UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TÉRCIO OLIVEIRA DE ALMEIDA

Ras4Nexus - Promovendo Reuso utilizando o Gerenciador de Repositórios Nexus com o padrão RAS

Projeto de Diplomação

Prof. Dr. Marcelo Soares Pimenta
Orientador

Porto Alegre, dezembro de 2009

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS 5

LISTA DE FIGURAS 6

LISTA DE TABELAS 7

RESUMO 8

ABSTRACT 9

1 INTRODUÇÃO 10

2 REUSO E REPOSITÓRIOS DE ARTEFATOS 12
2.1 Reuso: conceitos 12
2.1.1 Benefícios da prática de reuso 12
2.1.2 Obstáculos para prática de reuso 13
2.1.3 Tipos de reuso 13
2.1.4 O padrão RAS 16
2.1.5 Processo de reuso 20
2.2 Repositórios de Artefatos 21
2.3 Repositórios de Artefatos com suporte a Reuso 22
2.4 Ferramentas Existentes 23
2.4.1 Basic Asset Retrieval Tool (BART) 23
2.4.2 Component Repository (CORE) 24
2.4.3 Rational Asset Manager (RAM) 25
2.4.4 ARCSeeker 25
2.4.5 Asset Management Tool (AMT) 25
2.4.6 Maven 26
2.4.7 Archiva 26
2.4.8 Rasputin 26
2.4.9 Nexus 27

3 RAS4NEXUS: DESCRIÇÃO E USO 28
3.1 Sistema de Indexação 30
3.1.1 Disparadores para Indexação 30
3.1.2 Contextos de Indexação 30
3.1.3 Registrando índices RAS 31
3.2 Serviço de Busca 35
3.2.1 Critérios de Busca 37

3.3 Integridade dos Artefatos 37
3.4 Utilizando o Sistema: Passo a Passo 38
3.4.1 Submetendo um Pacote RAS 38
3.4.2 Pesquisando por Artefatos 40
3.4.3 Recuperando Artefato 40
3.4.4 Validação - Testes de Integração 42
3.4.5 Instalação do RAS4Nexus 42

4 CONCLUSÃO 45

REFERÊNCIAS 47

LISTA DE ABREVIATURAS E SIGLAS

OMG	*Object Management Group*
RAS	*Reusable Asset Specification*
IDE	*Integrated Development Environment*
JRE	*Java Runtime Environment*
XSD	*XML Schema Definition*
IDE	*Integrated Development Environment*
XML	*Extensible Markup Language*
CMMI	*Capability Maturity Model Integration*
MPSBR	Melhoria de Processos do Software Brasileiro
CBSD	*Component Based Software Development*

LISTA DE FIGURAS

Figura 2.1: Categorias de reuso 17
Figura 2.2: Seções principais do *Core RAS* 18
Figura 2.3: Dimensões de *assets* reusáveis 20
Figura 2.4: Fluxos de trabalho de um artefato reusável 20

Figura 3.1: Casos de Uso para disparadores de indexação 30
Figura 3.2: Classes responsáveis por tratamento de eventos 31
Figura 3.3: Diagrama de classes para indexação 32
Figura 3.4: Diagrama de sequência para o registro de repositórios 32
Figura 3.5: Diagrama de classes para extração de informações RAS 34
Figura 3.6: Diagrama de sequência para uso do RassetReader 34
Figura 3.7: Escolha do método de busca na interface 35
Figura 3.8: Classes responsáveis pelo direcionamento dos métodos de pesquisa . 36
Figura 3.9: Extensão no serviço de busca com a classe RasSearcher 36
Figura 3.10: Consulta aos campos RAS a partir do SearchEvent 37
Figura 3.11: Diagrama de Sequência para artefatos corrompidos 38
Figura 3.12: Seleção do repositório para envio 39
Figura 3.13: Estrutura do pacote com descritor RAS 39
Figura 3.14: Informações e seleção do artefato para envio 40
Figura 3.15: Conteúdo do arquivo rasset.xml 41
Figura 3.16: Pesquisa pelo termo “framework” no Nexus 41
Figura 3.17: Identificação do Artefato e ligação para baixar 42
Figura 3.18: Informações de dependência para o Maven 42
Figura 3.19: Testes de Integração - *RasPluginIndexingIT.java* 43

LISTA DE TABELAS

Tabela 2.1: Problemas para adoção do reuso 14
Tabela 2.2: Resumo comparativo 27

Tabela 3.1: Elementos do descritor RAS a serem indexados 33

RESUMO

O reuso de artefatos de software, apesar de ser uma prática advogada em todos os setores de desenvolvimento e defendida desde os primórdios da Engenharia de Software, não tem sido extensivamente aplicado nas organizações. Dentre as razões relacionadas ao aspecto técnico, está a falta de uma infra-estrutura que esteja integrada com todo o processo de desenvolvimento e suas diversas ferramentas, e que seja um ponto central para a aplicação do reuso. O objetivo deste trabalho é criar um protótipo deste ambiente, permitindo que os participantes de um processo de desenvolvimento possam adicionar, modificar, localizar e recuperar artefatos reusáveis a partir do mesmo repositório central. Para isto, foi implementada Ras4Nexus, uma extensão em um gerenciador de repositórios de software, o Nexus, para suportar o padrão de descritor de reuso criado pela OMG, o *Reusable Asset Specification* (RAS).

Palavras-chave: RAS, reuso de software, engenharia de software, repositório de reuso.

Promoting Reuse using Nexus Repository Manager with the RAS standard

ABSTRACT

The reuse of software artifacts, despite being a desired practice supported in every development sector and defended since the beginning of Software Engineering, has not been extensively applied in the organizations. Among the reasons related to the technical aspects, there is a lack of an infrastructure that integrates the entire development process and its various tools, being a central point to the reuse application. The objective of this work is to create a prototype of this environment, allowing participants of a development process to add, modify, find and recover reusable artifacts from the same central repository. To do so, an extension, named Ras4Nexus, was implemented in a repository manager software, Nexus, to support the standard created by OMG, the Reusable Asset Specification (RAS).

Keywords: RAS, software reuse, software engineering, reuse repository.

1 INTRODUÇÃO

O reuso de software vem sendo proposto desde o princípio dos estudos em Engenharia de Software (Mcilroy 1968). O objetivo desta prática é trabalhar de forma que artefatos planejados, construídos e testados possam ser reutilizados, evitando desperdiçar recursos em todas estas etapas novamente. Desta forma, sistemas podem ser construídos incrementalmente a partir de artefatos robustos e compartilhados, solucionando boa parte dos problemas do desenvolvimento de sistemas, visto que estes ocorrem devido a complexidade inerente do software (Brooks 1987).

Embora essa prática seja considerada por muitos autores a mais provável "bala de prata" (Cox 1990), existem muitos obstáculos que impedem sua aplicação extensiva. Como se torna necessário modificar toda uma cultura organizacional, surgem questões de cunho técnico, psicológico, sociológico e econômico que se interpõem à adoção da prática do reuso (Tracz 1988). Para sua efetivação, é necessário que artefatos sejam construídos com o objetivo de serem reusados. Além disso, precisam estar organizados de tal forma que sua busca e adaptação sejam mais eficientes do que a reconstrução destes (Krueger 1992).

Uma forma de organizar estes artefatos é mantendo-os em um repositório de reuso. Essa ferramenta tem como objetivo gerenciar estes artefatos, tornando-se um ponto central onde os participantes de um processo de desenvolvimento devem buscar por artefatos reusáveis. Quanto maior a quantidade e complexidade destes, mais fundamental para a adoção da prática essa ferramenta se torna (Sherif e Vinze 2003). Para lidar com isso, o repositório deve manter descrições úteis para a busca e critérios relevantes para a classificação relacionados a cada artefato. Ezran afirma que o material que descreve o artefato e o classifica deve estar empacotado juntamente com o artefato. Assim, outras ferramentas voltadas a reuso podem auxiliar na criação e manutenção dessas informações (Ezran 2002).

A OMG criou então o padrão RAS, para suprir a necessidade de uma especificação para empacotamento, descrição e classificação de artefatos reusáveis (OMG 2005). Porém, por ser um padrão recente, ainda são poucas ferramentas que o suportam, sendo a maior parte delas soluções proprietárias.

Este trabalho visa promover reuso utilizando um gerenciador de repositórios de software, o Nexus. Ele foi escolhido por ser de código aberto e fornecer características úteis para este objetivo, no tocante à gerência de artefatos: inserção, atualização, remoção, pesquisa e recuperação. Utilizando a mesma ideia de (da Rosa 2009), em RASPUTIN, criamos uma extensão para o Nexus, o Ras4Nexus, com o objetivo de adicionar suporte ao padrão RAS. Com isso, indexamos informações registradas no descritor RAS de cada pacote, capacitando o buscador do Nexus a recuperar artefatos a partir dessas informações. A principal diferença desta abordagem com relação ao trabalho em (da Rosa 2009), é que esta solução não está acoplada a repositórios Maven, permitindo lidar com diferen-

tes linguagens, projetos e repositórios de software. Já existe na versão paga do Nexus suporte a outros tipos de repositórios, como *OSGi Bundle* (OBR) e *Eclipse P2*, e caso os pacotes desses repositórios estejam no padrão RAS, serão detectados e tratados da mesma forma, aumentando a abrangência da solução.

O documento está estruturado como segue. No capítulo 2 apresentamos conceitos sobre reuso e sua adoção, buscando entender a função deste trabalho com relação a sua prática. Também, verificamos diferentes repositórios de software e repositórios de reuso, enfatizando suas características e permitindo uma comparação com essa proposta. No capítulo 3, descrevemos o Ras4Nexus, apresentamos sua modelagem e suas funcionalidades como ferramenta para aplicação do reuso, além de um passo-a-passo de sua utilização no gerenciador Nexus. No capítulo 5, concluímos apresentando os resultados, limitações e idéias de trabalhos futuros.

2 REUSO E REPOSITÓRIOS DE ARTEFATOS

Neste capítulo são explicados conceitos de reuso, as principais abordagens, sua importância, seus problemas e aspectos relacionados a sua aplicação prática. Também é feita uma revisão dos processos, os quais são: produção, identificação, consumo e gerenciamento. A partir daí, são analisadas as ferramentas existentes que se propõe a auxiliar nesses processos: os repositórios de artefatos.

2.1 Reuso: conceitos

Em um processo de desenvolvimento de software, surgem problemas recorrentes, que são resolvidos diversas vezes e por diferentes pessoas. A prática de reuso visa eliminar essa repetição de trabalho, focando recursos apenas em questões que ainda não foram tratadas (Krueger 1992).

O reuso pode ser aplicado em todas as etapas do processo de desenvolvimento e em diferentes níveis de abstração, tais como: código fonte, componentes, artefatos de desenvolvimento, padrões, *templates*, entre outros (Ambler 2009).

É necessário que haja planejamento prévio para que possa ser efetivamente praticado, pois a construção de artefatos reusáveis demanda mais tempo do que a construção dos demais artefatos. Eles precisam ser construídos, planejados e documentados para esse fim (Tracz 1988), para que o processo de busca, identificação e reutilização de um artefato seja menos custoso do que a construção completa de um novo artefato (Krueger 1992).

2.1.1 Benefícios da prática de reuso

Entre os diversos benefícios oferecidos pelo reuso, podemos citar (Almeida 2009):

- *Qualidade:* componentes reusáveis são sujeitos a rigorosos testes, por terem um risco maior, podendo prejudicar diversos sistemas. E como são usados diversas vezes em diferentes contextos, eles tendem a ficar cada vez mais robustos conforme os defeitos são encontrados e corrigidos, reduzindo a possibilidade de haver erros. Por isso, quanto mais um artefato é reusado, menor é a tendência de se encontrar novos defeitos neles.

- *Produtividade:* o trabalho fica voltado apenas a questões que ainda não foram resolvidas. Com isso, se torna possível aumentar a velocidade de produção, pois tanto o tempo de desenvolvimento e validação devem ser reduzidos. É comum observar um crescimento de 20% a 40% na produtividade.

- *Confiabilidade:* o uso de sistemas bem testados aumenta a confiabilidade do software desenvolvido. Além disso, o uso de um componente em diversos sistemas

aumenta a chance de que os erros sejam detectados e fortalece a confiança naquele componente. Além disso, os testes podem estar focados no software produzido, pois não é necessário testar novamente individualmente as subpartes reutilizadas, bastando testar a integração entre as partes e o funcionamento do todo.

- *Redução no tempo de desenvolvimento:* o reuso de artefatos prontos, como especificações, casos de uso e componentes, diminui o custo e tempo de desenvolvimento de um sistema.

- *Documentação:* a reutilização de software diminui a quantidade de documentação a ser escrita, porque apenas a parte desenvolvida de fato é que precisa ser documentada.

- *Tamanho das equipes:* problemas de comunicação e entrosamento impedem que uma equipe com o dobro do tamanho produza software duas vezes mais rápido. Se muitas partes desse sistema forem reutilizadas, ele pode ser desenvolvido por equipes menores, aumentando a produtividade e melhorando a comunicação.

2.1.2 Obstáculos para prática de reuso

Na busca de resolução dos problemas da Engenharia de Software, o reuso já foi considerado a grande "bala de prata" (Cox 1990). Entretanto, existe uma série de problemas que se interpõe a sua aplicação prática efetiva. Na tabela abaixo, estão descritos esses fatores, de acordo com (Kim e Stohr 1998), onde o foco do **Ras4Nexus** está em **negrito**.

2.1.3 Tipos de reuso

O reuso pode ser aplicado em todas as etapas de um processo de desenvolvimento, assim como em diversos níveis de abstração. Em (Ambler 2009), é feito um levantamento sobre as principais categorias de reuso, que estão descritas e comentadas a seguir.

- **Reuso arquitetado**

  O reuso arquitetado consiste na identificação, desenvolvimento e suporte de artefatos reusáveis de larga escala via uma arquitetura corporativa. A arquitetura corporativa pode definir componentes de domínio reusáveis, coleções de classes de domínio e negócio que funcionem juntas de forma a suportar um conjunto coeso de responsabilidades, ou domínios de serviço, agrupadas em um pacote coeso de uma coleção de serviços.

  Tem como característica fornecer um nível muito elevado de reuso entre aplicações. Mas também requer uma abordagem sofisticada para definir a arquitetura corporativa. E ainda requer uma equipe de reuso para dar suporte à evolução dos artefatos reusáveis ao longo do tempo.

- **Reuso de frameworks**

  Um *framework* pode ser definido como uma coleção de códigos-fonte, classes, funções, técnicas e metodologias, com o objetivo de facilitar o desenvolvimento de novas aplicações (Minneto 2007).

