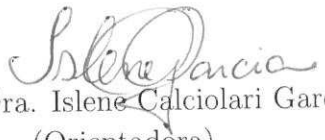
Busca e Compartilhamento de Componentes de *Software* em redes *Peer-to-Peer*

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Marcílio da Silva Oliveira e aprovada pela Banca Examinadora.

Campinas, 18 de janeiro de 2008.



Profa. Dra. Islene Calciolari Garcia
(Orientadora)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Júlia Milani Rodrigues – CRB8a / 2116

Oliveira, Marcílio da Silva
OL4b Busca e compartilhamento de componentes de software em redes peer-to-peer / Marcílio da Silva Oliveira -- Campinas, [S.P. :s.n.], 2007.

Orientador : Islene Calciolari Garcia
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Componentes de software. 2. Software compartilhado. 3. Software – Reutilização. 4. Arquitetura de redes de computador. I. Garcia, Islene Calciolari. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: Search and sharing software components in peer-to-peer networks.

Palavras-chave em inglês (Keywords): 1. Component software. 2. Software for sharing. 3. Software – Reusability. 4. Computer network architecture.

Área de concentração: Sistemas Distribuídos

Titulação: Mestre em Ciência da Computação

Banca examinadora: Prof. Dr. André Santanche (UNIFACS)
Prof. Dr. Edmundo Roberto Mauro Madeira (IC-UNICAMP)
Prof. Dr. Ricardo Anido (IC-UNICAMP)

Data da defesa: 15-12-2006

Programa de Pós-Graduação: Mestrado em Ciência da Computação

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 15 de dezembro de 2006, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. André Santanchè
UNIFACS.

Prof. Dr. Edmundo Roberto Mauro Madeira
IC – UNICAMP.

Profa. Dra. Islene Calciolari Garcia
IC – UNICAMP.

Instituto de Computação
Universidade Estadual de Campinas

Busca e Compartilhamento de Componentes de *Software* em redes *Peer-to-Peer*

Marcílio da Silva Oliveira

Dezembro de 2006

Banca Examinadora:

- Profa. Dra. Islene Calciolari Garcia (Orientadora)
- Prof. Dr. André Santanchè
  Departamento de Ciências Exatas – UNIFACS
- Prof. Dr. Edmundo Roberto Mauro Madeira
  Instituto de Computação – UNICAMP
- Prof. Dr. Ricardo de Oliveira Anido (Suplente)
  Instituto de Computação – UNICAMP

Para a inesquecível
Vovó Geralda.

Resumo

Arquiteturas *Peer-to-Peer* (P2P) têm sido uma alternativa bastante atraente para tornar a Internet mais acessível. Desde programas de compartilhamento de conteúdo a sistemas de comunicação vêm utilizando esta abordagem de arquitetura e dando importantes contribuições para melhorias nas tecnologias e metodologias relacionadas às redes P2P.

Neste contexto, o presente trabalho apresenta a definição da arquitetura de uma rede P2P para distribuição e compartilhamento de componentes de *software*, visando montar uma rede descentralizada, na qual qualquer participante possa se conectar, fornecer e buscar por componentes de *software*.

Os componentes de *software* são módulos independentes, com interfaces bem definidas, que podem ser reutilizados em diferentes situações. O reuso de *software* vem se destacando como grande promessa para aumentar a produtividade no desenvolvimento de sistemas. A aplicação do reuso de *software* se concentra principalmente na reutilização e integração de partes prontas e previamente testadas.

Este trabalho apresenta também a concepção de um modelo de repositório para armazenamento de componentes de *software*. Estes repositórios podem se comunicar através do protocolo de rede desenvolvido, montando uma rede de compartilhamento P2P.

Através da construção da arquitetura, do mecanismo de busca e modelagem dos repositórios, propomos aqui a estrutura fundamental para a criação de redes e sub-redes independentes, visando compartilhar componentes de *software* entre grupos de pesquisa, universidades, desenvolvedores e empresas. Esta arquitetura constitui a estrutura de distribuição da Rede de Compartilhamento de Componentes de *Software* (RCCS). A RCCS é um projeto de construção de uma rede pública para compartilhamento de componentes, desenvolvido no Laboratório de Inovação Ci&T/Unicamp.

Abstract

Peer-to-peer (P2P) architectures have become a very attractive alternative to make the Internet more accessible. This approach has been widely used, from content sharing programs through communications systems, which has provided the P2P networks with considerable improvements in the technology and methodologies related.

In this context, the current work presents the architecture definition of a decentralized P2P network for distribution and sharing of software components in which any participant can get connected, offer and search for software components.

Software components are interdependent modules with well-defined interfaces that can be reused in different situations. Software reuse has been considered the big boom when it comes to productivity in software development. Reusing is related to the integration of ready and previously tested pieces of software.

This work also presents the conception of a repository model for storing software components. These repositories can communicate with one another using the custom built network protocol, resulting in a strong and reliable P2P sharing network.

Through the architecture definition, the search mechanism and the modeling of the repositories, we propose the fundamental structure for the creation of independent network and sub-networks so as to share software components between research groups, universities and developers. Such architecture constitutes the distribution structure of the Software Component Sharing Network. The project focused on developing this public network for sharing components has been hosted at the Ci&T/Unicamp Innovation Lab.

Agradecimentos

Acima de tudo, agradeço a Deus, por não me deixar desistir nunca. Por reservar pra mim as melhores e maiores alegrias. Por me presentear com uma família especial e grandes amigos, e por nunca me deixar sozinho.

Gostaria de agradecer a minha família. Obrigado a meu pai Mauro e minha mãe Fátima. Vocês sempre foram e sempre serão minha fonte de inspiração. Obrigado ao Maurinho, meu irmão e melhor amigo. Como ninguém, acompanhou de perto minha luta e, agora, toda minha felicidade, e com quem posso sempre contar. Obrigado às meninas dos meus olhos, minhas irmãs Nane e Mari, sempre com palavras bonitas de carinho e incentivo. Quero agradecer aos meus tios e primos, especialmente ao tio Sebastião e tia Elza, que me viram crescer e que, de longe ou de perto, estão presentes em todas as etapas da minha vida. E um abraço especial ao Sávio.

Agradeço à Carol, por todo carinho e companhia. Sua presença e sua força foram muito importantes para mim durante todo este período.

Agradeço especialmente à Islene, que acreditou no projeto e me acolheu como seu orientado, mesmo sob situações especiais, com seu apoio irrestrito e enorme serenidade. Obrigado por sua disposição a ensinar e por sua coragem de aprender. Agradeço também ao Amin, por todo o apoio e orientação, principalmente pelos direcionamentos no início do projeto.

Gostaria de agradecer também aos professores Edmundo Madeira e Ricardo Anido, por suas contribuições valiosas na defesa da minha proposta de mestrado.

Obrigado à equipe do laboratório Ci&T pelo apoio desde o início, principalmente ao Zeh, Wanderley e Vessoni, que participaram da idealização da RCCS, me ajudaram muito e com os quais eu gostaria de comemorar. Sem vocês, provavelmente este projeto não teria se concretizado.

Agradeço ao pessoal da DigitalAssets pelo companheirismo e pelo incentivo. Valeu Kleber, pela ajuda de sempre. Valeu também Jonas, Ventura e toda a galera que está com a gente. Obrigado por proporcionarem um ótimo ambiente de trabalho. Esta nova jornada está só começando.

Não posso esquecer a galera do bem, os meus grandes amigos de Lu, Bruno e Laran-

jinha. Aconteça o que acontecer, vocês nunca estarão longe, saibam disso!

Quero agradecer ao pessoal da UFV. Foi lá que tudo começou. Lá ficaram grandes mestres e grandes amigos. Obrigado especialmente ao Prof. Mauro Nacif, pessoa com quem aprendi muito e a quem possuo profunda admiração. Obrigado também ao Prof. Zé Luis, conselheiro sempre presente e que me apoiou no sonho deste mestrado.

Muito obrigado a todos os amigos que fiz aqui no IC e que tornaram este período mais divertido e tranqüilo. Obrigado ao Ariel e à Vera, com quem eu pude sempre contar nos momentos emergenciais, mesmo em diferentes contextos.

E, por fim, faço questão de relembrar do meu grande e eterno amigo, o Zé Dobra, que viveu o suficiente para nos deixar lições inesquecíveis, para fazer grandes amigos e construir uma família especial. Ó Deus Bom!!!

É bem provável que eu tenha esquecido diversos nomes, e é aqui que eu me desculpo! Para todos que estiveram comigo, saibam que não me esqueci, mas ao final de uma dissertação a memória pode estar me pregando algumas peças! Recebam, com todo carinho, o meu muito obrigado!

"Penso que tudo aquilo que tem vida, beleza e valor
torna-se mais vivo, belo e precioso se puder ser compartilhado.
Por isso, ou para isso, decidi escrever."

– Iara L. Camaratta Anton

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Conceitualmente, *Peer-to-Peer* (P2P) é uma alternativa aos modelos computacionais do tipo cliente-servidor. O termo P2P se refere a uma classe de sistemas e aplicações que utilizam recursos distribuídos para realizar funções críticas de maneira descentralizada [MKL+03]. Através desta tecnologia, um sistema pode acessar diretamente recursos disponibilizados por outro, independente de terceiros atuando como servidores centrais.

Várias aplicações P2P podem ser vislumbradas, como processamento de dados, sistema de compartilhamento de arquivos ou algum outro tipo de aplicação distribuída. O primeiro passo para qualquer uma destas aplicações é permitir às maquinas saberem a respeito da existência de outras na rede. Em outras palavras, os participantes da rede precisam saber: "Quem, em toda a rede, gostaria de cooperar comigo?" Este primeiro passo, segundo [HBDL99], é conhecido como *resource discovery*.

Com a crescente utilização de aplicações de compartilhamento de conteúdo na Internet, o problema de *resource discovery* vem se tornando mais importante no contexto de redes P2P. Em redes P2P, em algum momento o ponto deve ser conhecido e conhecer outro ponto da rede, uma vez que a comunicação é feita diretamente, sem a intervenção de um servidor central.

No contexto deste trabalho, *resource discovery* seria o processo de descoberta de repositórios que disponibilizam componentes de *software*. Um dos objetivos deste trabalho é apresentar uma arquitetura P2P para distribuição e compartilhamento de componentes de *software*, visando resolver o problema de descoberta de repositórios e busca por componentes em uma rede distribuída.

Componentes são módulos de *software* construídos, testados e reutilizáveis que oferecem um conjunto de serviços através de uma ou mais interfaces bem definidas, sendo necessário um estudo cauteloso para viabilizar o armazenamento de todas as suas informações relevantes. Desta forma, sua representação não é simples e a modelagem criada

e utilizada para definição do protocolo foi desenvolvida visando oferecer uma solução mais completa possível.

Outro aspecto abordado é a forma que os objetos (componentes) compartilhados são armazenados e representados localmente, assim como o formato em que são transmitidos. Faz-se necessária a modelagem do repositório de componentes e criação de um protocolo de comunicação entre os pontos, baseados nas informações contidas sobre os objetos que trafegarão na rede.

Através de uma rede que implemente um protocolo P2P de busca, um desenvolvedor poderá, por exemplo, procurar por componentes de *software* para o desenvolvimento de um projeto. E também poderá compartilhar os componentes por ele criados, para que outros programadores utilizem. Várias são as aplicações para compartilhamento de conteúdo na Internet, como Napster [Ku02] [NAP06], Gnutella [Rip01] [GNU06], Kazaa [KAZ06] e eMule [eMu06], porém a idéia é oferecer uma solução aberta específica para compartilhamento de componentes de *software*.

1.1 Objetivos

Este projeto foi focado na concepção e modelagem de um repositório de componentes de *software* e desenvolvimento da arquitetura e protocolo de distribuição e compartilhamento. Desta forma, dois principais objetivos traçados são:

- ***Concepção do modelo de repositório para componentes de software.*** A modelagem do repositório é de vital importância para o compartilhamento de componentes. Deve-se adotar uma modelagem seguindo os requisitos iniciais de "padrões abertos, interoperabilidade e flexibilidade". Uma vez definido o modelo, toda a documentação gerada deve ser seguida para implementação de um repositório de compartilhamento para redes P2P.

- ***O desenvolvimento da arquitetura e mecanismo de resource discovery (descoberta de repositórios), aplicado ao compartilhamento de componentes de software.*** A arquitetura deve ser descentralizada, sem nenhum ponto totalmente responsável pela rede. É possível a existência de pontos de referência, mas a ausência destes não pode causar a queda da rede. O algoritmo de busca deve ser relativamente simples, descentralizado, e atento às características particulares de um componente de *software*, uma vez que um componente de *software* é sabidamente mais complexo que outros tipos de informações e materiais compartilhados em redes populares de compartilhamento como Napster, Gnutella, Kazaa, eMule.

1.2 Motivação e desafios

Frente à necessidade de se contruir *software* com qualidade em períodos cada vez mais curtos, o reuso de *software* pode ser visto como uma forte tendência para solucionar alguns problemas. Reuso pode ser considerado uma denominação genérica, envolvendo uma variedade de técnicas utilizadas desde as etapas de *design* até a implementação. O principal objetivo é evitar o retrabalho no desenvolvimento de um novo projeto, capitalizando trabalhos anteriores, fazendo com que as soluções já desenvolvidas sejam imediatamente implementadas em novos contextos. Desta forma, tem-se melhores produtos em um menor intervalo de tempo, com redução nos custos de manutenção pois as partes do sistema são independetes e facilitando a inclusão de novas funcionalidades. Além disso, a qualidade aumenta devido ao reuso de componentes previamente bem testados [DW99] [JGJ97].

Uma das principais motivações na utilização de desenvolvimento baseado em componentes é exatamente a premissa de reutilização. Isto implica no reaproveitamento de partes já desenvolvidas. Esta abordagem, por reduzir o tempo de desenvolvimento e aumentar a qualidade do produto final é, de fato, bastante atraente [Kru92].

Para que seja possível reusar algo, é necessário saber onde está. É sob esta perspectiva que este trabalho foi desenvolvido. O principal desafio é desenvolver uma arquitetura distribuída visando o compartilhamento de componentes de *software*, envolvendo também a concepção do repositório, dos serviços de compartilhamento, mecanismo de *resource discovery* e implementação de um padrão para representação e transmissão dos componentes na rede.

1.3 A Rede de Compartilhamento de Componentes de *Software*

A RCCS é uma rede aberta, baseada em padrões abertos para comunicação, arquitetura e modelagem dos componentes. A idéia é que qualquer entidade possa participar como fornecedora de componentes de *software* ou apenas consumidora dos componentes distribuídos na rede. A rede deve funcionar como um repositório ou biblioteca de componentes, porém de forma totalmente distribuída e independente de entidades centrais [OGN05a].

Devido ao grande crescimento das redes de interconexão de computadores para a troca de informações e serviços, várias aplicações têm exigido mecanismos mais otimizados para difusão de mensagens. As soluções atuais têm sido encontradas em algoritmos distribuídos que resolvem problemas clássicos como o de *resource discovery*. Porém, várias são as propostas que dificilmente são aplicadas na prática, devido à grande complexidade de implementação, como a proposta em [AACY04]. Desta forma, um dos principais focos no

desenvolvimento da solução de *resource discovery* e da busca é a simplicidade e eficiência com as quais essa solução possa ser implementada.

1.3.1 Requisitos iniciais da RCCS

Para atingir seu propósito, a rede foi idealizada pela equipe de pesquisadores do Laboratório de Inovação em *Software* Ci&T/Unicamp considerando-se algumas requisitos fundamentais. E é em torno destes requisitos que as pesquisas e o desenvolvimento se realizam. A seguir, são listados esses aspectos pré-definidos para a RCCS:

- ***Utilização de padrões.*** A rede deve ser modelada seguindo padrões abertos [OMG06] [RAS06] [IBM04] para facilitar o surgimento de novas implementações e participações na rede.
- ***Interoperabilidade.*** É importante que a rede não esteja restrita a uma plataforma de desenvolvimento. Como a comunicação deve ser possível entre pontos com diferentes implementações, é muito importante que a proposta proporcione interoperabilidade.
- ***Repositórios distribuídos e independência de entidade central.*** Não deve existir um conjunto de repositórios centrais onde os componentes são armazenados. Estes repositórios estarão de forma distribuída, descentralizando os recursos da rede. Deste modo, a RCCS não deve ser dependente de nenhuma entidade específica e a queda de um ponto (ou simplesmente sua ausência na rede) não torne a rede indisponível momentaneamente.
- ***Escalabilidade.*** Tendo em vista que a rede é aberta, é imprescindível que sua arquitetura suporte escalabilidade em relação ao número de participantes.

Considerando-se as características apresentadas como "independência de entidade central, repositórios distribuídos e escalabilidade", a RCCS foi concebida com arquitetura P2P. Neste cenário, foi necessário um estudo sobre as principais arquiteturas P2P, com o objetivo de obter em cada uma delas aspectos positivos que pudessem ser utilizados na nova arquitetura proposta.

1.3.2 Proposta de modelagem

Um dos aspectos mais importantes é a modelagem da rede. Como dito anteriormente, trata-se de uma rede P2P, sem um servidor central. Mas isto não é suficiente. É necessário identificarmos a arquitetura, determinarmos como os pontos da rede se comunicarão, qual

a tecnologia utilizada pra fazer a troca de informações e como estas informações serão transmitidas na RCCS.

A RCCS pretende atingir diversas categorias de entidades relacionadas a desenvolvimento e utilização de componentes de *software*. A figura 1.1 ilustra a forma com que a rede foi idealizada, com a participação de universidades e centros de pesquisa, órgãos governamentais, *hubs* de distribuição de *software* livre, fábricas de componentes, dentre outras entidades interessadas em compartilhar ou simplesmente buscar componentes na RCCS.

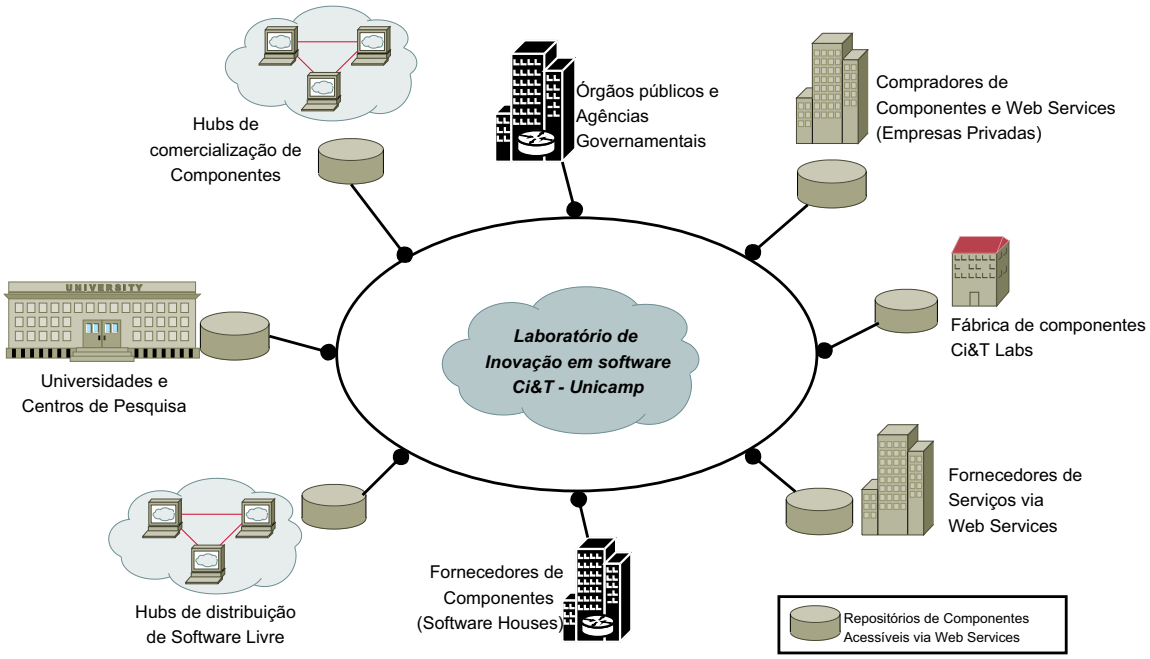


Figura 1.1: Arquitetura da RCCS.

Em cada repositório de componentes deve haver um servidor onde os serviços fiquem disponíveis através de serviços públicos (utilizando tecnologia de Web Services), e a comunicação entre os pontos da rede é feita através do acesso ao serviço disponibilizado. Desta forma, a arquitetura permite que os pontos da rede comuniquem-se diretamente, sem a interferência ou dependência de um servidor central. A inexistência deste servidor central proporciona à rede uma característica interessante de independência de qualquer entidade. Uma vez desenvolvido o protocolo de comunicação e uma implementação de referência, qualquer grupo de usuários pode montar a rede e mantê-la em funcionamento.

A RCCS deve atuar como uma ferramenta que impulsione práticas de utilização de componentes, sobretudo facilitando o compartilhamento e a busca de componentes de *software*.

1.4 Contribuições

As principais contribuições diretas deste projeto são:

- Concepção de um modelo de representação de repositórios de componentes de *software*. Tal modelagem pode ser utilizada em repositórios distribuídos, visando o intercâmbio de conteúdo, ou em repositórios locais, para armazenamento e organização de ativos digitais.
- Desenvolvimento de um mecanismo de descoberta e reconhecimento de participantes em redes P2P. Uma proposta que possa ser implementada e utilizada em sub-redes, sem inserir complexidade no desenovlvimento.
- Definição de um protocolo de comunicação através de Web Services para troca de componentes entre diferentes repositórios. Este protocolo é independente de plataforma e é definido especificamente para tráfego de componentes de *software*, segundo a modelagem desenvolvida.

1.5 Estrutura do texto

No próximo capítulo são apresentados os conceitos fundamentais do trabalho, que tratam principalmente de arquiteturas P2P e do modelo de representação de ativos digitais. Neste capítulo são discutidas as principais arquiteturas em redes P2P, que se dividem basicamente em puras e híbridas. Além disso, também apresentamos o RAS (*Reusable Asset Specification*), que é o padrão fundamental de modelagem de ativos digitais adotado neste trabalho para confecção do repositório e do protocolo de disbribuição.

O terceiro capítulo trata da concepção do modelo de repositório. São apresentados os fatores críticos relacionados à modelagem implementada, assim como um maior detalhamento do modelo adotado. Neste capítulo é possível avaliar com mais precisão a flexibilidade e extensibilidade do modelo RAS, assim como suas principais seções de classificação de ativos digitais.

A arquitetura de distribuição proposta é apresentada no capítulo 4. Neste capítulo é detalhado o mecanismo de descoberta de repositórios e busca de componentes na rede. Também é apresentado o protocolo de compartilhamento e comunicação. Este protocolo define os serviços (Web Services) publicados para busca, detalhamento e *download* de componentes, além de estabelecer quais objetos são transmitidos na comunicação entre os pontos da rede.

No quinto capítulo encontram-se os principais resultados do projeto, que são divididos em implementações, publicações e utilizações. Além disso, neste capítulo são apresentados os trabalhos futuros e possíveis evoluções do modelo de compartilhamento proposto.

Por fim, após o quinto capítulo, encontra-se o apêndice A, no qual é disponibilizado um glossário com definições para os principais termos técnicos encontrados no texto. Vale ressaltar que os termos apresentados são definidos no contexto deste trabalho.

