o que será compartilhado, o que serão os repositórios e demais componentes

presentes na arquitetura.

3.2.1 Recursos

Para fins de padronização, todo e qualquer item a ser compartilhado será denominado como **recurso**. A utilização de recursos compartilhados está presente no cotidiano da maioria dos usuários, ocorrendo de diferentes formas e está diretamente relacionada a experiência e facilidade destes com a tecnologia utilizada durante o compartilhamento. Portanto, qualquer recurso que se encontra armazenado pode vir a ser compartilhado com outros usuários para a criação de novos recursos.

Logo, a criação de recursos traz consigo pontos que devem ser explorados para potencializar sua utilização, principalmente, na otimização desta criação através da reutilização dos recursos já existentes. Por conseguinte, para que o reuso ocorra de forma controlada e, consistente, é necessário a definição de um processo padronizado de compartilhamento de recursos. Como os usuários alvos não são desenvolvedores e, na maioria das vezes, não possuem conhecimentos avançados em computação, a solução visa unir os tipos de repositórios apresentados nas subseções 2.2.1 e 2.2.2. Tal união tem por objetivo simplificar o processo de compartilhamento e torná-lo mais transparente para os usuários, tanto para disponibilização quanto para consumo de recursos.

Os recursos alvos da UAISharing são os arquivos gerados pelos usuários finais no exercício de suas atividades como planilhas, documentos de texto, projetos, apresentações, entre outros. Entendemos que estes arquivos podem ser vistos como pequenos recursos de software e, portanto, serem considerados pequenas aplicações. Assim, ao compartilhar uma planilha eletrônica, entendemos que o usuário está compartilhando tanto uma aplicação, que poderia ser distribuída por um repositório de aplicação, quanto o código-fonte desta aplicação, que poderia ser distribuída por um repositório de código.

Foi este entendimento que nos levou a propor a união destes repositórios em outro tipo de sistema para a distribuição e gerenciamento de recursos. O foco principal desta união é controlar as alterações e novas versões destes recursos, não visando resolver as dependências existentes entre estes recursos e tampouco os conflitos apresentados durante a utilização destes.

3.2.2 Pacotes

Um pacote pode ser definido como um único arquivo que contém toda estrutura necessária para que um recurso seja compartilhado. De maneira geral, o mesmo pode ser entendido como a representação do recurso, pois nele está presente o recurso, licença, suas dependências, arquivo de ajuda e, toda a estrutura necessária para que os recursos possam ser compartilhados pelos repositórios. Logo, um pacote é o próprio recurso em formato

para distribuição.

Uma vez definido o que é um pacote, para que o compartilhamento de recursos possa ser realizado é necessário definir a forma que este será empacotado. Este empacotamento deve definir todas as etapas envolvidas, abrangendo desde a criação até a utilização do recurso por outros usuários. Para o empacotamento realizado pela UAISharing, as seguintes etapas devem ser realizadas: i) definir o nome do pacote; ii) criar o descritor do pacote; iii) indicar um arquivo de ajuda; e iv) estabelecer uma licença para o pacote criado. Caso a licença seja omitida, assume-se a licença existente no repositório ao qual o recurso será compartilhado.

Assim sendo, um pacote será um único arquivo compactado contendo todas as informações necessários para sua utilização, como: recurso, descritor, arquivo de ajuda e licença. Logo, ao ser compartilhar qualquer recurso em um repositório este deve ser adequadamente empacotado para distribuição. O arquivo compactado com a definição do recurso a ser publicado deverá seguir o padrão de versionamento definido em (19) onde três incrementos (**MAJOR.MINOR.PATCH**) de versão podem ser utilizados para definir o recurso, representando:

1. **MAJOR**: quando fizer mudanças incompatíveis com outras versões do recurso;
2. **MINOR**: quando adicionar funcionalidades mantendo a compatibilidade com outras versões do recurso; e
3. **PATCH**: quando corrigir falhas mantendo compatibilidade com outras versões do recurso.

Como apresentando, o arquivo compactado representando o pacote deverá conter todas as informações necessárias para que o recurso compartilhado funcione adequadamente. Para tal, um arquivo descritor contendo as informações pertinentes a especificação, funcionamento e dependência do recurso distribuído se faz necessária. O mesmo será um arquivo que remete ao recurso compartilhado, porém com a extensão *.json*. O formato json consiste em um arquivo formatado utilizado para troca de informações, tal especificação foi escolhida pela simplicidade de utilização e legibilidade. O código 3.1 apresenta todos os campos presentes no descritor do recurso. A linha 1 contém o nome do recurso compartilhado, as linhas 2, 3 e 4 correspondem as informações de versão, edição e arquitetura associados ao recurso. Entre as linhas 5 e 20, são apresentadas as informações referentes a: tamanho (em bytes), data de lançamento, descrição, licença, grupo, palavras-chave, endereço de publicação, dependências e autores. Observe que nas linhas 11, 16 e 20 é possível informar várias palavras-chave, dependências e autores, respectivamente.

Código 3.1 – Exemplo Arquivo Descritor (Metadados)

```
{
  "name": "libmosaic-sound-1.0.0-release.x64.mcpkg",
  "version": "1.0.0",
  "edition": "release",
  "architecture": "x64",
  "size": 5000,
  "released_date": "2017-11-03T19:53:00Z",
  "description": "The library libmosaic-sound written in the
     'C' programming language based on the PortAudio API.
     This library provides resources for working with Sound
     Design.",
  "license": "GPLv3",
  "group": "sound-plugin",
  "keywords": [
    "Audio",
    "Sound"
  ],
  "url": "https://github.com/Mosaicode/libmosaic-sound",
  "authors": [
    "Luan Luiz Goncalves",
    "Flavio Luiz Schiavoni"
  ],
  "dependencies": [
    "portaudio19-dev",
    "libsndfile1-dev"
  ],
}
```

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 recurso. O arquivo de ajuda deve facilitar a interação do usuário com o recurso e o arquivo de licença deve deixar claro as regras de distribuição. Caso não informado uma licença pelo usuário, o recurso assumirá a licença padrão do repositório, por exemplo GNU GPLv3 (20).

3.2.3 Dependências

Dependências, em compartilhamento de recursos, pode ser definida como sendo tudo aquilo que é necessário para o adequado funcionamento do recurso a ser utilizado.

Por exemplo, uma planilha eletrônica pode estabelecer relação com muitas outras planilhas que não estão sendo disponibilizados no repositório de um usuário. Para que estas dependências sejam resolvidas é necessário que estas também estejam compartilhadas e instaladas pelo usuário solicitante e caso não se encontrem acessíveis a utilização do recurso ficará comprometida. O mesmo pode acontecer com um documento de texto que depende de um arquivo de fonte ou de um arquivo de macro.

No entanto, dada a complexidade de dependências que estes recursos podem gerar, não é o objetivo deste trabalho propor alguma solução para este problema. No contexto da UAISharing, para que as dependências sejam resolvidas é necessário que o usuário solicitante de algum recurso tenha acesso aos repositórios cujas dependências já foram disponibilizadas. Uma segunda possibilidade é que todas estas estejam presentes no mesmo repositório do recurso solicitado, mas como dito na subseção 3.3.1 os detalhes de implementação da interface de comunicação ficam a critério de quem implementá-la.

Portanto, a depender da forma que a interface for implementada, é possível que exista dependências inacessíveis para alguns usuários, possibilitando que o recurso solicitado não funcione de maneira adequada. Este cenário pode ocorrer quando as dependências de um recurso estejam acessíveis apenas em uma rede local, ao passo que este recurso seja acessível tanto pela rede local quanto para rede externa, desta forma somente os usuários presentes na rede local, onde as dependências se encontram, farão o uso adequado da dependência disponibilizada.

Um aspecto importante a ser definido sobre as dependências é a possibilidade da existência de um pacote apenas com dependências a serem instaladas, ou seja, é possível definir um pacote que realize a instalação de vários recursos de uma única vez.

3.2.4 Repositórios

No contexto da UAISharing um repositório é um diretório local, sendo que este pode ser, ou não, compartilhado. Repositórios não compartilhados são denominados repositórios locais e são acessíveis apenas localmente e diretamente via o próprio sistema de arquivos. Já os repositórios compartilhados oferecem recursos a outros usuários como um servidor de arquivos e possuem a sua acessibilidade limitada a abrangência da rede, podendo assumir duas subdivisões:

- **repositório em rede LAN:** são acessíveis apenas dentro da rede local e usuários externos a esta rede não interagem com qualquer recurso compartilhado neste repositório;
- **repositório em rede WAN:** são acessíveis a qualquer usuário que deseja interagir com os recursos compartilhados.

Vale ressaltar que a qualquer momento um repositório pode modificar a sua classificação dentro das definições anteriormente apresentadas, pois este pode ser adicionado a alguma rede de compartilhamento (LAN ou WAN), alterado entre redes de compartilhamento (LAN para WAN, e vice-versa) ou até mesmo ser removido de qualquer rede, tornando-o um repositório local.

Um repositório ao ser criado, além da localização dos recursos que serão armazenados, deve conter um nome, endereço de localização e uma licença. Tais informações são armazenadas em um arquivo de metadados, um exemplo deste arquivo pode ser observado em Código 4.1.

Logo, para que um usuário possa criar um repositório, ele deve preencher as configurações dos metadados apresentados e este passa a ser monitoramento pela aplicação. Um usuário pode possuir vários repositórios com diferentes níveis de acesso: apenas acesso local, acesso LAN ou acesso WAN, e desta forma distribuir seus recursos com diferentes visões de acesso. Desta forma um mesmo usuário poderá compartilhar vários recursos que estarão presentes em diferentes diretórios, ou seja, a estrutura do repositório local do usuário é dinâmica e novos recursos ou diretórios podem estar, ou não, acessíveis a cada instante. A soma de todos os recursos compartilhados em diversos caminhos é o que chamamos de repositório. Além das configurações dos diretórios a serem compartilhados, uma lista de repositórios conhecidos pode ser incluída para que novos recursos sejam localizados.