  Frameworks fornecem um bom ponto de partida para desenvolver uma solução para um problema em um domínio. Podemos revisar as boas práticas da indústria através

Tabela 2.1: Problemas para adoção do reuso

Categoria	*Aspectos Pesquisados*	*Impedimentos do Reuso*
Gerais	Definição de escopo	Falta de terminologia para a descrição de conceitos
	Aspectos econômicos	Investimento necessário para promover reuso de software; falta de modelo econômicos para explicitar os benefícios e custos do reuso de software
Técnicos	Processo de reuso	Falta de metodologia para criar e implementar o reuso de software
	Tecnologias de reuso	**Falta de recursos de software reutilizáveis e confiáveis; falta de tecnologias e técnicas para apoiar o reuso de software**
Não-técnicos	Aspectos comportamentais	Falta de comprometimento, encorajamento, treinamento e recompensas pelo reuso de software; síndrome do "não inventado aqui" (*not invented here*)
	Aspectos organizacionais	Falta de apoio organizacional para instituir o reuso de software; dificuldade para medir os ganhos do reuso
	Aspectos legais e contratuais	Problemas com direitos de propriedade intelectual e contratuais

deles. Funcionam encapsulando uma lógica complexa, permitindo que o foco do desenvolvimento esteja nos problemas pertinentes ao domínio.

O que ocorre também é que, por sua complexidade, a curva de aprendizado torna-se longa, demandando muito tempo e treinamento para que seja dominado em uma organização. Além disso, os *frameworks* geralmente funcionam em plataformas específicas, amarrando a organização a um único fornecedor. Além disso, *frameworks* raramente funcionam juntos, pois são construídos sobre bases arquiteturais diferentes.

- **Reuso de modelos**

  O reuso de modelos consiste na prática de utilizar um conjunto comum de *layouts* para artefatos-chave do desenvolvimento na organização, tais como documentos, modelos e códigos-fonte.

  A consequência de sua aplicação é um aumento na consistência e na qualidade dos artefatos de desenvolvimento, porque, como visto na seção 2.1.1, a qualidade dos modelos aumentará, evitando propagar erros para a construção dos artefatos.

  A sua prática é dificultada pela tendência dos participantes de não compartilharem as modificações aplicadas aos modelos com o restante da organização.

- **Reuso de padrões**

Padrões são abordagens documentadas para resolver problemas recorrentes de determinados contextos (Gamma 2000). Em geral, essa documentação possui quatro elementos essenciais:

- **Nome do padrão**: referência utilizada para descrever um problema de projeto, suas soluções e consequências em uma ou duas palavras;
- **Problema**: explica o problema e o seu contexto, descrevendo em que situação o padrão pode ser utilizado;
- **Solução**: descreve os elementos que compõem o padrão de projeto, seus relacionamentos, suas responsabilidades e colaborações;
- **Consequências**: descreve os resultados, vantagens e desvantagens na utilização do padrão, e como sua utilização afeta outros aspectos do sistema;

O uso de padrões melhora a manutenibilidade e a qualidade da aplicação por estar usando abordagens aplicadas inúmeras vezes por programadores experientes, que reconheceram ao longo do tempo seus pontos fortes e fracos, e os documentaram.

Dessa forma, para desenvolver código a partir de padrões o desenvolvedor não reutiliza código, mas sim os conceitos. Estes podem ser implementados em diferentes linguagens de programação e plataformas. Portanto não provêem uma solução imediata, pois precisam ainda ser aplicados e implementados corretamente.

- **Reuso de artefatos**

  O reuso de artefatos consiste em reutilizar, em um novo projeto, artefatos de desenvolvimento previamente criados. Esses artefatos podem ser casos de uso, documentos-padrão, modelos de domínio específico, procedimentos, recomendações ou, até mesmo, outras aplicações, como aplicações comerciais de "prateleira" (*commercial-off-the-shelf - COTS*).

  Existem artefatos resultantes de um processo de análise, chamado reuso de análise, como especificações lógicas completas ou subconjuntos delas (diagramas, descrições de dados ou descrições lógicas de conhecimento). Além disso, existem os artefatos resultantes do processo do projeto, como por exemplo, decisões de projeto. O seu reuso chama-se "reuso de projeto".

  O desenvolvedor pode optar por reutilizar o artefato tal como ele é ou utilizá-lo como exemplo para dar continuidade ao projeto. Além disso, o reuso de artefatos é fácil de ser realizado, porque a internet é facilitadora para se localizar e baixar artefatos.

- **Reuso de módulos**

  Módulos podem ser definidos como componentes, serviços ou bibliotecas de código. Os módulos são auto-suficientes e encapsulam somente um conceito.

  O reuso de módulos consiste em utilizar módulos pré-construídos e completamente encapsulados para o desenvolvimento de aplicações.

  A diferença entre o reuso de módulos e o reuso de código é que, no primeiro, não se tem necessariamente acesso ao código fonte. Além disso, o reuso de módulos oferece um escopo maior de reusabilidade porque basta "conectar" os módulos para funcionarem.

O uso em larga escala de determinadas plataformas e padrões provê um mercado grande o suficiente para que fornecedores criem e vendam módulos aos desenvolvedores a um custo baixo. Como geralmente são pequenos e muitos, surge a dificuldade de manutenção para os desenvolvedores, que precisam lidar com uma grande biblioteca deles.

É comum que o desenvolvedor não possa modificar um módulo, embora esta situação esteja mudando com o crescimento da popularidade da programação orientada a aspectos (POA). Também existe o problema da licença de uso, que podem dificultar, e até impedir, a reutilização de um módulo.

- **Reuso de código**

  A prática de reuso mais utilizada pelos desenvolvedores é a de reuso de código. Consiste no reaproveitamento de partes do código fonte.

  O reuso de código pode acontecer da melhor forma, com o compartilhamento de classes (herança e composição), coleções de funções e rotinas comuns. No pior caso, o reuso de código é executador através do "*copy&paste*", ou "copiar e colar", e eventualmente modificando-se parte do código copiado, o que pode dificultar a manutenção.

  Com a aplicação dessa técnica, vem a redução de esforços de desenvolvimento, pois a quantidade de código que o programador precisa escrever diminui. Porém, do ponto de vista de um projeto, códigos genéricos e reusáveis são geralmente mais complexos do que códigos feitos para um propósito específico, o que torna mais difícil seu desenvolvimento e manutenção. Um cuidado adicional necessário é evitar um alto nível de acoplamento ao reusar código, o que resultaria em dificuldades para realizar modificações nessas partes do sistema.

A Figura 2.1 apresenta os diferentes graus de benefícios que são obtidos em cada categoria de reuso (representados pela seta da esquerda), bem como os diferentes níveis de dificuldade para incorporar a prática de determinada categoria de reuso em um processo de desenvolvimento (representados pela seta da direita). O reuso de módulos, por exemplo, geralmente dá mais produtividade do que o reuso de código. A prática de reuso de código e de modelos é relativamente mais fácil de ser incorporada, pois as equipes de desenvolvimento somente precisam de um instrumento para localizar o artefato reusável para poder trabalhar com ele. Já a prática de reuso de um framework é bem mais difícil de ser absorvida, pois as equipes de desenvolvimento, além de conhecer uma forma de obtê-lo, precisam aprender a utilizá-lo.

2.1.4 O padrão RAS

Um *asset*, de acordo com o Object Management Group (OMG), é uma coleção de artefatos que fornecem uma solução para um problema em um determinado contexto (OMG 2005). Um artefato é um produto gerado no ciclo de desenvolvimento de software. Pode ser documento de requisitos, modelos, código-fonte, casos de teste, etc...

Para que *assets* possam ser utilizados pelas equipes de projeto, eles precisam ser documentados e disponibilizados em um repositório. Para documentar, empacotar e catalogar os diversos *assets*, o OMG criou a especificação RAS (*Reusable Asset Specification*) (OMG 2005). De acordo com esse padrão, eles são empacotados juntamente com um envelope de metadados que os descrevem. Um conjunto básico de metadados é definido pelo *Core RAS*. A Figura 2.2 apresenta as suas principais seções.

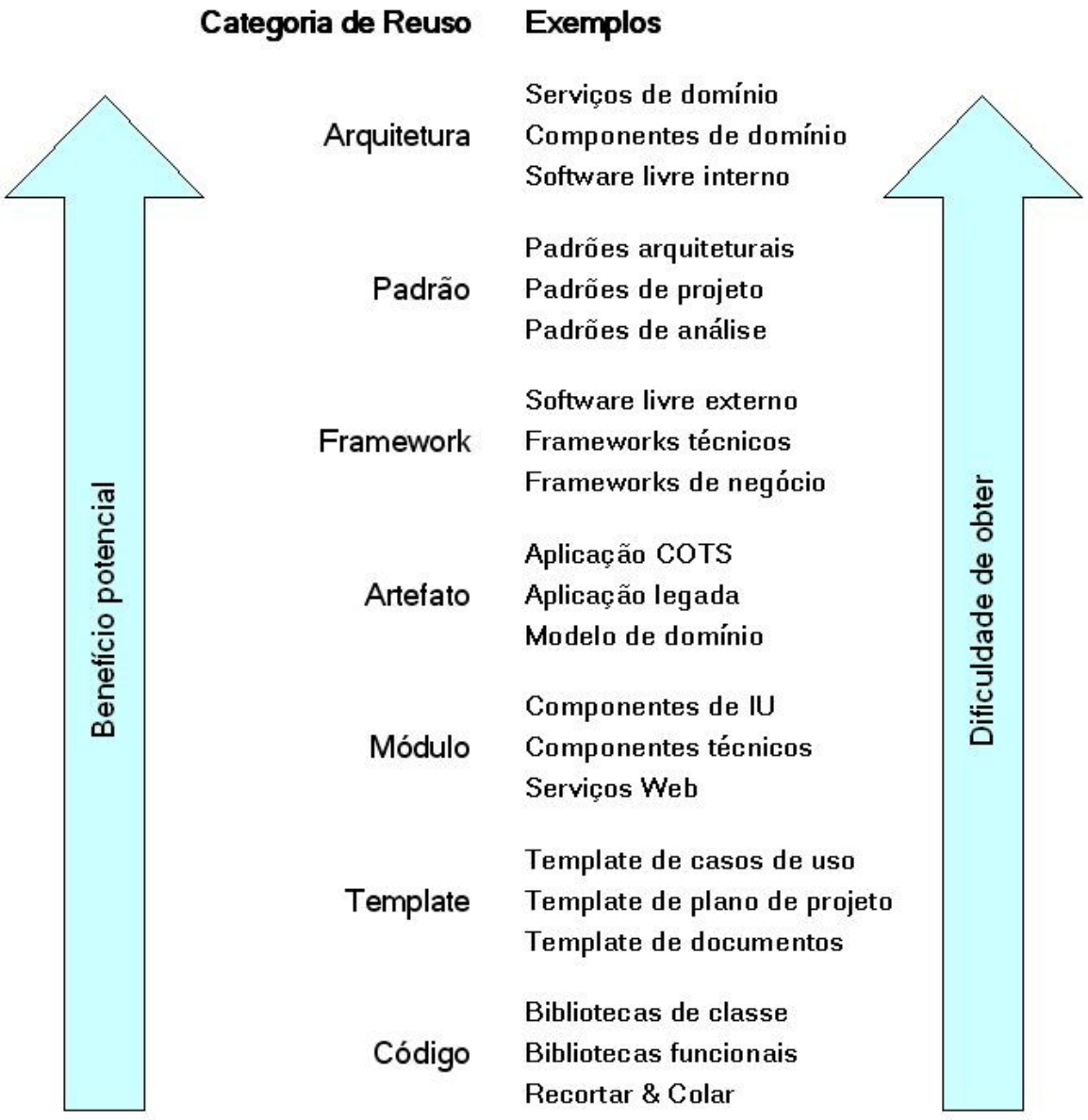

Figura 2.1: Categorias de reuso

Figura 2.2: Seções principais do *Core RAS*

Este modelo pode ser estendido por outros perfis permitindo a criação de tipos específicos para outros domínios de aplicação. A partir da Figura 2.2, podemos verificar o seguinte:

- **Perfil** (*profile*) na seção *Asset* identifica o perfil utilizado;
- **Classificação** (*classification*) contém descritores que descrevem as características e comportamentos chave dos elementos contidos no artefato. Isto é interessante para os consumidores e gerenciadores por conterem informações relativas ao domínio de aplicação, podendo facilitar a pesquisa e navegação por tópicos (*browsing*);
- **Solução** (*solution*) apresenta referências a artefatos que compõem a solução, como requisitos, modelos, especificação de interfaces, código fonte, etc;
- **Uso** (*usage*) contém informação de como aplicar e utilizar o *asset* usando os pontos de variabilidade, que permitem aos usuários a configuração de parâmetros. Um ponto de variabilidade é uma localização no *asset* que pode ter um valor fornecido ou customizado pelo consumidor. Algumas dessas orientações de uso podem ser automatizadas através de scripts ou assistentes, os quais também estarão armazenados na seção "Solução", juntamente com outros artefatos do *asset*. Estas regras visam a reduzir o tempo que os desenvolvedores levariam para descobrir, analisar, consumir e testar o *asset*;
- **Assets Relacionados** (*relatedAssets*) identifica relacionamentos entre este e outros possíveis *assets*. Este relacionamento pode ser, por exemplo, de dependência ou

sugestão de utilização em conjunto.

Para suportar os vários graus de reuso, formalidade e maturidade do processo de reuso nas organizações, muitas das seções RAS são opcionais.

Enquanto o *Core RAS* contém informações genéricas sobre os *assets*, eventualmente informações mais detalhadas são requeridas para especificar outros tipos deles, tais como *web services*, padrões, componentes e *frameworks*. Para isso, a especificação RAS pode ser estendida e customizada através de perfis (*profiles*). Estes perfis preservam e estendem a descrição do núcleo do RAS apresentada na Figura 2.2. Atualmente, o OMG define três perfis: *Default Profile*, *Default Component Profile* e *Default Web Service Profile*. O perfil *Default* é usado para empacotar qualquer tipo de *asset*, sendo a versão mais genérica dos perfis, herdando o *Core RAS* sem gerar nenhuma modificação especializada. Enquanto os demais são especificamente para componentes e *web services*, respectivamente.

Assets reusáveis podem ser divididos em três dimensões: granularidade, variabilidade e articulação. Essas dimensões podem ser visualizadas na Figura 2.3, caracterizando como exemplo três tipos de *assets*: componentes, padrões de projeto e *frameworks*.

- **Granularidade**: descreve quantos problemas particulares ou soluções alternativas podem ser cobertas pelo *asset*. Um mais simples oferece apenas uma solução para um único problema bem definido. Outro com granularidade maior, pode resolver múltiplos problemas e fornecer diversas soluções para estes problemas. Geralmente, com o aumento da granularidade, vem também o aumento do tamanho e complexidade. Na figura 2.3, podemos perceber que *frameworks* e padrões de projeto tem granularidade maior que componentes.

- **Variabilidade**: indica o quão variável um *asset* pode ser. Em um extremo do espectro, ele é invariável, ou seja, não pode ser alterado nem configurado. Geralmente esse é o caso de componentes binários. Esse tipo é chamado de caixa-preta (*black-box asset*), uma vez que o seu conteúdo interno não pode ser alterado. No outro extremo há os *assets* caixa-branca (*white-box assets*), que são criados com a expectativa de que os consumidores irão editar e alterar a sua implementação. Existem ainda duas variações que estão no meio termo: os caixa-clara (*clear-box assets*), e os caixa-cinza (*grey-box assets*). Os primeiros, expõem os detalhes de implementação via modelos, fragmentos de códigos ou documentação, porém não podem ser modificados. Os detalhes são expostos apenas para ajudar o consumidor a entender melhor o funcionamento do *asset*. Já os últimos expõem e permitem modificações apenas em um subconjunto dos seus artefatos, geralmente através de parâmetros. No exemplo, padrões de projeto possuem maior variabilidade pois seu código precisa ser feito por inteiro, permitindo customizações em variados aspectos.