Capítulo 2

Fundamentação Teórica

Podemos destacar como principais temas, que nortearam o desenvolvimento deste trabalho, arquiteturas *Peer-to-Peer* e componentização. Nesta seção apresentamos alguns conceitos fundamentais sobre as linhas de pesquisas relacionadas ao trabalho desenvolvido. São apresentadas as principais arquiteturas *Peer-to-Peer* existentes, assim como suas variações. Também apresentamos o conceito de desenvolvimento baseado em componentes e o *Reusable Asset Specification* (RAS), modelo de representação de ativos digitais.

2.1 Arquiteturas *Peer-to-Peer*

A tecnologia P2P foi fortemente impulsionada por aplicações de compartilhamento e troca de mensagens bastante populares, como Napster [Ku02], Gnutella [Rip01] e ICQ [ICQ06]. Tais aplicações deixam claro a potencialidade e as possibilidades de desenvolvimento. Neste contexto, o ambiente acadêmico se mostra interessante para realizar experimentos e se beneficiar dessa tecnologia, pois pode utilizar novas ferramentas que auxiliam a troca de informações sem despender recursos adicionais [RaACS+04].

Após o surgimento de diversas aplicações de compartilhamento de conteúdo na Internet e da popularização dos sistemas de mensagens instantâneas, o assunto P2P ganhou força e tem chamado a atenção. Porém, em grande parte das características estudadas dentro do P2P, poucas são novidades. O próprio conceito das aplicações não é novo. Podemos listar alguns exemplos como telefone, redes internas de jogos, servidores FTP (*File Transfer Protocol*) interligados, que sempre utilizaram o conceito de comunicação ponto a ponto. Podemos também listar outras características já conhecidas como descentralização, escalabilidade, realização de operações desconectadas, algoritmos distribuídos (muitos dos utilizados já são aplicados em outras arquiteturas), processamento distribuído, etc. Por outro lado, P2P traz consigo novas necessidades, das quais vale ressaltar necessidades tecnológicas (promover maior performance e interoperabilidade) e necessidade de maior

segurança [MKL+03].

Junto à idéia de utilização de recursos compartilhados, as redes P2P também oferecem uma série de vantagens, como baixo custo de obtenção, pois não é necessário nenhum gasto adicional com *hardware* (aquisição de servidores), pois a rede é formada apenas pelos computadores dos usuários. Além disso, algumas arquiteturas não estão vulneráveis a um crescimento descontrolado da rede, promovendo maior escalabilidade. Outra vantagem é a utilização de recursos ociosos de outros computadores participantes da rede. Por fim, a natureza descentralizada dos sistemas P2P os protege de um tipo de falha bastante comum em sistemas centralizados, nos quais a falha do servidor central pode ser fatal para o funcionamento do sistema.

A figura 2.1 apresenta a taxonomia dos sistemas P2P dentro dos sistemas computacionais, e suas quatro principais arquiteturas.

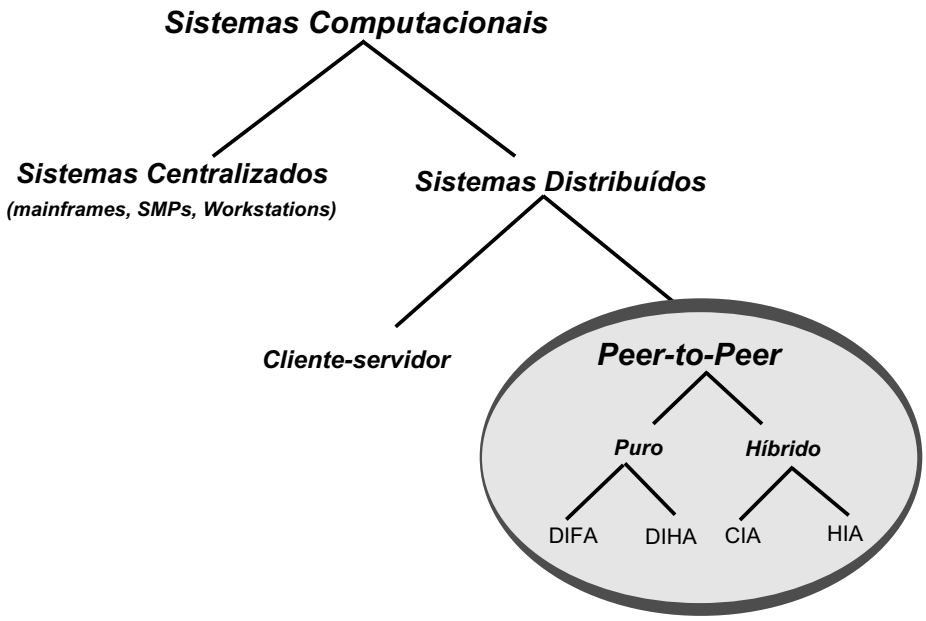


Figura 2.1: Taxonomia dos sistemas P2P [MKL+03].

Um sistema P2P pode ter sua arquitetura construída basicamente de duas formas: puro ou híbrido. Várias arquiteturas podem ser encontradas na literatura, porém, conforme visto em [WMS02], conceitualmente são resultado de uma modificação ou melhoria destas duas arquiteturas principais, que são apresentadas nos tópicos a seguir.

2.1.1 Arquitetura P2P Pura

Arquiteturas P2P puras formam redes totalmente descentralizadas, onde não existe nenhum ponto central de dependência, seja para busca ou para armazenamento de conteúdo.

Desta forma, a queda de qualquer ponto na rede não compromete o seu funcionamento, exceto pelos recursos que seriam oferecidos pelo ponto ausente.

Os principais exemplos de arquitetura pura são o *Distributed Indexing with Flooding Architecture* (DIFA) e *Distributed Indexing with Hashing Architecture* (DIHA), e a principal diferença entre eles é o algoritmo de busca utilizado. O DIFA utiliza algoritmos de busca baseados em *flooding*, ou seja, em busca por exaustão. E o DIHA utiliza realiza buscas baseadas em indexação por tabelas *hashing*. Maiores detalhes são apresentados a seguir.

Arquitetura DIFA - *Distributed Indexing with Flooding Architecture*

Em uma arquitetura P2P pura não existem servidores centrais ou supernós que sejam responsáveis por encontrar ou armazenar o endereço dos participantes da rede. A busca é realizada de forma totalmente descentralizada, segundo algum algoritmo distribuído utilizado. A figura 2.2 ilustra a busca em uma rede P2P com arquitetura pura DIFA.

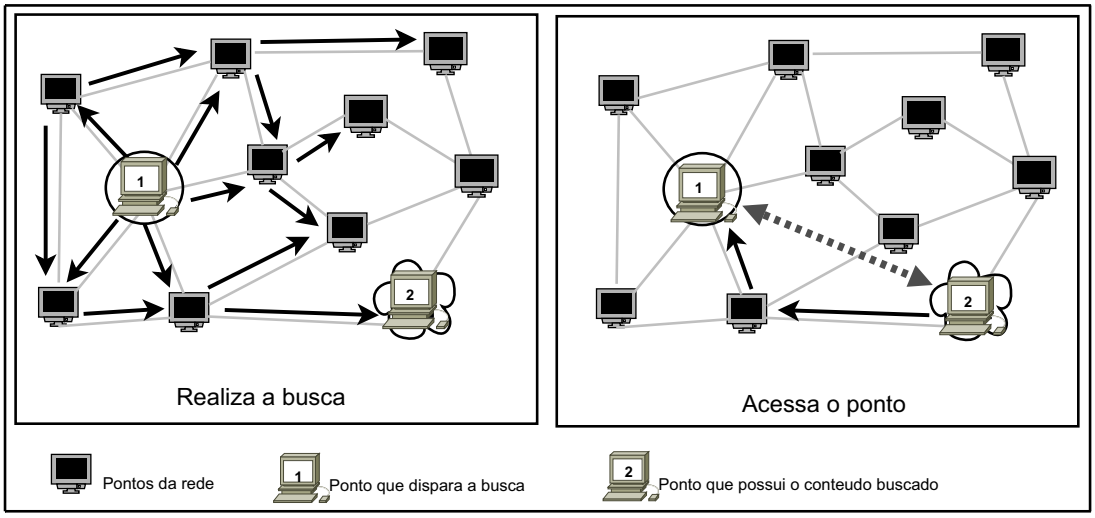


Figura 2.2: Busca e comunicação em uma rede P2P Pura (DIFA).

Um exemplo de sistema P2P com arquitetura DIFA é o Gnutella [Rip01]. Neste mecanismo de busca, um ponto da rede dispara a busca para os seus vizinhos, que é novamente repassada para os vizinhos dos vizinhos e assim em diante. Geralmente é estabelecido um tempo de vida ou um número de "saltos" para a busca, e isso pode fazer com que algum ponto da rede não seja visitado, como acontece na figura 2.2. Porém, o grande problema de arquiteturas que utilizam algoritmos de *flooding* é a sobrecarga causada na rede. Além disso, é difícil controlar onde a busca está sendo realizada, uma vez que ela é disparada diversas vezes por outros pontos da rede que recebem a busca

original.

Arquitetura DIHA - *Distributed Indexing with Hashing Architecture*

Arquiteturas DIHA são mais modernas e mais complexas. Utilizam sistema de indexação por tabelas *hashing* para distribuição e busca de conteúdo. Seu principal exemplo é o Chord [KR04].

A figura 2.3 ilustra um exemplo bastante simples de parte da busca em uma rede P2P com arquitetura DIHA.

Inicialmente, os identificadores dos pontos da rede são estabelecido através da aplicação de uma função *hash* no seu endereço IP. A rede é estruturada em forma de anel. Na busca por um nó com identificador = **X**, a tabela de identificadores é varrida até encontrar o valor do IP tal que o *hash(X)=IP*. Ou seja, tem-se os valores iniciais dos IPs de cada participante da rede, criando-se uma tabela com o mapeamento de identificadores "chave/IP"para os pontos. Vale observar que é possível que algum ponto da rede não exista. Na figura 2.3, por exemplo, não existem valores de IPs que gerem uma chave de valor `id = 3` , e este ponto fica "vago"na rede.

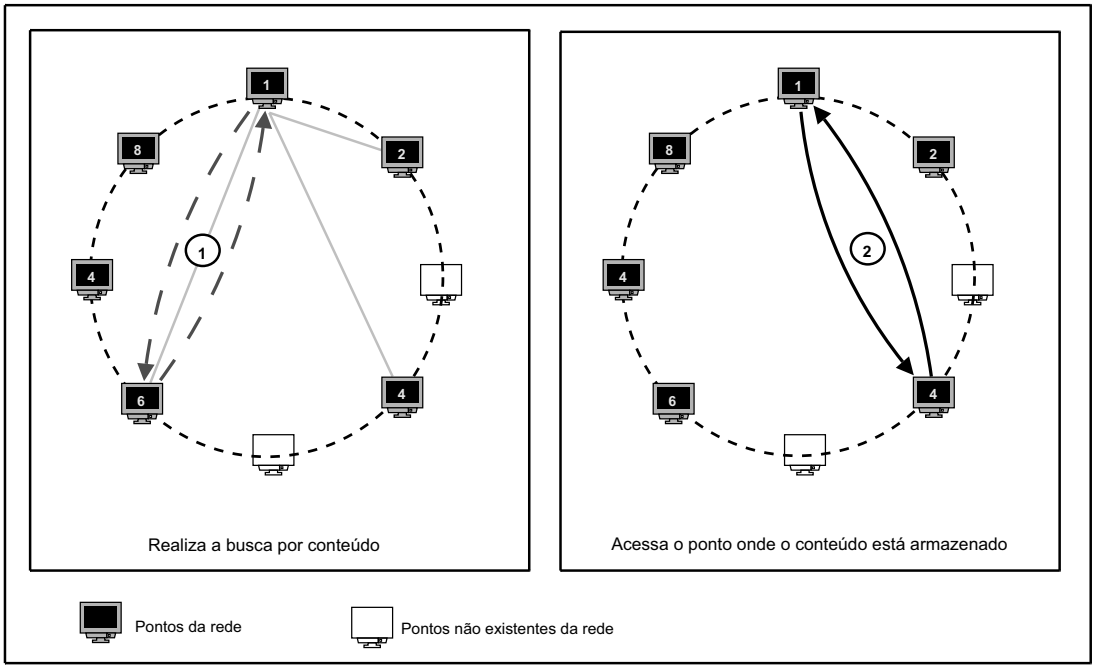


Figura 2.3: Arquitetura de uma rede P2P Pura (DIHA), utilizando Chord.

A busca por conteúdo é realizada de forma semelhante. Neste mesmo exemplo, o ponto da rede com `id=1` busca por um conteúdo que, pela função de indexação *hash("conteudo")*, tem valor **"6"**. Em um primeiro passo, é realizada a consulta ao ponto com `id = 6`, que

indica a localização do conteúdo. Em um segundo passo, o ponto que armazena o conteúdo é acessado.

Novamente, como o identificador de cada ponto da rede é determinado pela função *hash*, é importante ressaltar que podem existir diversos identificadores da rede que não possuam nenhum ponto associado, como ilustrado para os pontos 3 e 5 da figura 2.3.

O espaço de endereçamento do Chord aparece como um *loop* e os IPs dos computadores são utilizados para criar as chaves de identificação [KR04]. O que torna-se mais complexo quando o nó da rede não possui um IP fixo ou simplesmente não deseja armazenar nenhuma referência ou conteúdo, apenas buscar na rede.

2.1.2 Arquitetura P2P Híbrida

Em arquiteturas híbridas de compartilhamento de conteúdo existem um ou mais servidores centrais para realizar busca de conteúdo na rede. Ou seja, a busca acontece de maneira centralizada, cliente servidor. E a comunicação entre os pontos para *download* do conteúdo desejado é feita de forma direta, ponto a ponto.

A principal arquitetura híbrida é a *Centralized Indexing Architecure* (CIA), apresentada no próximo tópico. E uma importante variação desta arquitetura é o modelo *Hierarchic Indexing Architecure* (HIA).

Arquitetura CIA - *Centralized Indexing Architecture*

A característica principal de um sistema P2P híbrido é a existência de um ponto central, que é utilizado para busca de outros elementos (nós) da rede. A existência deste servidor central para busca não descaracteriza o sistema como P2P, pois, uma vez encontrado um recurso, a comunicação é feita diretamente entre os pontos, sem a necessidade do servidor.

A figura 2.4 ilustra a busca e a comunicação P2P em uma rede com arquitetura híbrida CIA.

Pela figura, percebe-se que a busca é realizada em um servidor central e, a partir do resultado obtido, um ponto se conecta diretamente ao outro ponto, caracterizando a conexão P2P.

Um exemplo de utilização da arquitetura CIA é o Napster [Ku02], que é um sistema de compartilhamento de música na Internet que foi bastante utilizado e hoje serve como uma forte referência para este tipo de rede.

Podemos observar que, uma vez que o processo de busca é centralizado, existe um ponto de vulnerabilidade, do qual depende toda a rede. Quando o servidor de busca estiver fora de funcionamento por algum motivo, os usuários não conseguirão encontrar os outros participantes, e consequentemente, conteúdo da rede, uma vez que a busca por conteúdo é feita em um único ponto e não há base de conhecimento local.

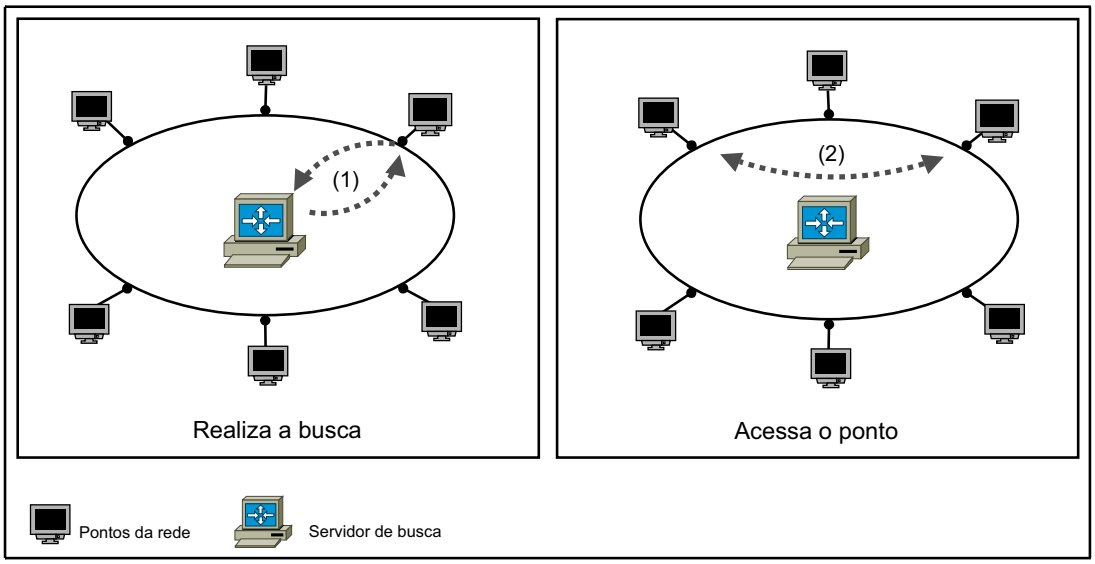


Figura 2.4: Busca e comunicação em uma rede P2P híbrida (CIA).

Arquitetura HIA - *Hierarchic Indexing Architecure*

O modelo hierárquico é uma derivação do modelo CIA. Cada ponto da rede possui um super-nó para realizar a busca, porém, quando um super-nó está indisponível, seus pontos realizam busca em outro. Desta forma, a ausência de um super-nó não indisponibiliza a busca na rede, pois existem outros responsáveis por respondê-las.

A figura 2.5 ilustra o mecanismo de busca de redes P2P hierárquicas.

O principal exemplo de redes P2P hierárquicas é o Kazaa [KAZ06], sistema bastante popular para compartilhamento de arquivos na Internet.

São utilizados algoritmos para identificar potenciais super-nós na rede. Geralmente baseados na capacidade de armazenamento e conexão dos participantes. Em aplicações como o Kazaa, estes algoritmos "elegem" nós na rede para serem responsáveis por atender as requisições de busca de outros participantes.

2.2 Desenvolvimento Baseado em Componentes

Em todo mundo, as áreas de TI buscam cada vez mais agilidade, economia e inovação tecnológica, que constituem o tripé da competitividade da indústria de *software*. Essa tendência foi destaque nas conferências do Gartner[1], um dos maiores grupos de desenvolvimento de pesquisa e análises de TI no mundo, realizada no final de março de 2005 e que reuniu especialistas de diferentes países. E dentro de inúmeras tecnologias e soluções

[1] `http://www.gartner.com/2_events/conferences/2005/brl221/brl221.jsp`

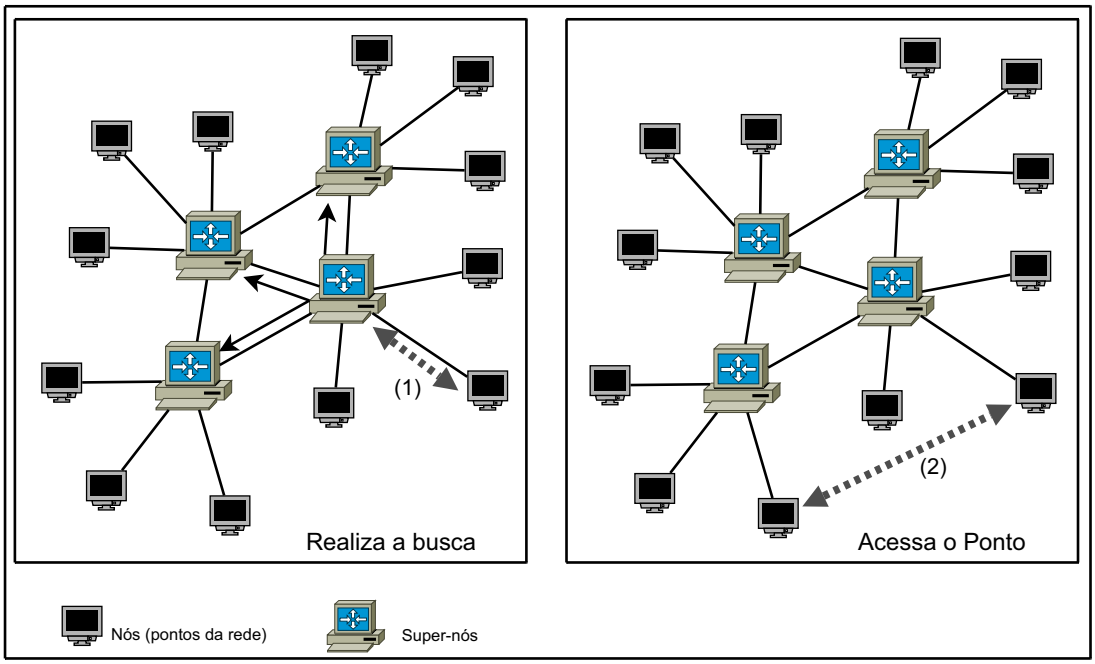


Figura 2.5: Busca e comunicação em uma rede P2P híbrida (HIA).

que aparecem todos os dias, a última edição do evento do Gartner enfocou a Integração de Aplicativos e Serviços na Web, demonstrando que grande parte dos especialistas apostam no reuso e em novas abordagens de construção como arquitetura orientada a serviços - SOA (*Service-Oriented Architecture*) e Web Services, buscando ganho de produtividade e otimização de investimentos.

Como citado anteriormente, componentes são módulos de *software* construídos, ptestados e reutilizáveis que oferecem um conjunto de serviços através de uma interface bem definida. Representam unidades funcionais auto-suficientes que se comunicam umas com as outras através de interfaces padronizadas e podem ser combinadas para formar sistemas complexos como peças de um quebra-cabeça ou, numa clássica analogia, como os inúmeros componentes em uma linha de montagem de automóveis.

Sabidamente, o principal objetivo do reuso de *software* é evitar o retrabalho no desenvolvimento de um novo projeto, capitalizando trabalhos anteriores, fazendo com que as soluções já desenvolvidas sejam implementadas em novos contextos. Desta forma, tem-se melhores produtos em um menor intervalo de tempo, com redução nos custos de manutenção pois as partes do sistema são independentes e facilitam a inclusão de novas funcionalidades. Além disso, a qualidade aumenta devido ao reuso de componentes previamente bem testados [DW99], [JGJ97].

Uma das principais motivações na utilização de desenvolvimento baseado em compo-

nentes é exatamente a premissa de reutilização. Esta abordagem, por reduzir o tempo de desenvolvimento e aumentar a qualidade do produto final é, de fato, bastante atraente a empresas de tecnologias [Kru92] e, segundo Weber et al. [WN02], a necessidade pela garantia de qualidade no desenvolvimento de *software* aumentou exponencialmente em alguns anos. Muitos apostam na solução baseada em reuso como uma das soluções para se atingir níveis mais elevados de qualidade.

Outro aspecto a ser citado é que, em um desenvolvimento baseado em componentes, novas funcionalidades podem ser facilmente adicionadas através da extensão de componentes pré-existentes ou da criação de novos componentes.

