UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

DIONATAN DE SOUZA MOURA

Software Profile RAS: Estendendo a Padronização do Reusable Asset Specification e Construindo um Repositório de Ativos

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Marcelo Soares Pimenta
Orientador

Porto Alegre, dezembro de 2013.

CIP – CATÁLOGO NA PUBLICAÇÃO

Moura, Dionatan de Souza

Software Profile RAS: Estendendo a Padronização do Reusable Asset Specification e Construindo um Repositório de Ativos / Dionatan de Souza Moura. – 2013.

132 f.:il.

Orientador: Marcelo Soares Pimenta.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2013.

1.Reúso de Software. 2.Reutilização de Software. 3.Modelo de Ativo Reutilizável. 4.Reusable Asset Specification. 5.RAS. 6.Repositório de Ativos Reutilizáveis de Software. 7.Repositório de Reutilização de Software. 8.Biblioteca de Reutilização de Software I. Pimenta, Marcelo Soares. II. Título.

AGRADECIMENTOS

De uma forma simples, porém muito importante:

- Agradeço aos meus pais, pela dedicação.

- Agradeço ao professor Marcelo Pimenta, pela orientação e motivação.

- Agradeço à UFRGS, pelo ensino e pesquisa de qualidade.

"Nada se perde, nada se cria, tudo se transforma."
(Antoine Lavoisier)

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ACM	Association for Computing Machinery
AJAX	Asynchronous JavaScript and XML
ALOAF	Asset Library Open Architeture Framework
AMT	Asset Management Tool
ANZSRC	Australian and New Zealand Standard Research Classification
API	Application Programming Interface
Arch.	Architecture
AUI	Abstract UI
CBSE	Component-based Software Engineering
CDM	Common Data Model
CVS	Concurrent Versions System
CUI	Concrete UI
ECM	Enterprise Content Management
ESDS	Earth Science Data Systems (NASA)
FUI	Final UI
GNU	GNU's Not Unix!
GoF	Gang of Four
GPL	GNU General Public License
HTTPS	HyperText Transfer Protocol Secure
ID	Identificador
IDE	Integrated Development Environment
i18n	Internationalization
IBM	International Business Machines
IEC	International Electrotechnical Commission

ISO	International Organization for Standardization
IU	Interface de Usuário
IT	Information Technology
J2EE	Java 2 Platform, Enterprise Edition
JACS	Joint Academic Coding System
JavaEE	Java Enterprise Edition
JBCL	Jade Bird Component Library System
JSF	JavaServer Faces
JPA	Java Persistence API
LDAP	Lightweight Directory Access Protocol
MPC	Multi Phased Component
MPS-BR	Melhoria de Processos do Software Brasileiro
MVC	Model View Controller
NASA	National Aeronautics and Space Administration
OMG	Object Management Group
PDF	Portable Document Format
PHP	PHP: Hypertext Preprocessor
RA	Reuse Assistant
RAS	Reusable Asset Specification
RAM	Rational Asset Manager (IBM)
RES	Reuse Enablement System (NASA)
ROI	Return on Investment
SOA	Service-oriented Architecture
SVN	Subversion
SUI	Sketched UI
SW-RAS	Software Profile RAS
TI	Tecnologia da Informação
T&C	Task & Concepts
X-ARM	XML-based Asset Representation Model
XML	eXtensible Markup Language
XUI	eXecutable UI