3.2.5 Área de Trabalho (Workspace)

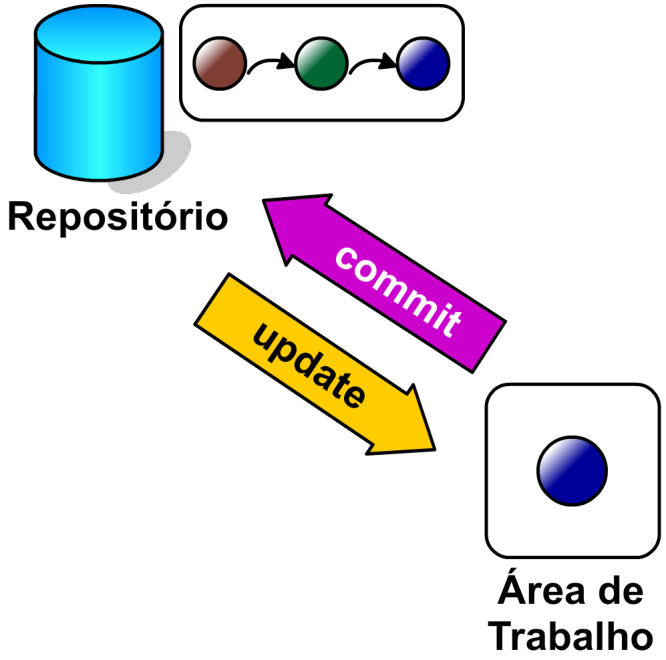
Uma vez definidos os repositórios, os usuários podem iniciar as operações sobre os recursos que estão, ou estarão contidos nestes. Todas as operações realizadas pelos usuários são efetuadas em um espaço de trabalho denominado como Área de Trabalho (Workspace). Na Área de Trabalho encontram-se armazenados todos os recursos que estão sendo utilizados pelos usuários durante seu processo de trabalho. Uma área de trabalho é, inicialmente, uma cópia de um repositório e por isto contém uma réplica de todos os recursos existentes no repositório. Conforme os recursos forem sendo modificados na área de trabalho, a mesma deixará de estar sincronizada com o seu repositório original e passará a estar dessincronizada com o mesmo.

Assim, o usuário pode trabalhar localmente em seus recursos e a qualquer momento publicar suas modificações no repositório sabendo que as modificações de qualquer recurso presente na área de trabalho só estarão disponíveis no repositório quando sua publicação for realizada. A publicação no repositório das modificações dos recursos presentes na área de trabalho faz com que a Área de trabalho volte a ser uma cópia do repositório estando, assim, sincronizado novamente com o mesmo.

Outra possibilidade é que as modificações locais, presentes apenas na área de trabalho, sejam descartadas ou atualizadas a partir do repositório. Para isto, é necessário que os recursos sejam novamente copiados do repositório, atualizando os recursos locais e mantendo a sincronização entre Área de trabalho e repositório.

A Figura 8 apresenta a estrutura básica de comunicação entre os recursos disponíveis na área de trabalho de um usuário e um repositório. As operações de commit e update ilustram o processo de envio e recebimento dos recursos, respectivamente.

Figura 8 – Estrutura básica de comunicação entre a Área de Trabalho e o Repositório



Fonte: o autor

3.2.6 Usuários

A estrutura da UAISharing para compartilhamento possui dois tipos de usuários que desempenham as atividades necessárias para o seu funcionamento, são eles: usuários de sistema (system) e usuários comuns (users). Os usuários de sistema são uma versão simplificada dos usuários comuns e atuam exclusivamente nas tarefas que são realizadas automaticamente, por exemplo: realização de backup de repositórios, processo de descoberta de repositórios presentes na rede local, dentre outras.

Portanto, há um conjunto de atividades específicas que cada tipo de usuário realiza dentro do processo de compartilhamento de recursos, mas a principal diferença consiste na interação de cada tipo de usuário com a atividade desempenhada. Os usuários de sistema realizam as atividades internas do processo de compartilhamento e não necessitam

de qualquer ação externa para que suas atividades sejam executadas, pois estão são realizadas com base em configurações existentes e servem de apoio para as atividades do outro tipo de usuário, visto que são responsáveis pela manutenção da estrutura de compartilhamento através de atividades de gerenciamento dos repositórios e recursos conhecidos e/ou utilizados pelos usuários. Já os usuários comuns são os que interagem diretamente com o sistema a fim de realizar alguma ação de seu interesse, como: configurar área workspace, localizar recursos, publicar recursos, dentre outras.

As figuras 9 e 10 apresentam as atividades desenvolvidas pelos usuários comuns e de sistemas, respectivamente.

3.3 Distribuindo Recursos

Tão logo tenha sido realizado o empacotamento, a distribuição do recurso pode ser realizada. A distribuição depende de algumas definições como a arquitetura da distribuição, a definição dos repositórios, o protocolo para comunicação e ações sobre estes repositórios. Destaca-se que mesmo sendo um repositório local, conforme apresentado na subseção 3.2.4, qualquer recurso presente em um repositório pode ser distribuído, pois sua acessibilidade muda conforme a visibilidade do repositório é modificada.

Os repositórios aos quais os recursos serão disponibilizados, são agrupados em dois tipos: **externos** e **locais**. Os repositórios externos devem ser adicionados manualmente, pois estes não serão acessíveis através das mensagens Multicast realizadas pela aplicação durante o processo de descoberta. Já os repositórios locais serão incluídos automaticamente à medida que forem sendo descobertos mediante a troca de mensagens envolvendo todos os usuários presentes na rede local. Independentemente do tipo de repositório presente nas configurações do usuário, uma cache com os recursos que estão disponibilizados serão mantidos para evitar que mensagens sobrecarreguem o tráfego de comunicação. Segundo (6), uma cache consiste em realizar um armazenamento dos dados recentemente utilizados em um local mais próximo de um usuário ou a um conjunto de usuário em particular. Por fim, uma porta de comunicação para que o repositório se torne acessível é fornecida. Maiores detalhes sobre a implementação do compartilhamento de recursos serão apresentados no Capítulo 4.

Inicialmente, pode-se pensar em duas arquiteturas para distribuir os recursos entre os usuários: cliente/servidor e P2P, cada uma com características específicas que podem afetar diretamente a forma de distribuição.

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 (21). Uma das desvantagens desse modelo é a necessidade de um servidor para intermediar a comunicação 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 estruturadas (22). Assim, neste modelo, os usuários podem trocar informações de forma distribuída assumindo que cada instância da aplicação pode vir a ser tanto um servidor, quanto um cliente.

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

3.3.1 Interface de Comunicação

Para a comunicação entre os envolvidos no processo de compartilhamento de recursos torna-se imprescindível o estabelecimento de uma interface padronizada que disponibilize as operações necessárias ao seu funcionamento. Ao se definir as operações, a responsabilidade de como a comunicação ocorrerá torna-se abstrata para seus usuários, cabendo aos desenvolvedores desta interface decidir qual a implementação mais adequada. Portanto, a interface pode ser implementada em diferentes protocolos, tais como: HTTP, FTP, SSH, dentre outros. Além destes, é possível que um protocolo específico seja desenvolvido para otimizar as necessidades de um determinado contexto. Devido as características apresentadas pode-se definir que a UAISharing é multiplataforma e independente de tecnologia.

As Figuras 15 e 16 trazem as operações disponíveis na interface de comunicação para manipulação de repositórios e pacotes, respectivamente. A abrangência das operações sobre os recursos compartilhados é definida no momento da implementação da interface, permitindo assim que uma determinada operação seja realizada apenas dentro da área de trabalho ou diretamente no repositório associado. Maiores detalhes sobre cada operação serão apresentados no Capítulo 4.

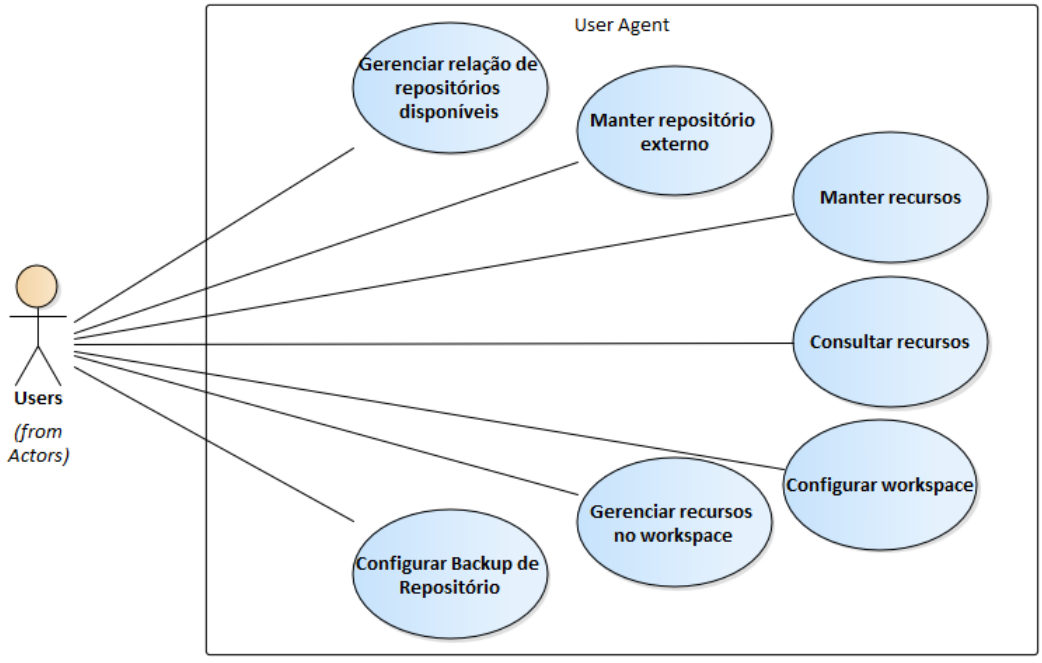
3.4 Operações sobre repositórios

Dentre as principais operações realizadas pelos usuários comuns, têm-se:

- **Gerenciar relação de repositórios disponíveis** → gerencia os repositórios disponíveis, com ações como: habilitar, desabilitar, remover, etc.

- **Manter repositório externo** → realiza as operações de criação, recuperação, atualização e remoção para os repositórios externos, ou seja, os repositórios não descobertos pelo System Agent através da troca de mensagens Multicast.
- **Manter recursos** → adiciona recursos disponíveis nos repositórios conhecidos, externo ou local, ao sistema para utilização.
- **Consultar recursos** → realiza a consulta dos recursos disponíveis nos repositórios gerenciados.
- **Configurar workspace** → configura o workspace para publicação, instalação de pacotes, dentre outros.
- **Gerenciar recursos no workspace** → gerencia os recursos publicados, tornando-os visíveis ou não para outros usuários.
- **Configurar backup de repositório** → realiza a configuração de replicação entre repositórios que será executada internamente pelo sistema (usuários system).