O desenvolvimento baseado em componentes é uma alternativa entre o desenvolvimento de aplicações sob-encomenda (a partir do zero) e a busca de soluções através da implantação de grandes pacotes de *software*. Se no primeiro caso temos um elevadíssimo custo de desenvolvimento associado ao grande risco de insucesso, no segundo caso, temos grandes (e em alguns casos intermináveis) esforços de implantação de soluções engessadas que raramente oferecem algum tipo de diferencial competitivo à empresa (visto que a solução de prateleira está sempre disponível para o concorrente). A terceira via da componentização permite a construção rápida e flexível de soluções customizadas através de componentes de *software* pré-construídos.

A utilização de partes prontas na construção de novas aplicações implica na existência de uma biblioteca de componentes prontos. As empresas podem organizar essas bibliotecas de diversas formas:

- Diretório de rede compartilhado
- Ferramentas de controle de versão
- Ferramentas específicas para administração de componentes

Diretórios de rede compartilhados, embora sejam uma opção para armazenamento e distribuição de componentes, deixam a desejar por diversos aspectos: ausência de controle de versão das informações existentes, dificuldade para se encontrar componentes (carência de mecanismos de busca eficientes), inexistência de ferramentas de colaboração que favoreçam a reutilização de componentes entre diversos outros. Já as ferramentas de controle de versão oferecem alguns pontos positivos, mas ainda são insuficientes na tarefa de promover a reutilização de componentes, uma vez que geralmente não fornecem mecanismos de busca, descrição e para rastreamento da utilização dos componentes armazenados. Então, a utilização de uma ferramenta específica para administração de componentes é altamente recomendada para empresas que buscam aumento de produtividade através da reutilização de componentes.

Neste contexto, a relação entre empresas e centros de pesquisa na busca de pesquisas sobre novas metodologias, conceitos e tecnologias, vem se intensificando, e não é raro vermos empresas investindo em pesquisas para encontrar novas soluções em relação à produtividade e qualidade de *software*, em um mercado cada vez mais competitivo através de laboratórios de pesquisa e parcerias com universidades [LAB06].

Em [MET02] foram apresentados e discutidos alguns fatores-chave em adotar um programa de reuso em empresas. Os fatores-chave são derivados de evidência empírica associada à analise de 24 projetos realizados de 1994 até 1997. No trabalho publicado são apresentados alguns fatores históricos de grande relevância para o estudo da aplicação de componentes.

Valer ressaltar sobre a existência iniciativas privadas e governamentais no sentido de incentivar a utilização e distribuição de componentes de *softwares*. Este fato pode ser observado através de um projeto nacional financiado pelo governo brasileiro, o CompGOV - Biblioteca Compartilhada de Componentes para e-GOV [2]. Os principais tópicos deste projeto são: (1) o desenvolvimento de um framework para reuso de *software* visando estabelecer um padrão para o desenvolvimento de componentes, (2) o desenvolvimento de um repositório de componentes e (3) desenvolvimento de um processo de certificação de componentes de *software*. Tal projeto está sendo desenvolvido por um consórcio de colaboração empresa/universidade visando a geração de um modelo bem definido de desenvolvimento, avaliação, qualidade e armazenamento de componentes e, após isso, tornar o modelo viável para fabricas de *software* que utilizam reuso no desenvolvimento de *software*.

2.3 *Reusable Asset Specification* (RAS)

O RAS é um modelo de representação de ativos digitais (assets) adotado pelo *Object Management Group* (OMG) [OMG06], e define uma forma padrão para empacotamento de ativos digitais. Ele tem como um de seus pontos positivos o fato de empacotar todos os arquivos que implementam uma solução e estruturar meta informações que definem e descrevem o seu comportamento por inteiro, possibilitando a geração de um único arquivo ".ras". O RAS surgiu de uma proposta conjunta das empresas Adaptive, Blueprint, ComponentSource, Flashline, IBM, LogicLibrary e OSTnet, afim de fomentar o mercado de ativos digitais, criando um padrão consolidado para representação de *assets*.

A figura 2.6 ilustra uma definição em alto nível de um *asset*. Ele pode ter um ponto

[2] Publicação do resultado do edital:
`http://www.finep.gov.br/fundos_setoriais/acao_transversal/resultados/resultado_Acao_transversal_Biblioteca_de_Componentes_05_2004.pdf`

de variação[3], que é o local no *asset* que pode ser customizado. As regras de utilização são as instruções descrevendo como o *asset* deve ser utilizado. Os artefatos podem estar também relacionados a diferentes contextos de utilização. Estes contextos podem ser utilizados para representar conjutos específicos de informações sobre de ativos digitais, ou sua aplicação em ferramentas específicas. Por exemplo, na geração de pacotes RAS para intercâmbio de dados entre ferramentas, uma ferramenta N pode criar um contexto para dados específicos desta ferramenta, como "contexto ferramenta N", que ao ser importado pela ferramenta M não serão importados, por não fazerem sentido no "mundo de execução da ferramenta M". Desta forma, os contextos são utilizados para "agrupar"conjunto de informações/artefatos específicos para determinadas situações (ou contextos, como o próprio nome sugere).

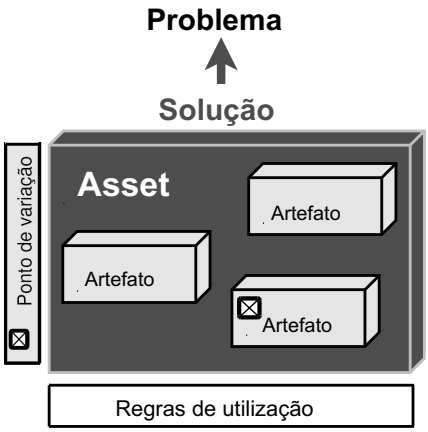


Figura 2.6: Estrutura de um asset segundo RAS.[RAS06]

Os artefatos são os produtos a para serem utilizados e reutilizados no desenvolvimento, como documento de requisitos, modelos, arquivos com código fonte, descritores, casos de teste, scripts, dentre outros. Em geral, o termo "artefato"está associado a um arquivo. O RAS define também a forma que um *asset* deve ser empacotado.

O principal objetivo é especificar a quantidade mínima de meta informações que um *asset* deve ter. Além disso, ele determina quais artefatos um *asset* possui. Possibilita também classificar estes artefatos. As meta informações ficam armazenadas em um arquivo definido como *manifest*, que é uma descrição em XML de várias características presentes no ativo representado. No pacote gerado para armazenamento e transmissão de componentes no formato RAS, o arquivo *manifest* pode ser nomeado como `rasset.xml`.

Para o entendimento do RAS, é importante entender alguns conceitos importantes relacionados à esta especificação: Core RAS e *profiles*. Core RAS representa os elementos

[3]do inglês *variability point*

fundamentais para uma especificação. Os *profiles* descrevem extensões a estes elementos fundamentais. Um profile não altera a semântica ou definição de um elemento definido pelo Core RAS, e pode ser estendido a novos *profiles*. Assim, em sua especificação o RAS fornece diferentes *profiles*. O `Default Profile` que é o conjunto de características básicas. O `Default Component Profile` é uma extensão do `Default Profile` que possui algumas características extras, relacionadas à representação de componentes. Outro profile existente é o `Default WebServices Profile`, que implementa características particulares de Web Services. Desta forma, utilizando o padrão RAS, um *asset* pode estar relacionado a um determinado profile, que pode conter características específicas para determinados tipos de *assets*, como Web Services, *patterns* e *frameworks*.

A figura 2.7 (a) ilustra o Core RAS e os *Profiles*, e 2.7 (b) mostra os relacionamentos entre o Core RAS e as implementações dos *Profiles*. O `Default Profile` é a realização do Core RAS. O `Default Component Profile` e o `Default WebServices Profile` são derivações do `Default Profile`.

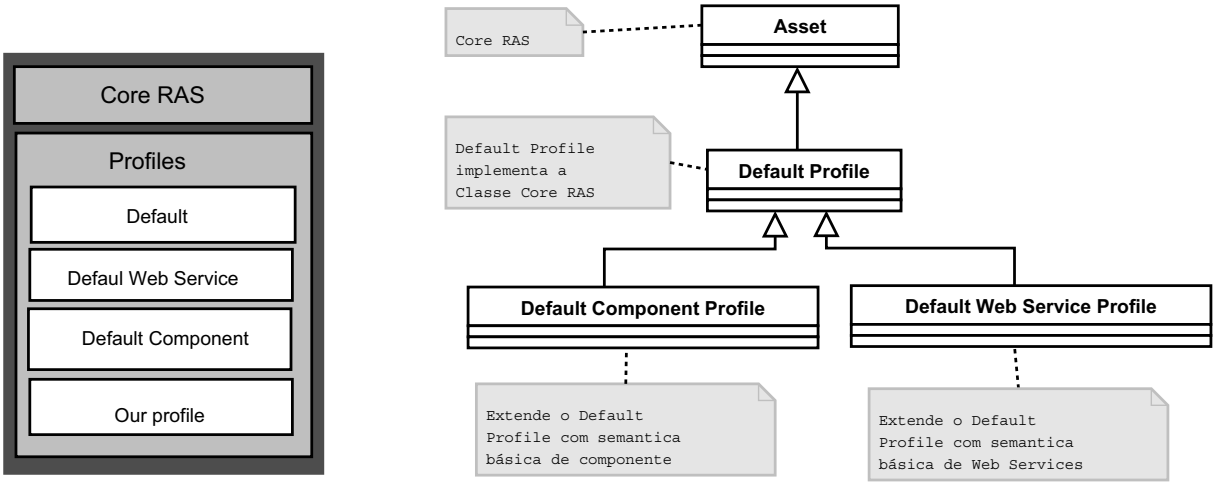


Figura 2.7: Core RAS e profiles(a). Relacionamentos do RAS profile(b).[RAS06]

O Core RAS define quatro seções principais para representação de um *asset*, incluindo `Classification, Solution, Usage` e `Related-Assets`. Um componente (ou ativo digital) é especificado por estas diferentes seções, que estão contida nos meta dados do *asset*, como mostra a figura 2.8.

Por utilizar meta informações, o RAS proporciona uma modelagem adaptável. Além de possibilitar extensibilidade através da criação de novos *Profiles*.

Vale a pena ilustrar as diferenças entre os *default profiles* existentes no RAS. Conforme informado na figura 2.7, além de implementar as classes padrão do `Default Profile`, os *profiles* implementam novas classes, específicas para diferentes tipos de *assets*. A seguir são rapidamente apresentados os dois principais *default profiles* do RAS, o `Default`

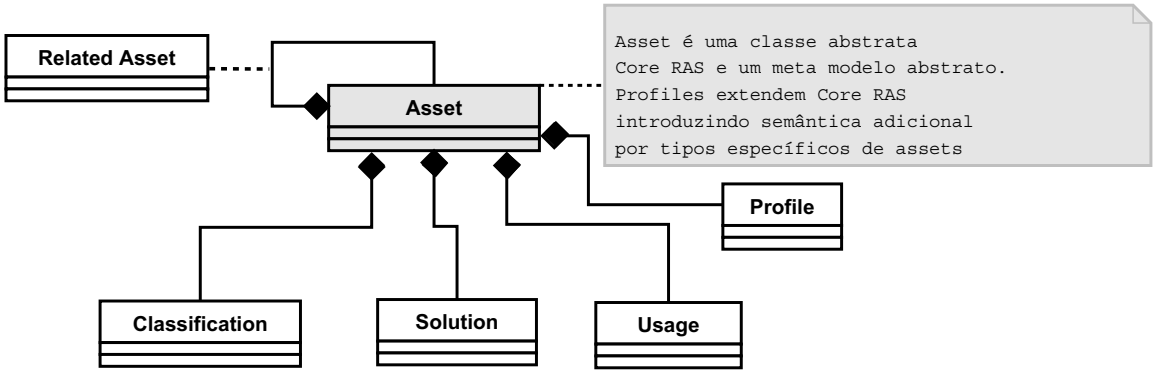


Figura 2.8: "Core RAS domain model". Seções principais.

`Component Profile` e o `Default Web Services Profile`.

A figura 2.9 apresenta o diagrama de classes do `Default Component Profile`. Este *profile* acrescenta um conjunto de classes relacionadas a componentes de *software*, principalmente ligadas às interfaces dos componentes representados. As classes inseridas neste *profile* são `InterfaceSpec, Operation, Condition, Parameter, InformationModel, Attribute` e `AssociationRole`.

A classe `InterfaceSpec` descreve uma interface do componente. Um componente pode possuir diversas interfaces, resultando em múltiplas instâncias desta classe. O atributo `name` é o nome da interface para o usuário. Esta classe está relacionada com `Operation` e `InformationModel`. Com a criação de uma instância de `InterfaceSpec` podem ser criadas/registradas uma ou mais operações.

O diagrama de classes de outro *profile*, o `Default Web Service Profile`, é exibido na figura 2.10. O `Default Web Service Profile` insere outras classes, que estão relacionadas à interface de publicação de Web Services, o WSDL (*Web Services Description Language*) [IBM04].

Dentro de um arquivo WSDL existe uma seção que descreve o *design* das interfaces. A classe `InterfaceSpec` armazena ou referencia estas seções. Podem haver múltiplas interfaces definidas em um Web Service, desta forma, pode-se criar múltiplas instâncias de `InterfaceSpec`. O atributo `name` representa o nome da interface para o usuário e o atributo `wsdl-name` é o nome da seção encontrada no WSDL. A classe `Wsdl` armazena a referência para uma descrição formal das operações disponíveis na interface.

No capítulo 3 são apresentados maiores detalhes técnicos sobre este padrão de representação de ativos, que foi adotado como principal referência na modelagem do repositório aqui proposto.

2.4 Considerações Finais

Neste capítulo foram introduzidos conceitos importantes sobre os quais foram desenvolvidas as pesquisas para a concepção da solução apresentada.

Foi aprentada a definição de aplicações P2P, suas principais arquiteturas e derivações, assim como sua localização dentro da taxonomia de sistemas computacionais.

Também foi apresentada uma visão inicial sobre o Desenvolvimento Baseado em Componentes, além do modelo de representação de ativos digitais adotado pelo OGM (RAS).

O próximo capítulo aborda o modelo de representação adotado e implementado para armazenamento dos componentes nos repositórios distribuídos. É um capítulo bastante relevante, considerando que o modelo adotado influencia diretamente nos serviços existentes nos repositórios e no protocolo de compartilhamento, apresentados no capítulo 4.

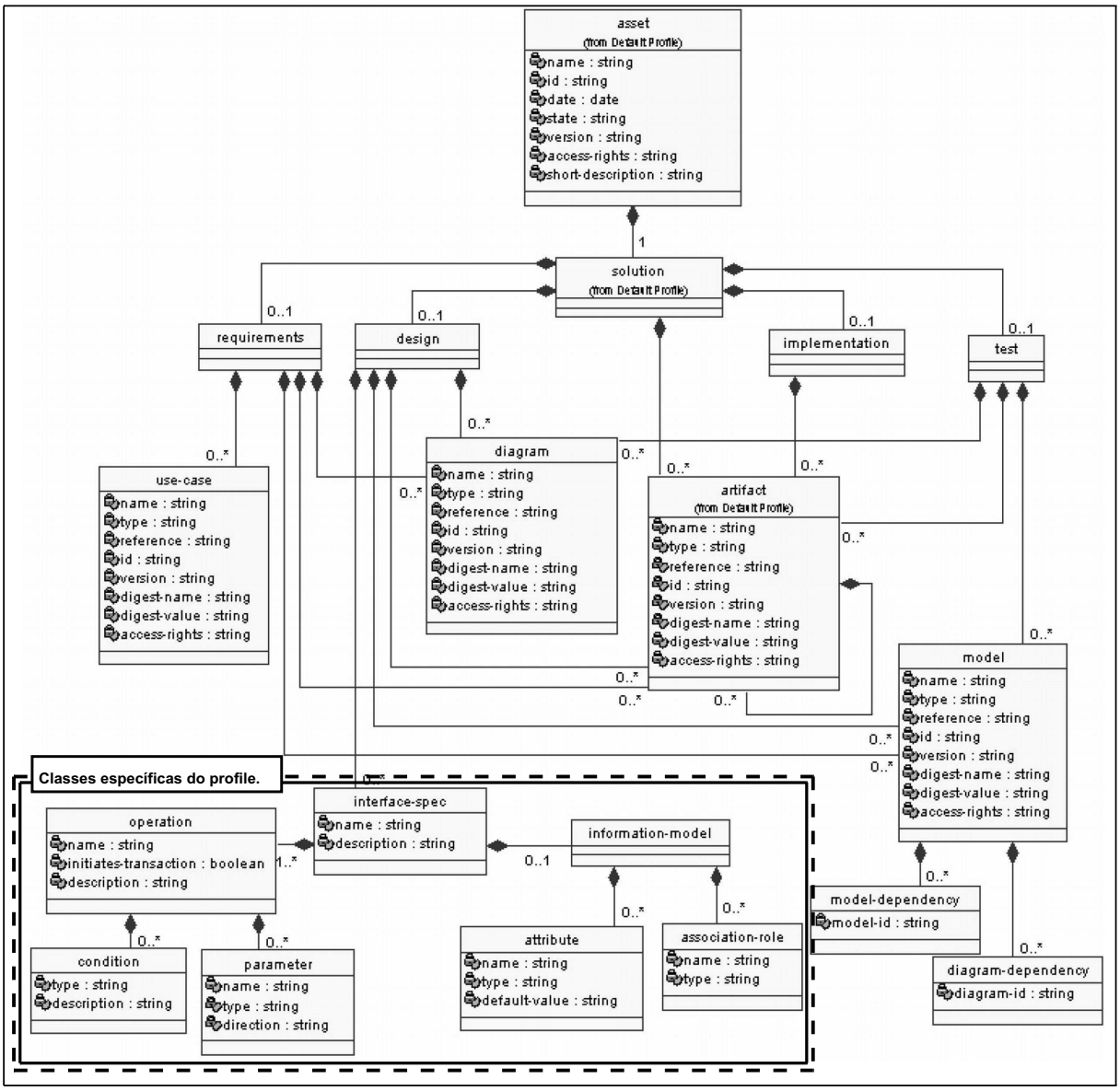


Figura 2.9: Default Component Profile.[RAS06]

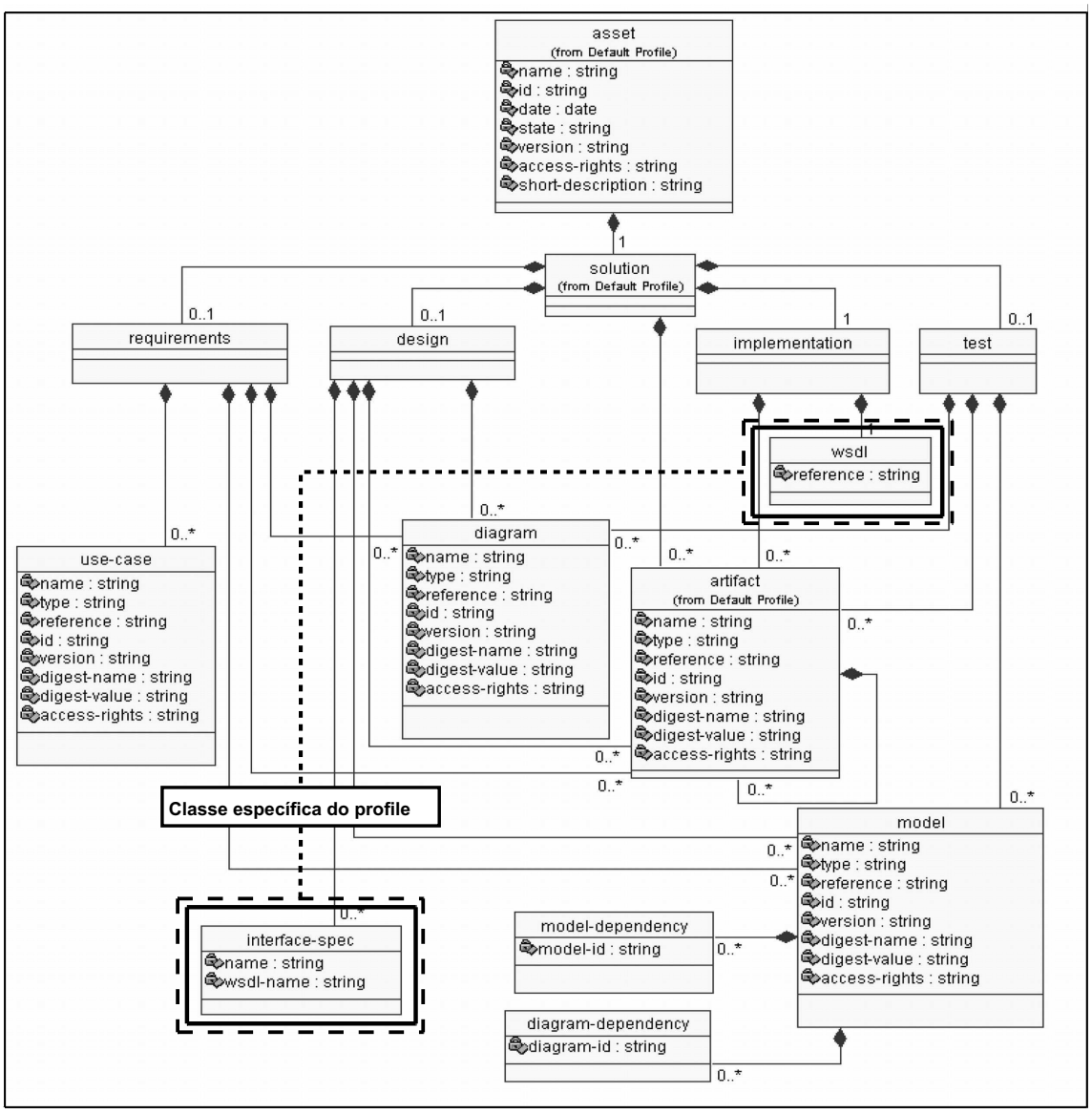


Figura 2.10: Default Web Service Profile.[RAS06]

Capítulo 3

Modelo de Repositório

No contexto de modelagem da arquitetura e do protocolo de transmissão de componentes de *software*, a definição do modelo utilizado para se armazenar os componentes torna-se relevante, uma vez que esta modelagem pode influenciar nos serviços oferecidos pelo repositório e, conseqüentemente, nos serviços disponíveis na rede de compartilhamento.

Para que seja possível conceber repositórios de componentes que sejam suficientemente flexíveis para armazenar diferentes tipos de componentes, é necessário que tenhamos a concepção de um modelo genérico, com flexibilidade para representar características e funcionalidades dos componentes, ou pelo menos aquelas consideradas mais significativas.

Uma visão que possibilita maior generalidade na representação de ativos é a utilização de atributos flexíveis. Esta abordagem admite que novas características sejam inseridas aos ativos sem necessidade de qualquer alteração no modelo de representação.

Antes de detalharmos sobre a solução, vale a pena discutir alguns aspectos que são relevantes na adoção de um modelo, considerando os objetivos do projeto. Estes aspectos são mostrados no tópico a seguir, fatores críticos.

3.1 Fatores críticos

Alguns aspectos são considerados vitais e nortearam várias decisões, sobretudo na modelagem e arquitetura da rede. Não é diferente quando tratamos do modelo de representação dos componentes.