WG	(NASA Software Reuse) Working Group
WSDL	Web Services Description Language
UDM	Uniform Data Model
UI	User Interface
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O reúso de software enfrenta inúmeras barreiras gerenciais, técnicas e culturais na sua adoção, e a definição da estrutura de ativos reutilizáveis de software é uma dessas barreiras técnicas. Para solucionar isso, o Reusable Asset Specification (RAS) é um padrão *de facto* proposto pela OMG. Uma especificação como o RAS define e padroniza um modelo de ativos (assets) reutilizáveis, e é a base para a construção e para o uso de um repositório de ativos que apoia a reutilização de software. No entanto, para ser adotado na prática, o RAS necessita resolver suas lacunas através da sua extensão e da definição de informações complementares. Essas lacunas estão detalhadas neste trabalho. Solucionando estas lacunas, o RAS torna-se útil para auxiliar efetivamente na padronização do empacotamento dos ativos reutilizáveis e para guiar a estrutura do repositório de reutilização de software. Alguns trabalhos anteriores já responderam parcialmente essa questão, porém eles atendiam propósitos muito específicos, não possuíam uma ferramenta de apoio ou não haviam sido avaliados em contexto real de (re)uso. Esse trabalho propõe o Software Profile RAS (SW-RAS), uma extensão do Profile de componentes do RAS, que propõe soluções para diversas de suas lacunas, incluindo informações úteis e artefatos relevantes apontados na literatura, baseados em outros modelos de ativos reutilizáveis, em outras extensões do RAS e na experiência do processo de reúso no desenvolvimento de software. Particularmente, o SW-RAS estende as categorias de classificação, solução, uso e ativos relacionados, cujos detalhes estão descritos no texto. Visando à experimentação da proposta através de um estudo de caso, desenvolveu-se o Lavoi, um repositório de ativos reutilizáveis baseado no SW-RAS, que foi avaliado num ambiente real de reutilização e desenvolvimento de software de uma grande companhia pública de TI. Uma descrição deste processo de avaliação em um contexto real é também apresentada neste trabalho. A principal contribuição desta dissertação é a proposta, a avaliação e a consolidação de uma extensão do RAS que atende várias de suas lacunas e é suportada por uma ferramenta de software livre.

Palavras-chave: Reúso de Software, Reutilização de Software, Modelo de Ativo Reutilizável, Reusable Asset Specification, RAS, Repositório de Ativos Reutilizáveis de Software, Repositório de Reutilização de Software, Biblioteca de Reutilização de Software

ABSTRACT

The software reuse faces numerous managerial, technical and cultural barriers in its adoption, and the definition of the structure of reusable software assets is one of these technical barriers. To solve this, the Reusable Asset Specification (RAS) is a *de facto* standard proposed by OMG. A specification such as the RAS defines and standardizes a reusable asset model, and it is the foundation for the construction and for the use of an asset repository that supports the software reuse. However, for being adopted in the practice, the RAS needs to solve its lacks through its extension and the definition of complementary information. These lacks are detailed in this work. Solving these lacks, the RAS becomes useful to help effectively in the standardization of packaging reusable assets and to guide the structure of the software reuse repository. Some previous works have already partially answered this question, but they attended very specific purposes, did not have a support tool or have not been evaluated in a real context of (re)use. This work proposes the Software Profile RAS (SW-RAS), an extension of the component Profile of RAS, which proposes solutions for its various lacks, including useful information and relevant artifacts pointed out in the literature, based on other reusable asset models, on other RAS extensions and on the experience in the reuse process at software development. Particularly, the SW-RAS extends the categories of classification, solution, usage and related assets, whose details are described in the text. Aiming at the experimentation of the proposal through a case study, the Lavoi was developed, a reusable asset repository based on the SW-RAS, which is was evaluated in a real environment of reuse and software development of a large public IT company. A description of this evaluation process in real context is also presented in this work. The main contribution of this dissertation is the proposal, the evaluation and the consolidation of an extension of RAS that addresses several of its lacks and is supported by a free software tool.