Figura 9 – Casos de uso acionados pelos usuários

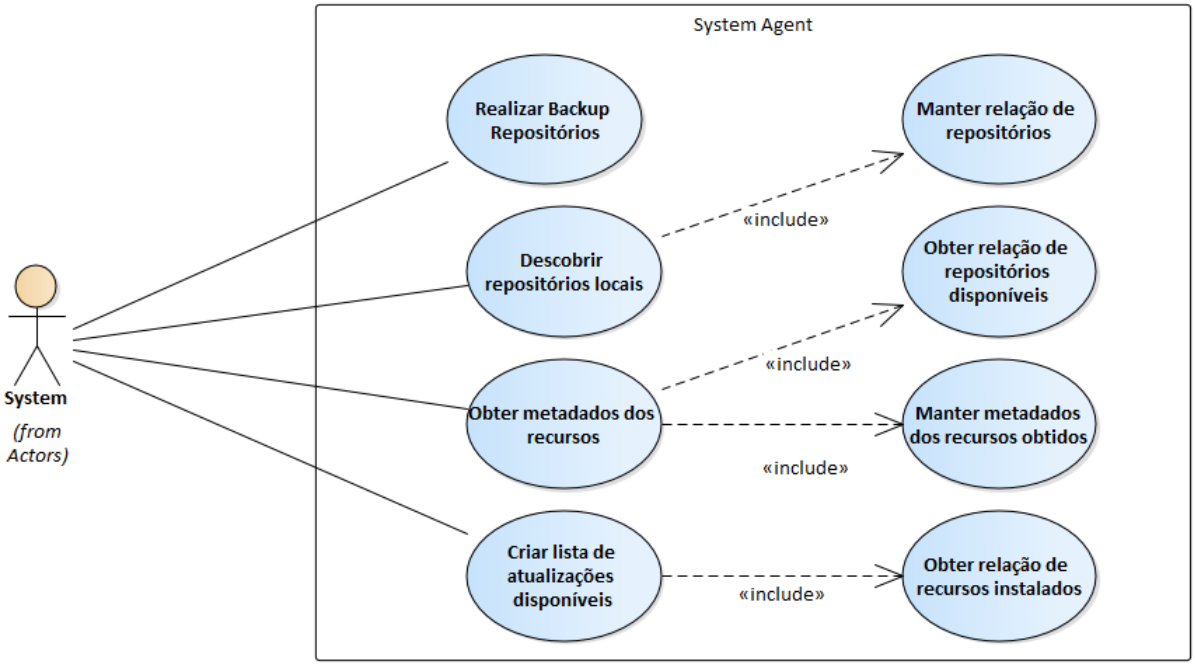


Fonte: os autores

Ao analisar a relação de atividades anteriormente apresentadas é possível observar que todas sofrem influência de um usuário que interage com o sistema e, portanto, necessitam de alguma ação para que sua execução seja realizada. Já as atividades apresentadas abaixo são executadas pelos usuários de sistema e independem de qualquer ação externa para sua execução, pois estas são realizadas internamente pelo sistema.

- **Realizar backup repositórios** → é realizado o backup de cada repositório configurado pelos usuários que devem ser replicados.
- **Descobrir repositórios locais** → através de mensagens Multicast obtém a relação de repositórios disponíveis na rede local.
- **Manter relação de repositórios** → para cada repositório obtido no processo de descoberta, este é mantido na relação de repositório.
- **Obter metadados dos recursos** → obtém os metadados disponíveis para cada recurso presente nos repositórios.
- **Obter relação de repositórios disponíveis** → obtém a relação de repositórios disponíveis/registrados.
- **Manter metadados dos recursos obtidos** → para cada metadado obtido este deve ser mantido para futura consulta.
- **Criar lista de atualizações disponíveis** → cria uma lista de atualização disponível com base nos recursos instalados.
- **Obter relação de recursos instalados** → obtém a relação de recursos instalados.

Figura 10 – Casos de uso em daemon

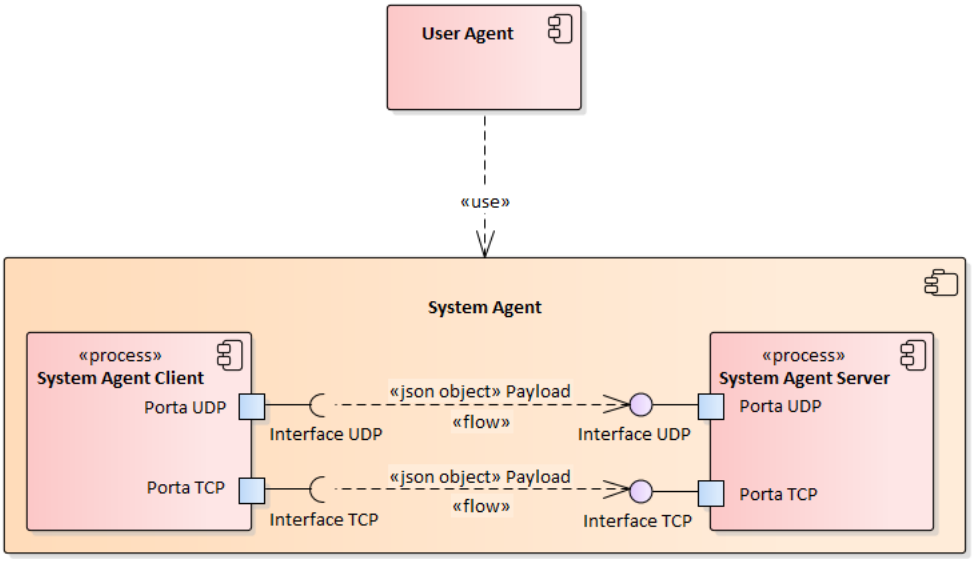


Fonte: os autores

4 Proposta de Implementação

Após o estabelecimento das atividades envolvidas no processo de compartilhamento de recursos apresentado no Capítulo 3, se faz necessária definição de uma estrutura capaz de suportá-las. Para tal, a utilização de componentes deve ser adotada com o objetivo de permitir que a criação e manutenção do sistema de compartilhamento seja eficiente. (23) define componentes como partes modulares de um sistema, que ocultam a sua implementação atrás de um conjunto de interfaces externas. Logo, em um sistema, os componentes que compartilham as mesmas interfaces podem ser substituídos ao mesmo tempo em que preservam o mesmo comportamento lógico. Com base nas atividades desempenhadas pelos tipos de usuários apresentados subseção 3.2.6 é possível definir dois macrocomponentes responsáveis pelas atividades realizadas por cada tipo. A Figura 11 ilustra o relacionamento entre estes componentes.

Figura 11 – Macrocomponentes utilizados no compartilhamento de recursos



Fonte: o autor

O macrocomponente denominado System Agent consiste em um processo que é iniciado juntamente com a aplicação e tem por objetivo executar atividades internas necessárias ao compartilhamento de recursos. Nele há dois subcomponentes principais que são incumbidos da comunicação entre as instâncias de compartilhamento, são eles: System Agent Server e System Agent Client. O subcomponente System Agent Server, durante a sua inicialização, cria dois sockets de conexão de rede, utilizando os protocolos UDP e TCP que serão responsáveis por receber e processar as requisições dos clientes. Já o subcomponente System Agent Client cria duas conexões, UDP e TCP, para estabelecer

comunicação com o servidor e realizar as operações de compartilhamento. O outro macrocomponente, denominado User Agent, engloba todas as operações que são realizadas mediante as ações que os usuários executam durante o compartilhamento de recursos, o mesmo disponibiliza uma CLI (Command-Line Interface - Interface de Linha de Comando) denominada **uaipkg**[1] que é responsável por todo o processo de compartilhamento, sendo que algumas dessas ações são intermediadas pelos sockets de comunicação criados pelo macrocomponente System Agent.

Torna-se necessário a definição para casos em que a comunicação ocorra dentro da mesma estação de trabalho, por exemplo, em repositórios locais, embora utilize a mesma interface externada pelos componentes apresentados, a implementação das funcionalidades ocorre sem a necessidade da troca de mensagens pela rede, mas toda estrutura das mensagens realizadas satisfaz às regras que serão apresentadas a seguir.