Tais aspectos foram baseados nos requisitos iniciais da RCCS, que utiliza o modelo proposto. São considerados fatores críticos para o desenvolvimento do modelo: Padrões abertos, distribuição, usabilidade, portabilidade, escalabilidade e simplicidade. Estes fatores são discutidos a seguir.

3.1.1 Padrões abertos

Uma das principais razões para adoção de desenvolvimento baseado em componentes é a premissa do reuso. A idéia é desenvolver *softwares* a partir de componentes existentes, através da integração de partes inter operáveis. A implicação de reduzir o tempo de desenvolvimento e aumentar a qualidade do produto torna esta abordagem bastante atrativa [DW99].

Padrões de *software* provêem um alto grau de reuso em arquiteturas e design de *software*. Com a utilização de padrões, um sistema torna-se mais compreensível, simplificando o desenvolvimento e manutenção. Outro objetivo da utilização de padrões de *software* é a facilidade na disseminação da experiência no desenvolvimento de soluções já padronizadas.

Considerando a idéia inicial de se propor uma solução aberta, para que o modelo seja utilizado livremente, toda a modelagem foi feita em torno da adoção de padrões abertos, para livre utilização sem a necessidade de adaptação para as necessidades de projeto específicas. Esta modelagem influencia inclusive no protocolo de comunicação entre os pontos da rede. Maiores informações sobre este protocolo de comunicação serão mostradas na seção 4.3, que descreve a arquitetura da rede.

A utilização de padrões abertos, como alguns propostos em [OMG06], [RAS06] e [IBM04] torna a proposta mais atrativa para novos desenvolvedores, e isso alavanca a possibilidade de novas implementações utilizando o modelo concebido.

Considerando que o RAS é um padrão já adotado pelo OMG como modelo de representação de ativos digitais, com flexibilidade para representação de um modelo específico de componentes, porém com generalidade suficiente para que diferentes modelos sejam representados e armazenados em um mesmo repositório, tal especificação tornou-se a principal referência na modelagem do repositório. O repositório e o protocolo de compartilhamento devem ser compatíveis com o modelo RAS.

Além da utilização de padrões abertos, faz-se necessária a adoção de padrões que já estejam estabelecidos e com boas referências.

3.1.2 Distribuição

Outro fator crítico e de forte impacto no desenvolvimento de uma solução para modelagem do repositório é a distribuição. Ou seja, o repositório deve estar modelado de forma que possibilite o desenvolvimento de serviços de distribuição e comunicação entre os participantes da rede.

O objetivo de distribuição está enraizado no propósito inicial da rede de compartilhamento, e desde o início do projeto se pensou na existência de repositórios distribuídos, e que o compartilhamento de ativos aconteça de forma totalmente distribuída. E foram ado-

tadas tecnologias como Web Services e arquiteturas P2P, que facilitem o desenvolvimento de um sistema computacional que seja de fato distribuído, sem nenhuma dependência central.

Deve-se identificar um padrão suficientemente genérico, para que diferentes tecnologias e plataformas possam implementá-lo, visando não comprometer a participação de diferentes usuários na RCCS, e que a rede possua diversos repositórios, implantados utilizando diferentes plataformas, porém seguindo a mesma modelagem aqui proposta.

O fator crítico distribuição é fortemente considerado, e é resolvido com a implementação da arquitetura P2P proposta na capítulo 4.

3.1.3 Portabilidade e interoperabilidade

Este é mais um aspecto que preza pela utilização de uma solução aberta e padronizada. O modelo aqui proposto baseia-se no RAS, que é um modelo e que não é dependente de nenhuma plataforma. Além disso, a tecnologia adotada na transferência de dados (Web Services) favorece a portabilidade e interoperabilidade da rede, criando uma nova camada de abstração acima da plataforma [IBM04].

A interoperabilidade é uma característica complementar à utilização de padõres, e sua principal vantagem é a possibilidade de desenvolver uma solução que funcione em diferentes plataformas, e desta forma a utilização de padrões é imprescindível. Porém, é importante que estes padrões não sejam específicos de uma plataforma, para não tornar a solução restrita e dificultar sua utilização.

3.1.4 Escalabilidade

Escalabilidade, de uma forma mais geral, é definida como “a habilidade ou eficiência da solução para um problema frente ao crescimento deste problema” [RS90]. Contextualizando esta definição, escalabilidade pode ser entendida como a habilidade de uma rede de compartilhamento de componentes continuar funcionando frente ao crescimento do número de pontos participantes desta rede.

Sem dúvida, o compartilhamento de ativos na RCCS não pode ser comprometido frente a um crescente número de pontos realizando buscas e número de repositórios disponibilizando componentes. Sob este aspecto, torna-se necessário que a arquitetura montada não esteja restrita a um número máximo de pontos, e que novos participantes possam entrar e sair livremente, sem que a rede fique inativa.

Como os recursos estão distribuídos, a desconexão de um ponto da rede faz com que seus recursos fiquem indisponíveis, mas a rede deve continuar ativa e a troca de outros componentes entre os pontos restantes não pode ser prejudicada, a não ser pela ausência dos recursos que seriam oferecidos pelo pontos então inativos.

3.1.5 Simplicidade

O modelo é proposto para ser adotado em diversos repositórios de componentes, que possam ser implementados por diferentes tecnologias.

Um fator importante é a simplicidade com que as implementações sejam realizadas, sobretudo na implementação da arquitetura da rede.

Desta forma, faz-se necessário o desenvolvimento de uma arquitetura da rede de forma relativamente simples, uma vez que o objetivo é que ela seja efetivamente aplicada e re-implementada em diferentes linguagens. E que a implementação de referência da RCCS desenvolvida pelo laboratório de inovação seja apenas uma das implementações existentes para a rede.

3.2 Modelagem do asset

Como citado anteriormente, levando em consideração as informações e fatores críticos apresentados na seção anterior, o RAS tornou-se a opção de modelagem adotada como principal referência, sobretudo devido a algumas características como flexibilidade, padronização e portabilidade. Também é o modelo adotado como base para a definição do protocolo de compartilhamento.

3.2.1 Definição do padrão

O RAS foi considerado adequado aos requisitos iniciais e fatores críticos do projeto. O banco de dados do repositório foi modelado como um banco de dados relacional, e as características das classes implementadas foram migradas para as tabelas do banco.

A classe `Asset` possui alguns poucos atributos, porém a característica flexível proporcionada pelo modelo foi utilizada para representar os atributos iniciais para um componente de *software*, como idioma, tecnologia, fabricante, palavras-chave, dentre outros atributos, considerados obrigatórios para uma descrição mínima de um componente de *software*. Ou seja, mesmo tendo poucos atributos "nativos"da classe `Asset` , é possível representar componentes através de criação de atributos extras.

O tópico a seguir apresenta uma visão mais aprofundada do RAS, já apresentando como cada uma de suas seções foi mapeada no banco de dados. Também são apresentados exemplos de representação de um componente com informações básicas no modelo criado.

3.2.2 Implementação do RAS - *Reusable Asset Specification*

Como afirmado anteriormente, o RAS define uma especificação padrão para componentes e seu principal objetivo é especificar a quantidade mínima de meta informações que um

ativo digital deve ter.

O propósito do modelo é ser totalmente compatível com o RAS. Desta forma, o banco de dados foi modelado com estas características, visando o armazenamento de componentes que sejam representados pelo RAS.

A figura 2.8, exibida no capítulo de fundamentação teórica (Capítulo 2), apresentou as principais seções do RAS. Neste capítulo apresentaremos uma visão detalhada das seções e como estas seções foram mapeadas para o modelo relacional do banco de dados do repositório de componentes.

Classificação

O benefício da utilização de uma classificação "flexível"para armazenar as informações fica mais claro na seção `Classification`. Nesta seção serão armazenadas informações (atributos) dos componentes que não são representados na classe `Asset`. A figura 3.1 exibe as classes da seção `Classification`.

Através desta seção podemos criar diversos atributos para um componente. Por exemplo, um atributo previsto para componentes é idioma, que pode possuir diferentes valores. Desta forma, o `descriptor-group` armazenaria a informação `name:` "idioma", e seria relacionado pelos idiomas armazenados em diferentes registros da classe `descriptor` com `name:` "inglês", "português", "alemão". Percebe-se que esta característica faz com que se tenha liberdade para adicionar quaisquer atributos na representação de um componente, e isto torna a solução mais genérica e adaptável à possíveis novas necessidades. Estes atributos podem estar relacionados também a contextos de utilização do componente.

Solução

Além da seção `Classification`, as seção `Solution` armazena informações sobre as soluções oferecidas pelo componente. Sobretudo, armazena os artefatos contidos neste componente.

No *Component Profile*, são inseridas classes de tipos de solução (*requirements, design, implementation* e *test*), e dentre estas, classes de tipos de artefatos (*use-case, diagram, artifact* e *model*). Além disso, a classe `Interface-spec` possibilita o armazenamento de informações sobre o *design* do componente, ou seja, interfaces providas e requeridas.

Para não tornar a implementação complexa e o banco de dados muito extenso, algumas simplificações foram feitas, porém sendo possível manter a divisão dos tipos de solução e artefatos, propostos pelo *RAS Component Profile*.

Para determinar o tipo de solução (*requirements, design, implementation, test*) foi utilizado o atributo `tiposolucao` da tabela solucao. E para determinar o tipo de documento (*use-case, diagram, artifact, model*), utiliza-se o atributo `tipodocumento` da tabela artefato. Não há impacto de representatividade nesta simplificação, ou seja, a capacidade

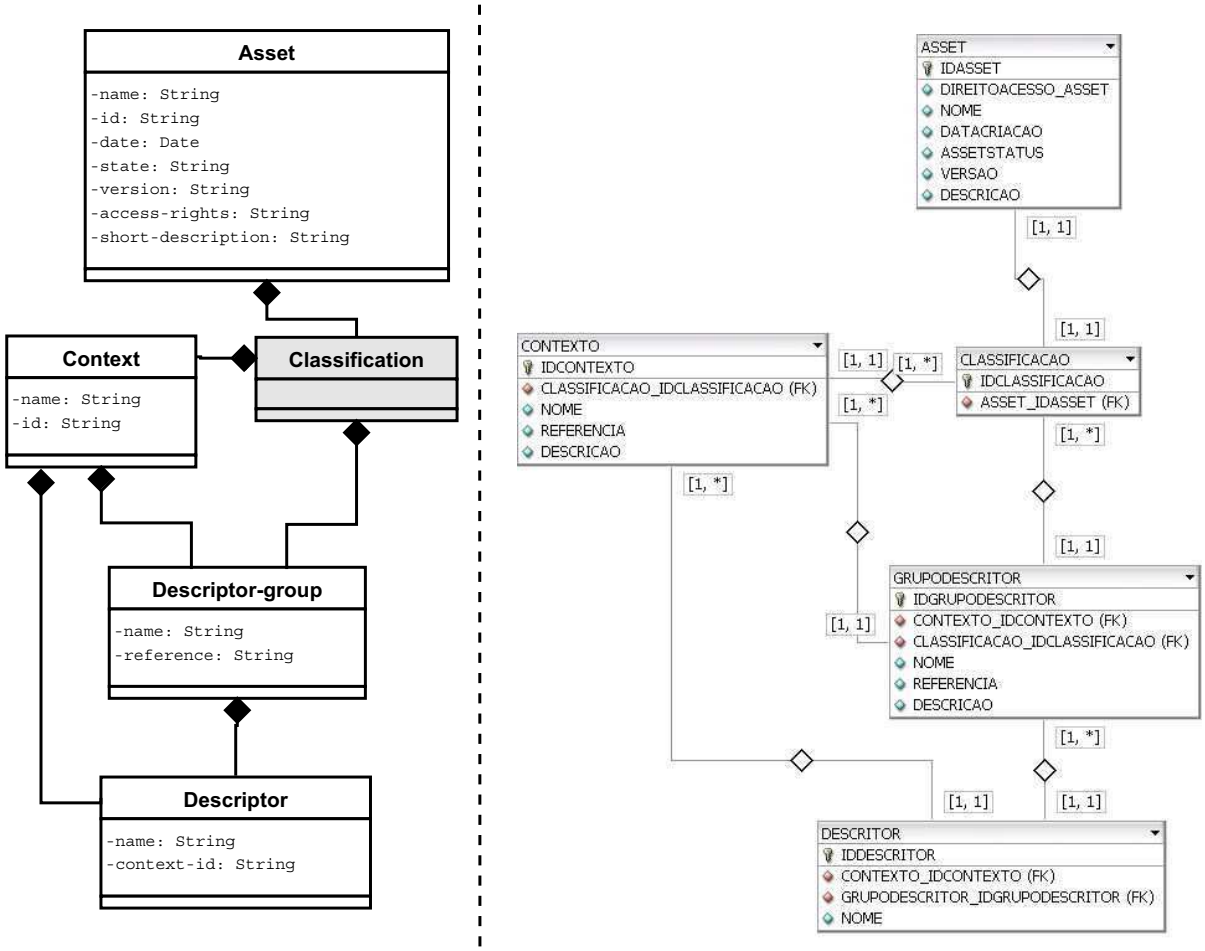


Figura 3.1: `Classification`. Diagrama de classes e tabelas no banco de dados

de manter informações é a mesma. Como as classes de tipo de solução são idênticas no modelo, nenhum atributo foi perdido, apenas criado um novo para definir o tipo da solução.

Desta forma, a modelagem do banco para armazenamento da seção `Solution` se assemelha com a descrição do `Default Profile`. A figura 3.2 exibe a seção `Solution`, com classe `Artifact`, com seus atributos.

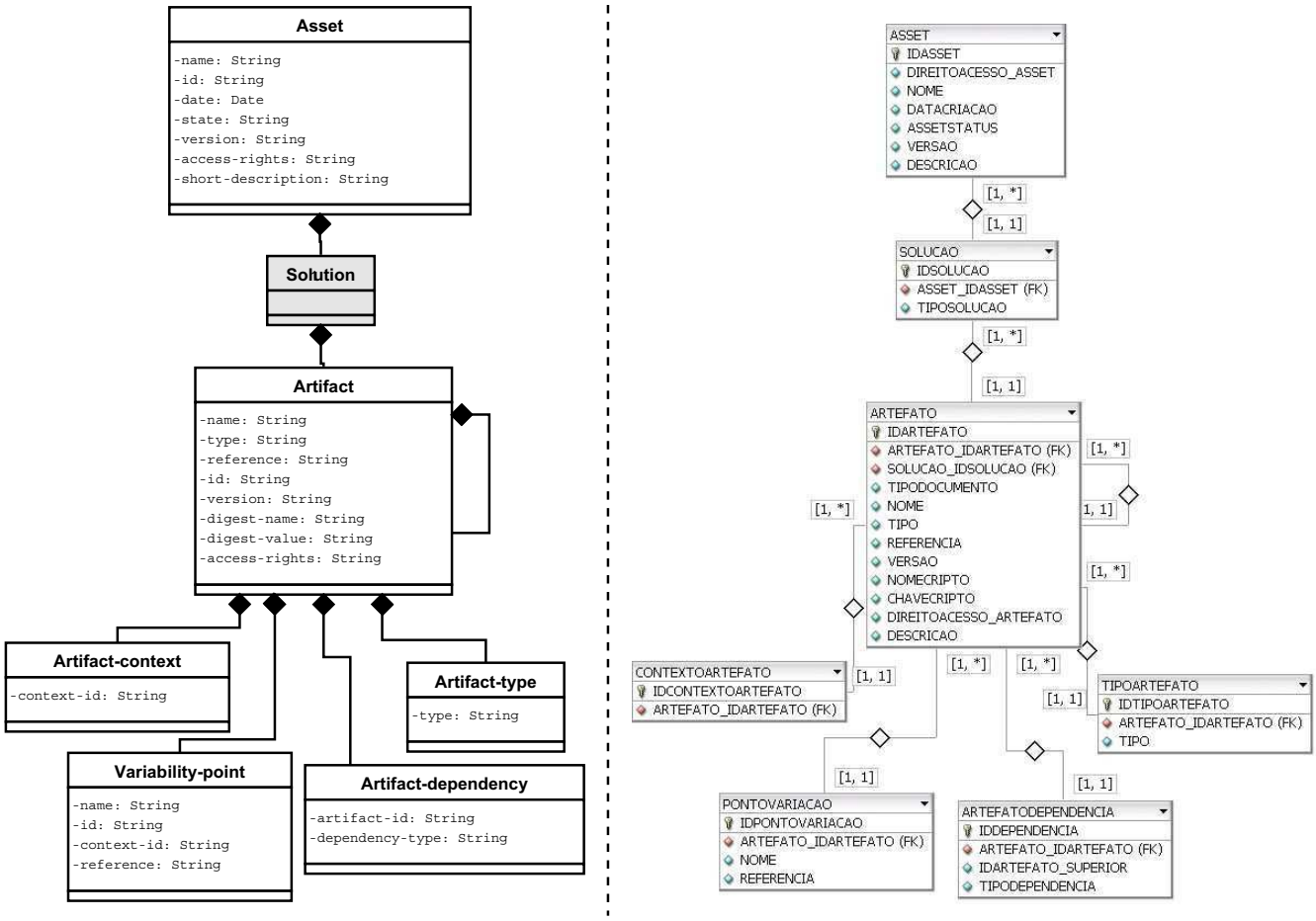


Figura 3.2: `Solution`. Diagrama de classes e tabelas no banco de dados

Utilização

A terceira seção para descrição do componente é a `Usage`, que define as possíveis utilizações do componente, assim como atividades relacionadas à sua utilização (como instalação, customização, etc.). Mas uma vez, a modelagem desenvolvida para esta seção segue a proposta do *Component Profile*. A figura 3.3 exibe o diagrama de classes da seção `Usage` .

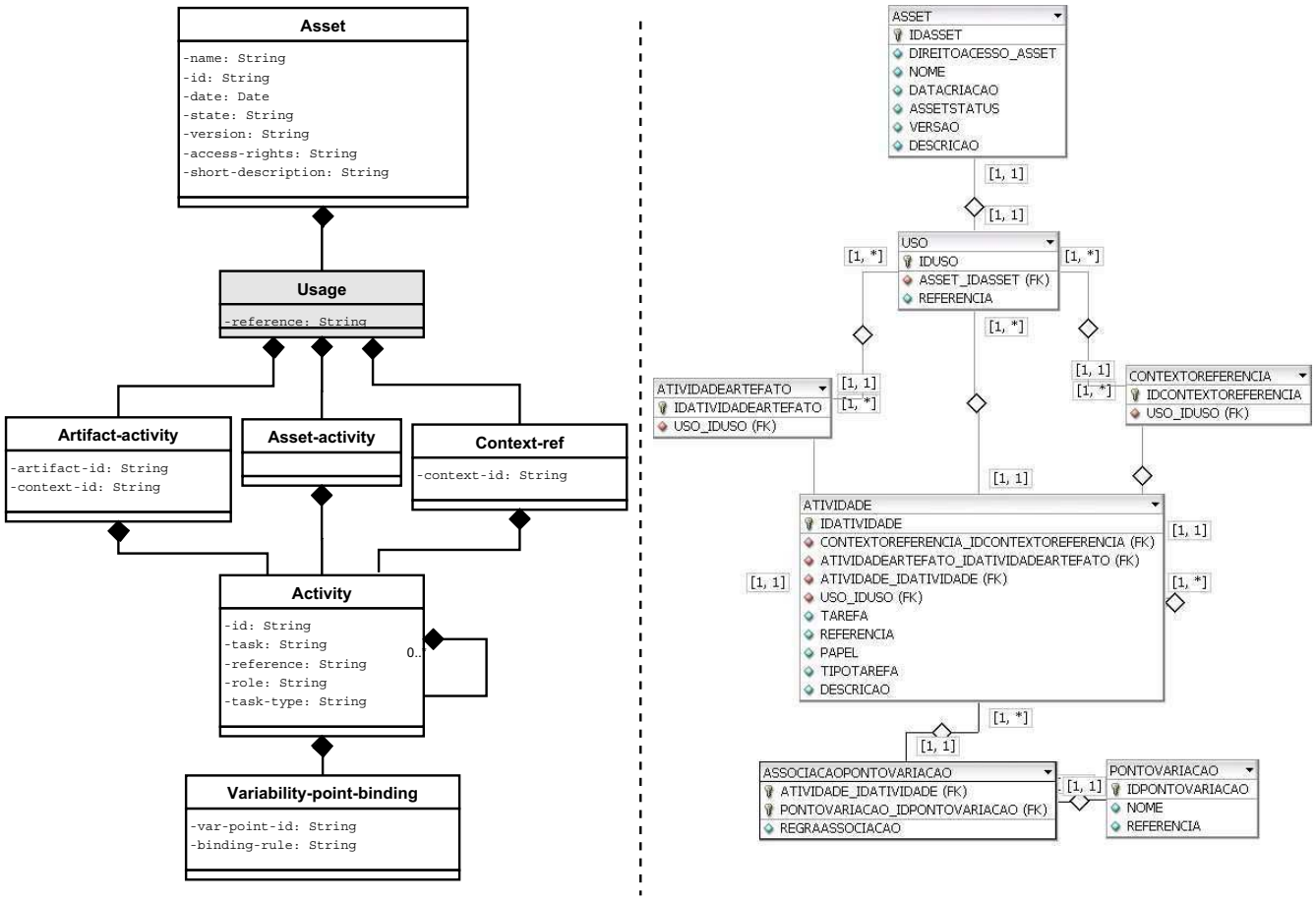


Figura 3.3: `Usage`. Diagrama de classes e tabelas no banco de dados

Relacionamento entre componentes

Além destas três seções principais que armazenam informações sobre o componente, ainda é possível registar o relacionamento entre componentes. Também foi possível manter a representação do relacionamento entre componentes de forma compatível ao modelo sugerido pelo RAS e atender as necessidades do repositório para a RCCS. Na verdade é apenas uma classe (`RelatedAsset`), que foi trasformada em uma tabela no modelo implementado para a rede de compartilhamento. Classe esta ilustrada na figura 3.4.

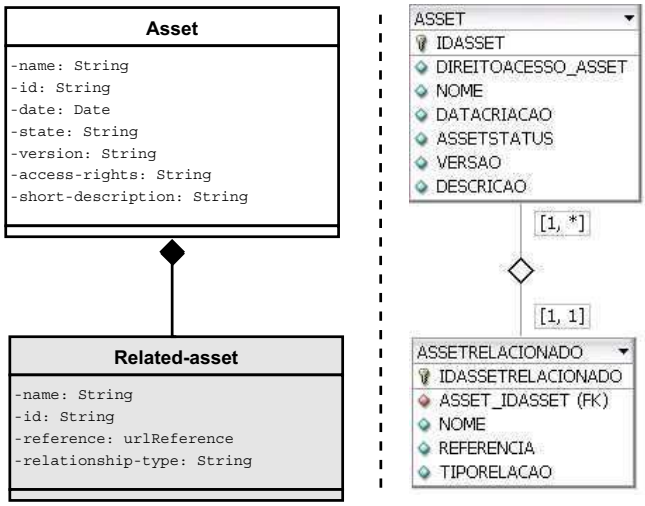


Figura 3.4: `Related-Asset`. Diagrama de classes e tabelas no banco de dados

Visão geral

Uma visão do banco de dados modelado e adotado como padrão para os serviços e implementação de referência da RCCS é ilustrado na figura 3.5.