- **Articulação**: descreve o grau de completude dos artefatos em fornecer solução. Os que especificam mas não fornecem a solução, possuem um baixo grau de articulação. Por outro lado, os que especificam e implementam a solução além de fornecerem documentos de suporte, como requisitos, casos de uso, artefatos de teste, possuem um grau maior de articulação. No exemplo, percebemos que padrões de projeto possuem um grau menor de articulação, e frameworks possuem a maior articulação dos três.

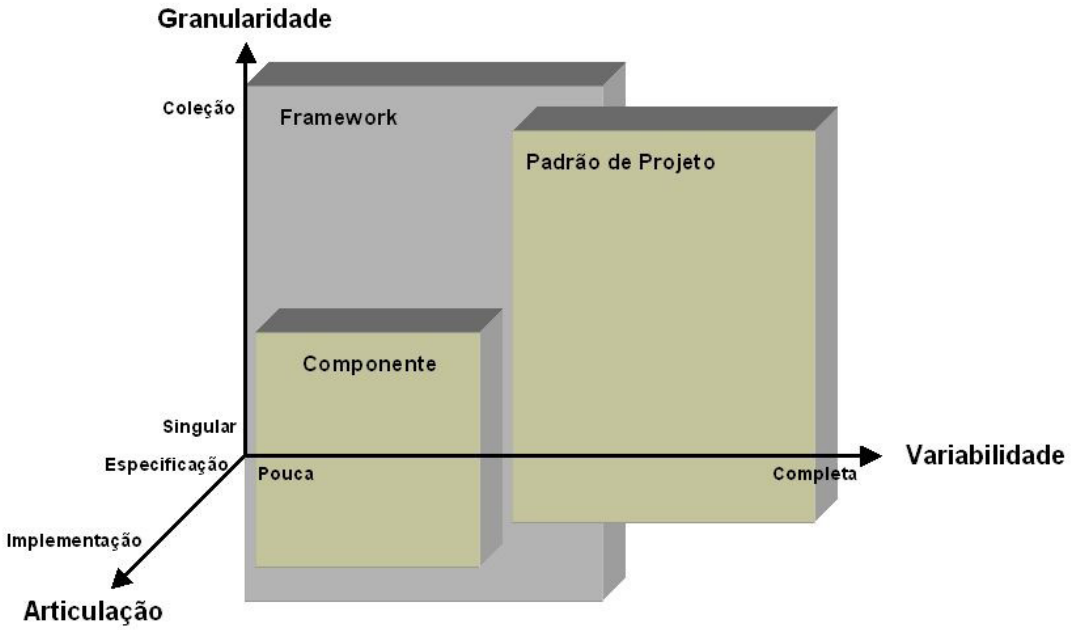

Figura 2.3: Dimensões de *assets* reusáveis

2.1.5 Processo de reuso

O ciclo de vida dos artefatos reusáveis compreende os processos pelos quais eles são identificados, obtidos, criados, documentados, certificados, publicados, reusados, mantidos, medidos e - finalmente - aposentados. O OMG tem trabalhado numa abordagem de desenvolvimento de software chamada desenvolvimento baseado em *assets* (*asset-based development - ABD*) que descreve todo esse ciclo de vida em quatro fluxos de trabalho distintos: identificação, produção, gerenciamento e consumo de artefatos reusáveis, como visto na figura 2.4. Esses fluxos de trabalho são congregados através de processos, ferramentas e padrões (Larsen 2009).

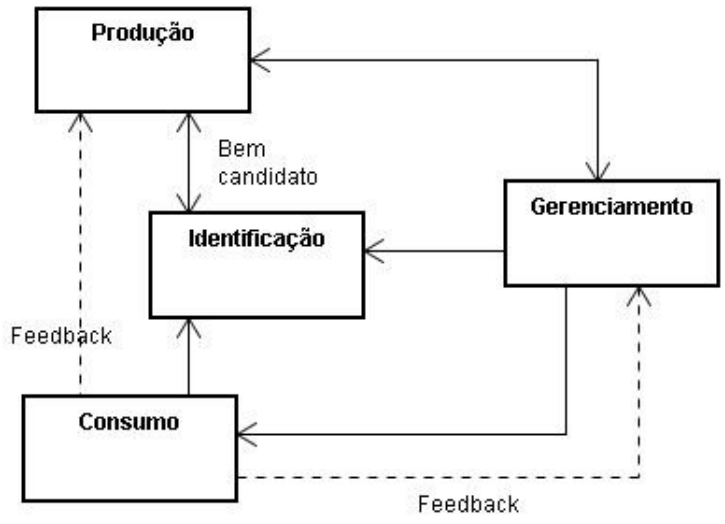

Figura 2.4: Fluxos de trabalho de um artefato reusável

O processo de reuso deve ocorrer de forma independente mas conectado (integrado)

com as demais atividades do processo de desenvolvimento do sistema. Podemos descrever cada uma destas etapas como segue:

- **Processo de identificação**: o primeiro passo é identificar o problema recorrente e uma solução em potencial. O responsável por esse passo, fará uma pesquisa por artefatos ou *assets* candidatos, e encontrando-os, os submeterá para serem preparados para reuso, empacotados de acordo com o padrão RAS e armazenados em um repositório. O objetivo é identificar todos os elementos reusáveis existentes que são úteis para a organização para serem reusados em futuros projetos de software;

- **Processo de produção**: envolve o trabalho de modelagem, codificação, teste e documentação. O indivíduo ou equipe responsável pelo processo de produção transformará o *asset* candidato em um reusável. Neste processo, poderá ser criado um novo *asset*, ou serão desenvolvidos artefatos complementares, ou refinados os já existentes. Ele deverá ser concebido de forma a ser consumível. Para isso, o responsável deverá levar em conta as três dimensões demonstradas na Figura 2.3: articulação, granularidade e variabilidade. Tendo sido construído e testado, será então documentado, empacotado, classificado e disponibilizado em um repositório RAS para consumo;

- **Processo de gerenciamento**: compreende as atividades que atuam em todo o ciclo de vida de um *asset*. Esse processo deve detalhar como um artefato é submetido como candidato a *asset* reusável, como são homologados e revisados, como são publicados e também aposentados. Deve definir métricas e criar esquemas de classificação, bem como estabelecer os papéis que serão desempenhados nos processos e as atividades desses papéis. Este processo, também pode descrever como criar perfis RAS customizados às necessidades da organização;

- **Processo de consumo**: aqui, uma das atividades chave é a localização de *assets*. Esta é realizada através de busca por critérios de pesquisa, que compreendem palavras-chave contidas no seu descritor, ou através da navegação por tópicos (*browsing*). Uma vez localizado um *asset* que atenda às necessidades do projeto, o consumidor poderá configurá-lo, usá-lo e fornecer *feedback* para os responsáveis pela produção e gerência, na forma de relatórios de defeitos, requisições de mudanças e casos de sucesso;

2.2 Repositórios de Artefatos

Um repositório é um depósito, mas pode também ser visto como uma coleção. No processo de desenvolvimento de software, um repositório é um lugar seguro para guardar o software produzido (Murta 2008). Frequentemente são associados ao aumento da memória organizacional, já que ao servirem como pontos de referência, guardam todo o conteúdo produzido em uma organização. Como funções fundamentais, os repositórios permitem o armazenamento e recuperação de artefatos.

Durante muito tempo, os repositórios estiveram ligados ao processo de Gerência de Configuração. Neste sentido, são abordados em diversos processos organizacionais como CMMI e MPSBR. Umas das partes chave daquele processo é chamada controle de versão, que define a necessidade de mantermos versões diferentes de um sistema de software ou controlar suas alterações. Aqui entram os sistemas de controle de versão, chamados

comumente de repositórios de gerência de configuração e muito utilizados durante o processo de desenvolvimento.

O processo de reuso também possui um repositório específico, apesar de não ser tão adotado sistematicamente. Um repositório de reuso vem para auxiliar o processo de busca de componentes prontos para serem reusados. Sua importância dentro do processo pode ser vista na Seção 2.1.5.

Em um repositório de controle de versão para gerência de configuração, estaremos interessados no processo de desenvolvimento de software, ou seja, sua construção. Assim, estaremos lidando com os arquivos que compõem uma determinada solução ou que satisfazem um determinado caso de uso ou requisito. Estes estarão sendo frequentemente alterados, dado que o processo de desenvolvimento acontece pela geração de várias versões de diversos arquivos, na maioria dos casos. Assim, os desenvolvedores estão cientes do que estão fazendo e onde as alterações estão localizadas, o que elimina a necessidade de pesquisa.

Já um repositório de reuso atua quando um artefato de software já está em produção. Nesta ocasião, estaremos interessados em reusar o componente, programa, biblioteca, ou outra versão de artefato que tenha passado por todas as etapas do processo de desenvolvimento, permitindo o seu reuso. Assim, a granularidade não é mais o arquivo que compõe o componente, mas o componente como um todo (ou um *build* do componente). Portanto, não estaremos interessados em alterar o componente, mas em reusá-lo, o que caracteriza um acesso de leitura, a partir do repositório.

Porém, para que se encontre um artefato reusável que forneça uma solução que seja aplicável a um contexto, é necessário mais do que apenas armazenamento e recuperação. É preciso um bom sistema de busca e navegação, voltados ao reuso. Nas seções seguintes, apresentaremos as características de repositórios de reuso e as principais ferramentas existentes no mercado.

2.3 Repositórios de Artefatos com suporte a Reuso

Um repositório de reuso é um repositório de artefatos que armazena artefatos reusáveis, mantendo um catálogo de artefatos. O repositório deve fornecer mecanismos para a organização acessá-lo e usá-lo de maneira rápida e fácil, de tal forma que se possa navegar e buscar um artefato reusável eficientemente. Suas funções são listadas abaixo em mais detalhes (Ezran 2002):

- **Identificação e descrição de artefato**: cada artefato deve ser identificado em um formato homogêneo. É importante estabelecer e manter uma interface comum para os artefatos, para que os membros de uma organização possam se familiarizar com isso;
- **Inserção de artefato**: um usuário deve ser capaz de inserir um novo artefato no repositório;
- **Navegação pelo catálogo**: um usuário deve ser capaz de navegar no catálogo para acessar a descrição dos artefatos;
- **Busca textual**: um usuário deve ser capaz de procurar um texto contido em qualquer parte da descrição de um artefato no catálogo;
- **Recuperação de artefato**: dado um artefato definido, um usuário deve ser capaz de recuperar uma cópia do artefato;

- **Organização e busca**: navegação pelo catálogo é claramente insuficiente para identificar um artefato quando o repositório tem um número grande de artefatos, enquanto que a pesquisa textual pode consumir muito tempo. Diversas técnicas podem ser usadas nesses casos para organizar a descrição dos artefatos em catálogos e facilitar a pesquisa. A Seção 2.1.4 apresenta critérios do descritor RAS que podem ser utilizados para melhorar a organização;

- **Histórico**: cada descrição de artefato deve gravar o autor, data de criação e de atualização, e uma lista de modificações. O uso do artefato também deve ser rastreado;

- **Métricas**: o repositório pode automatizar a coleta de dados para derivar métricas;

- **Controle de acesso**: cada uma das funções acima deve ser disponível apenas para perfis selecionados. Por exemplo, o direito de inserção deve ser garantido apenas aos gerentes de reuso, enquanto que navegação e recuperação podem ser garantidas a todos os usuários;

- **Gerenciamento de versão**: múltiplas versões do mesmo artefato devem ser definidas e gerenciadas, e seus relacionamentos gravados;

- **Controle de alterações**: procedimentos para requisitar, discutir, aceitar e programar alterações para os artefatos devem ser definidas e garantidas por um suporte automático;

- **Notificação de alterações**: alterações em artefatos e no repositório de reuso devem ser notificadas, com suporte automático, para todos os potenciais usuários;

Segundo (Ezran 2002), nem todas estas funções precisam necessariamente estar presentes em um repositório de reuso. Entretanto, há um consenso de que além das funcionalidades básicas de recuperação e armazenamento, devem possuir um sistema de navegação e localização eficiente (Mili 1998).

2.4 Ferramentas Existentes

As subseções a seguir descrevem algumas ferramentas presentes no mercado que possuem algum tipo de suporte a reuso.

2.4.1 Basic Asset Retrieval Tool (BART)

BART é um mecanismo de busca para auxiliar na recuperação de artefatos, que podem ser documentos, códigos, planilhas, entre outros. Esta busca pode ser realizada através de uma interface web ou através de extensões para Eclipse®, Visual Studio® e Microsoft Word® (RISE-BART 2009).

Seus principais atrativos são uma baixa intrusividade na implantação em uma organização, além de um formato de busca inteligente. Tem o funcionamento dividido em duas interfaces: busca e administração. A interface de busca só irá prover o acesso aos artefatos indexados. Os dados são armazenados logicamente em *workspaces* que garantem o controle de acesso e organização. As autorizações e permissões são concedidas pelo administrador. Assim, apenas quem possui as permissões definidas para um *workspace* pode acessar suas informações. As principais funcionalidades do ambiente de busca são:

- Busca (palavras-chaves e marcadores) e recuperação;

- Inclusão de marcadores (associados aos artefatos);
- Navegação por pastas e categorias;
- Interface *WebDAV*;
- Envio de artefatos;
- Nuvem de tags (baseada nos marcadores);

No ambiente de administração, é possível gerenciar o sistema a partir das seguintes funcionalidades:

- Cadastro de usuários, *workspaces* e repositórios;
- Controle de acesso aos *workspaces*;
- Envio de arquivos armazenados no servidor;
- Estatísticas e relatórios relacionados aos acessos, arquivos e usuários;
- Configurações dos tipos de arquivos disponíveis, banco de dados, e outras;

Essa aplicação não provê suporte ao padrão RAS e não possui código aberto.

2.4.2 Component Repository (CORE)

O CORE é um repositório de artefatos de software projetado para apoiar um processo de reuso sistemático (RISE-CORE 2009). Ele é mais intrusivo que o BART. Para o uso do CORE, é necessário a atribuição de papéis, como o de produtor. Este é responsável por criar e armazenar artefatos reusáveis no repositório. Análogo ao papel de produtor, temos o consumidor, responsável por identificar um artefato para ser reutilizado em um projeto, através da pesquisa. Complementando o quadro de papéis, temos o certificador, que valida os artefatos que são submetidos ao repositório, sendo responsável pela qualidade dos mesmos.

As funcionalidades do CORE incluem inserção de artefatos, normalmente pelo produtor. Junto com o artefato ficam armazenados seus metadados. Também se pode realizar a navegação pelo catálogo de artefatos reusáveis, que são agrupados em diferentes categorias. Isso permite uma visão mais simplificada e organizada da estrutura de componentes.

Classificação e busca também são possíveis no CORE, onde a pesquisa pode ocorrer por palavras-chave, textual ou através de facetas. A ferramenta provê um mecanismo de geração de relatórios, onde se pode obter uma visão geral de como o repositório está sendo utilizado.