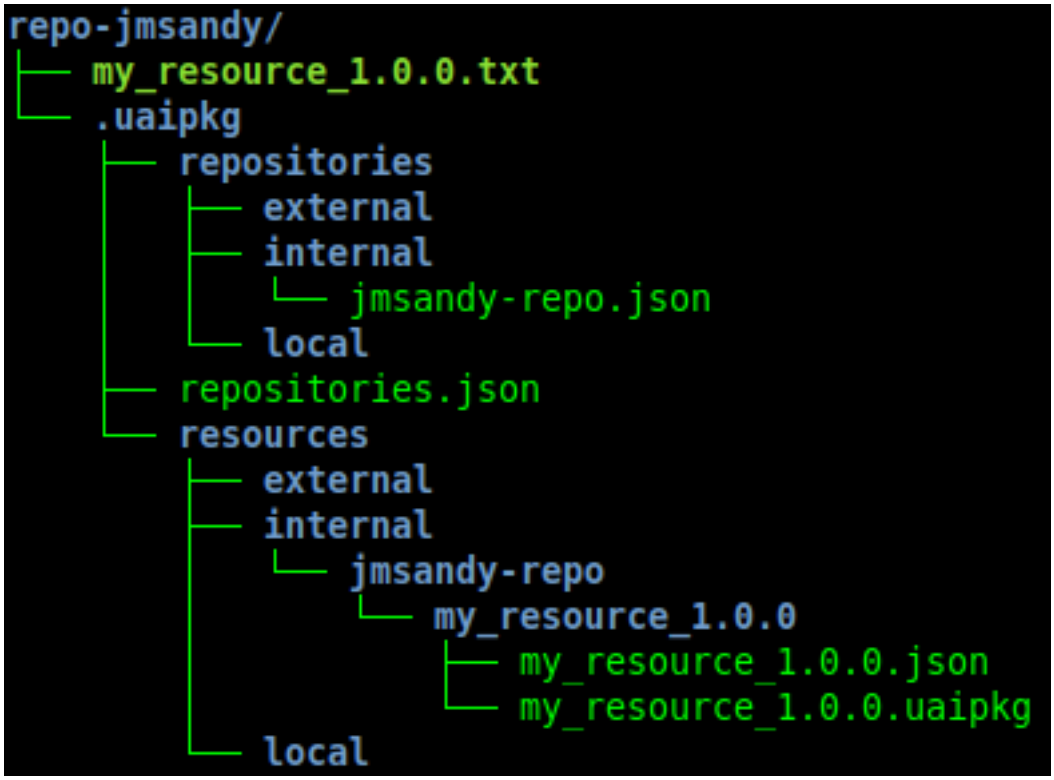
4.1 Estrutura da Área de Trabalho

Cada Área de Trabalho permite o usuário a se associar a vários outros repositórios para obtenção de recursos. Cada repositório, possui uma lista de recursos que disponibiliza ou é disponibilizado pelo usuário, podendo existir um mesmo recurso em várias versões distintas, cabendo ao usuário definir qual a mais adequada para as suas necessidades atuais. A Figura 12 apresenta um usuário que possui um único repositório nomeado como *repo-jmsandy* e com um recurso, *my-resource_1.0.0.txt* na versão 1.0.0. Este arquivo encontra-se tanto na Área de Trabalho do usuário quanto na lista de recursos disponibilizados localmente. Dentro da Área de trabalho, há um diretório oculto chamado *uaipkg* que armazena toda a estrutura de repositórios e recursos. Na raiz desse diretório fica o arquivo *repositories.json* que contém o mapeamento para todos os repositórios presentes na Área de Trabalho e estes por sua vez a referência para todos os recursos aos quais estão vinculados. Ainda dentro deste diretório oculto há dois subdiretórios, **repositores** e **resources**, que armazenam os vínculos com os repositórios e recursos, respectivamente. Dentro de cada subdiretório, há outros três subdiretórios que indicam o tipo de repositório vinculado a cada estrutura (repositório ou recurso). Nos subdiretórios internos a **repositories**, possuem apenas arquivos json com as informações de cada repositório existente, por exemplo: jmsandy-repo.json. Por sua vez, os subdiretórios internos a **resources** armazenarão os recursos associados a cada repositório, sendo que cada recurso possui o arquivo compactado, extensão *.uaipkg* e o seu descritor, disponíveis dentro de um subdiretório com o nome e versão do recurso que ficam internos ao repositório ao qual se associa, neste caso, estão no seguinte caminho: **resources/internal/jmsandy-repo/my_resource_1.0.0**, onde

[1] **Uaipkg** é um acrônimo para UAISharing Package e tem por objetivo controlar o processo de compartilhamento de recursos.

jmsandy-repo e **my_resource_1.0.0**, representam o nome do repositório e recurso, respectivamente.

Figura 12 – Estrutura do repositório proposta para implementação.

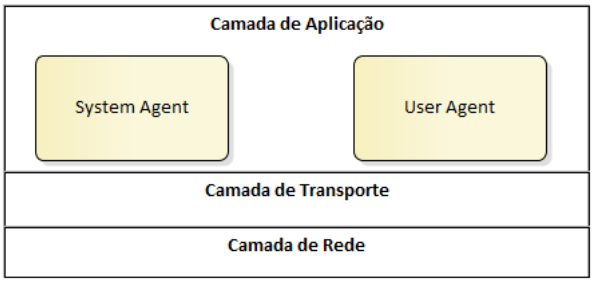


Fonte: o autor

4.2 Protocolo de Comunicação

Como apresentado na seção anterior há dois macrocomponentes que estabelecem a comunicação entre si e com outras instâncias de compartilhamento. Quando a comunicação é realizada entre componentes da mesma instância de compartilhamento, ou seja, ambos se encontram em um mesmo usuário, esta ocorre através da resposta a eventos solicitados diretamente pelos usuários, como exemplo, podem ser citadas as operações destinadas a manipulação de recursos, dentre elas: obter recursos, instalar recursos, etc. Embora exista um padrão para que esta comunicação ocorra internamente, este pode ser definido com maior simplicidade, pois qualquer mudança em sua definição não afeta a comunicação com outras instâncias de compartilhamento. Já a segunda forma de comunicação, entre instâncias de compartilhamentos distintas, necessita de um protocolo bem estabelecido para que ambas as instâncias possam se comunicar. Para tal, o protocolo criado se situará na camada de aplicação da pilha de protocolos TCP/IP. A Figura 13 apresenta um resumo das principais camadas envolvidas na comunicação.

Figura 13 – Localização dos componentes na pilha de protocolos TCP/IP



Fonte: o autor

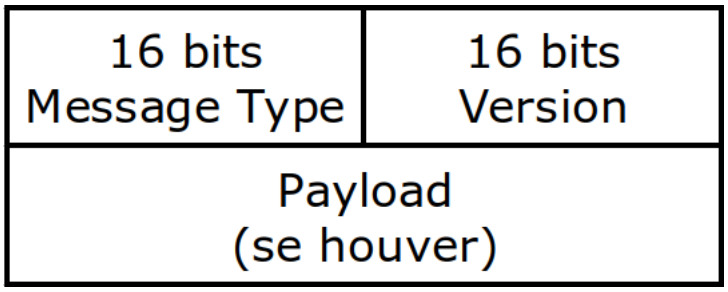
- **Camada de aplicação**: o protocolo utilizado para comunicação entre as instâncias de compartilhamentos está localizado na camada de aplicação na pilha de protocolo TCP/IP.
- **Camada de transporte**: os protocolos mais utilizados para comunicação, TCP e UDP, serão utilizados pelo protocolo estabelecido na camada de aplicação. O UDP será utilizado exclusivamente durante o processo de descoberta através da troca de mensagens de Multicast, já o protocolo TCP será utilizado para as demais funcionalidades.
- **Camada de rede**: a camada de rede está representada para resumir o processo de comunicação.

Uma vez estabelecida a necessidade do protocolo e onde ele se situa na pilha de protocolo TCP/IP, se faz necessário sua definição. O mesmo possuirá uma estrutura de mensagem simples, mas ao mesmo tempo amplamente customizável, com os seguintes campos:

- **Message Type**: possuindo 16 bits, o campo é utilizado para indicar o tipo de mensagem trafegada pelo protocolo, podendo representar até 65.536 mensagens diferentes;
- **Version**: envia a versão atual dos dados contidos no campo payload, indicando ao destinatário a forma que a mensagem deve ser tratada;
- **Payload**: consiste em um arquivo json com os dados de envio e retorno associados. Juntamente, com os campos: Message Type e Version, o conteúdo nele presente é extraído e manipulado no recebimento das mensagens.

A Figura 14 apresenta a estrutura da mensagem anteriormente apresentada.

Figura 14 – Mensagem Padrão do Protocolo de Comunicação



Fonte: o autor

4.3 Mensagens de Comunicação

As próximas subseções apresentarão detalhes de cada mensagem utilizada pelo protocolo de comunicação. Um resumo destas mensagens pode ser visualizado na Tabela 2:

Tabela 2 – Tipos de Mensagens Utilizadas pelo Protocolo de Comunicação

Message Type	**Descrição**
1	Descoberta de repositórios na rede local
2	Obtenção dos metadados dos recursos
3	Obtenção dos metadados dos recursos (Resposta)
4	Verificação das atualizações disponíveis
5	Verificação das atualizações disponíveis (Resposta)
6	Obter recurso por identificador
7	Obter recurso por identificador (Resposta)
8	Publicar informações do recurso
9	Publicar informações do recurso (Resposta)

Fonte: o autor

4.3.1 Descoberta de repositórios na rede local

A mensagem de descoberta de repositórios na rede local tem por objetivo montar uma lista de repositórios conhecidos que estão localizados nas proximidades dos usuários. Ao iniciar a aplicação a periodicidade para o envio destas mensagens é obtido nas configurações e sempre que ela for atingida uma mensagem de descoberta é submetida ao endereço de Multicast conhecido pela aplicação. O conteúdo enviado no campo Payload desta mensagem é um json contendo os dados do repositório para ser registrado nas demais instâncias. O Código 4.1 apresenta o conteúdo submetido no payload de uma mensagem: nas linhas 2 e 3, são informados o identificador único do repositório a ser gerado pela aplicação e o nome que identifica o repositório, respectivamente. Já na linha 4, são fornecidas as informações para conexão com o repositório: endereço IP e as portas TCP e UDP.

Código 4.1 – Payload da Mensagem de Descoberta de Repositórios na Rede Local

```
1  {
2    "id": "unique-identifier",
3    "name": "my-repository",
4    "address":
5      {
6        "ip": "192.168.1.12",
7        "tcpport": "10101",
8        "udpport": "10102"
9      }
10 }
```

4.3.2 Obtenção dos metadados dos recursos

Uma vez localizados os repositórios, é necessário verificar quais são os recursos que estão sendo compartilhados. Para cada repositório conhecido, seja ele local ou externo, um conjunto de mensagens são enviadas para obtenção das informações dos recursos. No campo payload desta mensagem deve ser enviado um json contendo o número da página atual e o tamanho de cada página envolvida na comunicação, linhas 2 e 3, respectivamente. As informações relativas as páginas são utilizadas para controlar o processo de comunicação de modo a não sobrecarregar o tráfego de comunicação. Sempre ao enviar a primeira mensagem de obtenção de recursos, o valor contido no atributo **page** será 1 e o atributo **pagesize** será um valor lido nas configurações da aplicação e à medida que as mensagens forem sendo trocadas o atributo **page** é incrementado até que o número total de páginas seja atingido. O Código 4.2 apresenta a mensagem inicial para obtenção dos metadados destinados a um repositório.

Código 4.2 – Payload da Mensagem de Obtenção dos Metadados dos Recursos

```
1 {
2   "page": 1,
3   "pagesize": 10
4 }
```

4.3.3 Obtenção dos metadados dos recursos (Resposta)

Ao receber as mensagens para obtenção dos metadados (ver subseção 4.3.2) o atributo payload é extraído para processamento, este consiste em obter os valores da página a ser retornada e a quantidade de recursos presentes em cada página. De posse destes, o número total de páginas é calculado levando em consideração a quantidade de

recursos compartilhados existentes no repositório, por exemplo, ao receber um tamanho de 10 recursos por página e o repositório conter 150 recursos, a quantidade total de páginas será 15. Desta forma, o processo de envio das mensagens apresentado na subseção 4.3.2 enviará 15 mensagens para obter todos os recursos presentes no repositório. O Código 4.3 apresenta o conteúdo enviado no atributo payload da mensagem de resposta, nas linhas 2 e 3 são enviados o número da página de resposta e o total de páginas disponíveis para consulta, respectivamente. Na linha 4, um conjunto com os metadados dos recursos é retornado, os valores contidos nas linhas 5 e 6 são apenas para ilustrar que os metadados serão fornecidos individualmente para cada recurso, o conteúdo exato destes metadados podem ser visualizados no Código 3.1.