Keywords: Software Reuse, Reusable Software Asset Model, Reusable Asset Specification, RAS, Reusable Asset Repository, Software Reuse Repository, Software Reuse Library

1 INTRODUÇÃO

A ideia de reutilizarmos software tem sido estudada por décadas, desde que pioneiramente McIlroy (1968) instigou que o estudo de componentes de software deveria ser um ramo específico da Engenharia de software, propondo catálogos padronizados de diferentes rotinas intercambiáveis, classificadas de diversas maneiras, de fácil adaptação, definindo explicitamente graus de confiabilidade na qualidade destas rotinas. O reúso de software não é só um desejo, mas uma necessidade que enfrenta inúmeras barreiras técnicas, gerenciais e culturais para ser adotado. Uma das principais barreiras técnicas está em como definir a estrutura de um ativo (*asset*) reutilizável de software, empacotando as informações e os artefatos necessários no processo de reutilização de software, abrangendo todo seu ciclo de vida, incluindo sua produção, certificação, reutilização, manutenção e evolução. Um modelo de ativos reutilizáveis de software define e padroniza a estrutura dos ativos reutilizáveis, e é a base para a construção e uso de um repositório de ativos que apoia a prática de reúso de software.

Ativos reutilizáveis são compostos de uma coleção de produtos de trabalho relacionados que podem ser reutilizados de uma aplicação para outra (EZRAN, 2002). Complementando, um ativo é um conjunto de informações e artefatos que contém ao menos uma solução para um problema específico (OMG, 2005). Os ativos podem abranger quaisquer artefatos reutilizáveis no ciclo de vida do desenvolvimento de software, tais como códigos-fontes, componentes, serviços, frameworks, padrões de projeto, padrões de processos de negócio, interfaces de usuários, documentos de requisitos, modelos de domínio, scripts e casos de testes. Em geral, um artefato é associado a um arquivo, ou a uma referência. A especificação de um ativo deve demonstrar como descrevê-lo, armazená-lo e empacotá-lo através de um modelo, proporcionando um melhor entendimento através da documentação e da descrição de seu conteúdo, facilitando a catalogação, a pesquisa, a manutenção e a reutilização do software. Tais ativos muitas vezes são chamados de componentes, por ser em uma das formas mais comuns de ativo. Os ativos relacionados a este trabalho se referem aos desenvolvidos e projetados para serem reutilizáveis em software, mesmo que por algumas vezes sejam chamados simplesmente de ativos. A Figura 1.1 ilustra um ativo com três artefatos e um ponto de variabilidade para um determinado contexto, com regras para seu uso.

Figura 1.1: Um ativo contém uma solução para um problema específico – adaptada e traduzida de (OMG, 2005).

O reúso de software traz os benefícios de aumentar a confiança do software, reduzir o risco do processo, usar efetivamente o conhecimento de especialistas, estar em conformidade com padrões e ter um desenvolvimento de software acelerado (SOMMERVILLE, 2011). Porém, alguns problemas podem ocorrer, tais como o custo elevado de manutenção por não se possuir o código-fonte dos artefatos binários, a carência de suporte de ferramentas, e a síndrome "*Not-invented-here*" em desenvolvedores que preferem reescrever componentes acreditando que podem criá-los de uma maneira melhor. A criação, a manutenção e a utilização de tais repositórios também pode ser um problema, assim como encontrar, compreender e adaptar componentes reutilizáveis. Um modelo de ativos auxilia nesses benefícios definindo itens de qualidade e de certificação dos ativos, estruturando a documentação do conhecimento de especialistas e padronizando a estrutura dos ativos com artefatos reutilizáveis para acelerar o desenvolvimento de software. Um modelo de ativos também pode reduzir os riscos na reutilização de software definindo o empacotamento de códigos-fontes além dos artefatos executáveis, gerando mais confiança aos desenvolvedores através de artefatos de teste e guias de utilização, estruturando a classificação, a solução e o uso dos ativos reutilizáveis para que sejam facilmente encontrados em repositórios, também sendo compreendidos e adaptados pelos desenvolvedores de software.