Alterações geram notificações através de um serviço de notificação no qual os usuários podem se registrar. Também é mantido um histórico de múltiplas versões de um mesmo arquivo. Relações entre os artefatos podem ser visualizadas, na forma de “usa” ou “é composto por”.

O repositório possui mecanismos para promover a cultura de reuso pela organização, oferecendo serviços para manutenção e destaque de notícias relacionadas ao reuso, como iniciativas, melhores produtores e artefatos mais reusados.

Assim como o BART, ele não possui suporte ao padrão RAS, e também não é em código aberto.

2.4.3 Rational Asset Manager (RAM)

O RAM é o repositório de tempo de desenvolvimento que a IBM oferece para tarefas de desenvolvimento de software relacionadas aos gerentes, analistas, arquitetos, desenvolvedores e testadores. Este repositório é responsável por auxiliar as tarefas de submissão e categorização de artefatos, prover controle de acesso e medir o nível de atividade em termos do seu uso (IBM 2007).

Tarefas, necessidades individuais e comunidades são suportadas. As categorias de usuários do RAM incluem negócios e gerenciamento técnico (líderes de equipe e gerentes de projeto), administradores e praticantes. Estes podem ser analistas, arquitetos, desenvolvedores e testadores. Cada um deles possui um cenário de uso dentro do RAM, o que pode visualizado na forma de um cliente web ou através da interface do Eclipse®.

O gerenciamento de metadados é um dos cenários principais, onde o RAM utiliza o padrão RAS como sua estrutura de metadados. Informações adicionais permitem que o RAM seja integrado com outras ferramentas da IBM, como o registro de serviço e repositório WebSphere®. Portanto, essa ferramenta oferece suporte total à especificação RAS.

Para identificar os melhores artefatos, o RAM utiliza um sistema de pontuação no qual os usuários podem avaliar os artefatos reusados. Essas avaliações também alimentam o sistema de busca.

2.4.4 ARCSeeker

O ARCSeeker suporta reuso de modelos UML gerados a partir da ferramenta Enterprise Architect, da Sparx Systems. A ferramenta permite relacionar modelos UML com documentos, arquivos-fonte e armazená-los como componentes. A partir daí, eles podem ser visualizados, pesquisados e recuperados (SPARX 2009).

Podemos destacar as funcionalidades de criação de componentes e configuração de suas propriedades, armazenamento em pastas virtuais, histórico de componentes por controle de versão, navegação por categorias, pesquisa por termos e notificação de alterações via e-mail.

O ARCSeeker faz uso do padrão RAS para empacotar o modelo UML e os arquivos relacionados, oferecendo suporte à especificação.

2.4.5 Asset Management Tool (AMT)

Esta ferramenta foi proposta e implementada em (Wang et al. 2008) com o objetivo de permitir o desenvolvimento de software baseado em componentes (*Component Based Software Development - CBSD*) utilizando uma gerência de *assets* baseados em RAS.

Para isso, a ferramenta estende o perfil padrão de componente (*Default Component Profile*) do RAS, adicionando as seções: *Domain*, *Architecture*, *Analysis*, *Pattern* e *Test Cases*. O objetivo principal dessas seções é permitir a separação de mais artefatos reusáveis dentro de um mesmo *asset*.

O AMT é dividido em duas partes: o sistema principal (*Main System - MS*), e o sistema assistente (*Assistant System - AS*).

O sistema principal é composto de submissão, recuperação, e uso de *assets*. Na submissão, o AMT indexa informações encontradas na seção *Classification* do descritor RAS, para que possam ser encontradas na etapa de recuperação. Nesse sistema, é possível navegar nas informações de descrição e uso, como *variability-points*, para que o usuário conheça pontos nos quais pode realizar alterações relevantes.

O sistema assistente é composto de gerenciamento de usuários e privilégios, informações de estatísticas, gerência de informações de projeto e um sistema de suporte a decisões. Este último é capaz de gerar métricas de reuso para que responsáveis por decisões estratégicas saibam quais componentes são mais reusados, e outras informações que ajudem a decidir estratégias de reuso.

Esta ferramenta é em código aberto, podendo ser estendida. Porém, os comentários de código estão em chinês, o que pode dificultar "um pouco" quem não domina a língua.

2.4.6 Maven

Maven é uma ferramenta de gerenciamento e compreensão de projetos de software. O gerenciamento da construção, a geração de relatórios e documentação de projetos é realizada a partir de um objeto de modelo de projeto (*Project Object Model* - POM).

Seu objetivo principal é permitir ao desenvolvedor compreender o estado completo do esforço de desenvolvimento em um curto período de tempo. O Maven atinge o objetivo simplificando o processo de geração de artefatos, provendo um sistema de geração uniforme. Fornece informação sobre a qualidade do projeto, através de orientações para melhores práticas de desenvolvimento, como relatórios de cobertura de testes e auditoria de código. Além disso, permite uma migração transparente para novas funcionalidades.

O Maven facilita o processo de busca por dependências, localizando-as em seus repositórios remotos através de quatro coordenadas: *groupId*, *artifactId*, *version* e *packaging*. Portanto, o seu mecanismo de busca é bastante simples, não sendo suficiente para uma aplicação em um processo de reuso sistematizado.

Para concluir, é um projeto de código aberto, mas não suporta a especificação RAS.

2.4.7 Archiva

O Archiva é um gerenciador de repositórios de software extensível que lida com repositórios Maven. Ele funciona como *proxy* remoto para repositórios, possuindo gerenciamento de acesso seguro, armazenamento, recuperação, navegação, indexação e relatório de uso de artefatos.

Como gerenciador de repositórios, o Archiva pode conter diversas entradas para repositórios Maven, as quais tem seu conteúdo indexado, permitindo também a navegação através da interface web. Como *proxy*, mantém cópias locais de artefatos acessados em repositórios remotos, aumentando a velocidade de recuperação destes, e diminuindo o consumo de banda.

Assim como o Maven, possui um mecanismo de busca bastante simples, não sendo suficiente para um processo de reuso. É um projeto de código aberto e não suporta a especificação RAS.

2.4.8 Rasputin

O Rasputin implementa uma extensão para o Archiva, para prover suporte ao padrão RAS. Fornece assim uma maneira padrão de armazenar, consultar e recuperar artefatos (da Rosa 2009).

Utiliza o mecanismo de pesquisa do Archiva para localizar informações relevantes no descritor RAS dos artefatos. Mas ainda assim, este mecanismo é bastante simples, faltando critérios para ordenar quando são localizados vários artefatos.

Essa extensão mantém a restrição do Archiva de ser atrelado a projetos Maven. Ainda, por ser uma versão bastante recente, não utiliza os mecanismos fornecidos no padrão RAS

para facilitar a navegação nos repositórios de artefatos.

O Rasputin é um projeto em código aberto, permitindo que trabalhos futuros implementem essas funcionalidades.

2.4.9 Nexus

O Nexus tem uma proposta similar a do Archiva, como gerenciador de repositórios Maven, com a vantagem de poder lidar mais facilmente com outros tipos de repositórios, tanto através de extensões, quanto com suporte nativo na versão paga. Possui menos consumo de memória e é mais rápido que o Archiva, principalmente quando existe um número grande de repositórios e artefatos.

É também uma ferramenta em código aberto, com uma arquitetura bastante flexível. Possui um sistema de extensões utilizando um padrão de inversão de controle, o que permite reforçar o princípio de inversão de dependências (Fowler 2009).

Além disso, não utiliza banco de dados, mas o sistema de arquivos, indexando as informações através da biblioteca Apache Lucene. Permitindo mais liberdade à customização na busca, avaliação dos critérios e ordenação dos resultados.

Por estes e outros detalhes encontrados em (Sonatype-Nexus 2009), o Nexus foi escolhido como gerenciador de repositórios base para ser estendido, com adição do suporte ao padrão RAS, pela extensão Ras4Nexus, proposta nesse trabalho.

Tabela 2.2: Resumo comparativo

Ferramenta	*Código Aberto*	*Suporte a RAS*
BART	Não	Não
CORE	Não	Não
RAM	Não	Sim
ARCSeeker	Não	Sim
AMT	Sim	Sim (apenas perfil definido)
Maven	Sim	Não
Archiva	Sim	Não
Rasputin	Sim	Sim (apenas perfil Default)
Nexus	Sim	Não

3 RAS4NEXUS: DESCRIÇÃO E USO

Podemos observar que um fator indispensável na adoção prática e efetiva do reuso de software sistematizado é a utilização de ferramentas adequadas. Nesse contexto, percebemos os seguintes desafios com relação a sua adoção extensiva, os quais são trabalhados nesse capítulo:

- Ferramentas geradoras de descritores para reuso possuem um padrão próprio, dificultando a interação com ferramentas existentes. Um sistema que gere artefatos reusáveis não será aproveitado caso o descritor desse artefato não seja reconhecido em um gerenciador de artefatos reusáveis com um diferente padrão de descritor;

- Necessidade de integração com um repositório pré-existente. Normalmente as ferramentas exigem um tipo específico de projeto e/ou repositório com seus metadados e informações. Assim, essas informações são replicadas como também os artefatos são replicados para os sistemas de reuso. Surge por consequência o requisito de sincronização desses sistemas, o que dificulta sua aplicação prática;

- Com uma quantidade crescente de artefatos e seus produtores e consumidores, surge a necessidade de uma série de funcionalidades já existentes em outros sistemas que gerenciam artefatos, mas não são voltados a reuso. Por exemplo, controle de acesso, gerenciamento de versão, controle de alterações e notificação de alterações. Dessa forma, o sistema para reuso que não seja integrado precisará replicar essas funcionalidades;

- Migração a partir de artefatos sem informações de reuso. Organizações que desejam iniciar a aplicação do reuso sistematizado geralmente possuem um grande número de artefatos em diferentes repositórios e padrões. Iniciar a utilização de uma determinada ferramenta para reuso requer que esses artefatos sejam migrados e descritos com o respectivo padrão de reuso. Assim, existe um grande esforço inicial para que esse artefatos legados seja suportados por completo na ferramenta;

- Código aberto e extensibilidade. Nem todas as soluções possuem essas características, que permitem a extensão e personalização das funcionalidades da ferramenta de acordo com as necessidades de cada organização;

- Custos envolvidos. Considerando-se principalmente o valor das ferramentas e consultoria necessárias para a efetivação do reuso sistematizado;

Analisando estes desafios, verificamos que a adoção de uma ferramenta de gerência de repositórios que já esteja sendo usada em larga escala no âmbito das organizações, e que

dê a possibilidade de extensão para a adoção de um padrão mundialmente reconhecido para reuso, pode resolver esses problemas. E ainda, devido a capacidade de extensão, pode ser adotado um processo de melhoria contínua com relação ao reuso, personalizando o sistema e gerando novas extensões para atacar as questões que surgirem e que poderiam dificultar essa adoção.

O ponto de partida da solução proposta, o **Ras4Nexus**, é a extensão do gerenciador de repositórios Nexus para suporte ao padrão RAS (ambos citados no capítulo anterior). O Nexus foi escolhido entre os gerenciadores de repositórios pelos seguintes motivos:

- Tem seu código aberto;
- Possui uma arquitetura flexível e projetada para extensões;
- Vem substituindo o Archiva e se tornando o principal gerenciador de repositórios Maven;
- Pode suportar qualquer tipo de repositório através de extensões, e já existe suporte a outros tipos na versão paga;
- Possui um sistema de indexação flexível e eficiente;
- Possui um serviço de busca por artefatos eficiente e capaz de integração com outras ferramentas (IDEs);
- Possui versão gratuita completa;

Já o RAS foi escolhido entre os padrões por ter sido elaborado pela OMG, que já estabeleceu padrões mundialmente aceitos como UML e MOF. Igualmente esperamos que essa definição seja aceita e aplicada a todas as ferramentas cujo objetivo é promover o reuso de software. Poderemos assim armazenar e gerenciar esses artefatos através do Ras4Nexus, sejam quais forem, independente de linguagem de modelagem, programação, documentação, e projeto de software, desde que estejam descritos no padrão RAS.

O sistema de indexação e o serviço de busca nativos do Nexus, apesar de eficientes em termos de performance, são muito limitados em relação a quantidade de informações que são guardadas relativas a cada artefato e que podem ser pesquisadas para recuperação. Atualmente, o Nexus indexa o nome do artefato (busca por *Keywords*), o nome das classes (busca por *Classnames*), coordenadas GAV (busca por *GAV*) e pelo *hash* do artefato (busca por *Checksum*). Nenhuma informação descritiva, esteja ela nos descritores *pom.xml* dos projetos Maven ou nas próprias classes, será indexada. O que torna inviável a aplicação do reuso de software de maneira extensiva, pois os desenvolvedores precisariam conhecer os nomes de cada artefato ou de suas classes para chegarem a eles.

Descrições relativas ao artefato, que explicitam para que ele se propõe, suas características e aplicações, que possam ser recuperadas por um serviço de busca, favorecem a aplicação prática do reuso. Essas descrições estão presentes nos descritores RAS. Inicialmente, o Ras4Nexus irá estender o Nexus adicionando informações úteis à pesquisa em seus índices, buscadas a partir do descritor RAS (Seção 3.1). A seguir, modificará o serviço de busca para que esses novos índices sejam consultados (Seção 3.2). Além disso, serão considerados os casos nos quais os artefatos estariam corrompidos, seja na sua estrutura de arquivos ou no próprio descritor RAS (Seção 3.3).

3.1 Sistema de Indexação

Num primeiro momento, iremos analisar como o Nexus realiza a indexação, quais são os disparadores do evento, os tratadores das informações e as classes responsáveis pelo armazenamento em si. A partir de então, criaremos uma estratégia para adicionarmos as informações desejadas a partir do descritor RAS, estendendo esse sistema pré-existente. O restante da seção apresentará as classes implementadas, o fluxo do sistema resultante e a classe responsável em recuperar as informações a partir do artefato RAS.

3.1.1 Disparadores para Indexação

Existem dois eventos gerados pelo usuário que disparam a indexação: a adição (ou remoção) de um artefato através da interface web, e o comando para reindexar todos os artefatos de um determinado repositório. Este último pode ser gerado por atualização em configurações de repositórios e adição de repositórios. Todos estes comandos podem ser executados explicitamente pela interface web. Além disso, um sistema externo também pode adicionar artefatos no Nexus. Essa é a forma mais comum de envio de artefatos e, no caso do Maven, é realizada pelo comando *deploy*. A Figura 3.1 mostra os casos de uso que disparam eventos de indexação.