Código 4.3 – Payload da Mensagem de Obtenção dos Metadados dos Recursos (Resposta)

```
1 {
2   "page": 1,
3   "totalpage": 15,
4   "metadados": [
5       "Metadados do Recurso 1",
6       "Metadados do Recurso 2"
7   ]
8 }
```

4.3.4 Verificação das atualizações disponíveis

Após a instalação de algum recurso, ainda que este esteja completamente funcional, é desejável mantê-lo atualizado para se ter acesso as novas funcionalidades e correções que porventura foram realizadas. Para tal, torna-se necessário verificar nos repositórios de origem dos recursos instalados, a existência de novas versões disponíveis para atualização. Logo, uma mensagem é enviada com uma lista de identificadores dos recursos instalados no atributo payload do protocolo de comunicação. O Código 4.4 apresenta o envio de dois identificadores de recursos para verificação das atualizações em um determinado repositório. As linhas 3 e 7 indicam quais os recursos a serem verificados e as linhas 4 e 8 suas referidas versões. Em uma mesma mensagem pode ser verificado um conjunto de recursos instalados em um mesmo repositório, ou seja, para cada repositório registrado pelo menos uma mensagem de verificação deve ser enviada, o número de mensagens é definido de acordo com as configurações da aplicação que determinará o máximo de recursos verificados em cada mensagem.

Código 4.4 – Payload da Mensagem de Obtenção das atualizações disponíveis

```
1 [
2   {
```

```
    "resourceId": "resource-identifier-1",
    "version": "1.0"
  },
  {
    "resourceId": "resource-identifier-2",
    "version": "1.2"
  },
  {
    "resourceId": "resource-identifier-3",
    "version": "1.3"
  }
]
```

4.3.5 Verificação das atualizações disponíveis (Resposta)

Ao receber a mensagem para verificação das atualizações, o destinatário extrai do atributo payload recebido quais os recursos que devem ser verificados e para cada um é realizada uma checagem para definir se há alguma atualização disponível. Tal checagem consiste em verificar a existência de novas versões com base na versão do recurso recebida na mensagem, sendo que cada nova versão é retornada para o remetente, por exemplo: caso um determinado recurso esteja com a versão 1.0 instalada em algum cliente e no repositório exista duas novas versões, 1.1 e 1.2, a mensagem de retorno conterá as duas versões, pois desta forma o remetente poderá escolher especificamente qual versão deseja atualizar. Além das possíveis versões para atualização a mensagem de retorno indicará se o recurso solicitado não existe mais no repositório e se o recurso estiver atualizado ele é suprimido da resposta. Assim sendo, define-se três situações para os recursos:

1. **Não há atualizações**: o recurso não é enviado na mensagem de retorno para não sobrecarregar o processo de comunicação;
2. **Há atualizações**: uma lista com as versões disponíveis para atualização é enviada ao remetente;
3. **Recurso não localizado**: caso o recurso não seja localizado o mesmo é devolvido para o remetente com a lista de versões vazia. Desta forma, o remetente entende que o mesmo não existe mais no repositório de origem e novas instalações ou atualizações deste recurso não é mais possível.

O Código 4.5 apresenta o atributo payload da mensagem de resposta da mensagem indicada na subseção 4.3.4. O recurso cujo identificador é **resource-identifier-1** obteve

duas novas versões, já o identificador **resource-identifier-2** não possui nenhuma nova versão e por isso foi suprimido do retorno, por fim, o recurso **resource-identifier-3** não foi localizado e uma lista de versões vazia foi retornada para indicar a não existência.

Código 4.5 – Payload da Mensagem de Obtenção das atualizações disponíveis (Resposta)

```
[
  {
    "resourceId": "resource-identifier-1",
      "versions": {
        "1.1",
        "1.2"
      }
  },
  {
    "resourceId": "resource-identifier-3",
    "versions": {
    }
  }
]
```

4.3.6 Obter recurso por identificador

Para que algum recurso seja instalado, ou atualizado, para utilização do usuário é necessário que este seja obtido junto ao seu repositório de origem e armazenamento no local desejado para o respectivo uso. Há vários protocolos que podem ser utilizados para facilitar o processo de transferências de arquivos, havendo inclusive um específico para esta funcionalidade FTP (File Transfer Protocol - Protocolo de Transferência de Arquivos). Outra forma bastante comum de se obter recursos é através do uso do protocolo HTTP (Hypertext Transfer Protocol - Protocolo de Transferência de Hipertexto). Qualquer um dos protocolos citados, inclusive há outros não apresentados, podem ser utilizados internamente a aplicação para se obter o recurso que se deseja, bastando que a implementação seja modificada para utilizá-los. No entanto, a implementação aqui proposta faz uso de sockets através de mensagens trocadas com base na estrutura apresentada. Assim como nas outras mensagens, o tipo e a versão indicam como a mensagem deve ser tratada no destinatário e o atributo payload indica qual o recurso deve ser obtido informando o identificador do recurso e sua referida versão, caso a versão seja omitida, a última disponível no repositório é retornada. O Código 4.6 apresenta o atributo payload da mensagem para obtenção de recursos por identificador.

Código 4.6 – Payload da Mensagem de Obtenção do recurso por identificador

```
{
  "resourceId": "resource-identifier",
  "version": "1.2"
}
```

Após receber a mensagem de retorno (ver subseção 4.3.7), existindo o recurso, a localização deste é extraída e uma nova mensagem desta vez para obtenção do recurso para utilização é realizada.

4.3.7 Obter recurso por identificador (Resposta)

Após receber a solicitação para obtenção do recurso, o repositório de destino verifica se existe algum recurso com a versão desejada e caso exista as informações para download do arquivo é enviada na mensagem de retorno. No atributo payload da mensagem de retorno são enviadas as seguintes informações: localização para download do recurso, versão e situação do recurso. A situação do recurso indica se o mesmo foi ou não localizado e, se há alguma versão mais recente, podendo assumir os valores: 1 - Localizado, 2 - Não Localizado e 3 - Localizado, mas possui uma recente. O Código 4.7 apresenta a resposta para a solicitação de um recurso que foi localizado no repositório e não possuía uma versão mais recente.

Código 4.7 – Payload da Mensagem de Obtenção do recurso por identificador (Resposta)

```
{
  "resource": "resource-identifier",
  "version": "1.2",
  "status": "1"
}
```

4.3.8 Publicar informações do recurso

Para que um recurso se torne acessível aos usuários, se faz necessário que este seja publicado em algum repositório. Inicialmente, uma mensagem com o atributo payload contendo os metadados do recurso deve ser submetida para que a estrutura para armazenamento seja montada para o recebimento do recurso que ocorrerá por um outro canal de comunicação. Maiores detalhes sobre os metadados citados podem ser observados no Código 3.1.

Após o recebimento da mensagem de retorno (ver subseção 4.3.9) referente ao envio dos metadados, ocorrendo sucesso na criação da estrutura para recebimento do recurso, a localização para o envio é extraída e uma nova mensagem, desta vez somente com o recurso, é submetida para que o mesmo seja mantido no repositório.

4.3.9 Publicar informações do recurso (Resposta)

Ao receber a mensagem inicial para publicação do recurso, os metadados são extraídos e a unicidade do recurso é verificada, assim como as informações obrigatórias para sua disponibilização. Para garantir a unicidade é analisada a existência de algum recurso com base em seu identificador e versão, caso nada seja encontrado os campos obrigatórios são verificados, são eles: nome, versão, descrição, licença e autores. Uma vez satisfeitas as regras supracitadas uma estrutura para o recebimento do recurso é criada com base em seu nome e versão, ou seja, o nome dará origem a um diretório e para cada versão deste recurso um subdiretório será criado, dentro deste subdiretório se encontrará os metadados, além do recurso publicado.

Após a realização dos passos apresentados, uma mensagem de retorno é submetida ao remetente com a resposta da solicitação. No atributo payload desta mensagem são enviadas as seguintes informações: situação da solicitação, descrição do processamento e localização para publicação do recurso. A situação da solicitação indica o resultado final do processamento da solicitação, podendo assumir os valores: 1 - OK e 2 - Ocorreu um erro. O campo com a descrição do processamento conterá o motivo da solicitação não ter sido processada e assim ter o problema corrigido em sua origem. Em casos de sucesso, o campo localização conterá o local de destino do recurso. O Código 4.8 apresenta a resposta para a publicação de um recurso que foi atendido com sucesso.

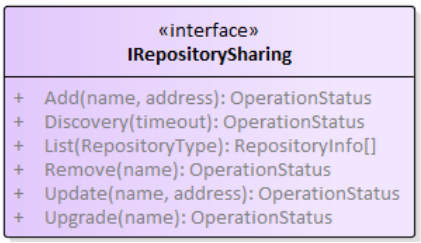
Código 4.8 – Payload da Mensagem para Publicar informações do recurso (Resposta)

```
{
  "resource": "resource-localization",
  "message": "Processado com sucesso",
  "status": "1"
}
```

4.4 Operações com Repositórios

Para o processo de compartilhamento de recursos se faz necessário a existência de um conjunto de repositórios conhecidos e que estes sejam fornecedores de conteúdo. Logo, para que qualquer recurso seja acessível, este deve ser publicado em um repositório que por sua vez deverá ser registrado em todos os usuários que possuam interesse nos recursos nele publicado. Nas subseções a seguir serão apresentadas as operações disponíveis (ver Figura 15) para configuração dos repositórios de recursos.

Figura 15 – Operações existentes na Interface de Manipulação de Repositórios



Fonte: os autores

4.4.1 Listar Repositórios

A listagem de repositórios é uma atividade costumeira ao usuário que consiste em verificar quais repositórios estão acessíveis a ele. Para facilitar o processo organizacional dos repositórios existirão três grupos de repositórios:

1. **Interno (internal)**: são repositórios configurados pelo usuário para publicação de seus recursos e obtenção de recursos por terceiros;
2. **Locais (local)**: são repositórios que foram localizados através das mensagens de Multicast e automaticamente inseridos na lista de repositórios acessíveis do usuário;
3. **Externos (external)**: são repositórios que foram adicionados explicitamente pelo usuário, podendo ou não, estar em uma rede local. Sua acessibilidade é verificada apenas ao se obter alguma informação nele presente.