O Reusable Asset Specification (OMG, 2005) é um padrão de facto para modelo de ativos reutilizáveis, porém seu modelo possui lacunas para satisfazer as informações e artefatos necessários no processo de reutilização de software. Possui lacunas relacionadas à qualidade do ativo, à reusabilidade, à licença de software, a artefatos de interfaces de usuários, de códigos-fontes e de testes, a perfis de usuários, a guias de usuários, entre outras identificadas neste trabalho. O problema em questão encontra-se na seção 1.1 e a introdução ao trabalho proposto encontra-se na seção 1.2.

1.1 O Problema

Alguns modelos de ativos reutilizáveis já foram propostos na literatura (vide seção 2.2), sendo parte deles utilizados em repositórios de reutilização de software. Além disso, diversos repositórios de ativos reutilizáveis foram construídos sem adotar explicitamente um modelo de ativos para sua estrutura e padronização (vide seção 2.2). Dentre os modelos existentes, o mais utilizado e promissor é o Reusable Asset Specification (RAS) (OMG, 2005), pois é uma especificação do Object Management Group (OMG). O RAS visa a ser o padrão da descrição e empacotamento de ativos reutilizáveis de software, da mesma forma que a Unified Modeling Language (UML) é

definida pelo OMG para linguagens de modelagem. O RAS estrutura um conjunto de informações e artefatos mínimos para um ativo através Core RAS, o qual é instanciado pelo Default Profile. Dois outros Profiles estendem o Default Profile, um para componentes (o Default Component Profile) e outro para web services (o Default Web Service Profile). Na literatura, alguns trabalhos estenderam o RAS, através do Default Profile ou do Default Component Profile, assim como alguns repositórios adotaram o RAS ou alguma das suas extensões.

O problema em questão é que, mesmo sendo um padrão *de facto* na comunidade internacional, o RAS possui lacunas em seu modelo de ativos para que a reutilização de software seja satisfatoriamente atendida (vide extensões do RAS na seção 2.10 e as lacunas no Capítulo 3). Esse problema é significativo porque informações e artefatos de software para a reutilização não são empacotados nos ativos por não serem especificados no modelo, ficando apenas na ferramenta do repositório baseado no RAS, e portanto sua descrição é mal estruturada ou inexistente. Alguns exemplos de lacunas são relacionados ao tipo do ativo, à qualidade, à reusabilidade, a domínios de aplicação, a licenças de software, à classificação colaborativa, a padrões de projetos, a interfaces de usuários, a códigos-fontes, a linguagens de programação, a artefatos e métodos de testes, a perfis de usuários, a guias de usuários e ao relacionamento de ativos pela versão anterior. Os trabalhos que estenderam a especificação do RAS já responderam parcialmente esse problema em questão (vide seção 2.10), porém não são satisfatórios porque ou atendem focos muito específicos, contendo uma baixa especificação de suas soluções para as lacunas, ou não foram adotados em repositórios suportados por ferramentas, nem utilizados em contexto real, sem uma avaliação ou experimentação efetiva.

1.2 Trabalho Proposto

O objetivo desse trabalho é apresentar o Software Profile RAS (SW-RAS), uma extensão do RAS proposta para resolver várias de suas lacunas, contribuindo com uma maior padronização das informações e artefatos empacotados nos ativos para atender o processo de reutilização no desenvolvimento de software. O SW-RAS estende o Default Component Profile de uma forma compatível, através de extensões na classificação dos ativos, na solução que os ativos abrangem, no uso dos ativos e nos ativos relacionados. Essas extensões propostas se baseiam em características relevantes de outros modelos de ativos, nos artefatos reutilizáveis no ciclo de vida do software, no processo de reutilização de software e em outras extensões do RAS, visando a serem enxutas (simples) e suficientes.