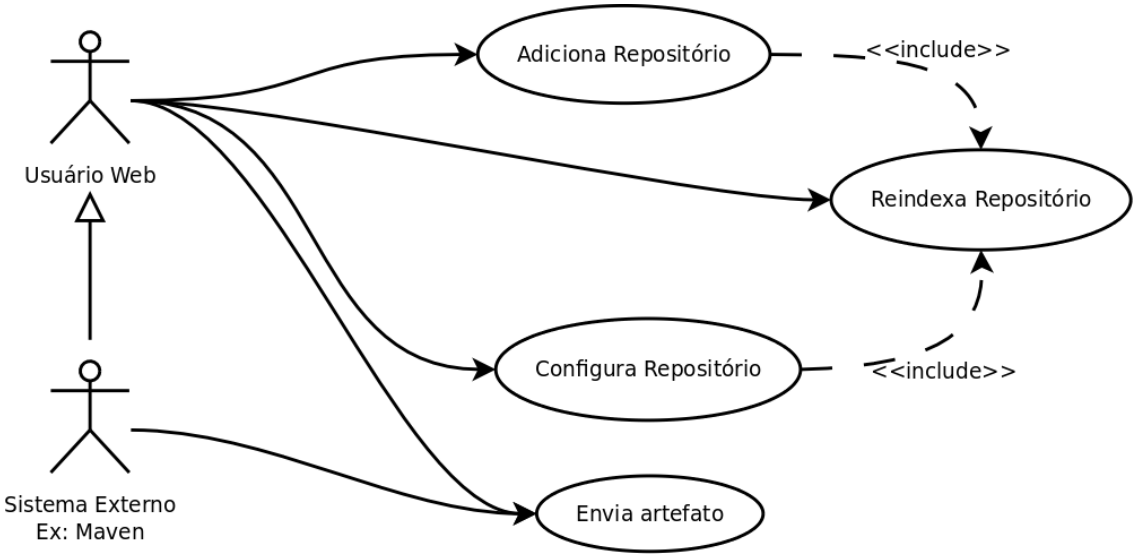

Figura 3.1: Casos de Uso para disparadores de indexação

As extensões do Nexus devem implementar a classe **EventInspector** para tratar eventos. O Ras4Nexus implementa duas classes que herdam da EventInspector: **RasIndexerManagerEventInspector** e **RasRegistryRepositoryEventInspector**. Suas responsabilidades são, respectivamente, tratar eventos de inserção e remoção de artefatos e tratar eventos de atualização, adição e remoção de repositórios.

3.1.2 Contextos de Indexação

Com relação à adição de repositórios, é também necessário registrar um novo contexto de indexação, representado pela classe **IndexContext**. Aqui são armazenados estados e informações relacionados a um repositório, que possibilitam que os artefatos sejam indexados e atualizados. Essa classe é composta por um contêiner de interfaces **IndexCreator**, cujas implementações são responsáveis por salvar, recuperar e atualizar informações

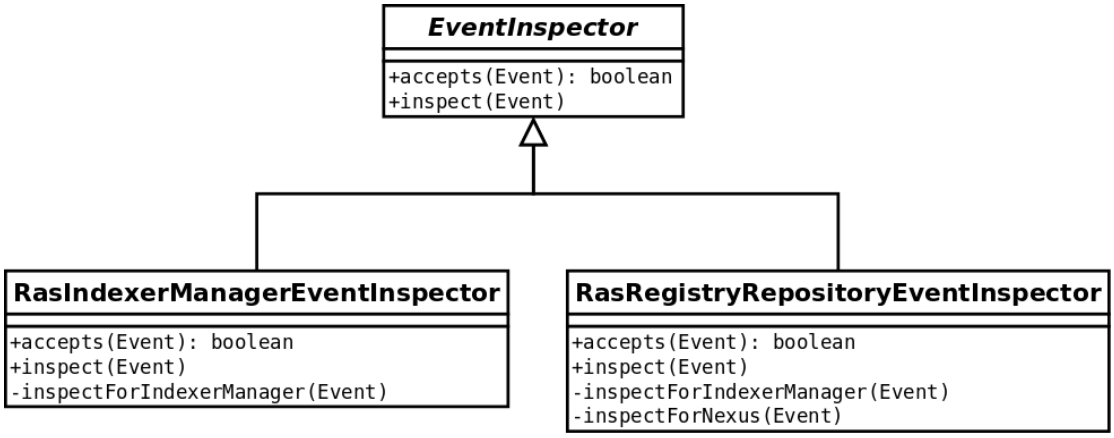

Figura 3.2: Classes responsáveis por tratamento de eventos

de índice, para um artefato em um IndexContext. Este é um ponto importante para o objetivo do Ras4Nexus, pois desejamos que todos os pacotes sejam tratados, independente de tipo de repositório. Registramos, então, a classe **RasIndexCreator**, implementando o contrato da interface IndexCreator, em objetos IndexContext para todo repositório registrado. A partir daí, estes repositórios passam a contar também com a capacidade de gerir informações RAS. Porém, ainda não é suficiente para que o sistema funcione, visto que o IndexContext existe, com objetos RasIndexCreator capazes de lidar com RAS, mas ninguém ainda o está usando de fato.

O próximo passo é utilizar nosso IndexContext registrado para cada repositório. Precisamos capturar os disparadores para indexação de artefatos individualmente. E como a classe de gerenciamento de índices do Nexus **DefaultIndexerManager** não permite o registro de novos contextos de índice ou de novos IndexCreators, o que seria suficiente para resolvermos esse problema, tornou-se necessário criar o nosso próprio gerenciador de índices, representado pela classe **RasIndexerManager**. Essa classe será requisitada sempre que for necessário manipular os índices, e todos os eventos tratados até agora utilizarão os métodos dessa classe. Isolando essa responsabilidade aqui, podemos finalmente usar os contextos de índice registrados para tratamento do RAS, consultando para cada ação em um determinado repositório, o seu IndexContext registrado. Este que já deverá estar composto por RasIndexCreator. As extensões podem ser melhor visualizadas no diagrama de classes simplificado da Figura 3.3.

O diagrama de sequência da Figura 3.4 apresenta como o sistema se comporta a partir de um evento de algum tipo de alteração nos repositórios, seja adição, atualização da configuração ou remoção. Sendo esse um tipo de evento que a classe RasRepositoryEventInspector retorna *true*, representado pela primeira mensagem e retorno no diagrama de sequência. Um ponto a destacar é o momento em que o contexto de indexação é configurado ao repositório. Ali, registramos o nosso criador de índice específico para o RAS. Tanto essa responsabilidade quanto a de configurar os índices de cada item são da classe RasIndexerManager.

3.1.3 Registrando índices RAS

Nest seção, utilizaremos o conjunto de componentes IndexCreators para criar índices que serão estruturados e armazenados pela biblioteca Lucene, da Apache, que é a biblioteca usada para registrar e buscar índices. Além do RasIndexCreator, utilizaremos também um criador de índices padrão do Nexus, o **MinimalArtifactIndexCreator**. Esta

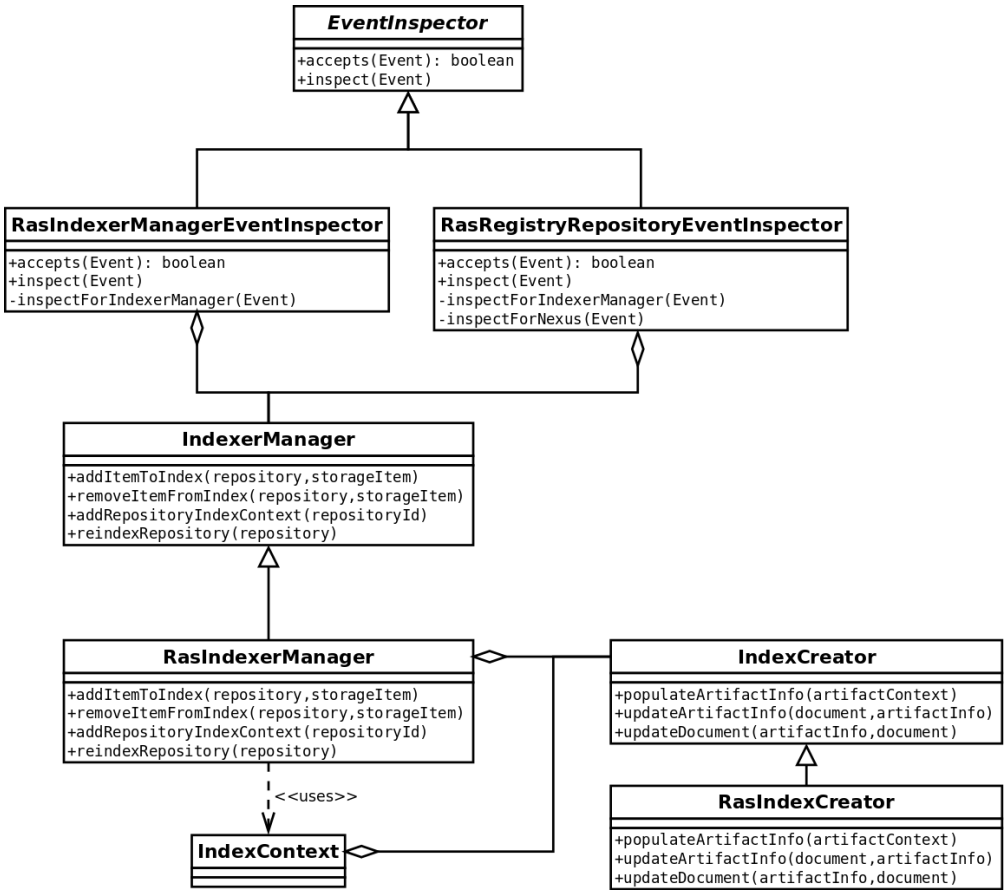

Figura 3.3: Diagrama de classes para indexação

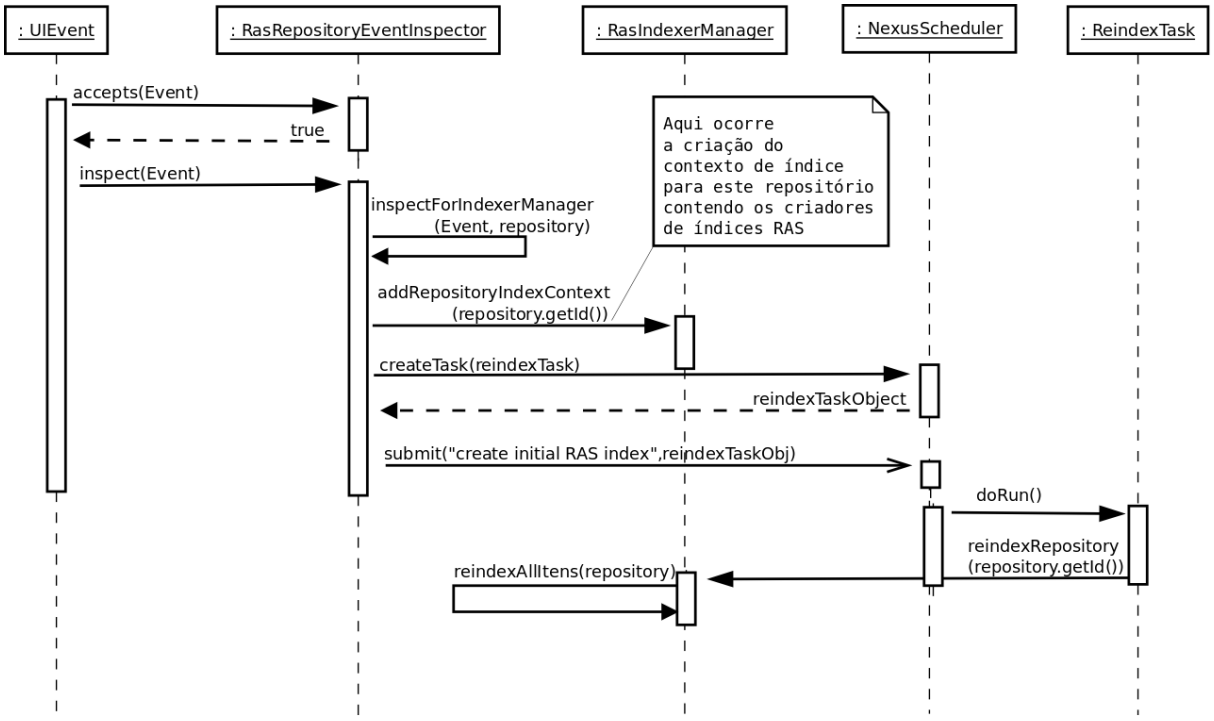

Figura 3.4: Diagrama de sequência para o registro de repositórios

classe pega itens comuns aos artefatos e do descritor de projeto (*pom.xml*, se existir) e também os adiciona como índices recuperáveis. Sua principal função é permitir a recuperação de um artefato através dos índices referenciados.

Para registrar índices Lucene, precisamos criar campos no documento de índices. Criando campos para cada informação do descritor RAS, temos a vantagem de manter as informações necessárias ao RAS separadamente. Sua consulta pode, dessa forma, ser realizada de maneira diferente, com critérios que favoreçam a organização dos resultados de pesquisa. Aqui temos a liberdade de criar outros campos, como por exemplo, um **RAS_SCORE**. Este campo poderia inclusive ser preenchido em um pós-processamento, onde um evento disparado após uma inserção ou modificação de um item verificaria quantas dependências e referências existem para esse artefato, avaliando sua pontuação. A pontuação é um critério útil na ordenação de resultados de pesquisa, mas está fora do escopo deste trabalho, sendo referenciada na seção sobre trabalhos futuros, no Capítulo 4.

Para preencher os campos, utilizamos a classe **RassetReader**. Essa classe é responsável por passar por todos os itens contidos em um artefato gerado pelo algoritmo ZIP, buscando por descritores *rasset.xml*. Encontrado esse descritor, é extraído do arquivo ZIP, para então percorrermos as informações relevantes à indexação através dos XPaths descritos na Tabela 3.1, associados a seus campos Lucene de indexação.

Tabela 3.1: Elementos do descritor RAS a serem indexados

Elemento	XPath equivalente	Campo Lucene
Nome do Ativo	/asset/@name	RAS_NAME
Identificador do Ativo	/asset/@id	RAS_ID
Versão do Ativo	/asset/@version	RAS_VERSION
Descrição breve do Ativo	/asset/@short-description	RAS_SHORT_DESC
Descrição do Ativo	/asset/description	RAS_DESCRIPTION
Descrição dos Contextos	/asset/classification//context/description	RAS_CONTEXT_DESC

O diagrama de classes abaixo (Figura 3.5) representa a estrutura das classes responsáveis pela indexação, e o diagrama de sequência da Figura 3.6 mostra como eles interagem.

O método getIndexWords da classe RassetReader recebe um objeto **RasPathMapBean**, que representa o conjunto de informações extraídas, separadas por cada XPath e campo Lucene, vistos na Tabela 3.1. Portanto, o RassetReader não possui conhecimento dos XPaths, mas estes são repassados através deste bean. Assim, o RassetReader passa por todos os XPaths registrados no bean previamente pela classe RasIndexCreator, buscando essas informações no descritor RAS. Após o retorno, o RasIndexCreator itera por todas as informações do conjunto, salvando-as em seus respectivos campos Lucene.