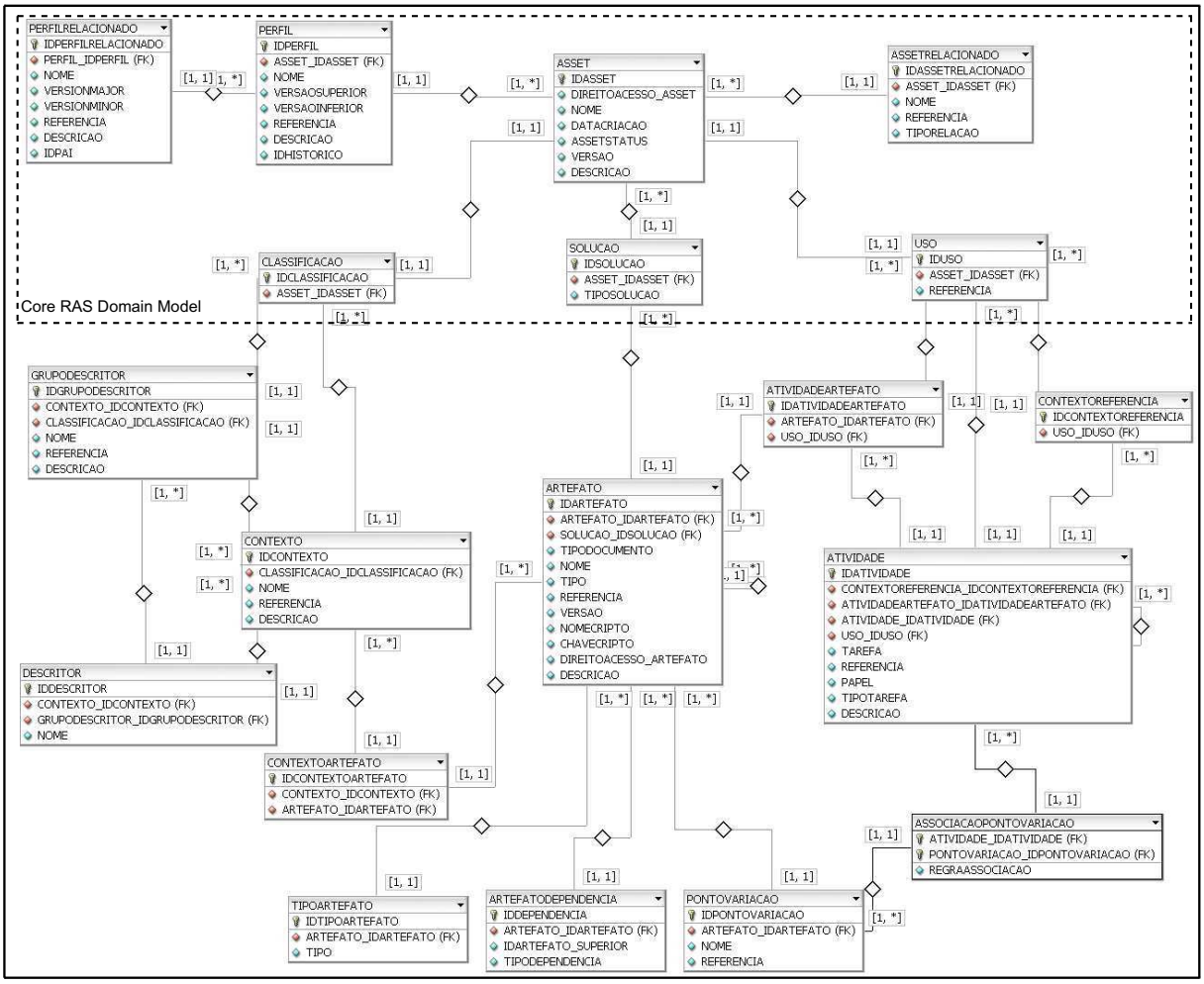


Figura 3.5: Diagrama ER do repositório de componentes.

Enfim, a modelagem proposta sugere a implementação simplificada e funcional para repositórios de componentes de *software*. A simplificação se deve à representação das classes `Requirements, Design, Usecase` que são idênticas pela única classe `Artifact`, indicando qual classe é representada através do novo atributo ("tipodocumento"). Desta forma, nenhuma informação é perdida, uma vez que os atributos são os mesmos e um novo atributo foi inserido para informar se as informações são de *requirements, design, usecase* ou *artifact*. A implementação pode ser considerada funcional pelo fato do modelo já ser estudado, implementado e testado em uma aplicação real, a RCCS.

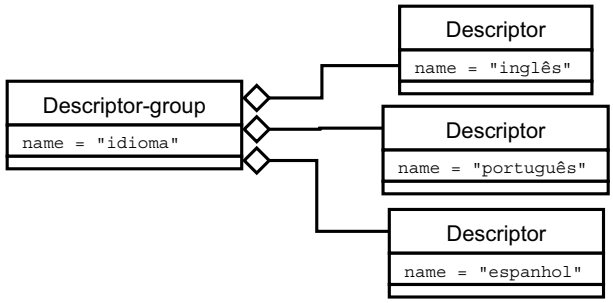
A modelagem proposta surgiu de observações sobre as informações necessárias a um

componente em uma rede P2P, onde muitas vezes não se tem contato direto com o responsável ou desenvolvedor, e as informações básicas sobre este ativo devem estar contidas no "pacote" no qual ele é transmitido. Levando em consideração estas informações julgadas necessárias e a observação de um padrão aberto e adotado pelo OMG, o RAS, foi possível utilizar um modelo que mantenha as características deste padrão, contenha todas as informações consideradas relevantes e seja implementável.

Na próxima seção, apresenta-se um exemplo de mapeamento das informações de um componente básico no modelo desenvolvido.

3.2.3 Mapeamento de atributos

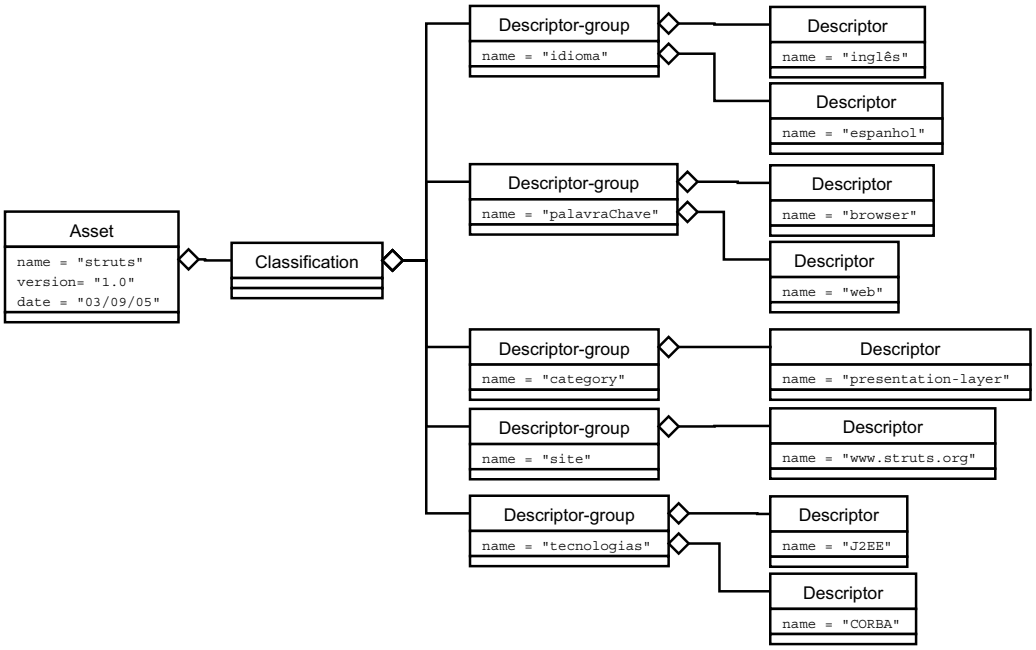
A partir da flexibilidade oferecida pelas meta informações, sobretudo da seção `Classification`, foram criados diversos atributos. Estes atributos foram mapeados armazenando a chave na classe `Descriptor-group` e seus valores na chave descriptor. Um exemplo simples é exibido na figura 3.6.



Considerando que gostaríamos de representar os valores de banco convencional:
Asset.idomas = {portugues, ingles}
Reprentaríamos da seguinte forma:
Descriptor-group[j].name = idioma
Descriptor[i].descriptorGroup = Descriptor-group[j]
Descriptor[i+1].descriptorGroup = Descriptor-group[j]
Descriptor[i].name = portugues
Descriptro[i+1].name = ingles

Figura 3.6: Exemplo de utilização dos atributos flexíveis

Porém, não é necessário fixar à tabela asset todos os atributos, principalmente porque não se têm o conhecimento do domínio completo de atributos relacionados a um componente de *software* que possa ser considerado completo em todos os contextos. Desta forma, a modelagem possibilita que as diferentes implantações ou ferramentas criem seus atributos de forma customizada, sem necessidade de alterar o banco de dados ou criar atributos em excesso. A figura 3.7 apresentam um exemplo mais completo de mapeamento de atributos do modelo conceitual para o modelo do banco de dados implementado.



Chave	***Valor***	⇔	***Chave***	***Valor***
Nome	Struts	⇔	Assets.name	Struts
versão	1.1	⇔	Assets.version	1.1
data de criação	01/05/2006	⇔	Assets.creationDate	01/05/2006
idiomas	ingles, espanhol	⇔	descriptor-group[1].name=“idiomas”	descriptor[1].descriptorGroup = “1” descriptor[2].descriptorGroup = “1” descriptor.name=“ingles” descriptor.name=“espanhol”
palavras-chave	web, browser	⇔	descriptor-group[2].name=“palavraChave”	descriptor[3].descriptorGroup =“2” descriptor[4].descriptorGroup = “2” descriptor.name=“web” descriptor.name=“browser”
site	www.struts.org	⇔	descriptor-group[3].name=“site”	descriptor[5].descriptorGroup = “3” descriptor[5].name=“www.struts.org”
categoria	apresentação	⇔	descriptor-group[4].name=“categoria	descriptor[6].descriptorGroup = “4” descriptor[6].name=“apresentação”
tecnologia	J2EE	⇔	descriptor-group[5].name=“tecnologia”	descriptor[7].descriptorGroup = “5” descriptor[7].name=“J2EE” descriptor[8].descriptorGroup = “5” descriptor[8].name=“J2EE”

Figura 3.7: Exemplo de mapeamento de atributos.

A partir da modelagem criada para o repositório, foi desenvolvido o protocolo de comunicação da rede. O objetivo foi criar um protocolo compatível com os padrões de comunicação, e que seja implementável independente da plataforma onde a rede esteja sendo utilizada. Este protocolo é compatível com o RAS, e, conseqüentemente, compatível com modelo desenvolvido. Desta forma, repositórios que utilizarem o RAS como padrão de representação de componentes poderão trocar informações com os outros pontos da RCCS. Maiores informações sobre este protocolo são exibidas no tópico 4.3, e no documento de RFC de descrição [RFC06].

3.3 Funcionalidade e Usabilidade

Um dos principais objetivos deste projeto é criar um modelo apto para oferecer funcionalidades básicas de gerenciamento e compartilhamento de componentes.

Visou-se modelar uma rede que pudesse trazer, de fato, funcionalidades que impulsionem e incentivem a utilização de componentes. Para isso, algumas funcionalidades são básicas e, sobretudo, essenciais. Pensando na forma mais natural de utilizarmos um sistema de compartilhamento, vimos que a busca de material é o primeiro passo. No capítulo 4, sobre a arquitetura da rede, são apresentadas as principais funcionalidades oferecidas no repositório e protocolo de comunicação: A busca, o AssetInfo e o *download*. Levando em consideração alguns sistemas de busca [GOO06] e sistemas populares de compartilhamento de material, observamos que uma busca por conteúdo na Internet geralmente é feita através de um texto informado.

Além de encontrar um componente, o usuário deve ter a opção por fazer o *download*. Para isso, a funcionalidade de *download* também se faz necessária. Visando agilizar a busca e diminuir o tráfego de dados, uma etapa é inserida entre a busca e o *download*: o *asset info* ou "detalhamento do componente". Ou seja, uma funcionalidade para capturar todas as informações sobre o componente. A busca retornará informações básicas sobre os componentes encontrados, e, a partir disso, o usuário poderá requisitar maiores informações a respeito. Isso é feito através do *asset info*, que retornará todas as informações do componente, baseadas na modelagem completa. A partir das informações fornecidas, o usuário poderá decidir por fazer ou não o *download* dos artefatos (que também são retornados no detalhamento) ou do componente completo.

3.3.1 Serviços de compartilhamento

O modelo deve proporcionar informações suficientes para habilitar o funcionamento da rede de forma P2P, com comunicação entre os pontos de forma independente de plataforma e seguindo padrões de modelagem e comunicação. O serviço essencial para o funciona-

mento da rede é a busca por componentes. Porém, alguns cuidados são necessários em relação a busca por componentes. Diferentemente de sistemas populares de compartilhamento de conteúdo [NAP06], [GNU06] e [KAZ06], o compartilhamento envolve muito mais informações a serem consideradas.

Desta forma, o protocolo de comunicação transporta mais informações, e de forma estruturada. Na busca podem ser informadas diversas características de um componente, não apenas um texto, como é comum em sistemas P2P. Além disso, o resultado da busca é um conjunto de dados estruturados, e não apenas um arquivo de música ou documento. Considerando estas características, fez-se necessária a definição de um conjunto de objetos a serem transmitidos pelos serviços de compartilhamento. Maiores detalhes sobre este protocolo definido são apresentados na seção 4.3, que apresenta a modelagem de objetos propostos para o compartilhamento na RCCS.

3.4 Considerações finais

Neste capítulo foi apresentada a modelagem adotada para o repositório de componentes. Esta modelagem foi implementada seguindo algumas premissas como utilização de padrões, interoperabilidade e flexibilidade na representação dos ativos. A criação do banco foi feita totalmente baseada no RAS (*Reusable Asset Specification*), mantendo todas as suas características originais.

A principal contribuição da modelagem é o fato de utilizar criação dinâmica de atributos para classificação dos ativos, atribuindo uma característica de flexibilidade na definição dos atributos que devem ser utilizados para descrever o componente. Além disso é uma especificação extensível, através da criação de *Profiles* específicos.

A adoção do RAS como padrão foi também uma forma de validá-lo através de uma aplicação real. E a partir da modelagem criada foi possível desenvolver o protocolo de transmissão de componentes em uma rede distribuída.

No próximo capítulo apresenta-se a arquitetura proposta para compartilhamento de componentes, com definição do mecanismo de descoberta de repositórios e implementação do protocolo.

Capítulo 4

Rede de Compartilhamento

Como visto no capítulo 2, um sistema P2P pode ter sua arquitetura construída de duas formas: pura ou híbrida. A arquitetura aqui proposta é uma arquitetura híbrida e foi baseada em duas arquiteturas principais, a arquitetura CIA e a arquitetura DIFA.

Tendo em vista características positivas nas duas arquiteturas citadas, uma boa solução seria fazer um sistema que reunisse o melhor de ambas as arquiteturas (CIA e DIFA), montando um método simples e bastante funcional de realizar o *resource discovery*. O método de realizar esta descoberta de repositórios e a busca de componentes são apresentados mais adiante, neste capítulo, com detalhes na seção 4.2.

4.1 Modelo de distribuição

Em cada repositório de componentes deve haver um servidor onde os serviços fiquem disponíveis e a comunicação seja feita na forma de Web Services. A arquitetura permite que os pontos da rede comuniquem-se diretamente, sem a interferência ou dependência de um servidor central.

4.1.1 Papéis dos participantes da rede de compartilhamento

Levando-se em consideração que um ponto da rede pode simplesmente procurar por componentes, não sendo, obrigatoriamente, um repositório, existe uma diferença de papéis entre seus participantes.

Reusable Asset Browser. Um ponto pode ser simplesmente um consumidor de componentes, disparando pesquisas na rede para o acesso aos componentes compartilhados. É o que chamamos de *Reusable Asset Browser* (RAB), um agente da rede que faz a busca.

Reusable Asset Repository. Existem os repositórios distribuídos de componentes, denominados *Reusable Asset Repository* (RAR), que disponibilizam seus componentes na rede para serem acessados por outros participantes. É possível que um ponto da rede funcione como RAR e como RAB ao mesmo tempo, ou seja, compartilhando e buscando por componentes na rede.

Yellow Pages. A RCCS possui servidores de referência, ou yellow pages, que são registros de repositórios distribuídos. As yellow pages funcionam como catálogos de repositórios, que podem ser consultados para que se tenha maior conhecimento dos participantes da rede. Tecnicamente, são servidores UDDI (*Universal Description Discovery and Integration*) [UDD06], com referências para os Web Services de busca disponibilizados pelos repositórios.

4.1.2 Políticas de compartilhamento

Algumas redes P2P dão uma atenção especial à política de compartilhamento de conteúdo, buscando evitar a existência de nós caronas (também chamados parasitas ou *free riders*) [RRRW05].

Segundo [AH00], quase 70% dos usuários não compartilham recursos em uma rede P2P Gnutella e aproximadamente 50% de todas as respostas são retornadas por 1% dos hospedeiros compartilhadores.

Tendo em vista esta possível "injustiça" ocorrida, diversas são as políticas adotadas por redes *peer-to-peer*. Em [PH04] foi projetado um middleware que funciona como elemento intermediário entre os participantes da rede P2P, promovendo um ambente P2P controlado. O trabalho foi estudado por outros pesquisadores [RRRW04], visando sua melhoria e aplicação em redes P2P que necessitassem de alguma política para acesso a recursos.

Por outro lado, em alguns casos, esta abordagem não faz tanto sentido. No contexto de distribuição de componentes, neste projeto, a decisão foi de que não se pode exigir que todos os desenvolvedores ou grupos de desenvolvimento possuam componentes para compartilhar. E como a rede é específica para componentes de *software*, não é possível exigir que um novo participante já possua componentes para compartilhar na rede. A idéia é facilitar a distribuição de componentes de *software* e também sua troca entre desenvolvedores. Isto se torna claro quando, na definição dos papéis, é possível que participantes da rede sequer possuam um repositório. Ou sejam, se conectem à RCCS apenas para encontrar componentes. Outros, por sua vez, podem apenas fornecer componentes, para serem testados ou avaliados, sem o objetivo de realizar buscas por componentes em outros repositórios.

Enfim, na arquitetura aqui proposta e desenvolvida, não foi criada nenhuma restrição para seus participantes buscarem e obterem material. Além disso, nenhuma restrição é feita a novos repositórios que compartilhem conteúdo. Acredita-se que esta abordagem torna a rede mais acessível e atrativa para novos adeptos. Trata-se de uma rede livre, mas novas implementações de protocolos poderiam criar políticas específicas, caso faça sentido em diferentes contextos.

4.1.3 Tecnologia de comunicação

A comunicação entre um repositório e um consumidor será feita através de Web Services. Levando em consideração a premissa de utilizarmos padrões abertos, temos nos Web Services fortes características, promovendo também a interoperabilidade. Eles oferecem uma nova camada de abstração, permitindo integrar aplicações, mudar a interface com o usuário e evoluí-las. Sabendo que a necessidade de comunicação entre os usuários é real, devemos considerar que é mais do que desejável que se criem mecanismos para que isso aconteça da melhor forma possível, com velocidade e segurança.

No centro desta visão está o conceito de operabilidade conjunta, ou seja, a capacidade de sistemas diferentes se comunicarem e compartilharem dados sem estarem ligados entre si. Este é um dos objetivos dos Web Services.

O grupo de trabalho da W3C (World Wide Web Corsortium) de Arquitetura dos Web Services chegou a um acordo com relação à definição de um Web Service [IBM04]. Sua definição é: "um Web Service é uma aplicação de *software* identificada por um URI (*Uniform Resource Identifier*), cujas interfaces e bindings são capazes de serem definidos, descritos e descobertos como artefatos XML (*Extensible Markup Language*) e são publicadas através do padrão WSDL (*Web Services Description Language*). Um Web Service permite que haja interações diretas com outros agentes de *software* utilizando mensagens baseadas em XML, enviadas e recebidas através de protocolos baseados na Internet".

Isso significa dizer que é possível acessar um Web Service disponível na Internet e utilizar todas as suas funcionalidades públicas. O acesso será na maioria das vezes via HTTP, mas internamente existe uma cadeia de caracteres XML que está empacotada em um protocolo SOAP (*Simple Object Access Protocol*). O SOAP é um padrão aberto criado por grandes empresas da indústria de tecnologia para padronizar a transferência de dados em diversas aplicações. Para isso, usou-se o XML, que é um padrão amplamente aceito pela indústria de *software* e, principalmente, utilizado na definição dos demais protocolos [MSZ01]. Toda a comunicação entre os pontos da rede é feita através de Web Services.

4.2 Descoberta de repositórios e busca por componentes

A busca é o primeiro passo para o funcionamento da rede. Podemos considerá-la como a parte mais importante do sistema, uma vez que ela determinará, muitas vezes, a qualidade do serviço. Uma busca deve ser simples e rápida, porém eficiente.

Uma vez que temos uma arquitetura P2P para a rede de compartilhamento e baseados nas duas principais arquiteturas existentes (CIA e DIFA), a arquitetura de busca utiliza o que consideramos como pontos fortes de cada arquitetura existente.

Se por um lado a arquitetura pura sobrecarrega a rede em suas buscas por recursos, ela forma uma rede totalmente independente de um servidor central e isso deve ser levado em consideração na nossa solução. Da mesma forma, a arquitetura CIA utiliza um único servidor central para busca, tornando a rede mais vulnerável, porém ela não possui problemas clássicos como sobrecarga da rede, falta de controle sobre a busca (pois o usuário sabe onde está buscando) e conhecimento global da rede (todos os pontos da rede estão registrados no servidor), e assim, também devemos levar em consideração vários aspectos da CIA em nossa arquitetra.

Tendo em vista todos os aspectos destas principais arquiteturas originais, foi montada uma arquitetura de busca onde existe uma base inicial de conhecimento em cada ponto da rede (seus vizinhos, segundo a DIFA) e a busca pode ser disparada diretamente para estes vizinhos, sem necessidade de uma consulta prévia em um servidor central. Mas esta busca não é propagada pelos vizinhos, evitando a sobrecarga da rede. Porém, a arquitetura da rede deve torná-la apta a conviver com fatores como o crescimento do número de participantes, uma vez que a busca é feita apenas em pontos conhecidos. Neste contexto, surge a necessidade da existência de servidores de referência (as yellow pages), onde um ponto pode escolher por consultar sobre novos pontos da rede e disparar as chamadas de busca para estes outros pontos também, até então desconhecidos. Observe que isso não torna o servidor imprescindível, porém, acrescenta maior poder à busca. Desta forma, um usuário (ponto da rede) pode escolher onde realizar a busca: se esta busca é realizada em seus vizinhos conhecidos (selecionando, inclusive, quais estes vizinhos serão consultados) e, além disso, pode optar por realizar a busca também nos pontos da rede ainda desconhecidos e capturados através de uma consulta às yellow pages da rede. Uma vez feita esta busca nas yellow pages, os repositórios encontrados (RARs) podem ser acrescentados ao banco de conhecimento daquele ponto, aumentando sua base de conhecimento sobre os integrantes da rede.