Para fins de listagem um quarto grupo é criado de forma lógica e engloba todos os grupos de repositórios apresentados acima. Abaixo é apresentada a instrução para listagem dos repositórios:

```
$ uaipkg list repository {all|local|internal|external}
```

4.4.2 Adicionar Repositório

Novos repositórios podem ser adicionados de duas formas: i) automaticamente através de mensagens de Multicast; e ii) manualmente pelo usuário. O nome do repositório será considerado como identificador único, logo não é possível registrar vários repositórios com o mesmo nome. A adição de um novo repositório é simples e basta que seu nome, localização e tipo sejam fornecidos, onde os tipos são os apresentados na seção 4.4: local, internal ou external. Para tal, a instrução abaixo deve ser executada:

```
$ uaipkg add repository "name" "address" "type"
```

Internamente, cada repositório é mantido em um arquivo (*nome_repositorio.json*) com suas informações de acesso e o conjunto de recursos nele compartilhados. O Código 4.9 apresenta a estrutura do arquivo de configuração para o repositório "Meu Primeiro Repositorio".

Código 4.9 – Exemplo Arquivo Repositório ("Meu Primeiro Repositorio.json")

```
{
  "repository": {
    "id": "my-repository-id",
    "name": "Meu Primeiro Repositorio",
    "address": "192.168.1.100:8000",
    "address": {
      "ip": "192.168.1.100",
      "tcpport": "8000",
      "udpport": "8001"
    }
    "resources": [
        "resource1",
        "resource2",
    ]
  }
}
```

No entanto, um arquivo de configuração de repositórios (*repositories.json*) com todos os repositórios conhecidos e controlados pelo usuário é utilizado. Neste arquivo, há os três grupos de repositórios apresentados na subseção 4.4.1 e cada arquivo individual do repositório é acrescentando em seu referido grupo. O Código 4.10 apresenta o arquivo *repositories.json*.

Código 4.10 – Exemplo Arquivo Configuração de Repositórios (repositories.json)

```
{
  "internal": {
    "repositories": [
      "Meu Primeiro Repositorio.json"
    ]
  },
  "local": {
    "repositories": [
    ]
  },
  "external": {
```

```
12      "repositories": [
13      ]
14    }
15  }
```

4.4.3 Atualizar Repositório

O procedimento para atualização de repositório é extremamente parecido com o processo de atualização, isto é, informa-se o nome do repositório que se deseja atualizar e o respectivo endereço. Vale ressaltar, que o nome é o identificador do repositório dentro da estrutura, portanto ele não pode ser atualizado, por isso apenas o endereço será modificado. Caso não seja localizado algum repositório com o nome informado uma mensagem de erro é informada ao usuário e este pode realizar o processo de adição, visto que o repositório solicitado para atualização não foi encontrado. A instrução abaixo apresenta a atualização do endereço do repositório nomeado como "Repositorio".

```
$ uaipkg update repository "Repositorio" "192.168.1.200:8888"
```

4.4.4 Remover Repositório

A remoção de um repositório consiste em torná-lo inacessível para os usuários, inclusive aos que o detém. Logo, não é possível publicar recursos nos repositórios locais e tampouco obter recursos em qualquer repositório, independente de qual grupo este esteja enquadrado. Abaixo é apresentada a instrução para remoção de um repositório nomeado como "Repositorio".

```
$ uaipkg remove repository "Repositorio"
```

Como a remoção não realiza a exclusão dos recursos já compartilhados uma vez adicionado o repositório, na mesma localização, todos os recursos presentes já se encontrarão disponíveis.

4.4.5 Descobrir Repositórios

Embora o processo de descoberta de repositórios possa ser realizado automaticamente, o usuário pode solicitá-lo manualmente quando o processo automatizado estiver desativado. Assim sendo, o usuário deve executar a instrução para procurar os repositórios presentes na rede local, informando um tempo, em segundos, que a descoberta deve permanecer ativa. A instrução abaixo apresenta o processo de descoberta por um período de 30 segundos.

```
$ uaipkg search repository -t 30
```

4.4.6 Atualizar Lista de Recursos Existentes

Feita a adição dos repositórios que se deseja ter acesso, seja ela realizada de forma automática ou manual, se faz necessário a obtenção dos recursos presentes em cada um. Assim como a descoberta de repositórios presentes na rede interna, todo esse processo pode ser automatizado mediante a configuração, mas uma vez desabilitado ou se houver a necessidade de execução imediata, o usuário pode acionar o processo de atualização da lista de recursos existentes manualmente. Para tal, a instrução abaixo deve ser executada informando a cláusula *all* se todos os repositórios conhecidos devem ser atualizados ou informar especificamente o repositório desejado.

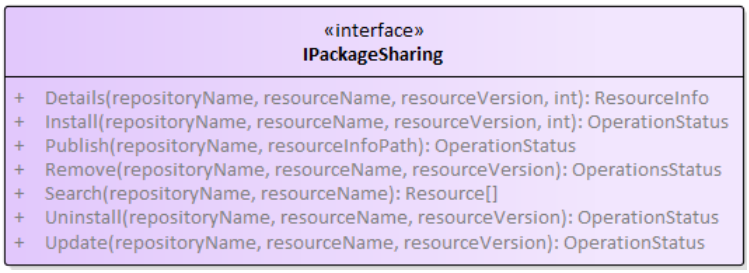
```
$ uaipkg upgrade repository {all|repository}
```

Este processo consiste em obter os metadados de todos os recursos compartilhados nos repositórios desejados. Para cada recurso existente, os metadados são adicionados separadamente com a nomenclatura: *"NomeRecurso_versao.json"* e a localização de cada um é adicionada dentro do arquivo apresentado pelo Código 4.9 para futuras consultas.

4.5 Operações com Recursos

Após a configuração dos repositórios que estarão envolvidos no processo de compartilhamento, os usuários tornam-se capazes de compartilhar de recursos entre si. Nas próximas subseções serão apresentadas as ações (ver Figura 16) que os usuários podem realizar estabelecer o compartilhamento de recursos.

Figura 16 – Operações existentes na Interface de Manipulação de Pacotes



Fonte: os autores

4.5.1 Publicar Recursos

A publicação consiste em disponibilizar algum recurso em um repositório que esteja presente no grupo de repositórios denominado **local**, ou seja, em repositórios que estão sob o controle do usuário, permitindo as operações de escrita. Ao selecionar um recurso para publicação um arquivo com as informações com de seus metadados (ver Código 3.1) deve ser criado no mesmo diretório que o recurso a ser publicado se encontra, após a sua criação a publicação pode ser efetuada. A instrução abaixo ilustra a publicação de um recurso denominado *"MeuPrimeiroRecurso"* em um repositório cujo nome é *"MeuRepositorio"*.

```
$ uaipkg publish package "MeuRepositorio" "MeuPrimeiroRecurso.json"
```

Feita a publicação, o recurso torna-se disponível para ser utilizado por outros usuários.

4.5.2 Remover Recursos

Uma vez publicados, os recursos tornam-se visíveis para os usuários que tenham acesso ao repositório no qual eles se encontram, se por alguma razão o proprietário do recurso não queira mais disponibilizá-lo para outros usuários, a remoção deste recurso deve ser realizada. No entanto, assim como a publicação, para que a remoção seja possível é necessário que o repositório cujo recurso esteja presente esteja sobre o controle do usuário, isto é, seja um repositório presente no grupo **local**. Para efetuar a remoção o usuário deve informar o nome do repositório, nome e versão do recurso, conforme instrução abaixo:

```
$ uaipkg remove package "MeuRepositorio" "MeuRecurso" "1.0.0"
```

A instrução apresentada realiza a remoção do recurso denominado *"MeuRecurso"* cuja versão é *"1.0.0"* que está presente no repositório *"MeuRepositorio"*.

4.5.3 Localizar Recursos

A localização de recursos é de extrema importância para o processo de compartilhamento, pois através dele o usuário é capaz de localizar os recursos que melhor atenda suas necessidades. Um usuário pode realizar a procura por recursos em todos os repositórios conhecidos de uma única vez - informando a cláusula *all* - ou em algum específico informando o nome, a instrução abaixo apresenta a sintaxe para localização de recursos.

```
$ uaipkg search package {all|repository} "NomeRecurso"
```

Como a consulta pode ser realiza em múltiplos repositórios ao mesmo tempo, o retorno da consulta além do recurso com as versões disponíveis também apresenta o repositório ao qual o recurso pode ser obtido.

4.5.4 Obter Informações do Recursos

A subseção 4.5.3 trata da localização dos recursos de uma maneira ampla, não se importando com os detalhes existentes em cada recurso apresentando apenas as informações mínimas necessárias para as demais operações. Devido as estas características, se faz necessário uma instrução mais abrangente que apresente todas as informações relevantes do recurso desejado para o usuário solicitante. Portanto, para se obter tais informações, o usuário deve executar a instrução abaixo informando o nome do repositório, nome e versão do recurso, sendo a versão opcional e caso não seja fornecida a última versão existente do recurso é retornada.

```
$ uaipkg info package "MeuRepositorio" "NomeRecurso" "1.0.0"
```

A instrução apresentada obtém a informação do recurso denominado *"NomeRecurso"* cuja versão é *"1.0.0"* presente no repositório *"MeuRepositorio"* e as apresenta ao usuário. Tais informações são as contidas no arquivo de metadados utilizado durante a publicação.

4.5.5 Instalar Recursos

A instalação visa obter o recurso desejado em algum repositório conhecido, tornando-o disponível para utilização do usuário. Com base nas informações do recurso, nome, versão e repositório, o usuário solicita a instalação através da instrução abaixo:

```
$ uaipkg install package "RepositorioExterno" "NomeRecurso" "1.0.0"
```

Assim como na subseção 4.5.4 a versão é opcional e quando não informada a última versão do recurso é instalada. Uma vez executada a instalação, o recurso é copiado para a área de trabalho do usuário sendo organizado por repositório, nome e versão, ou seja, pode existir várias versões de um mesmo recurso na área de trabalho do usuário e também pode existir recursos com o mesmo nome e versão, mas em repositórios diferentes. Caso o usuário realize alguma modificação neste recurso e queira compartilhar, basta que este seja publicado em seu repositório interno, a partir deste momento qual outro usuário pode obter as modificações realizadas.