Aqui percebemos uma nova possibilidade de extensão. Implementando um novo IndexCreator, poderíamos adicionar novos XPaths e campos Lucene que seriam preenchidos pelo RassetReader, permitindo assim a indexação ilimitada de extensões dos perfis RAS. O RasIndexCreator é suficiente para indexar as informações básicas e funciona em qualquer perfil RAS, visto que todos os perfis devem herdar do Core RAS, que é a partir de onde essas informações (XPaths) estão definidas. Porém, surgindo a necessidade de se registrar para recuperação descrições que só existem em determinados perfis, essa pode

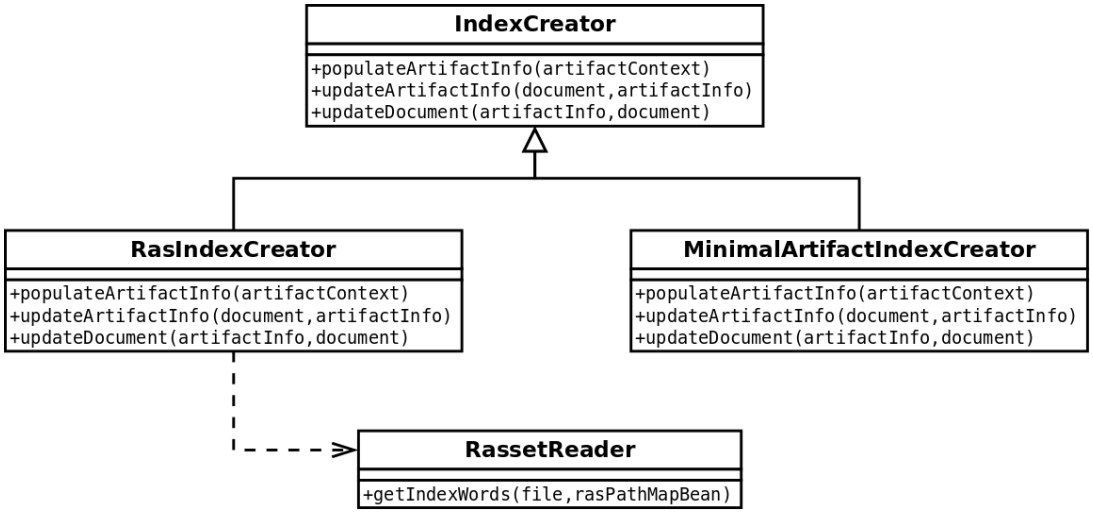

Figura 3.5: Diagrama de classes para extração de informações RAS

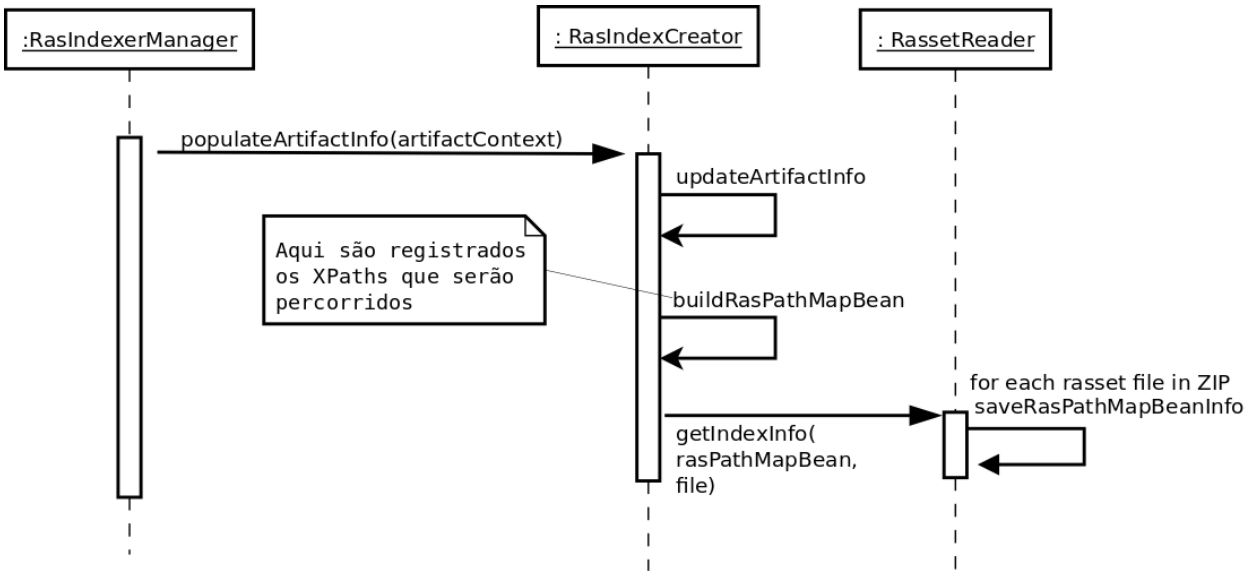

Figura 3.6: Diagrama de sequência para uso do RassetReader

ser uma saída interessante e bastante simples de ser executada, pois basta implementar essa única classe. Para que seja instanciada pelo motor de injeção de dependência, ela precisa estar anotada como *@Component(role=IndexCreator.class, hint='ras')*. Assim estará pronta para ser utilizada pelo RasIndexerManager, citado na Seção 3.1.2. Sendo instanciada juntamente com a RasIndexCreator para tratamento de artefatos RAS.

3.2 Serviço de Busca

O serviço de busca do Nexus pode ser utilizado através da interface web ou por meio de extensões que gerem esta requisição para o servidor web. As duas formas convergem nas classes que implementam a interface **Searcher**, que indicam como será feita a busca em si. A Figura 3.7 mostra o campo que permite a escolha do método de busca pela interface.

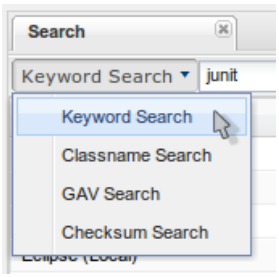

Figura 3.7: Escolha do método de busca na interface

Existem três classes que implementam as requisições de cada um desses métodos, são: **KeywordSearcher**, **ClassnameSearcher** e **MavenCoordinatesSearcher**. Percebemos que o método *ChecksumSearch* não tem uma classe que representa a sua implementação. É um critério de busca mais simples, que apenas utiliza um *hash*, não trataremos sobre ele. A Figura 3.8 apresenta o diagrama de classes e como todas elas são compostas pelo **DefaultIndexerManager**.

Seria suficiente para o Ras4Nexus utilizar a KeywordSearcher, e a partir daí buscar as descrições nos seus índices usando o RasIndexerManager. Mas como não é possível devido a todos os Searchers estarem acoplados com uma instância do DefaultIndexerManager, que não tem acesso aos índices RAS, precisamos implementar um novo Searcher, o **RasSearcher**. O RasSearcher funciona da mesma forma que o KeywordSearcher, com exceção de que o gerenciador de índices não é o padrão, mas o RasIndexerManager. A extensão fica evidenciada na Figura 3.9. Aqui é importante destacar a flexibilidade do sistema de extensões do Nexus, e a possibilidade de novos critérios para busca e avaliação de resultados. Basta implementar a interface Searcher e adicionar a anotação *@Component(role = Searcher.class)*, e o motor de injeção de dependências instanciará um objeto dessa classe para tratar eventos de busca. O RasSearcher, fazendo isso, realiza uma busca pelo critério *Keyword Search*, e juntamente com a KeywordSearcher padrão do Nexus, devolve resultados recuperados por palavra-chave. A diferença é que estas palavras-chave são buscadas dos índices gerados a partir dos descritores RAS.

Portanto, futuras extensões relativas ao serviço de busca devem decidir entre dois pontos de mudança: o primeiro é uma nova implementação da interface Searcher. Essa abordagem permite adicionar novos critérios, métodos de busca e comportamentos sem interferir na implementação atual do Ras4Nexus, podendo ser uma extensão separada. O segundo ponto seria modificar o gerenciador de índices RasIndexerManager. Deve-se usar

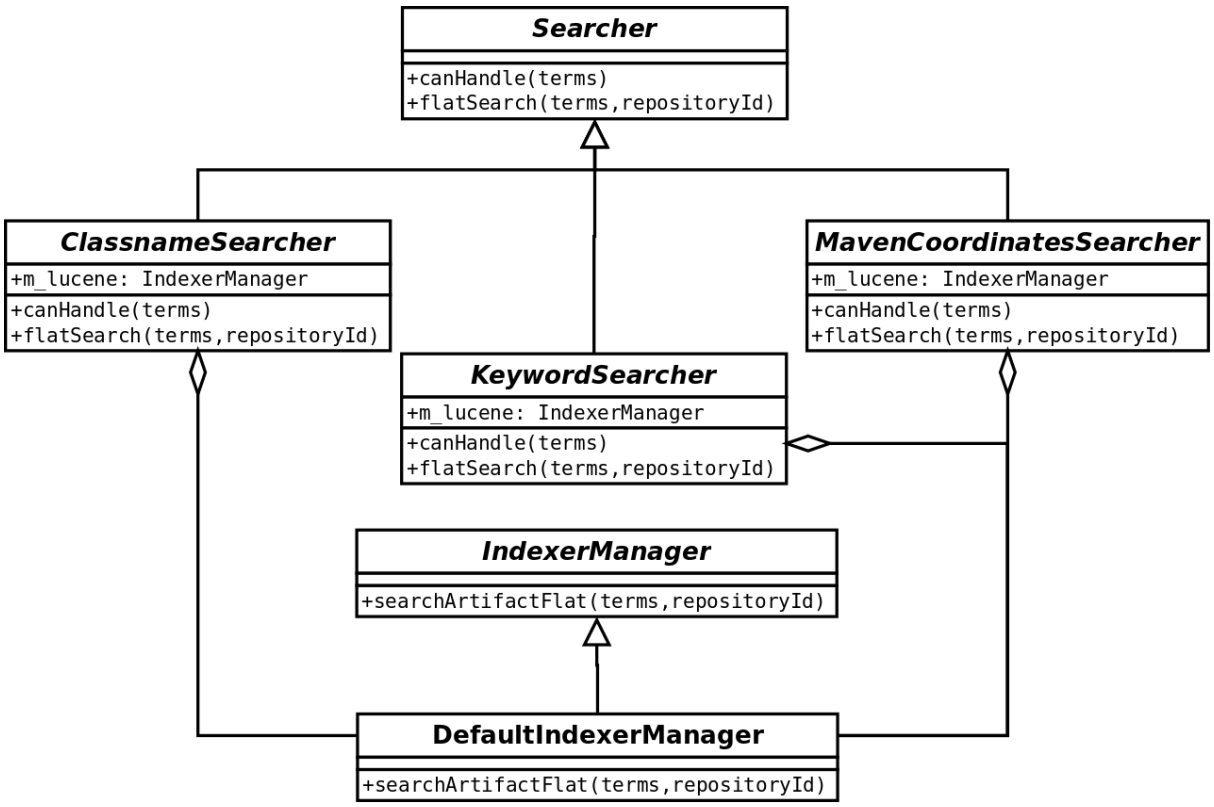

Figura 3.8: Classes responsáveis pelo direcionamento dos métodos de pesquisa

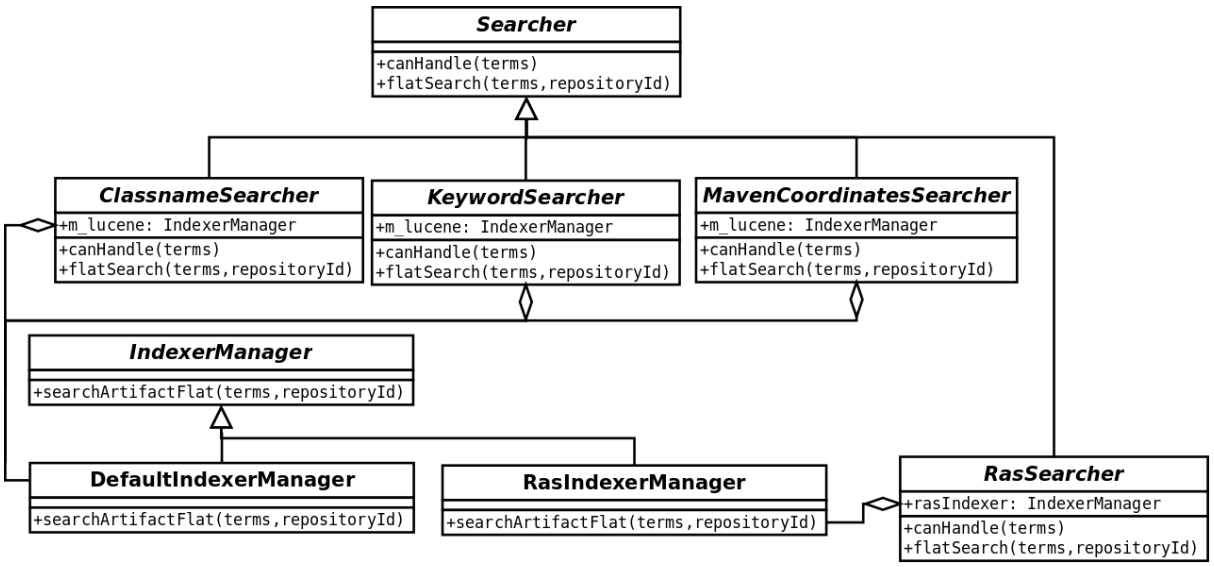

Figura 3.9: Extensão no serviço de busca com a classe RasSearcher

essa abordagem se a necessidade for mudança na forma de consulta aos índices Lucene e critérios de ordenação dos resultados. Isto implica uma alteração na extensão Ras4Nexus, não podendo ser uma extensão separada.

3.2.1 Critérios de Busca

O Ras4Nexus implementa a busca através do RasIndexerManager da mesma forma que a busca por palavra chave padrão do Nexus, isto é, utiliza o mesmo motor de busca, Apache Lucene, com os mesmos critérios. A única diferença, portanto, é onde é realizada esta busca. A Tabela 3.1 apresenta os novos campos criados para as informações RAS, que são os campos que interessam. O gerenciador de índices RAS cria consultas em cada um desses campos, e a união dos resultados da busca nesses campos é devolvida. O diagrama de sequência da Figura 3.10 apresenta as mensagens geradas a partir do evento de busca recebido pelo buscador RAS.

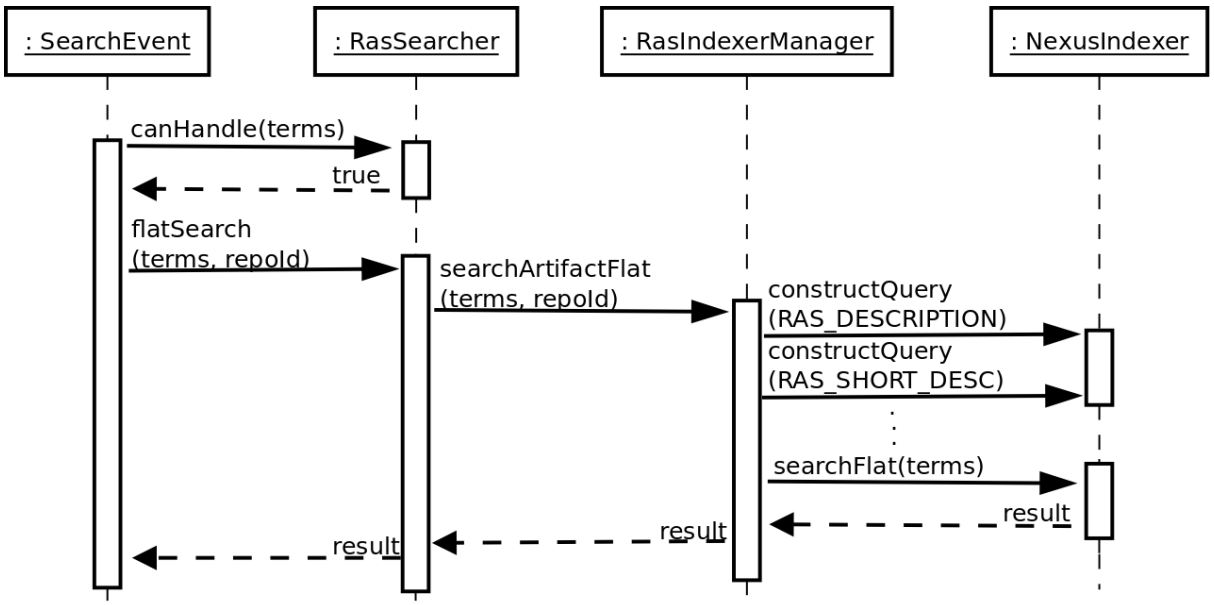

Figura 3.10: Consulta aos campos RAS a partir do SearchEvent

3.3 Integridade dos Artefatos

A integridade dos artefatos é verificada na etapa de submissão. Essa verificação é realizada em duas etapas: a primeira na extração do artefato (no formato ZIP) após a submissão pelo cliente, e a segunda na leitura do arquivo *rasset.xml* contido dentro desse pacote.