A figura 4.1 ilustra a arquitetura da rede, onde um *Reusable Asset Browser* conhece os repositórios (RARs) 1, 2 e 6 e desconhece os repositórios 3, 4, e 5, porém, através de uma consulta nas yellow pages, ele toma conhecimento de todos os repositórios da

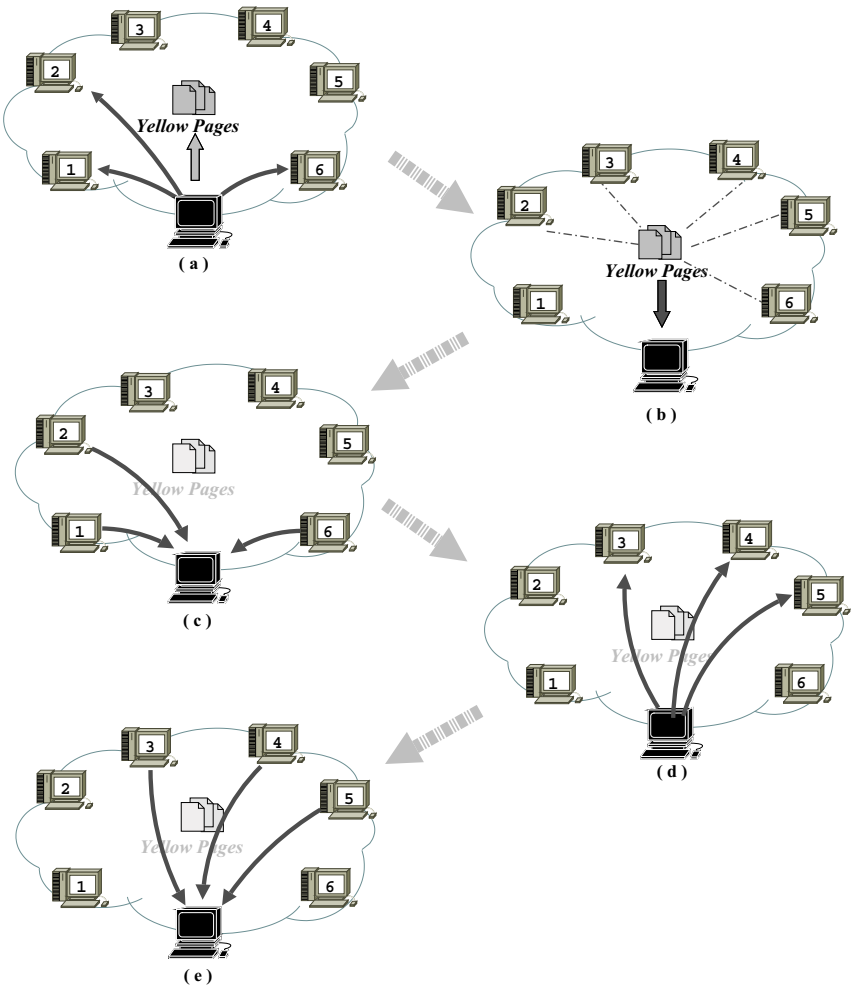


Figura 4.1: Descoberta de repositórios e realização da busca na RCCS

rede, podendo fazer com que sua busca atinja a rede inteira. A figura apresenta alguns passos em etapas separadas, porém a ordem temporal não é exata, uma vez que o tempo de resposta de cada repositório depende da sua capacidade, e não acontece de forma sincronizada com os outros repositórios.

As etapas do processo de descoberta por repositórios e busca por componentes apresentadas na figura 4.1 são as seguintes:

1. No momento da busca, o usuário seleciona, entre um conjunto de pontos conhecidos, onde deseja realizar a busca e, além disso, pode optar por buscar por novos participantes da rede em servidores yellow pages, que armazenam serviços disponibilizados, utilizando a especificação de publicação de serviço dos Web Services, o UDDI (Figura 4.1 a).

2. O servidor yellow page, que armazena as referências para os participantes, retorna os endereços dos repositórios (Figura 4.1 b).

3. A terceira etapa (Figura 4.1 c) mostra o momento em que o resultado da busca realizada nos repositórios já conhecidos é recebido.

4. Conhecendo os demais participantes da rede, é disparada novamente a busca, mas somente nos participantes da rede que ainda não haviam sido acessados (Figura 4.1 d).

5. A resposta da busca realizada nos participantes descobertos na yellow page é exibido na figura 4.1 e.

A existência de um servidor de referência torna mais prática e rápida a descoberta de repositórios. Porém, como os pontos da rede possuem sua própria base de conhecimento, a consulta neste servidor central é opcional e sua inexistência ou indisponibilidade não faz com que a rede deixe de funcionar, ou seja, a busca pode se realizar apenas com os passos **a** e **c**, sem a necessidade da consulta no servidor de referência (yellow page).

4.2.1 Interfaces gráficas da RCCS.

A seguir, serão apresentadas as interfaces gráficas de busca e recuperação de componentes na RCCS. As interfaces permitem executar o mecanismo de descoberta de repositórios e busca por componentes apresentado na seção anterior. São apresentadas as interfaces de preferências, busca simples e avançada, visualização do resultado de busca e visualização dos detalhes do componente.

Preferências

A figura 4.2 apresenta a interface de preferências da RCCS. Nesta interface são definidos onde a busca será realizada (repositórios da base de conhecimento) e Yellow Page, caso o usuário queira buscar em outros repositórios que não estejam em sua base de conhecimento.

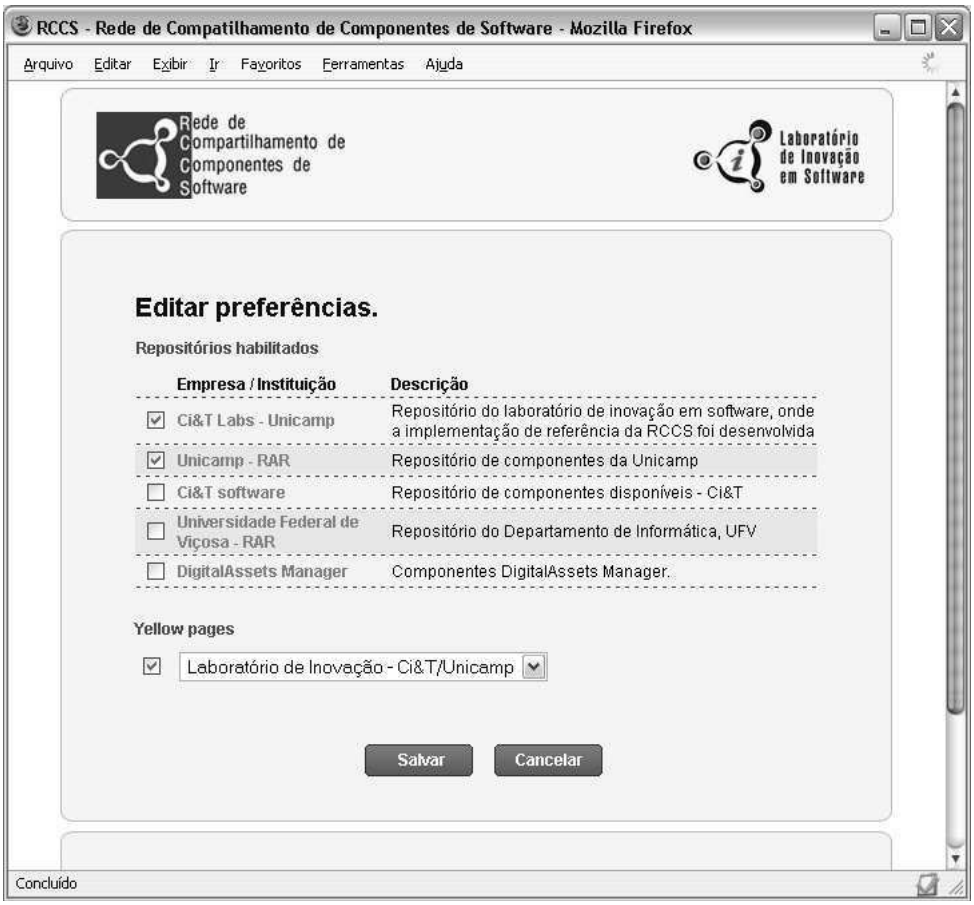


Figura 4.2: Interface de preferências da RCCS.

Realizando uma busca simples

A figura 4.3 apresenta o início da busca utilizando uma única palavra chave "`data`". Caso seja escolhida a busca remota, a busca será realizada nos repositórios segundo a configuração definida nas preferências, ilustradas pela figura 4.2.

Figura 4.3: Interface de busca RCCS.

Vale ressaltar que é possível realizar a busca apenas localmente. Desta forma, a RCCS pode ser utilizada também para o gerenciamento do acervo de componentes local, de uma instituição ou grupo de pesquisa que possua um elevado número de componentes.

Resultado da busca

A figura 4.4 mostra o resultado da busca executada em diferentes repositórios.

Vale observar que o repositório *SourceForge.net* não estava na base de conhecimento, mas foi atingido através da referência fornecida pela *Yellow Page* selecionada na tela de preferências. Esta interface apresenta a relação de componentes encontrados, com algumas informações básicas sobre cada componente (descrição, categoria, tecnologia e distribuição). Para maiores detalhes, o usuário clica sobre o nome do componente, e o serviço de detalhamento deste componente é chamado no repositório responsável por ele.

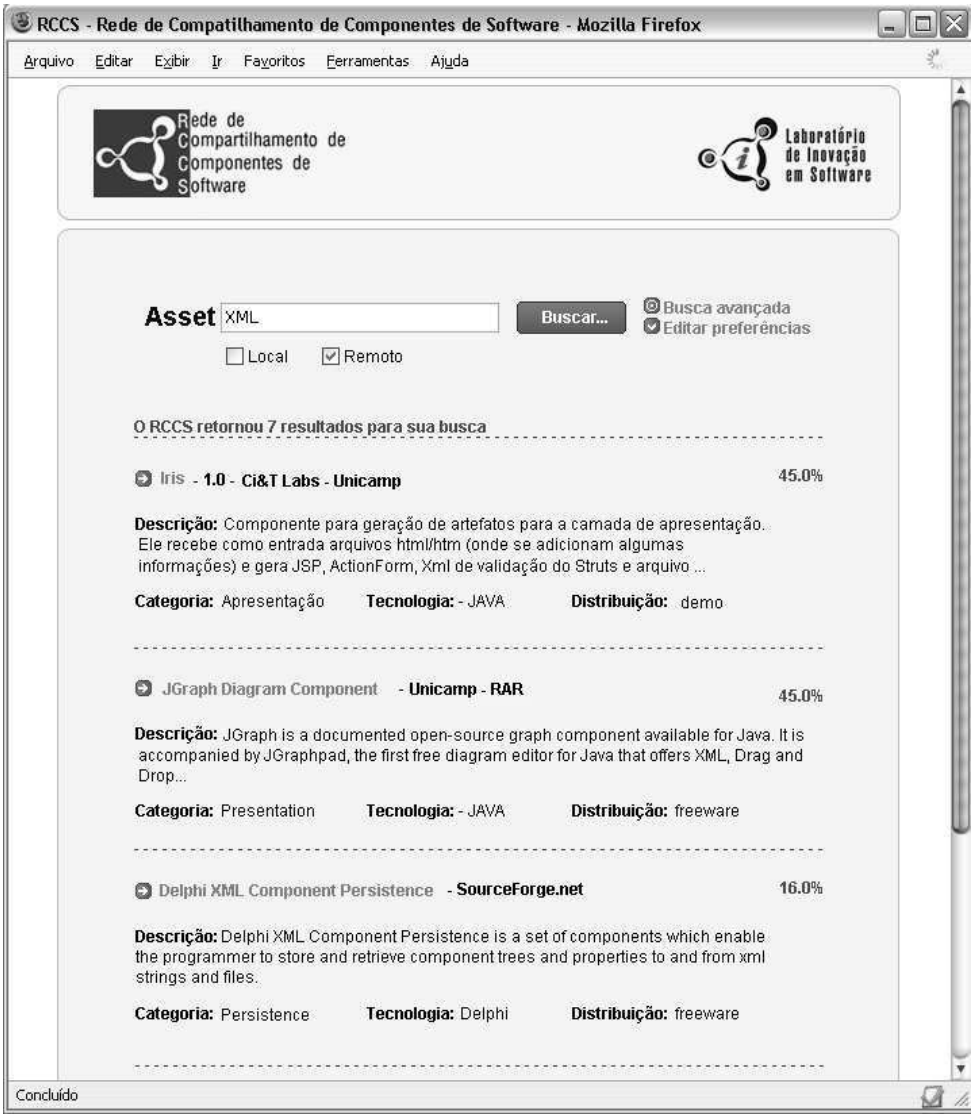


Figura 4.4: Interface de resultado da busca.

Detalhamento do componente

A funcionalidade de detalhar ou pedir detalhes de um componente pode chamada remotamente a partir do resultado da busca. Para isto, é chamado um WebService que retorna todas as informações (em formato RAS) sobre um componente, e estas informações são apresentadas na interface, como ilustra a figura 4.5.

Nesta interface são exibidas as informações de classificação do componente (atributos), como descrição, categoria, idiomas, fornecedor, status, etc.

Também são relacionados os artefatos do componente. Para os artefatos que forem

Figura 4.5: Detalhamento de um componente (*asset info*).

links, é possível acessá-los diretamente a partir da interface. Os artefatos que forem arquivos podem ser baixados, chamando o Web Service de *download* de artefato diretamente no repositório responsável (esta chamada é disparada através de um clique sobre o link "ver", na frente do nome do artefato). Além disso, é possível visualizar o conjunto de componentes relacionados a este componente.

Também é possível fazer o *download* completo do componente, através do botão `Exportar Assset`. Este botão dispara a chamada do Web Service de *download* do componente diretamente no repositório onde ele está armazenado. O componente é compactado segundo o padrão de empacotamento RAS, e pode ser baixado. Maiores informações sobre

este empacotamento são apresentadas na seção 4.3.3, que fala sobre o serviço de *download* de artefatos e componentes na RCCS.

Busca avançada

A busca por componentes pode conter maiores informações que simplesmente um texto. Para isso, foi implementada a interface de busca avançada. Nesta interface, conforme ilustra a figura 4.6, além da opção de se determinar um número máximo de resultados, são oferecidas opções de seleção por:

- Categoria
- Tecnologia principal
- Distribuição
- Idiomas suportados

No momento da busca, o objeto que é trasmitido para os repositório, definindo a característica da busca realizada, é o "`SearchDescription`", conforme apresentado na seção a seguir, sobre a modelagem de objetos, mais especificamente na subseção 4.3.1, que detalha o objeto enviado na busca por componentes.

Resultado da busca avançada

A interface de resultado da busca avançada é a mesma da busca simples. Porém, vale a pena representar o resultado da busca ilustrada na figura 4.6, onde é definido que a busca será realizada apenas por componentes com tecnologia "`J2EE`". O resultado desta busca avançada é exibido na figura 4.7

4.3 Comunicação e modelagem de objetos

O protocolo de comunicação determina quais são e como serão representadas as informações transmitidas na rede. Como a comunicação é feita através de Web Services, as informações serão trocadas através de objetos, ou seja, o protocolo trata de quais objetos são transportados na rede. Para descrever tais objetos, primeiro precisamos saber como os componentes são representados, uma vez que serão sobre os componentes as informações transmitidas na rede.

A modelagem do protocolo seguiu o mesmo padrão da representação no repositório e foi feita baseada no RAS.

Figura 4.6: Interface de busca avançada.

Algumas alterações foram feitas no padrão proposto pelo RAS, visando atender as necessidades da rede. As principais alterações foram simplificações feitas no extenso modelo proposto, porém mantendo a fidelidade ao padrão do OMG.

Considerando a complexidade e quantidade de informações contidas num modelo RAS, que será utilizado para representar o componente, o processo de busca foi simplificado. Para isso, entre a busca e o *download* de um componente foi introduzida a etapa denominada *asset info*. Desta forma, tem-se três etapas durante a procura por componentes:

- ***Busca***. São retornadas apenas informações básicas sobre o componente, ou seja, as seções Asset e Classification, esta decisão foi tomada baseada na possibilidade de uma única busca retornar uma grande quantidade de componentes, o que poderia sobrecarregar a rede se todos os componentes fossem transmitidos por completo.

- ***Detalhamento (asset info)***. O usuário solicita os detalhes de um único componente, ou seja, o usuário seleciona um componente na interface e pede maiores

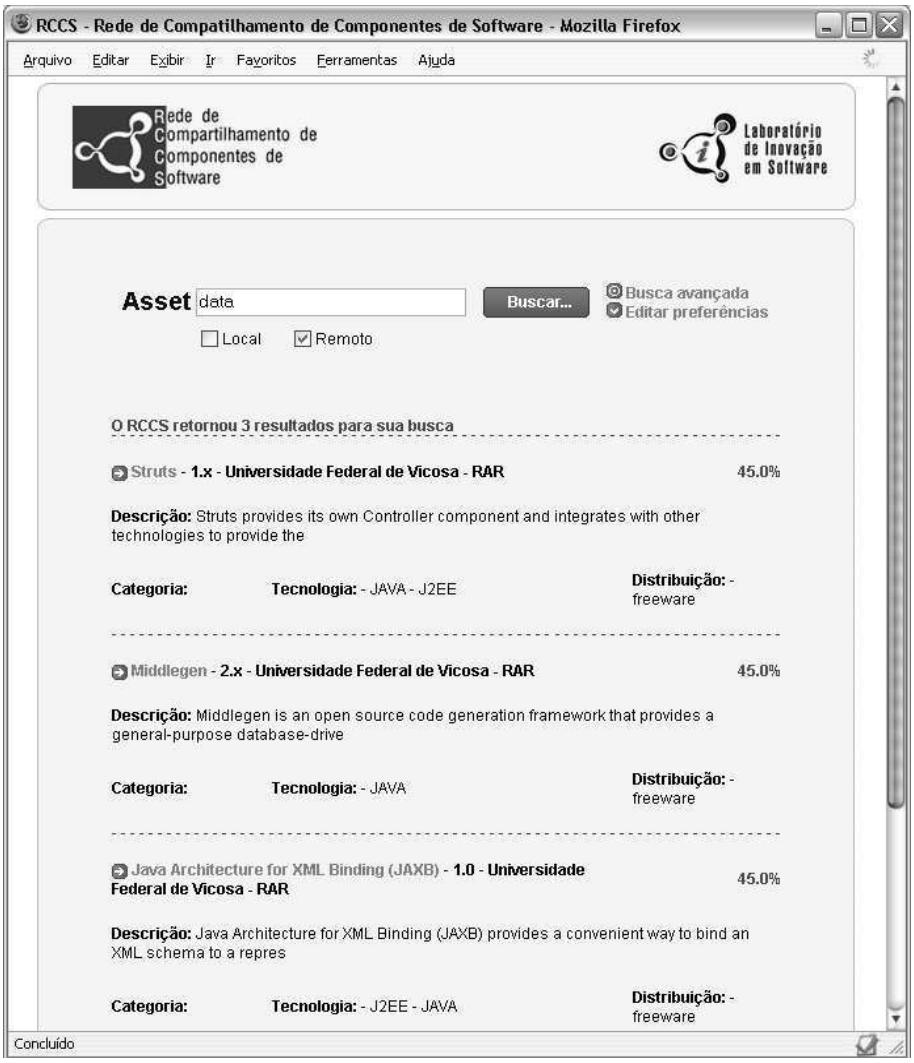


Figura 4.7: Interface de resultado da busca avançada.

informações sobre ele. Nesta etapa todas as informações sobre um determinado componente são retornadas. Como é apenas um componente, a grande quantidade de informações não interfere no desempenho da rede.

- ***Download***. Através da visualização detalhada do componente, é possível que o usuário faça *download* deste componente (através do empacotamento), ou de algum de seus artefatos, que constituem a seção `Solution` (diagramas, casos de uso, códigos fonte, modelos de teste, etc.).

Os repositórios disponibilizam Web Services para serem acessados pelos Buscadores (RAB). Estes serviços são acessados para realizar buscas, visualizar mais detalhes de um

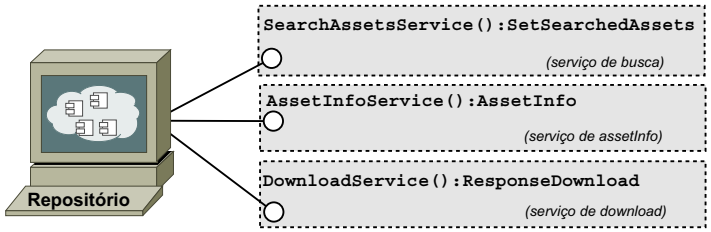


Figura 4.8: Serviços de compartilhamento de um repositório.

componente, fazer *downloads* e demais iterações necessárias entre os pontos da rede. A figura 4.8 ilustra um repositório e os serviços oferecidos.

A descrição dos objetos retornados pelos serviços e demais objetos envolvidos na comunicação entre os pontos da rede pode ser encontrada no documento de descrição do protocolo, disponível em [RFC06]

Na definição da arquitetura, foram considerados os dois principais modelos de distribuição P2P, dos quais se adotou algumas importantes características, como busca direta entre pontos da rede e utilização de um servidor de referência, porém sem criar nenhuma dependência de qualquer ponto.

4.3.1 Busca por componentes

Além da busca simples, existe a opção de busca avançada, onde são fornecidas informações adicionais, e não apenas a busca por texto. Desta forma, é necessário ter um objeto para envio destas informações para o repositório.

Para disparar a busca, o usuário envia para os repositórios conhecidos (e desejados) o objeto `SearchDescription` , que tem os dados de descrição da busca. A figura 4.9 apresenta as classes `SerachDescription` e `SetSerachedAssets`.

O atributo SCORE, da classe `SearchedAsset` é calculado pelo repositório, de acordo com as ocorrências do texto de busca (atributo TEXT de SearchDescription).

Na classe SearchDescription são passadas as informações de busca, incluindo a parte de `Classification` na qual é possível passar parâmetros que funcionam como filtro para os componentes buscados, por exemplo, alguma tecnologia ou categoria específica de componente. A classe `SetSearchedAssets` possui um conjunto de `SearchedAsset`, classe que contém as informações básicas do componente e a seção "classification", que transmite os meta-dados. Vale ressaltar que informações das seções `Solution` (como os artefatos) e `Usage` não são transmitidas neste objeto, para melhorar o desempenho da etapa de busca. A figura 4.9 apresenta estas classes com formato proposto para documentos no padrão RFC, utilizado para descrever protocolos de distribuição.