4.5.6 Desinstalar Recursos

A desinstalação de um recurso é uma operação realizada na área de trabalho do usuário e descarta todas as alterações que porventura tenham sido efetuadas, mas não afeta qualquer versão que já tenha sido publicada em algum repositório. Caso o usuário queira desinstalar um recurso presente em sua área de trabalho, este deve fornecer o repositório de origem, o nome e a versão do recurso, sendo a versão opcional e, se não informada, a última instalada para o recurso dentro do repositório de origem. A instrução abaixo realiza a remoção do recurso denominado *"NomeRecurso"* cuja versão é *"1.0.0"* dentro do repositório *"RepositorioExterno"*.

```
$ uaipkg uninstall package "RepositorioExterno" "NomeRecurso" "1.0.0"
```

4.5.7 Atualizar Recursos

Normalmente, um recurso receberá um conjunto de atualizações ao longo de sua existência e estas podem ser replicadas para os usuários que possuem versões mais antigas. Para tal, em conjunto com o processo apresentado na subseção 4.5.4, novas versões podem ser localizadas e atualizadas na área de trabalho dos usuários. A atualização é realizada informando o nome do repositório para obtenção do recurso, bem como o novo e versão do recurso desejado, sendo a versão opcional e caso esta não seja informada o recurso é atualizado para a última disponível. A instrução abaixo apresenta a atualização do recurso denominado *"NomeRecurso"* presente no repositório *"RepositorioExterno"* para a versão *"2.0.0"*.

```
$ uaipkg update package "RepositorioExterno" "NomeRecurso" "2.0.0"
```

5 Cenário de Uso

Este capítulo apresenta um cenário de compartilhamento envolvendo todas as etapas apresentadas nos capítulos 3 e 4, proposta e a implementação, respectivamente. Para tal, todas as atividades necessárias para que os usuários compartilhem recursos entre si serão descritas a fim de validar a estrutura proposta.

5.1 Descrição do Cenário

Em um cenário hipotético, os usuários: Alice, Bob e Carlos, desejam compartilhar recursos entre si. Alice e Bob estão presentes em uma rede local que permite o uso de mensagens Multicast para o processo de descoberta, ao passo que Carlos se encontra em uma outra rede, portanto, em uma rede externa. Portanto, a rede de compartilhamento estabelecerá a seguinte estrutura de comunicação:

- **Alice:** o usuário Alice possuirá três repositórios: i) repositório local; ii) repositório interno de Bob; e iii) repositório externo de Carlos;
- **Bob:** o usuário Bob possuirá três repositórios: i) repositório local; ii) repositório interno de Alice; e iii) repositório externo de Carlos;
- **Carlos:** o usuário Carlos possui apenas o repositório local, sendo que este é acessado pelos demais usuários.

A Tabela 3 resume as configurações dos usuários que estão envolvidos no compartilhamento.

Tabela 3 – Configurações dos Usuários para Compartilhamento

Usuário	**Endereços**		
	TCP	**UDP**	**Multicast**
Alice	192.168.1.153:8800	192.168.1.153:58800	224.0.0.100
Bob	192.168.1.154:8800	192.168.1.154:58800	
Carlos	203.0.113.1:8800	203.0.113.1:58800	

Fonte: o autor

5.1.1 Configurando Repositórios

Inicialmente, nenhum dos usuários que estarão envolvidos no compartilhamento de recursos possuem qualquer repositório. Portanto, cada usuário deve adicionar um repositório local para controlar seus próprios recursos, para facilitar o cenário proposto os

repositórios conterão o nome do usuário possuidor do repositório, bem como seu tipo, por exemplo: **AliceRepoLocal**.

Alice adiciona o repositório local:

```
$ uaipkg add repository AliceRepoLocal 192.168.1.153:8800 local
```

Bob adiciona o repositório local:

```
$ uaipkg add repository BobRepoLocal 192.168.1.154:8800 local
```

Carlos adiciona o repositório local:

```
$ uaipkg add repository CarlosRepoLocal 203.0.113.1:8800 local
```

Uma vez adicionados os repositórios locais todos os usuários estão aptos a publicarem recursos em seus próprios repositórios, mas como ainda não adicionaram repositórios internos ou externos, não são capazes de acessar recursos de outros usuários. A próxima etapa consiste em estabelecer a rede de compartilhamento, ou seja, os repositórios internos devem ser descobertos e adicionados, assim como os repositórios externos conhecidos devem ser adicionados. Assim sendo, Alice, Bob e Carlos acionam o processo de descoberta de repositórios internos e assim se obter repositórios acessíveis por mensagem de **multicast**. Para tal, cada um executa a instrução abaixo, que consiste em iniciar o processo de descoberta por um período de 30 segundos.

```
$ uaipkg search repository -t 30
```

Após a execução da instrução supracitada, Alice e Bob obtêm como resposta o repositório um do outro, visto que estão na mesma rede local e são acessíveis por mensagens de Multicast. Já Carlos, por estar em uma rede externa, não consegue localizar nenhum repositório automaticamente. De posse dos repositórios obtidos, Alice e Bob, realizam a inclusão dos repositórios obtidos pelo processo de descoberta, através das instruções abaixo.

Alice adiciona o repositório interno de Bob:

```
$ uaipkg add repository BobRepoLocal 192.168.1.154:8800 internal
```

Bob adiciona o repositório interno de Alice:

```
$ uaipkg add repository AliceRepoLocal 192.168.1.153:8800 internal
```

O repositório de Carlos, por ser um repositório externo conhecido, também será adicionado por Alice e Bob, através da instrução abaixo.

```
$ uaipkg add repository CarlosRepoLocal 203.0.113.1:8800 external
```

Uma vez executadas as instruções apresentadas até este ponto, os repositórios estão configurados e os usuários estão aptos a compartilharem recursos entre si.

5.1.2 Compartilhando Recursos

Uma vez estabelecida a estrutura para compartilhamento, cada usuário pode realizar a troca de recursos entre si. Com o objetivo de simplificar os passos que serão apresentados, os arquivos com os metadados serão omitidos, portanto sempre que o arquivo com metadados for citado, o Código 3.1 deve ser assumido como exemplo. Alice, publicará 2 recursos em seu repositório local, denominados *RecursoAlice1.pdf* e *RecursoAlice2.png*, ambos na versão 1.0.0.

```
$ uaipkg publish package AliceRepoLocal RecursoAlice1.json
$ uaipkg publish package AliceRepoLocal RecursoAlice2.json
```

Já Bob, publicará em seu repositório um outro recurso em seu repositório local, denominado *RecursoBob1.py* cuja versão é 1.1.0.

```
$ uaipkg publish package BobRepoLocal RecursoBob1.json
```

Por fim, Carlos publicará um recurso denominado *RecursoCarlos1.odt*, versão 1.0.0, que será obtido por todos os demais usuários.

```
$ uaipkg publish package CarlosRepoLocal RecursoCarlos1.json
```

A Tabela 4 apresenta um resumo dos recursos existentes nos repositórios após a publicação inicial de cada usuário.

Tabela 4 – Recursos existentes em cada repositório após a configuração inicial

Repositório	**Recurso**	**Versão**
AliceRepoLocal	RecursoAlice1.pdf	1.0.0
	RecursoAlice2.png	1.0.0
BobRepoLocal	RecursoBob1.py	1.1.0
CarlosRepoLocal	RecursoCarlos1.odt	1.0.0

Fonte: o autor

Após a publicação dos recursos pelos usuários é necessário que cada um realiza a atualização de seus repositórios para que os recursos publicados sejam reconhecidos pelos demais usuários. Logo, cada usuário executa a instrução abaixo para se obter o conjunto de informações (metadados) de cada repositório publicado na lista de repositórios conhecidos.

```
$ uaipkg upgrade repository all
```

Uma vez obtidas as informações de todos os recursos existentes nos repositórios conhecidos, os usuários podem localizá-los e realizar a instalação dos que lhes são interessantes. Com base nas configurações apresentadas, o usuário Carlos está isolado dos demais usuários, logo ele apenas fornece seus recursos aos outros. Neste cenário, Alice fará uma busca pelos recursos de Bob e Carlos, e após encontrá-los efetuará a instalação de ambos em sua área de trabalho. Já Bob, buscará os recursos *RecursoAlice1.pdf* e *RecuroCarlos1.odt*, presentes nos repositórios de Alice e Carlos, respectivamente, e após localizá-los os instalará em sua área de trabalho.

Alice realiza a pesquisa dos recursos em todos os repositórios conhecidos executando as instruções abaixo:

```
$ uaipkg search package all RecursoBob1
$ uaipkg search package all RecuroCarlos1
```

Bob realiza a pesquisa individualizada do recurso em cada repositório executando as instruções abaixo:

```
$ uaipkg search package AliceRepoLocal RecursoAlice1
$ uaipkg search package CarlosRepoLocal RecuroCarlos1
```

Os recursos pesquisados serão localizados devido a existência de todos nos repositórios envolvidos. Com base no resultado de cada pesquisa os usuários realizam a instalação dos recursos em suas respectivas áreas de trabalho.

Alice adiciona os recursos localizados, com as instruções abaixo:

```
$ uaipkg install package BobRepoLocal RecursoBob1 1.1.0
$ uaipkg install package CarlosRepoLocal RecursoCarlos1 1.0.0
```

Bob adiciona os recursos localizados, com as instruções abaixo:

```
$ uaipkg install package AliceRepoLocal RecursoAlice1 1.0.0
$ uaipkg install package CarlosRepoLocal RecursoCarlos1 1.0.0
```

Vale observar que a obtenção e instalação dos recursos realizadas são efetuadas nas áreas de trabalhos de cada usuário e qualquer modificação efetuada nestes recursos não são disponibilizadas automaticamente para outros usuários e, tampouco, podem ser replicadas para os repositórios de origem sem a ação explícita do proprietário do repositório. Suponhamos, que Alice tenha efetuado uma modificação no recurso *RecursoBob1.py* obtido no repositório de Bob, após as modificações um novo arquivo de metadados deve ser criado informando a nova versão do recurso, que neste caso será *1.1.1*, para publicação. A publicação ocorre no próprio repositório de Alice e, posteriormente, pode ser obtido por outros usuários que tenha acesso ao repositório, até mesmo o Bob. A instrução abaixo representa a publicação do recurso modificado no repositório de Alice.

```
$ uaipkg publish package AliceRepoLocal RecursoBob1.json
```

Para obter a nova versão modificada do recurso, o usuário Bob deve atualizar as informações conhecidas do repositório de Alice executando a instrução abaixo:

```
$ uaipkg upgrade repository AliceRepoLocal
```

Uma vez executada a instrução, os metadados referente ao recurso *RecursoBob1* são obtidos do repositório *AliceRepoLocal* e pode ser instalado e, em um segundo momento, publicado em seu repositório. As instruções abaixo apresentam os passos para instalação e publicação do recurso citado no repositório local de Bob.

```
$ uaipkg install package AliceRepoLocal RecursoBob1 1.1.1
$ uaipkg publish package BobRepoLocal RecursoBob1.json
```

A Tabela 5 apresenta todos os recursos existentes nos repositórios após as instalações apresentadas.