A verificação do arquivo ZIP é realizada pela classe *java.util.zip.ZipFile* contida na JRE. Caso o arquivo esteja corrompido, uma exceção é lançada, e o tratamento é feito pelo próprio Nexus. Um detalhe importante é que não necessariamente um arquivo com extensão *.jar* será um arquivo de pacote Java compactado com o padrão ZIP, por exemplo. Dessa forma, devemos permitir a inserção desse artefato pelo Nexus sem interferir no processo. Assim, a carga do artefato é realizada, abandonando a busca pelo arquivo *rasset.xml* e também a etapa de indexação dos metadados internos.

Já a consistência do descritor *rasset.xml* é mantida utilizando-se arquivos XSDs fornecidos pela OMG, que definem estrutura, conteúdo e semântica do descritor RAS. Da

mesma forma, essa etapa também lança uma exceção, em caso de inconsistência na estrutura do XML, que é tratada pelo Nexus e registrada em log.

O diagrama de sequência da Figura 3.11 demonstra como o motor de indexação padrão (classe *DefaultIndexerEngine*) realiza a chamada para o indexador RAS, definido por *RasIndexerCreator*, e, após, receber um artefato corrompido, a exceção é repassada para o motor do Nexus.

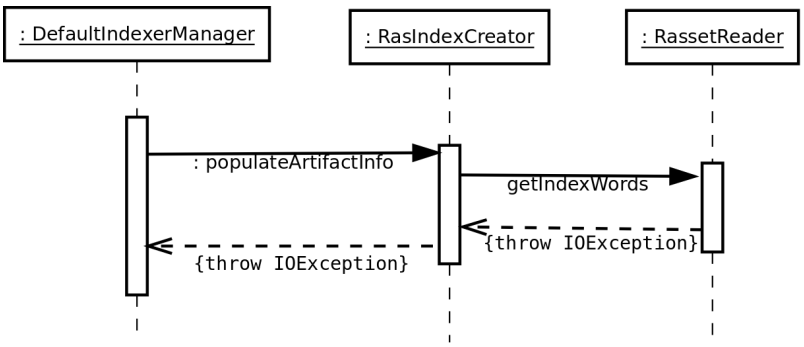

Figura 3.11: Diagrama de Sequência para artefatos corrompidos

Garantimos então as seguintes propriedades em caso de erro na submissão:

- O pacote corrompido é inserido sem efeitos colaterais ao restante do sistema.
- O descritor RAS corrompido ou fora do padrão é descartado da fase de indexação, porém é mantido no repositório.
- Os erros (exceções) gerados são salvos em arquivo de log para posterior consulta. Esses erros podem também ser consultados através da interface do Nexus.

A rastreabilidade é importante em caso de erro, principalmente para que o usuário não perca tempo tentando discernir a razão do problema, mas possa isolá-lo eficientemente para uma possível correção. Após uma busca sem sucesso por um artefato que foi submetido, por exemplo, pode-se inicialmente verificar os registros de erro, através da interface, buscando a partir da data de inserção. Encontrado o registro de erro, com seus detalhes, sua data e hora, a causa raiz pode ser localizada e tratada rapidamente.

3.4 Utilizando o Sistema: Passo a Passo

Nesta subseção apresentaremos um passo a passo de como o sistema é utilizado. Para isso, iremos submeter um pacote Java simples, contendo um descritor RAS. Em seguida, faremos pesquisas por termos que estão contidos nesse descritor, verificando o retorno na interface. Finalmente, baixaremos o arquivo para simular o uso por um sistema, e verificaremos seu conteúdo.

3.4.1 Submetendo um Pacote RAS

Existem diversas maneiras de se submeter artefatos para o Nexus. Uma forma bastante usada é através do comando *deploy* do Maven. Para facilitar nossa demonstração, iremos utilizar a própria interface gráfica do Nexus para submeter um artefato RAS. A Figura 3.12 apresenta a tela de repositórios, na qual selecionaremos um repositório local para o envio do arquivo.

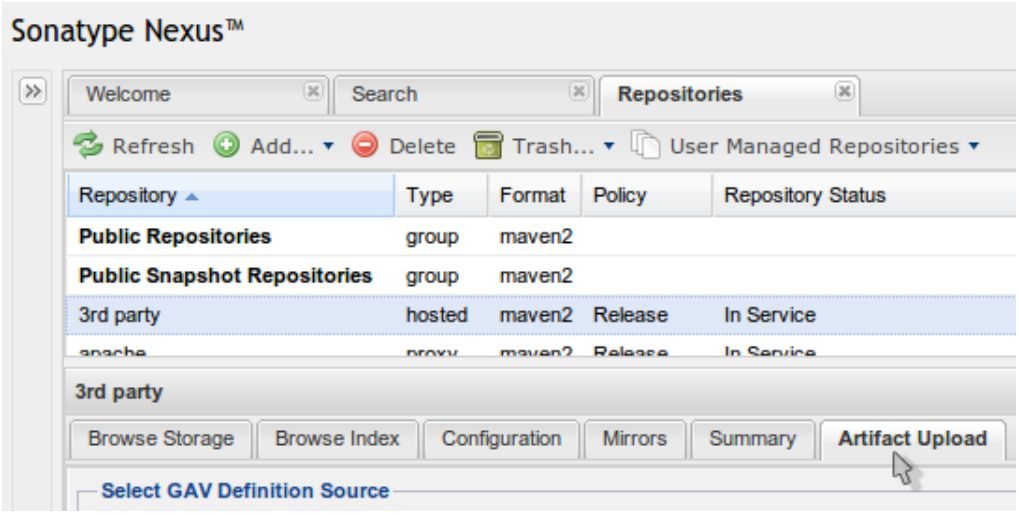

Figura 3.12: Seleção do repositório para envio

Escolhemos um repositório do tipo Maven armazenado localmente (*hosted*). Agora através da aba *Artifact Upload* iremos selecionar o artefato **Junit**, um conhecido framework para testes unitários, com a adição de um descritor RAS. A figura 3.13 mostra como o descritor está armazenado com relação às outras pastas do pacote, compactado com o algoritmo ZIP. Após configurar o envio conforme a Figura 3.14, o passo seguinte é realizado pelo sistema. A indexação é construída a partir do descritor fornecido no pacote.

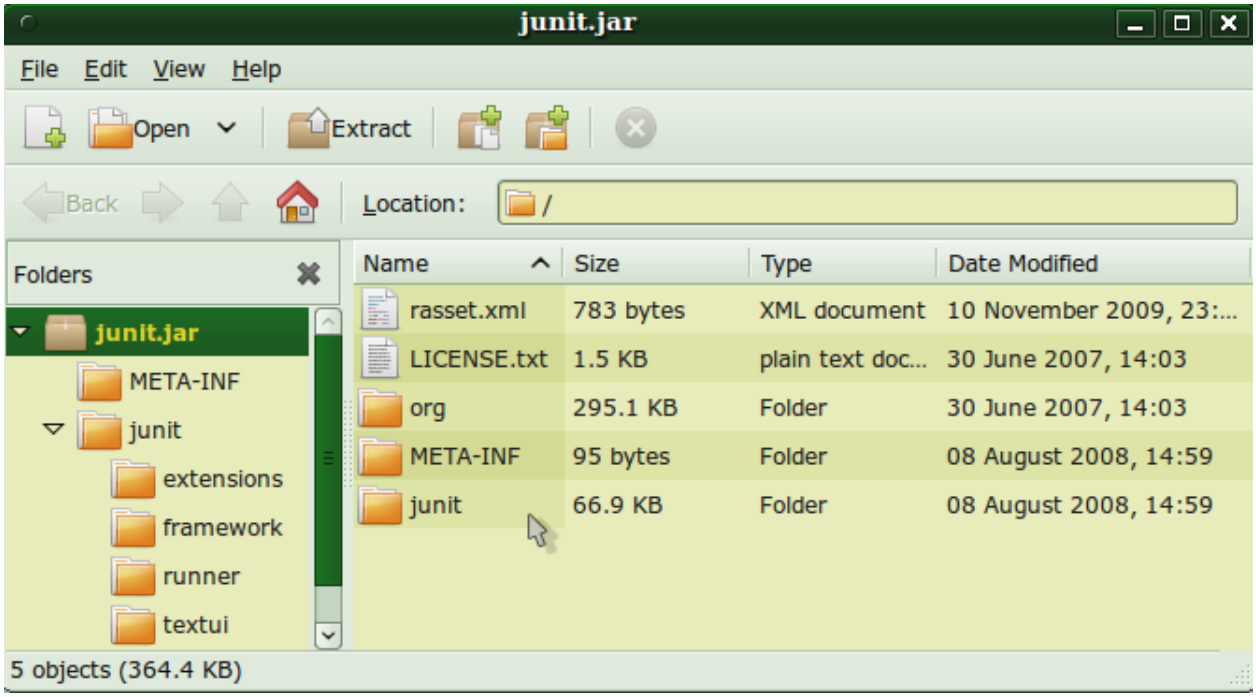

Figura 3.13: Estrutura do pacote com descritor RAS

Basta configurar o pacote dessa forma, com o descritor RAS na estrutura de pastas de um arquivo compactado, e o sistema irá indexar as informações relevantes para a pesquisa. Perceba que o sistema não exige nenhuma extensão específica de pacote, sendo apenas necessário que o arquivo esteja compactado com o algoritmo ZIP e um descritor esteja definido na sua estrutura de pastas. A descrição das operações realizadas e caminhos para

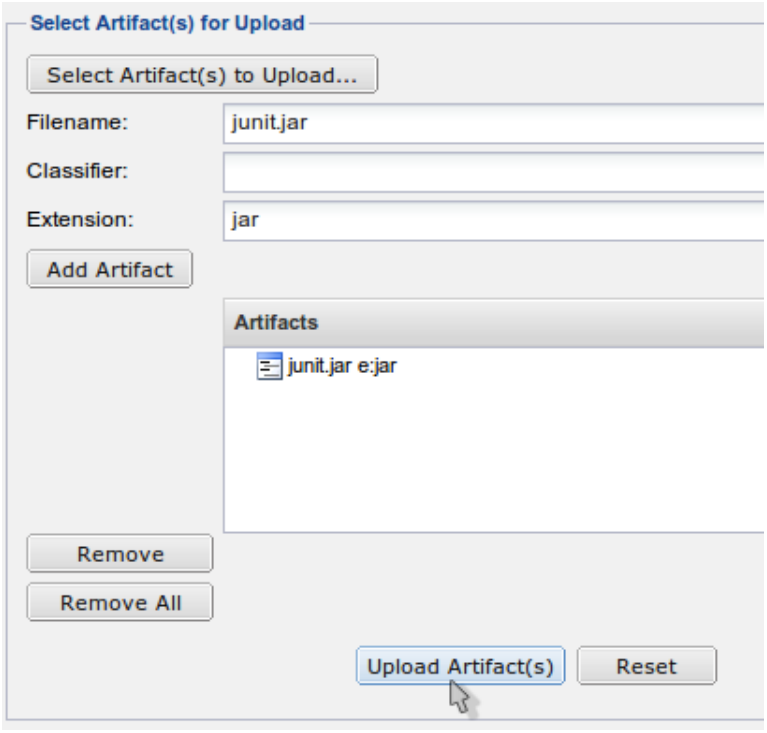

Figura 3.14: Informações e seleção do artefato para envio

indexação dentro do descritor estão especificadas na Seção 3.1. O conteúdo do arquivo *rasset.xml* é uma versão simplificada do **Default Profile**, para simplificar o entendimento e a sequência da demonstração do sistema (Figura 3.15).

3.4.2 Pesquisando por Artefatos

Existem extensões que permitem realizar pesquisas diretamente nos índices do Nexus diretamente dos ambientes de desenvolvimento. Para esse exemplo, porém, iremos novamente usar a interface web para realizar a pesquisa para recuperação do framework de testes *junit*. Na Figura 3.16, a pesquisa pelo termo "framework" é executada, retornando o artefato que adicionamos na seção anterior.

O pacote *junit* foi retornado nesse caso, porque o termo "framework" está contido em um dos XPaths considerados, nesse exemplo, o *//asset/@short-description*, do descritor RAS. Mas, conforme mostraremos no capítulo seguinte, todas as propriedades do artefato no Nexus são mantidas, o que permite a sua recuperação pelos índices padrão do Nexus, mantendo a compatibilidade com qualquer que seja o sistema de repositórios do artefato. No próximo capítulo será demonstrada a capacidade desse sistema de se utilizar o resultado da pesquisa para recuperar o artefato de diferentes maneiras.