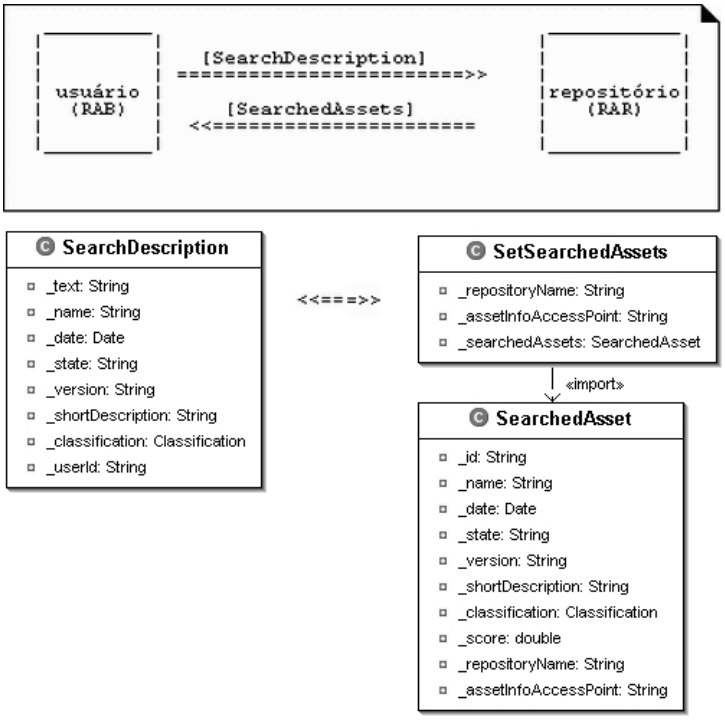


Figura 4.9: Classes `SearchDescription` e `SetSearchedAssets`

4.3.2 Asset Info

Após ter o array de `SearchedAsset`, o usuário do RAB pode pedir mais informações sobre um determinado Componente. Para isso ele deve enviar uma mensagem ao RAR, como ilustrado na figura 4.10.

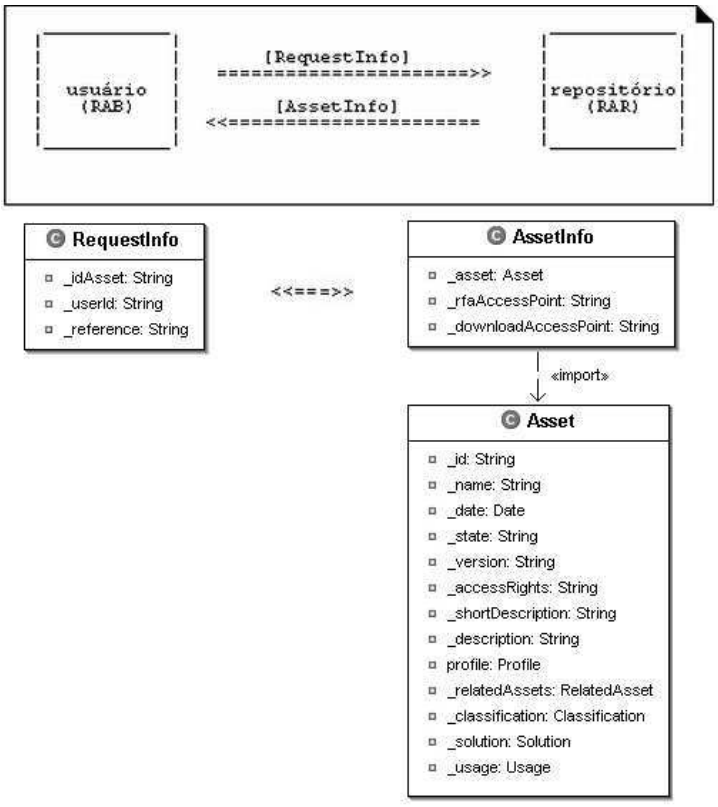


Figura 4.10: Classes `RequestInfo` e `AssetInfo`

Um objeto da classe `AssetInfo` trasporta todas as informações do componente, através do atributo "asset". A classe `Asset` é a classe descrita no RAS, contendo todas as seções: `RelatedAsset, Classification, Solution, Usage`. A quantidade de informações pode ser muito grande, dependendo do nível de detalhamento cadastrado no repositório. Porém, este é um serviço chamado exclusivamente para um único componente.

4.3.3 *Download*

Após a visualização dos detalhes de um componente (*asset info*), pode-se fazer a chamada do serviço de *download* de componentes ou artefatos. O resultado do *download* é um arquivo no formato de *array* de `byte`, como mostra a figura 4.11.

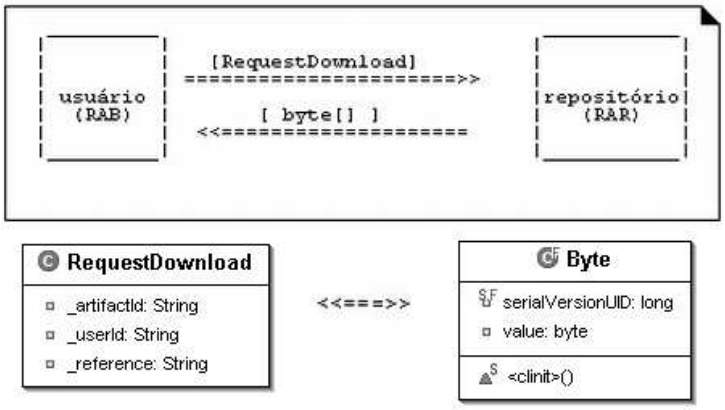


Figura 4.11: Classe `RequestDownload`

O objeto enviado `RequestDownload` apenas especifica o artefato a ser baixado, ou, caso seja passado para o Web Service responsável pelo *download* do pacote RAS, o `RequestDownload` estabelece qual componente será baixado.

Como apresentado na figura 4.5, seção 4.2.1, a interface de detalhamento do componente exibe a lista de artefatos que o compõe. Esta visualização torna possível realizar o *download* de cada artefato separadamente, chamando o serviço de *download* e informando o artefato desejado.

Da mesma forma, é possível realizar o *download* de todo o componente empacotado. Esta trasmissão é possível através do empacotamento do componente em formato RAS. Neste formato, as informações relacionadas ao ativo são descritas em um arquivo `rasset.xml` e os artefatos são armazenados em diretórios relacionados ao tipo de cada artefato. Ou seja, artefatos classificados como `requirements`, são armazenados no diretório `requirements`. O arquivo `rasset.xml` é armazenado no diretório raiz, que é compactado gerando um arquivo `.ras` , por exemplo `struts.ras`. Um exemplo desta estrutura é exibido na figura 4.12, onde são compactados arquivos de `requirements, test, implementation` e `model`, além do arquivo `rasset.xml`.

No exemplo apresentado, é gerado um pacote RAS. O arquivo `struts.ras` é um arquivo compactado contendo todos os arquivos exibidos na figura, com destaque para o arquivo `rasset.xml`, que fornece as informações de classificação do componente. Em ferramentas que utilizam o padrão RAS, este pacote gerado pode ser importado, mesmo que tal ferramenta não faça parte da rede de compartilhamento, justificando mais uma vez o requisito inicial e a preocupação constante da utilização de padrões para maior usabilidade do modelo gerado.

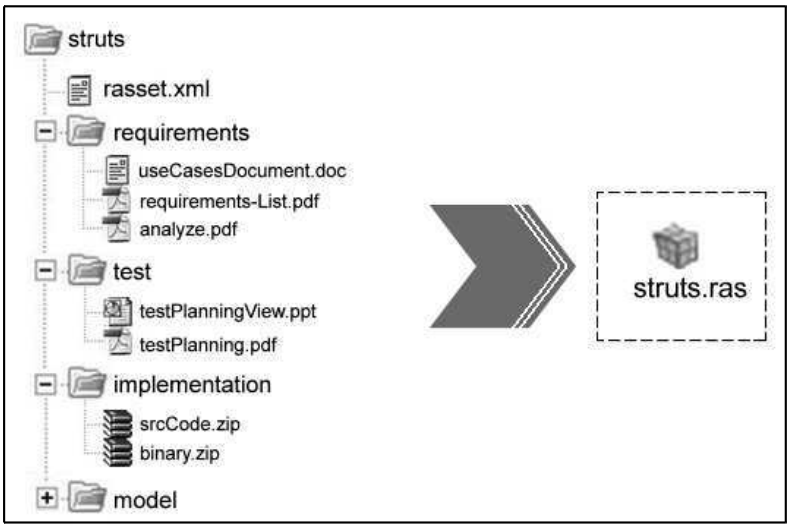


Figura 4.12: Estrutura do pacote RAS gerado para *download* do componente

4.4 Avaliação de desempenho

Os testes de desempenho foram realizados para se ter uma melhor avaliação de como os repositórios se comportam com múltiplos acessos. E se os RABs apresentam algum resultado inesperado quando realizamos a busca em múltiplos repositórios.

Um dos fatores importantes dos testes de desempenho é especificar o escopo dos testes, principalmente os gargalos do sistema. Os testes devem ser feitos em cima destes gargalos, para avaliação de situações críticas.

4.4.1 Definindo o escopo

O objetivo deste tipo de teste é verificar se o sistema pode manipular o volume de dados e transações recebidas especificadas no modelo de implementação do usuário e verificar se apresenta o tempo de resposta necessário. Testes de desempenho são fundamentais para que se conheça as capacidades de uma arquitetura.

Como se trata de uma arquitetura distribuída e acessível, onde todos os interessados possam participar e todos os participantes possam escolher livremente onde buscar os componentes, poderemos encontrar situações onde um grande número de usuários faz busca em um único repositório. Consideramos que o principal gargalo da rede seja exatamente este, e desta forma os testes se concentrarão em avaliar a capacidade de um repositório para atender um conjunto de requisições de busca.

Serão simulados diversos acessos distribuídos a um repositório. Este tipo de simulação também avalia, de certa forma, o poder de escalabilidade da rede, uma vez que para

que a rede seja realmente escalável quanto ao número de participantes, situações com concentração de busca em um repositório não podem acarretar situações inesperadas, como resultado incorreto da busca.

Além da busca, o assetInfo é outra funcionalidade que pode limitar a escalabilidade. Se vários RABs fazem busca em um único RAR, provavelmente as requisições de assetInfo também serão em grande número. Desta forma, o escopo dos testes de desempenho envolve também esta funcionalidade.

4.4.2 Ferramenta de testes

Como ferramenta para a realização dos testes de performance foi escolhido o JMeter [JME06], da Jakarta[1]. Além de ser uma ferramenta gratuita e open-source, o JMeter é especializado na execução de testes de stress em aplicações Web e já foi utilizado com sucesso em projetos corporativos, como no centro de desenvolvimento da Ci&T [2].

O JMeter possui dois conceitos fundamentais: *thread groups* e *threads*. O primeiro representa uma seqüência de requisições, numa ordem determinada, que deve ser disparada contra o sistema a ser testado. Uma analogia válida é considerar o *thread group* igual ao cenário de teste. Já a *thread* representa um agente de execução do *thread group*. Pode-se considerar que uma *thread* representa um usuário, apesar do número de usuários que se pretende simular não ser igual ao número de *threads* configurada na ferramenta.

Para maiores informações sobre o JMeter e sua utilização, uma ótima referência é o Guia de Utilização da ferramenta, disponível no diretório de instalação da mesma ou na página do produto na Internet [JMU06].

4.4.3 Cenários de teste

Alguns cenários foram criados, visando atender todo o escopo definido para os testes de desempenho. Antes de executar os testes, vale definir as configurações sob as quais os testes serão realizados, levando-se em consideração a infra-estrutura do laboratório. As tabelas 4.1, 4.2, 4.3 apresentam os pontos que foram utilizados na rede, e seus *access points* correspondem aos endereços dos Web Services, disponíveis através de interfaces WSDL. O *acess point* é acessível apenas internamente na rede onde os testes foram realizados. Não foram executados testes de desempenho com usuários externos, embora a rede já tenha sido acessada externamente.

É importante considerar também que os testes foram realizados em computadores populares dos membros do laboratório, com configuração bastante simples. Em média, as máquinas possuiam 512M de memória RAM, com processadores PentiumIII 550Mhz.

[1] Jakarta: http://jakarta.apache.org
[2] Ci&T Software - http://www.cit.com.br

Tabela 4.1: Yellow Pages habilitadas para os testes.

AccessPoint	**Nome**
http://lab05:8080/rar/inquiry	Unicamp RAR

Tabela 4.2: Repositórios cadastrados nas Yellow Pages de teste.

AccessPoint	**Nome**
http://galileo:8080/rar/services/SearchWebService?wsdl	Marcilio RAR
http://cit05:8080/rar/services/SearchWebService?wsdl	Labs RAR

Tabela 4.3: Repositórios cadastrados na base de conhecimento do RAB utilizado no teste.

AccessPoint	**Nome**
http://galileo:8080/rar/services/SearchWebService?wsdl	Marcilio RAR
http://epicuro:8080/rar/services/SearchWebService?wsdl	Isabelle RAR
http://mendel:8080/rar/services/SearchWebService?wsdl	Edy RAR

Foram definidos quatro cenários sob os quais os testes deveriam ser realizados. Todos focaram na busca distribuída e no detalhamento, considerando a utilização ou não da base de conhecimento e também considerando ou não a conulta nas yellow pages.

Os cenários testados foram os seguintes:

- Cenário 1 - Busca simples - local e distribuída
- Cenário 2 - Busca utilizando base de conhecimento e Yellow Pages.
- Cenário 3 - Busca avançada - local e distribuída
- Cenário 4 - Busca utilizando apenas Yellow Pages.

Em todos, encontrou-se o mesmo comportamento com o crescimento da rede. No subtópico a seguir apresentamos maiores detalhes sobre os testes do cenário 1, com gráficos apresentando o tempo de resposta para cada passo de acordo com o número de acessos simultâneos (*threads*).

Vale ressaltar que o *asset info* apresentou grande tendência a tornar a rede mais lenta, afirmando a decisão correta em criar esta etapa de detalhamento de forma separada, e que a busca só retorne informações básicas de cada componente, para não comprometer a performance da rede.

Busca simples - local e distribuída (testes no cenário 1)

A tabela 4.4 apresenta os passos realizados na execução do teste:

Tabela 4.4: Passos de teste, Cenário 1.

Passo	**Descrição**
OPÇÃO DE BUSCA	1. Seleciona, em preferências, que a busca deve ser em todos os repositórios e não utilizar busca nas yellow pages. 2. Escolhe que a busca será local e distribuída 3. Digita o texto de busca : "banco de dados".
RESULTADO	1. Visualiza os resultados e clica em hibernate, do repositório RAR Laboratório.
ASSET INFO	1. Visualiza os detalhes e volta no navegador 2. Clica no componente "ADOConnection" 3. Visualiza o detalhes e clica em "busca avançada".

Os gráficos a seguir apresentam os testes para diferentes quantidades de acessos remotos simultâneos:

1. Execução de 1 *thread*
2. Execução de 10 *threads* concorrentes
3. Execução de 25 *threads* concorrentes
4. Execução de 50 *threads* concorrentes

A figura 4.13 apresenta o resultado de uma busca com uma *thread* apenas.

A configuração das preferências, como são tratadas apenas localmente, é muito rápida. Assim como a busca e o *asset info*.

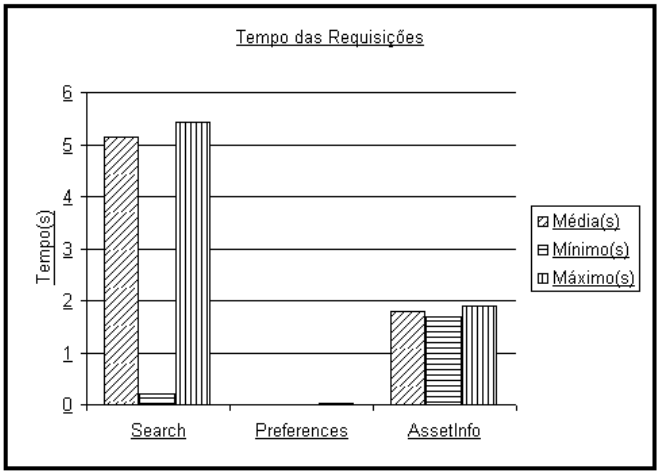
Neste caso, o tempo máximo da busca é o mais alto, pois a busca é disparada para vários repositórios. E, propositadamente, um dos repositórios da rede era uma máquina com servidor de banco de dados e ferramentas de desenvolvimento sendo utilizadas. Ou seja, simulando máquinas em utilização e com pequeno porte.

A figura 4.14 exibe o resultado para a simulação de 10 *threads*. Ou seja, o JMetter executa 10 vezes, paralelamente, o script de teste apresentado na tabela 4.4. Percebe-se ainda que a busca ainda é realizada no tempo satisfatório, sendo limitada (tempo máximo) devido a realização em computadores muito sobrecarregados. O *asset info*, agora feito 10 vezes em paralelo em uma máquina, ainda possui um tempo de resposta satisfatório.

Com o crescimento do número de execuções simultâneas para 25, conforme ilustra a figura 4.15, acarreta na sobrecarda da máquina na qual era realizado o *asset info*.

Número de Threads: 1

URL	Contador	Média(s)	Mínimo(s)	Máximo(s)
Search	4	5,155	0,200	5,435
Preferences	2	0,008	0,000	0,016
AssetInfo	2	1,788	1,687	1,890

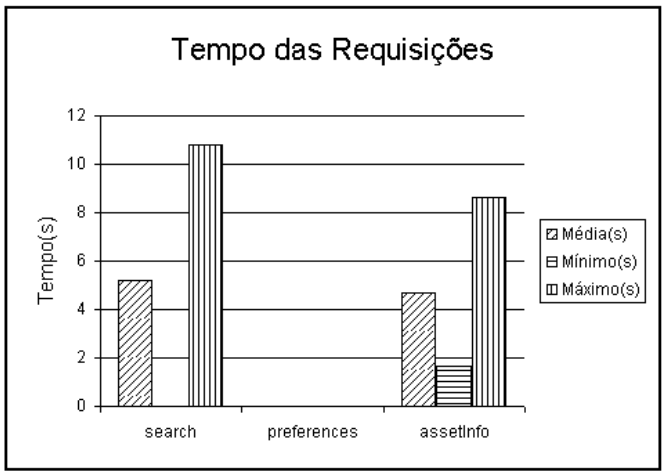


Tempo médio: 3,027 s *(média de tempo por número de requisições)*

Figura 4.13: Gráfico relacionado à execução de uma *thread*

Número de Threads: 10

URL	Contador	Média(s)	Mínimo(s)	Máximo(s)
search	40	5,195	0,000	10,781
preferences	20	0,007	0,000	0,016
assetInfo	20	4,704	1,672	8,656

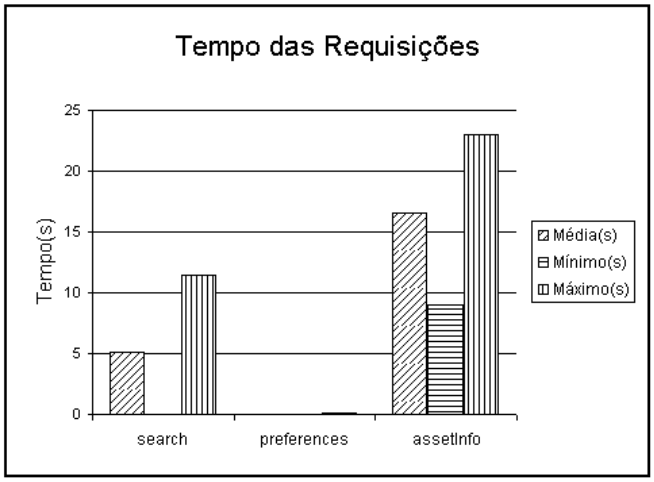


Tempo médio: 3,775 s *(média de tempo por número de requisições)*

Figura 4.14: Gráfico relacionado à execução de 10 *threads*

Número de Threads: 25

URL	Contador	Média(s)	Mínimo(s)	Máximo(s)
search	100	5,116	0,000	11,436
preferences	50	0,020	0,000	0,110
assetInfo	50	16,607	8,975	23,012



Tempo médio: 6,715 s *(média de tempo por número de requisições)*

Figura 4.15: Gráfico relacionado à execução de 25 *threads*

Ou seja, a busca continua mostrando o mesmo comportamento, uma vez que é uma operação simples e se limita apenas a ser distribuída. Porém, o *asset info* é realizado diretamente em uma única máquina e trata de uma operação muito mais "cara" tratando-se de processamento. Este comportamente justifica a decisão de desenvolver uma operação separada para o detalhamento dos componentes, já prevendo que esta operação sobrecarregaria a rede caso fosse disparada para todos os repositórios paralelamente. Esta situação, de 25 chamadas de *asset info* paralelas é mais rara, uma vez que o usuário pede este detalhamento apenas de um componente por vez, e isto é feito pontualmente em um único repositório.

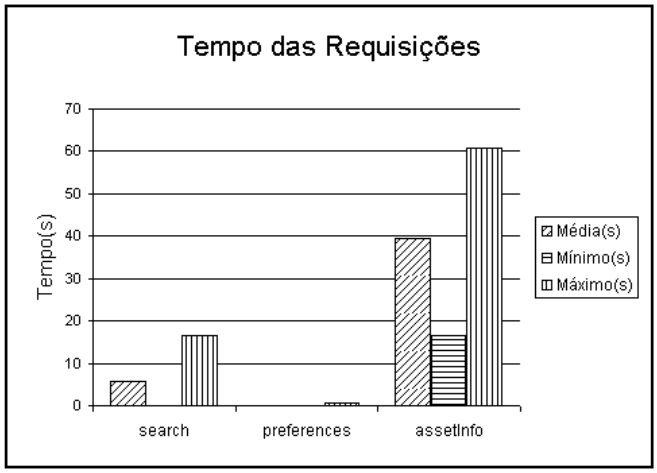
Por fim, a figura 4.16 apresenta o gráfico que ilustra o teste com realização de 50 execuções paralelas. Os testes realizados foram bastante proveitosos, ilustrando bem a previsão antes realizada. A funcionalidade de *asset info* é realmente muito pesada, uma vez que busca em todo o banco de dados e retorna todas as informações de um componente específico. Uma máquina realizando esta busca muitas vezes em paralelo tem seu funcionamento comprometido. Nos testes não havia duplicidade de conteúdo, ou seja, cada componente encontrava-se apenas em um único repositório.

A figura 4.17 apresenta um gráfico geral, com o comportamento da rede de acordo com o crescimento do número de requisições.

Percebe-se que as chamadas paralelas de *asset info* a um computador comum poderiam

Número de Threads: 50

URL	Contador	Média(s)	Mínimo(s)	Máximo(s)
search	200	5,613	0,000	16,482
preferences	100	0,053	0,000	0,547
assetInfo	100	39,587	16,600	60,797



Tempo médio:	12,716 s	*(média de tempo por número de requisições)*

Figura 4.16: Gráfico relacionado à execução de 50 *threads*

Número de threads	tempo médio de execução
1 Thread	3,027 s
10 Threads	3,775 s
25 Threads	6,715 s
50 Threads	12,716 s
75 Threads	21,163 s
100 Threads	39,840 s

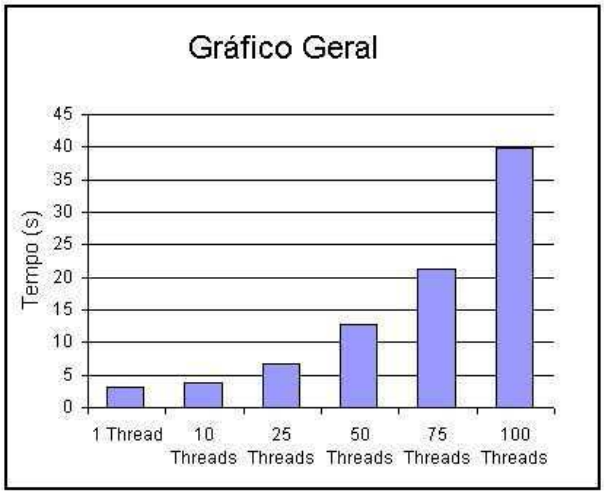


Figura 4.17: Ttempo de execução médio no cenário 1.

sobrecarrega-lo.