Tabela 5 – Recursos existentes em cada repositório após as instalações

Repositório	**Recurso**	**Versão**
AliceRepoLocal	RecursoAlice1.pdf	1.0.0
	RecursoAlice2.png	1.0.0
	RecursoBob1.py	1.1.1
BobRepoLocal	RecursoBob1.py	1.1.0
		1.1.1
CarlosRepoLocal	RecursoCarlos1.odt	1.0.0

Fonte: o autor

6 Conclusão e Trabalhos Futuros

6.1 Conclusão

Ao se expandir o compartilhamento de recursos entre os diferentes usuários um ambiente de cooperação é criado, o que possibilita que cada usuário possa focalizar em novos aspectos de um determinado recurso, visto que todas as características já conhecidas podem ser obtidas através de recursos de outros usuários que já as distribuíram. No entanto, não existem apenas aspectos positivos, o compartilhamento de recursos traz consigo um conjunto de desafios que devem ser enfrentados para que sua adesão ocorra entre os mais diversos tipos de usuários. Comumente, devido a estes desafios, somente os usuários com conhecimentos um pouco mais aprofundados fazem uso de todas as características intrínsecas ao compartilhamento, tais como: versionamento e distribuição. Neste cenário, novas arquiteturas devem ser propostas para permitir que cada vez mais usuários possam aderir o compartilhamento de recursos, independentemente de qual seja o objetivo ou o grau de conhecimento destes usuários, sendo este o principal objetivo deste trabalho.

Como esperado, o desenvolvimento de um processo de compartilhamento de recursos se apresentou como uma tarefa complexa devido as particularidades que cada tipo de usuário possui, destacando-se, principalmente, o **conhecimento computacional** e a **especificidade de cada recurso**. Para contornar a complexidade envolvendo o conhecimento computacional, uma padronização de arquivos no formato json foi proposta por se tratar de um padrão de fácil assimilação, podendo ser editado por qualquer editor de texto existente nas diversas plataformas. Além disso, um conjunto sucinto de instruções é utilizado para realizar todo o processo de compartilhamento, o que o torna de simples adoção. Já em relação a complexidade em gerir os diferentes tipos de recursos que cada usuário manipula, neste primeiro momento, a proposta objetivou em propiciar que estes sejam compartilhado sem a preocupação em resolver qualquer diferença entre uma versão e outra, portanto, ao se publicar um recurso uma nova versão deve ser gerada cabendo aos usuários resolver as possíveis diferenças entre as versões. Tal abordagem foi adotada pois boa parte dos recursos produzidos são binários o que dificulta a geração dos deltas com as modificações realizadas. Tendo em vista os objetivos propostos, a arquitetura para compartilhamento de recursos apresentada mostrou-se ser simples o suficiente para atrair novos usuários, além de permitir que várias implementações sejam realizadas sobre as interfaces exibidas no Capítulo 4.

De maneira geral, a arquitetura permite o compartilhamento de recursos entre os mais diversos tipos de usuário de maneira simples e eficiente, mas algumas melhorias precisam ser realizadas para tornar a solução mais atrativa para uma parcela de usuários.

Na subseção a seguir serão apresentados alguns aspectos que devem ser tratados em trabalhos futuros.

6.2 Trabalhos Futuros

Ainda que o propósito inicial tenha sido alcançado um conjunto de melhorias deve ser realizado para corrigir aspectos que podem ser melhorados e outros que, por definição, não foram objetivados na solução.

A implementação proposta apresenta uma série de instruções que devem ser executadas pelos usuários para que o compartilhamento possa ser estabelecido. Além disso, os arquivos de metadados dos recursos para publicação e algumas configurações são realizadas manualmente sem o auxílio de qualquer ferramenta facilitadora. Tendo em vista que grande parte dos usuários possuem familiaridade na utilização de ambientes gráficos, uma aplicação que encapsule as instruções apresentadas e, também, auxilie na configuração dos arquivos de metadados e configuração deve ser desenvolvida. Tal ferramenta definitivamente aumentará o número de utilizadores, pois tornará o processo de compartilhamento mais acessível para uma parcela maior de usuários.

Um segundo aspecto a ser desenvolvido é a possibilidade de resolver as diferenças entre os recursos de forma automática. Ainda que existam ferramentas que possam ser utilizadas para facilitar a resolução destas diferenças, vale ressaltar que uma parte considerável dos recursos compartilhados são binários o que dificulta, ou até mesmo impossibilita, tal procedimento.

Todo o processo de publicação baseia-se na existência de um repositório local onde somente o proprietário pode realizar publicação com os novos recursos ou atualizações. Além do mais, não há qualquer controle de autenticação e permissão nos repositórios, sendo que a única forma de isolamento é que o usuário seja detentor do repositório que realiza suas publicações. Tal característica, torna o compartilhamento bastante vulnerável em aspectos de segurança e, devido a isto, o protocolo de comunicação deve ser modificado para inserir uma camada de segurança que permitir o controle de autenticidade e permissionamento para acesso aos recursos existentes nos repositórios.

Referências

1 SILVA, M. A. G. T. da; FERREIRA, R. B.; LEITE, O. dos S.; MACEDO, S. da H.; SANTOS, S. L. dos. *Segurança e confiabilidade para ambientes SOHO*. dez 2017. HOLOS, [S.l.], v. 4 (2013), p. 66-76, agosto 2013. ISSN 1807-1600. Disponível em: <http://www2.ifrn.edu.br/ojs/index.php/HOLOS/article/viewFile/1042/705>. Acesso em: 15 dez. 2017. http://dx.doi.org/10.15628/holos.2013.1042. Disponível em: <http://dx.doi.org/10.15628/holos.2013.1042>. Citado na página 14.

2 MELL, P.; GRANCE, T. The nist definition of cloud computing. n. 800-145, sep 2011. Citado na página 16.

3 SYSTEMS, I. C. *Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are*. 2015. Disponível em: <https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf>. Citado na página 16.

4 Hejderup, J.; Deursen, A. v.; Gousios, G. Software ecosystem call graph for dependency management. In: *2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER)*. [S.l.: s.n.], 2018. p. 101–104. Citado na página 21.

5 Ben Seghir, N.; Kazar, O. A new framework for web service discovery in distributed environments. In: *2017 First International Conference on Embedded Distributed Systems (EDiS)*. [S.l.: s.n.], 2017. p. 1–6. Citado na página 21.

6 COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.; BLAIR, G. *Sistemas Distribuídos - 5ed: Conceitos e Projeto*. [S.l.]: Bookman Editora, 2013. ISBN 9788582600542. Citado 3 vezes nas páginas 21, 22 e 41.

7 TANENBAUM, A.; STEEN, M. van. *Distributed Systems: Principles and Paradigms*. [S.l.]: Pearson Prentice Hall, 2007. ISBN 9780132392273. Citado na página 22.

8 NUGET. *What is NuGet?*. fev 2019. Disponível em: <https://www.nuget.org/>. Acesso em: 22 fev. 2019. Disponível em: <https://www.nuget.org/>. Citado na página 27.

9 NUGET. *An introduction to NuGet*. fev 2019. Disponível em: <https://docs.microsoft.com/en-us/nuget/what-is-nuget>. Acesso em: 22 fev. 2019. Citado na página 27.

10 RAEMAEKERS, S.; DEURSEN, A. van; VISSER, J. Semantic versioning versus breaking changes: A study of the maven repository. In: *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*. [S.l.: s.n.], 2014. p. 215–224. Citado na página 27.

11 PYPI. *PyPI - The Python Package Index*. fev 2019. Disponível em: <https://pypi.python.org/pypi>. Acesso em: 22 fev. 2019. Disponível em: <https://pypi.python.org/pypi>. Citado na página 28.

12 PYPI. *Packaging and Distributing Projects*. fev 2019. Disponível em: <https://packaging.python.org/tutorials/distributing-packages/>. Acesso em: 22 fev. 2019. Disponível em: <https://packaging.python.org/tutorials/distributing-packages/>. Citado na página 28.

13 PRESSMAN, R.; MAXIM, B. *Engenharia de Software - 8ª Edição*. [S.l.]: McGraw Hill Brasil, 2016. ISBN 9788580555349. Citado na página 29.

14 HUMPHREY, W. S. *Managing the Software Process*. first. [S.l.]: Addison-Wesley Professional, 1989. (SEI series in software engineering). Citado na página 29.

15 CLEMM, G. Versioning extensions to webdav. In: *WWW 1999*. [S.l.: s.n.], 1999. Citado na página 29.

16 KUROSE, K. W. R. J. F. *Redes de Computadores e a Internet: uma abordagem top-down*. sexta. [S.l.]: Pearson Education do Brasil, 2013. Citado na página 30.

17 GIT. *Git –everything-is-local*. mai 2019. Disponível em: <https://git-scm.com/>. Acesso em: 06 mai. 2019. Disponível em: <https://git-scm.com/>. Citado na página 30.

18 SUBVERSION. *Apache™ Subversion*. mai 2019. Disponível em: <https://subversion.apache.org/>. Acesso em: 06 mai. 2019. Disponível em: <https://subversion.apache.org/>. Citado na página 31.

19 SEMVER. *Semantic Versioning 2.0.0*. mar 2019. Disponível em: <http://semver.org>. Acesso em: 10 mar. 2019. Disponível em: <http://semver.org>. Citado na página 36.

20 FOUNDATION, F. S. *GNU General Public License*. mar 2019. Disponível em: <https://www.gnu.org/licenses/gpl-3.0.en.html>. Acesso em: 03 mar. 2019. Disponível em: <https://www.gnu.org/licenses/gpl-3.0.en.html>. Citado na página 37.

21 ZHU, Y.; WU, W.; LI, D. Efficient client assignment for client-server systems. *IEEE Transactions on Network and Service Management*, v. 13, n. 4, p. 835–847, Dec 2016. ISSN 1932-4537. Citado na página 42.

22 LEHMANN, M. B.; ANTUNES, R. S.; BARCELLOS, M. P. Exploring your neighborhood: Comparing connection strategies in swarming networks through evolving graphs. In: *IEEE P2P 2013 Proceedings*. [S.l.: s.n.], 2013. p. 1–10. ISSN 2161-3559. Citado na página 42.

23 BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML: guia do usuário*. [S.l.]: Elsevier Brasil, 2006. Citado na página 45.