3.4.3 Recuperando Artefato

A recuperação do artefato para uso pode ser feita por meio da interface web, apesar de essa não ser a maneira mais comum. Geralmente, o usuário configura seu projeto no seu ambiente de desenvolvimento, incluindo os artefatos gerados e as dependências necessárias, e o gerenciador de projetos se encarrega de buscar essas dependências nos repositórios configurados. No nosso exemplo, o artefato é recuperado a partir de um repositório Maven. Portanto, poderemos utilizá-lo em um descritor de projeto Maven através de suas coordenadas *GroupId*, *ArtifactId* e *Version*.

```
<?xml version="1.0" encoding="utf-8"?>
<asset xmlns="http://www.omg.com/RAS"
  xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
  name="JUnit"
  id="junit"
  version="3.8.1"
  short-description="JUnit is a regression testing framework.">
  <profile name="Default"
                id-history="F1C842AD-CE85-4261-ACA7-178C457018A1"
                version-major="2"
                version-minor="1"/>
  <description>JUnit is a regression testing framework
                    written by Erich Gamma and kent Beck.
                    It is used by the developer who implements
                    unit tests in Java.
       </description>
  <solution>
    <artifact name="junit.jar" type="jar">
      <description>
                        This is the JAR package for JUnit 3.8.1
                        </description>
      <artifact-type type="xs:string">
                        Java Package
                        </artifact-type>
    </artifact>
  </solution>
</asset>
```

Figura 3.15: Conteúdo do arquivo rasset.xml

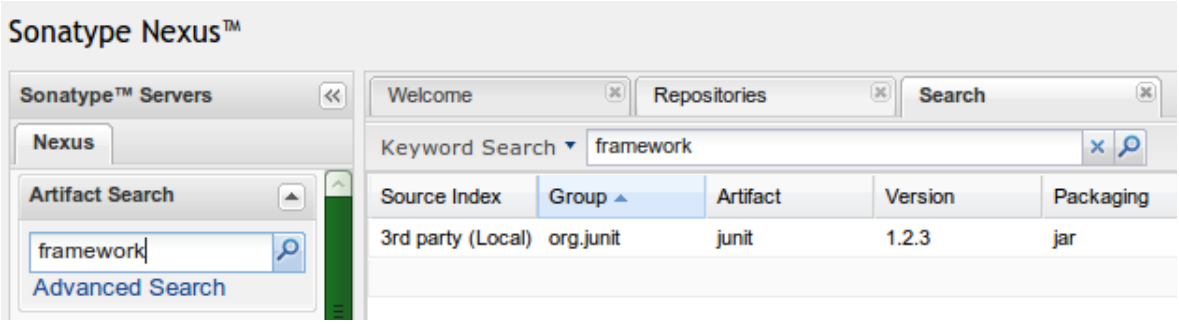

Figura 3.16: Pesquisa pelo termo “framework” no Nexus

A Figura 3.17 mostra os links para baixar direto da interface do Nexus. Clicando no link *artifact*, recuperaremos o artefato que foi adicionado na seção **Submetendo um Pacote RAS**. O link *pom* permite recuperar o descritor do projeto, o qual mantém informações de dependências e outros aspectos. Porém, para que possamos adicionar o artefato num projeto Maven como dependência, precisaremos adicionar um trecho de código XML no arquivo *pom.xml*. Esse trecho é apresentado na interface de acordo com a Figura 3.18, facilitando o papel do mantenedor do projeto. Portanto, copiar esse código presente na interface para o descritor de projeto, é suficiente para que o gerenciador recupere a dependência em questão e, posteriormente, verifique atualizações desta e de suas próprias dependências.

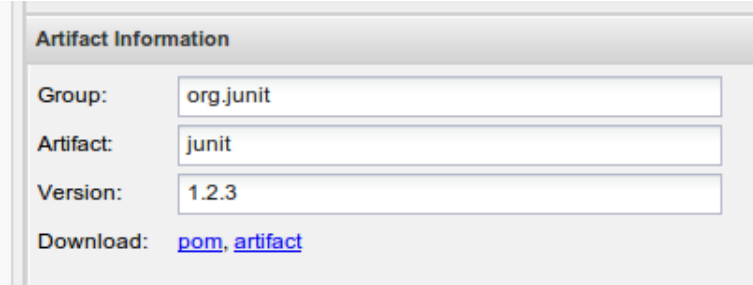

Figura 3.17: Identificação do Artefato e ligação para baixar

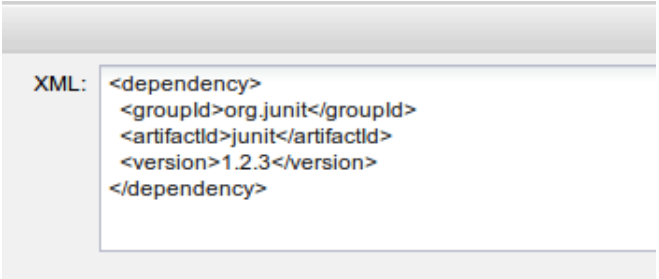

Figura 3.18: Informações de dependência para o Maven

3.4.4 Validação - Testes de Integração

Os seguintes testes de integração, inicializam o Nexus, adicionando uma informação contida em um artefato reusável, e o recuperando a partir de uma busca no repositório. Sua execução demonstra que o sistema pode corretamente adicionar e buscar por informações contidas no descritor RAS de artefatos reusáveis.

3.4.5 Instalação do RAS4Nexus

O *RAS4Nexus* pode ser baixado a partir do repositório SVN em *http://code.google.com/p/ras-plugin-for-nexus/*. Para gerar a extensão para o Nexus:

- Entre na pasta *project*;
- Execute o comando do Maven: *mvn install*

  Tenha certeza que seu Maven está configurado com os repositórios da Sonatype: *http://repository.sonatype.org/index.html#view-repositories*

```
1  public class RasPluginIndexingIT
2  extends AbstractNexusIntegrationTest
3  {
4      protected SearchMessageUtil messageUtil;
5      @Before
6      public void ini() {
7          this.messageUtil = new SearchMessageUtil();
8      }
9      @Test
10     public void deployPomlessArtifact()
11         throws Exception {
12         File artifactFile = getTestFile( "artifact.jar" );
13         DeployUtils.deployWithWagon( this.container, "http",
               baseNexusUrl + "content/repositories/"
14             + REPO_TEST_HARNESS_REPO, artifactFile, "nexus983/nexus983-
                   artifact1/1.0.0/nexus983-artifact1-1.0.0.jar" );
15         List<NexusArtifact> artifacts = messageUtil.searchFor( "
               nexus983-artifact1" );
16         Assert.assertEquals( "Should find one artifact", 1, artifacts.
               size() );
17         artifacts = messageUtil.searchFor( "framework" );
18         Assert.assertEquals( "Should find one artifact", 1, artifacts.
               size() );
19     }
20     @Test
21     public void copyPomlessArtifact()
22         throws Exception {
23         File artifactFile = getTestFile( "artifact.jar" );
24         FileUtils.copyFile( artifactFile, new File( nexusWorkDir, "
               storage/" + REPO_TEST_HARNESS_REPO
25             + "/nexus983/nexus983-artifact2/1.0.0/nexus983-artifact2
                   -1.0.0.jar" ) );
26         RepositoryMessageUtil.updateIndexes( REPO_TEST_HARNESS_REPO );
27
28         List<NexusArtifact> artifacts = messageUtil.searchFor( "
               nexus983-artifact2" );
29         Assert.assertEquals( "Should find one artifact", 1, artifacts.
               size() );
30     }
31     @Test
32     public void searchExtraIndexData()
33         throws Exception {
34         File artifactFile = getTestFile( "artifact.jar" );
35         FileUtils.copyFile( artifactFile, new File( nexusWorkDir, "
               storage/" + REPO_TEST_HARNESS_REPO
36             + "/nexus983/nexus983-artifact2/1.0.0/nexus983-artifact2
                   -1.0.0.jar" ) );
37         RepositoryMessageUtil.updateIndexes( REPO_TEST_HARNESS_REPO );
38
39         List<NexusArtifact> artifacts = messageUtil.searchFor( "
               framework" );
40         Assert.assertEquals( "Should find one artifact", 1, artifacts.
               size() );
41     }
42 }
```

Figura 3.19: Testes de Integração - *RasPluginIndexingIT.java*

- Copie o arquivo ZIP gerado na pasta *target* para a pasta *plugin-repository* dentro do diretório de trabalho do Nexus.
- Reinicie o serviço do Nexus.

A partir de então, o Nexus inicializará o RAS4Nexus, permitindo que todos os artefatos existentes e os adicionados sejam verificados pelo padrão RAS, e as informações relevantes indexadas.

4 CONCLUSÃO

A integração da especificação RAS no gerenciador de repositórios Nexus foi bem sucedida, auxiliando na sistematização do reuso de artefatos de software. A infra-estrutura gerada a partir da extensão Ras4Nexus, permite a gerência de uma variedade de artefatos, criados em quaisquer etapas de desenvolvimento, para diferentes contextos, devido a capacidade descritiva do padrão RAS. A independência de tipo de projeto, repositório e linguagem dos artefatos dá ainda mais liberdade para a aplicação prática do reuso em uma organização, eliminando restrições que poderiam dificultar sua disseminação. Essa independência foi atingida pois a arquitetura do Ras4Nexus, como descrita no Capítulo 3, não foi implementada utilizando classes responsáveis pelo tratamento de um repositório específico, como o Maven, mas a partir de interfaces de alto nível, desacopladas do tipo de repositório. As demais funcionalidades, como controle de versão, controle de dependências e gerenciamento de usuários e permissões, continuam inalteradas, sendo reutilizadas por sua utilidade especificamente na sitematização do reuso.

Apesar de todas essas funcionalidades, existem limitações no Ras4Nexus que o impedem de ser aplicado, nessa versão, em um ambiente de produção. O sistema de busca, visto no Capítulo 3 é muito simplificado, e pode ser insuficiente em um repositório com um número crescente de artefatos. O sistema de navegação nativo do Nexus permite que se navegue pela estrutura de pastas do repositório, mas esse critério por si só não facilita a localização de artefatos desejados. Não é possível visualizar informações RAS diretamente da interface web. Estas informações são necessárias para decidir se um artefato em questão pode ou não satisfazer determinados requisitos.

Portanto, a partir destas avaliações, podemos indicar as seguintes melhorias e trabalhos futuros:

- Modificar o sistema de pesquisa do Ras4Nexus, criando critérios mais complexos de busca, utilizando técnicas de recuperação de informações como *stemming* e similaridade. Criar também critérios de pontuação, para exibir ordenadamente os artefatos mais relevantes a uma determinada pesquisa, facilitando a localização de um artefato relevante;

- Estender a interface web do Nexus para permitir a navegação nas classificações e descrições recuperadas do descritor RAS de um artefato. Com isso um usuário pode rapidamente verificar se um determinado artefato atende ou não a suas necessidades. Permitir também que se possa navegar em informações de descritores RAS de diferentes perfis, não se prendendo ao perfil padrão (*Default Profile*);

- Estender a interface web, permitindo que se navegue pelo repositório não apenas por pastas, mas também por critérios como classificação, contexto, etiquetas e pa-

râmetros personalizáveis. Além da busca textual, como citado em (Ezran 2002), é interessante permitir uma busca através de navegação na estrutura de um repositório. Mas isto só é possível a partir de critérios que permitam o entendimento dessa estrutura. Esses critérios podem ser buscados a partir dos descritores RAS dos próprios artefatos;

- Modificar o Ras4Nexus para coleta de dados que automatizem a geração de métricas. Essas métricas podem definir o quão reusável é um artefato ao longo do tempo, frequência de atualização, quantidade de dependentes e quais são os critérios de busca que mais chegam a este artefato. Essas métricas não só servem para avaliar um artefato individualmente, mas também para auxiliar o sistema de pesquisa;

REFERÊNCIAS

[Almeida 2009]ALMEIDA, E. *Component Reuse in Software Engineering. [S.l.]:R.I.S.E.* 2009. Disponível em: <http://cruise.cesar.org.br/>. Acesso em: dezembro 2009.

[Ambler 2009]AMBLER, S. *Types of Reuse In Information Technology*. 2009. Disponível em: <http://www.ambysoft.com/essays/typesOfReuse.html>. Acesso em: dezembro 2009.

[Brooks 1987]BROOKS, F. P. No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, v. 20, p. 10–19, 1987.

[Cox 1990]COX, B. J. Planning the software industrial revolution. *IEEE Softw.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 7, n. 6, p. 25–33, 1990. ISSN 0740-7459.

[da Rosa 2009]da Rosa, F. R. *RASPUTIN Uma Infra-estrutura de Suporte para Promover o Reuso de Software através do padrão RAS*. Dissertação (Projeto de Diplomação em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre, RS, 2009.

[Ezran 2002]EZRAN, M. *Practical software reuse*. London, UK: Springer-Verlag, 2002. ISBN 1-85233-502-5.

[Fowler 2009]FOWLER, M. *Inversion of Control Containers and the Dependency Injection pattern*. 2009. Disponível em: <http://www.martinfowler.com/articles/injection.html>. Acesso em: dezembro 2009.

[Gamma 2000]GAMMA, E. *Design Patterns. Elements Of Reusable Object-Oriented Software*. [S.l.]: Addison Wesley, 2000.

[IBM 2007]IBM. *Federated metadata management with IBM Rational and WebSphere software*. 2007. Disponível em: <ftp://ftp.software.ibm.com/software/rational/web/whitepapers/10709147_Rational_RAM_WP_ACC.pdf>. Acesso em: dezembro 2009.

[Kim e Stohr 1998]KIM, Y.; STOHR, E. A. Software reuse: survey and research directions. *J. Manage. Inf. Syst.*, M. E. Sharpe, Inc., Armonk, NY, USA, v. 14, n. 4, p. 113–147, 1998. ISSN 0742-1222.

[Krueger 1992]KRUEGER, C. W. Software reuse. *ACM Comput. Surv.*, ACM Press, New York, NY, USA, v. 24, n. 2, p. 131–183, June 1992. ISSN 0360-0300.

[Larsen 2009]LARSEN, G. *Asset lifecycle management for service-oriented architectures.* 2009. Disponível em: <http://www-128.ibm.com/developerworks/rational/library/oct05/wilber>. Acesso em: dezembro 2009.

[Mcilroy 1968]MCILROY, D. Mass-produced software components. In: *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany*. [S.l.: s.n.], 1968. p. 88–98.

[Mili 1998]MILI, A. A survey of software reuse libraries. *Ann. Softw. Eng.*, J. C. Baltzer AG, Science Publishers, Red Bank, NJ, USA, v. 5, p. 349–414, 1998. ISSN 1022-7091.

[Minneto 2007]MINNETO, E. L. *Frameworks para Desenvolvimento de Software em PHP.* [S.l.]: Novatec, 2007.

[Murta 2008]MURTA, L. *Repositórios de Componentes*. Porto Alegre, RS: [s.n.], 2008. Palestra ministrada no Segundo Simpósio Brasileiro de Componentes, Arquitetura e Reutilização de Software, SBCARS.

[OMG 2005]OMG. *Reusable Asset Specification, Version 2.2*. 2005. Disponível em: <http://www.omg.org/cgi-bin/doc?formal/2005-11-02g>. Acesso em: dezembro 2009.

[RISE-BART 2009]RISE-BART. *Basic Asset Retrieval Tool*. 2009. Disponível em: <http://www.rise.com.br/whitepaper/whitepaper_BART.pdf>. Acesso em: dezembro 2009.

[RISE-CORE 2009]RISE-CORE. *Component Repository*. 2009. Disponível em: <http://www.rise.com.br/whitepaper/whitepaper_CORE.pdf>. Acesso em: dezembro 2009.

[Sherif e Vinze 2003]SHERIF, K.; VINZE, A. Barriers to adoption of software reuse a qualitative study. *Inf. Manage.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 41, n. 2, p. 159–175, 2003. ISSN 0378-7206.

[Sonatype-Nexus 2009]SONATYPE-NEXUS. *Nexus: A Repository Manager*. 2009. Disponível em: <http://nexus.sonatype.org>. Acesso em: dezembro 2009.

[SPARX 2009]SPARX. *Reusable Asset Management Tool: ARCSeeker*. 2009. Disponível em: <http://www.arcseeker.com>. Acesso em: dezembro 2009.

[Tracz 1988]TRACZ, W. Software reuse myths. *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, USA, v. 13, n. 1, p. 17–21, 1988. ISSN 0163-5948.

[Wang et al. 2008]WANG, Z.-S. et al. Design and implementation of ras-based reusable asset management tool. In: *ICICSE '08: Proceedings of the 2008 International Conference on Internet Computing in Science and Engineering*. Washington, DC, USA: IEEE Computer Society, 2008. p. 363–366. ISBN 978-0-7695-3112-0.