Observamos que acontece um crescimento muito grande do tempo de resposta à medida que o número de requisições aumenta. A partir dos gráficos já apresentados, pode-se concluir que este aumento se deve aos pedidos de *asset info* em um único repositório. Considerando que estes pedidos serão descentralizados, o *asset info* não comprometerá tanto a rede.

Este problema de desempenho acontece por não terem sido criados mecanismos específicos, nesta primeira proposta, para tratar possíveis sobrecargas em um único repositório. Para não comprometer pontualmente a rede, com a queda temporária de repositórios devido à sobrecarga, pode se montar mecanismos de controle das entrada de requisições. Neste contexto, um repositório montaria localmente uma fila de requisições de detalhamento de componentes, executando apenas 10 em paralelo. Seria uma solução com baixa complexidade e que proporcionaria maior estabilidade aos repositórios mais requisitados da rede.

4.5 Considerações finais

Este capítulo apresentou a arquitetura proposta para compartilhamento de componentes de *software*. Na definição da arquitetura, foram considerados os dois principais modelos de distribuição P2P, dos quais se adotou algumas importantes características, como busca direta entre pontos da rede e utilização de um servidor de referência, porém sem criar nenhuma dependência de qualquer ponto.

Foi apresentado o modelo de distribuição proposto, com definição dos papéis para diferentes participantes, definição de alguns pontos importantes em redes P2P como política de compartilhamento e tecnologia de comunicação.

Também foi apresentado o protocolo de distribuição implementado, totalmente baseado no modelo de representação definido no capítulo 3.

Por fim, foram exibidos os resultados de parte dos testes de desempenho realizados na rede, já em funcionamento. O resultado foi bem previsível, mostrando um comprometimento de desempenho nos servidores que recebem muitos pedidos de *asset info* paralelos.

O próximo e último capítulo ressalta os principais resultados obtidos, trabalhos futuros e conclusões.

Capítulo 5

Conclusão e trabalhos futuros

Este projeto foi focado na concepção do modelo de repositório para componentes de *software* e no desenvolvimento da arquitetura e mecanismo de *resource discovery* (descoberta de repositórios), aplicado ao compartilhamento de componentes de *software*.

Foi apresentada a definição da arquitetura de uma rede P2P para distribuição e compartilhamento de componentes de *software*, visando montar uma rede descentralizada, na qual qualquer participante possa se conectar, fornecer e buscar componentes.

Apresentou-se também a concepção de um modelo de repositório para armazenamento de componentes. Estes repositórios podem se comunicar através de um protocolo de rede desenvolvido, montando uma rede de compartilhamento P2P.

A arquitetura proposta foi implementada e aplicada em uma rede real, a RCCS, que já está integrada em ferramentas e novos projetos. Além da arquitetura, o modelo de repositório proposto e implementado está sendo utilizado também em novos trabalhos de P&D.

Vale ressaltar que, durante o projeto, observou-se a crescente quantidade de pesquisas e projetos relacionados ao reuso de *software*, sobretudo da aplicação de componentes. Também foi possível notar o surgimento de eventos específicos para *Peer-to-Peer*, como o Workshop de *Peer-to-Peer*, que acontece junto ao SBRC (Simpósio Brasileiro de Redes de Computadores). A primeira edição do Workshop foi em 2005, na qual foi apresentada parte deste trabalho. Além disso, algumas aplicações P2P têm se tornado extremamente úteis, em diferentes áreas, como ferramentas de comunicação instantânea de voz sobre IP [SKY06].

O RAS, principalmente por ser o padrão de modelagem de ativos do OMG, vem se tornando o principal padrão para ferramentas comercias. O modelo é constantemente atualizado e novas versões são disponibilizadas no site do OMG [RAS06].

As ferramentas de compartilhamento de conteúdo sempre foram vistas como ferramentas para *download* de música e vídeos. A proposta deste projeto foi de utilizar o conceito

de aplicações P2P em prol da produtividade dos desenvolvedores, visando facilitar o intercâmbio de informações úteis, neste caso componentes de *software*. Ou seja, foi possível utilizar tecnolgias e arquiteturas já existentes, em um contexto diferente, proporcionando uma solução específica para novas necessidades.

A seguir são apresentadas as principais realizações e trabalhos futuros do projeto.

5.1 Implementações

Os modelos propostos foram implementados com sucesso.

- Foi realizada a implementação da arquitetura distribuída proposta neste trabalho, utilizando-se o mecanismo de descoberta de repositórios e busca por componentes, juntamente com servidores *Yellow Page*, repositórios de componentes (RARs) e buscadores (RABs).

- A modelagem de banco de dados proposta foi implementada para o repositório de componentes da RCCS. O modelo mostrou-se suficientemente completo para suportar as informações das funcionalidades previstas para a rede.

- Os serviços de compartilhamento estabelecidos no protocolo proposto também foram implementados. Os repositórios disponibilizam tais serviços através de Web Services e podem comunicar entre si, realizando busca, detalhamento e *download* de componentes. O protocolo foi documentado e disponibilizado no site do laboratório de inovação. Trata-se de um documento (em formato pdf), que pode ser baixado na área de *downloads* do site[1].

As interfaces apresentadas na seção 4.2.1 foram as interfaces da implementação de referência desenvolvida. A partir desta implementação de referência, foi possível realizar testes de funcionalidade da rede.

5.2 Utilizações

Alguns resultados aqui atingidos já são utilizados em outros projetos.

- Atualmente, o modelo de repositório e banco de dados é utilizado no projeto de construção de um **Identificador Automático de Componentes de Software (IACS)**. O foco central do projeto consiste em montar um mecanismo capaz de reconhecer, em parques de aplicações já desenvolvidas, grupos de artefatos que

[1] Ci&T Labs - Laboratório de Inovação em Software Ci&T/Unicamp - `http://labs.cit.com.br/`

componham componentes. Visando, sobretudo, a extração destes componentes para catalogação e reuso. Através de pesquisas desenvolvidas no laboratório, estão sendo montados mecanismos capazes de identificar o agrupamento de artefatos relacionados sugerindo ativos de software (componentes, serviços etc.) candidatos à reutilização. Os principais tópicos de pesquisa relacionados são: "Metodologias de Representação de Ativos Digitais e Componentes de Software", "Mecanismos de identificação e classificação de componentes de *software*", "Algoritmos de identificação automática de componentes"e "Ferramentas escaláveis para exibição gráfica da dependência entre artefatos". O mecanismo de identificação de componentes de software (*component mining/harvesting*) permite realizar a reengenharia dos sistemas legados em sistemas baseados em componentes, identificando partes (componentes) reusáveis do sistema legado e permitindo reestruturá-lo em torno destas peças [LAB06].

- A arquitetura de distribuição foi aplicada em outra ferramenta de gerência de componentes de *software*. Atualmente, o DigitalAssets Manager[2], também implementa a arquitetura compartilhamento proposta. Com isso, através do DigitalAssets é possível realizar buscas distribuídas em redes montadas através da RCCS. Da mesma forma, através da RCCS é possível buscar componentes armazenados no DigitalAssets.

- Um novo projeto acadêmico pretende utilizar a RCCS para implementação de agentes de identificação e categorização de componentes. Este projeto é tema de mestrado no Departamento de Informática, da Universidade Federal de Viçosa[3].

5.3 Publicações

Além da realização de apresentação em eventos técnicos, como o III Fórum de Inovação Ci&T e na IX Semanda de Informática UFV, o trabalho originou duas publicações nacionais e uma internacional:

- O mecanismo de *Resource Discovery* foi publicado no I WorkShop de P2P, em Fortaleza-CE, sob o título de "*Resource discovery* em redes *Peer-to-Peer* aplicado à busca e compartilhamento de componentes de *software*" [OGN05b].

- A rede de compartilhamento de componentes de *software* foi apresentada no Salão de Ferramentas do 23 SBRC [OGN05a]. Neste salão, além de uma apresentação formal,

[2]DigitalAssets Manager - http://www.digitalassetsmanager.com.br
[3]DPI-UFV - http://www.dpi.ufv.br

a RCCS foi instalada em diferentes computadores para realização de demonstrações práticas.

- A integração da RCCS com o DigitalAssets Manager foi publicada no salão de demonstrações do OOPSLA (21st *Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*) [KRMS06]. Neste evento também foram feitas exibições práticas do mecanismo da RCCS integrado a outra ferramenta, e foi possível demonstrar ambas realizando buscas distribuídas, uma vez que o protocolo de compartilhamento é o mesmo.

O perfil prático do projeto pôde ser afirmado através das publicações, que foram, em sua maioria, em salões de demonstração. Nestes salões foi possível instalar e permitir a iteração dos participantes do evento com a ferramenta.

5.4 Trabalhos futuros

O trabalho aqui apresentado abriu uma série de possibilidades de pesquisa e deixa algumas oportunidades de melhoria nos conceitos e implementações. Algumas questões podem ser levantadas, como a utilização do RAS frente a banco de dados estáticos, considerando que o RAS pode ser extendido e isso não seria tarefa trivial para ser implementada em bancos estáticos. Seriam necessárias pesquisas específicas na área de banco de dados e modelagem para conseguir medir o impacto que estas extensões (os *profiles*) poderiam causar.

Da mesma forma, considerando que o RAS possibilita a representação de interfaces dos componentes, vale pensar, futuramente, em considerar características específicas de cada interface no momento da busca. Isso tornaria a busca mais complexa e, possivelmente, mais lenta. Porém, possibilitaria a utilização de atributos de interface no momento de se encontrar um componente na rede.

É possível identificar outras possíveis evoluções. São destacadas a seguir dois pontos importantes, que tratam de replicação de conteúdo (quando for aplicável) e melhorar a escalabilidade e alcançabilidade do mecanismo de descoberta de repositórios.

5.4.1 Replicação de conteúdo

Uma notável desvantagem de redes *peer-to-peer* é a instabilidade que possa encontrar durante a busca por recursos. Ou seja, uma vez que o conteúdo está distribuído em pontos específicos e, muitas vezes, sem nenhuma responsabilidade de ficar 100% do tempo disponível, não é raro que um recurso existente na rede não esteja disponível devido a queda de um único ponto.

Este aspecto não foi abordado na arquitetura proposta, uma vez que os componentes de *software* podem ser parcialmente compartilhados, com disponibilização apenas de alguns artefatos. A duplicação deveria ser também parcial, e seria necessário o desenvolvimento de políticas mais avançadas e específicas para determinar o que replicar e onde replicar.

Além disso, é possível também a criação de sub-grupos de compartilhamento, onde o repositório pode ou não exibir componentes para determinados pontos da rede.

Novas pesquisas seriam bem vindas, no sentido de determinar políticas seguras para tais replicações, sem a violação do direito de um repositório querer ou não distribuir e replicar seu conteúdo em outros repositórios.

5.4.2 Escalabilidade e alcançabilidade

Através de sugestões recebidas em apresentações sobre o trabalho, foi possível identificar algumas possibilidades de evolução do modelo, promovendo maior escalabilidade e, ao mesmo tempo, aumentando o alcance das buscas realizadas em um ponto da rede.

Uma das sugestões seria utilizar mecanismos *hashing* para indexação dos componentes na rede, sem a necessidade da busca pontual em cada repositório. A característica de complexidade na representação de componentes torna esta tarefa mais difícil, uma vez que é necessário escolher um ou mais campos a indexar, e conforme visto no capítulo 3, a modelagem do componente possui inúmeros campos e meta informações que também deveriam ser indexadas para não perder a eficiência da busca.

Desta forma, podemos visualizar evoluções consideráveis nos mecanismos de busca distribuída aplicados a componentes de *software*.

Apêndice A

Glossário

A seguir são apresentados alguns termos técnicos utilizados no texto. Os conceitos apresentados se referem às definições técnicas no contexto deste trabalho:

- ***Artefato***: São os documentos que constituem a solução proposta em um componente. Segundo o RAS, os artefatos são arquivos empacotados junto ao componente de *software*.
- ***Asset Info***: Detalhamento do componente. É a funcionalidade que fornece todas as informaçõs de um componente de *software*. Neste objeto retornado no *asset info*, além dos atributos (*Classification*), são transmitidos as informações de *Solution* e *Usage*.
- ***Ativo digital***: É um conteúdo digital. Desde um arquivo simples até um conjunto de arquivos.
- ***Busca por componentes***: Busca realizada diretamente nos repositórios, a partir dos serviços disponibilizados.
- ***CIA***: *Centralized Indexing Architecture*. Arquitetura P2P híbrida, com um servidor central onde é realizada a busca por conteúdo, e uma vez encontrado, o servidor central informa em qual ou quais pontos armazenam o conteúdo buscado. A partir deste momento a comunicação é feita direta entre os pontos, independente do servidor central.
- ***Classification***: No RAS, é o conjunto de informações e meta dados sobre um componte.
- ***Componente de software***: São módulos de *software* construídos, testados e reutilizáveis que oferecem um conjunto de serviços através de uma ou mais interfaces definidas.

- ***Detalhamento***: Chamada do serviço que captura informações detalhadas de um único componente (*Asset Info*).
- ***DIFA***: *Distribuited Indexing with Flooding Architecture*. Arquitetura P2P pura, com busca baseada em algoritimos de *flooding*.
- ***Digital Asset***: Ativo digital. No contexto do trabalho, trata-se de componente de *software*.
- ***DIHA***: *Distribuited Indexing with Hashing Architecture*. Arquitetura P2P pura, com busca baseada em algoritimos de indexação por tabela *hashing*.
- ***Download***: Chamada do serviço de obtenção de um arquivo do repositório. O usuário pode realizar o *download* de um componente inteiro ou de apenas um de seus artefatos.
- ***Flooding***: Tipo de algoritmo de busca no qual o ponto da rede dispara a busca para os seus vizinhos, que é novamente transmitida para os vizinhos dos vizinhos e assim por diante. Como trata-se de uma busca por exaustão e que sobregarregaria a rede, geralmente é estabelecido um tempo de vida (número de saltos) na busca, e isso pode fazer com que algum ponto mais distante da rede não seja visitado.
- ***Hashing***: Uma função de espalhamento (função *hash*) que transforma uma determinada chave em um endereço. Este endereço é usado como a base para o armazenamento e recuperação de registros.
- ***HIA***: *Hierarchic Indexing Architecture*. Arquitetura P2P híbrida. A principal diferença para a CIA é que neste caso existem vários servidores de busca, conhecidos como super-nós. Estes super-nós são responsáveis pela busca de algum recurso e indicam onde o recurso se encontra. Caso algum super-nós esteja ausente na rede, seus nós utilizam outro super-nós para realizar a busca.
- ***Peer-to-Peer*** *(P2P)*: Comunicação ponto a ponto, onde os participantes atuam como servidor e como clientes, e a comunicação se realiza diretamente entre os pontos, sem dependência de um servidor central.
- ***RAS***: *Reusable Asset Specificaton*. Modelo adotado pelo *Object Management Group* (OMG) como padrão internacional de representação de ativos digitais.
- ***RCCS***: Rede de Compartilhamento de Componentes de Software. Projeto desenvolvido no Laboratório de Inovação em *Software* da Unicamp, com o principal objetivo de montar uma rede pública para compartilhamento de componentes de *software*, baseada em padrões abertos e em comunicação através de Web Services.

- ***Repositório***: Computador ou cluster que armazena componentes de *software* e provê serviços através dos quais são disponibilizados os componentes.
- ***Resource discovery***: Busca por repositórios que compartilham componentes na RCCS. Através do *resource discovery* são encontrados os repositórios nos quais é possível buscar componentes.
- ***Solution***: No RAS, é o conjunto de artefatos que constituem o ativo digital. O *solution* contém e classifica os artefatos, que podem ser artefatos de requisitos, modelagem, implementação e teste.
- ***Usage***: No RAS, é o conjunto de informações sobre a utilização de um ativo digital. Apresenta as possíveis customizações e configurações necessárias para utilização do ativo.
- ***Yellow Pages***: São os servidores de referência da RCCS. Nas *Yellow Pages* são armazenadas as referências para os repositórios da rede e disponibiliza serviços para que um repositório encontre o outro.

Referências Bibliográficas

[AACY04] D. Angluin, J. Aspnes, L. Chen, and Y. Yin. Fast construction of overlay networks. Scientific Literature Digital Library, 2004. Disponível em `HTTP://citeseer.ist.psu.edu/angluin04fast.html`. Último acesso em 30/03/2006.

[AH00] E. Adar and B. Huberman. Free Rinding on Gnutella. *First Monday. Peer-Reviewed Journal on the Internet*, 5(10):1–14, 2000.

[DW99] D. F. D'Souza and A. C. Wills. Objects, Components, and Frameworks With UML: The Catalysis Approach. Addison Wesley. USA., 1999.

[eMu06] eMule-Project.net. Site oficial do emule, 2006. `http://www.emule-project.net/`. Último acesso em 12/11/2006.

[GNU06] Gnutella.com, 2006. `http://www.gnutella.com/`. Último acesso em 02/11/2006.

[GOO06] Sobre o Google, 2006. `http://www.google.com.br/intl/pt-BR/about.html`. Último acesso em 02/11/2006.

[HBDL99] M. Harchol-Balter, T. Leighton D., and Lewin. Resource Discovery in Distributed Networks. *Proc. 15th ACM Symp. on Principles of Distributed Computing*, pages 229–237, 1999.

[IBM04] Technology options for application integration and extended enterprise patterns. IBM Red books, 2004. Disponível em `http://www.redbooks.ibm.com/redpapers/pdfs/redp3840.pdf`. Último acesso em 02/11/2006.

[ICQ06] About the ICQ, 2006. `http://company.icq.com/info/community.html`. Último acesso em 02/11/2006.

[JGJ97] L. Jacobson, M. Griss, and P. Jonsson. Software Reuse: Architecture, Process and Organization for Business Success. Addison Wesley. USA., 1997.

[JME06] JMeter Jakarta, 2006. `http://jakarta.apache.org/jmeter`. Último acesso em 12/11/2006.

[JMU06] JMeter - User manual, 2006. `http://jakarta.apache.org/jmeter/usermanual/index.html`. Último acesso em 12/11/2006.

[KAZ06] Kazaa, 2006. `http://www.kazaa.com/us/index.htm`. Último acesso em 02/11/2006.

[KR04] D. R Karger and M. Ruhl. Diminished chord: A protocol for heterogeneous subgroup formation in peer-to-peer networks. The 3rd International Workshop on Peer-to-Peer Systems, San Diego, CA, USA, February 26-27, 2004.

[KRMS06] Bacili K. R. and Oliveira M. S. Digitalassets manager, managing and sharing software components. OOPSLA - 21st Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications - October 22-26, 2006 - Portland, Oregon, 2006.

[Kru92] C. W. Krueger. Software Reuse. *ACM Computing Surveys*, 24(02):131–182, 1992.

[Ku02] Raymond S. R. Ku. The creative destruction of copyright: Napster and the new economics of digital technology. University of Chicago Law Review, 2002. Disponível em `http://ssrn.com/abstract=266964`. Último acesso em 12/11/2006.

[LAB06] Ci&t labs - laboratório de inovação em software ci&t/unicamp, 2006. `http://labs.cit.com.br/`. Último acesso em 02/11/2006.

[MET02] M. Morisio, M. Ezran, and C. Tully. Success and Failure Factors in Software Reuse. *IEEE Transactions on Software Engineering*, 28(04):340–357, 2002.

[MKL+03] D. S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja1, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-peer computing. HP Labs, Rutgers University. University of California at Santa Barbara, 2003.

[MSZ01] S. McIlraith, T. C. Son, and H. Zeng. Semantic Web Service. *IEEE Intelligent Systems*, 16(02):45–53, 2001.

[NAP06] Napster, 2006. `http://www.napster.com/`. Último acesso em 02/11/2006.

[OGN05a] M. S. Oliveira, I. Garcia, and A. Nunes. RCCS, Rede de Compartilhamento de Componentes de Software. X Salão de Ferramentas - XXIII Simpósio Brasileiro de Redes de Computadores - Fortaleza, CE - maio de 2005, 2005.

[OGN05b] M. S. Oliveira, I. Garcia, and A. Nunes. *Resource Disocovery* em uma rede *Peer-to-Peer* aplicado a busca e compartilhamento de componentes de *software*. I Workshop de *Peer-to-Peer*- XXIII Simpósio Brasileiro de Redes de Computadores - Fortaleza, CE - maio de 2005, 2005.

[OMG06] OMG, 2006. Object Management Group. `http://www.omg.org`. Último acesso em 02/11/2006.

[PH04] J. S. Park and J. Hwang. Roled-based access control for collaborative enterprise in peer-to-peer computing environments. Proceedings of 8th ACM Symposium on Access Control Models and Technologies (SACMAT 2003), pages 93-99., 2004.

[RaACS+04] J. Rocha, M. Domingues ad A. Callado, E. Souto, G. Silvestre, C. Kamienski, and D. Sadok. Peer-to-Peer : Computação Colaborativa na Internet. Minicurso, SBRC 2004, 2004.

[RAS06] RAS. Reusable Asset Specification, OMG Available Specification. Object Management Group, 2006. Disponível em `http://www.omg.org/technology/documents/formal/ras.htm`. Último acesso em 01/11/2006.

[RFC06] RFC-RCCS - Ci&T Labs - Laboratório de Inovação em Software Ci&T/Unicamp, 2006. `http://labs.cit.com.br/sitelab/downloads/RFC-RCCS_Protocol.pdf`. Último acesso em 02/11/2006.

[Rip01] M. Ripeanu. Peer-to-Peer Architecture case study: Gnutella Network. International Conference on Peer-to-peer Computing, 2001.

[RRRW04] F. R. Pellissari R. R. Righi and C. M. Westphall. P2p-role:uma arquitetura de controle de acesso baseada em papéis para sistemas colaborativos peer-to-peer. VI Workshop em Segurança de Sistemas Computacionais (WSeg/SBRC2004), pages 285-296. Gramado, Rio Grande de Sul, Brasil, 2004.

[RRRW05] F. R. Pellissari R. R. Righi and C. M. Westphall. Controle de acesso e combate a usuários caroonas em sistemas peer-to-peer. I Workshop de Peer-to-Peer-XXIII Simpósio Brasileiro de Redes de Computadores (WP2P/SBRC)- Fortaleza, CE - maio de 2005, 2005.

[RS90] OF Rana and K Stout. What is Scalability in Multi-Agent Systems? Fourth Annual International Conference on Autonomous Agents, 1990.

[SKY06] Skype, 2006. `http://www.skype.com/`. Último acesso em 02/11/2006.

[UDD06] Uddi.org - specification, 2006. `http://www.uddi.org/specification.html`. Último acesso em 12/11/2006.

[WMS02] J. Walkerdine, L. Melville, and I. Sommerville. Dependability Properties of P2P Architectures. In the Second International Conference on Peer-to-Peer Computing (P2P'02), 2002.

[WN02] K. C. Weber and C. J. Nascimento. Brazilian Software Quality 2002. he 24th IEEE International Conference on Software Engineering (ICSE), EUA, pp. 634-638., 2002.