São as extensões propostas pelo SW-RAS:

- Classificação pelo Tipo
- Classificação pela Qualidade Avaliada por Usuários
- Classificação pela Reusabilidade
- Classificação por Domínios de Aplicação
- Classificação por Organizações e Projetos
- Classificação por Licenças de Software

- Classificação por Custos e Esforços
- Classificação por Social Tagging
- Solução com Requisitos Funcionais e Requisitos Não-funcionais
- Solução com Padrões de Projeto
- Solução com Interfaces de Usuários e seus Níveis de Abstração
- Solução com Serviços
- Solução com Códigos-fontes e Linguagens de Programação
- Solução com Artefatos e Métodos de Testes
- Uso com Perfis de Usuários Consumidores, Produtores e Certificadores
- Uso com Guias de Usuário e Descrições de Uso
- Uso com Comentários de Usuários
- Relacionamento de Ativos pela Versão Anterior

Para a experimentação e avaliação do SW-RAS, implementou-se o sistema de repositório ativos reutilizáveis de software denominado Lavoi, estruturado pelas extensões do SW-RAS de uma forma integrada e padronizada. Para o estudo de caso, o Lavoi está sendo utilizado no ambiente de desenvolvimento de software de uma grande companhia pública brasileira de TI. Como resultado, a proposta de solução para as lacunas no RAS foi sendo refinada e avaliada pelo uso real, mas tentando preservar a generalidade e evitando se tornar específica apenas para atender as características intrínsecas do processo de reutilização de software e da maturidade do ambiente de desenvolvimento dessa companhia.

Como contribuição secundária desse trabalho, o sistema de repositório Lavoi está disponibilizado como software livre para a comunidade internacional de software. Atualmente existem raras ferramentas semelhantes disponíveis para uso, sendo de uso restrito ou limitado a soluções proprietárias. Para suprir essa indisponibilidade, os ambientes de reúso de software comumente utilizam outras ferramentas não específicas, tais como de gerenciamento de conteúdo (ECM – gestão de conteúdo empresarial) ou sistemas de versionamento (CVS, SVN, Git). Em formas menos estruturadas ainda, os artefatos dos ativos são armazenados em um servidor de arquivos com baixa documentação, versionados em cópias manuais, sem mecanismos de pesquisa e de gerenciamento. Desse modo, acredita-se ser interessante que uma solução padronizada pelo RAS na forma de software livre seja disponibilizada para a comunidade internacional. Por ser software livre, qualquer pessoa ou entidade pode utilizar o Lavoi ou modificá-lo para seu contexto, podendo contribuir com sua evolução.

O Lavoi é um sistema de repositório de contexto amplo que provê mecanismos de armazenamento e busca de ativos reutilizáveis de software, podendo conter ativos em níveis horizontais e verticais, e em diferentes pontos nas dimensões de granularidade, variabilidade e articulação dos ativos. Exemplos de ativos podem ser componentes, ativos baseados em códigos-fontes de implementação e testes, interfaces de usuários, serviços, padrões de projeto, ativos de domínio, entre outros. Seu mecanismo de

armazenamento se dá através de um wizard (passo a passo) que conduz o cadastro dos ativos, guiado pela estrutura do SW-RAS. Os mecanismos de busca são o textual com relevância, o textual com relevância campo a campo, a nuvem de tags e a navegação por tags. O texto é buscado dentre todas as informações de classificação, de solução, de uso e de ativos relacionados. A relevância leva em conta a média da qualidade avaliada pelos usuários, a reusabilidade (quantidade de consumos do ativo), o estado do ativo e o peso semântico de cada atributo no ativo. O estado de ativo com maior relevância é o estado certificado, e o com menor relevância é o estado descontinuado. O Lavoi notifica (por e-mail) os usuários produtores, consumidores e certificadores sobre o ciclo de vida e utilização dos ativos. Sua arquitetura Java web é escalável e utiliza um sistema de indexação para as buscas dos ativos, permitindo o armazenamento e a pesquisa de ativos reutilizáveis em larga escala.

Este trabalho é estruturado como segue. No capítulo 2, conceitos gerais são definidos e os modelos de ativos reutilizáveis existentes na literatura são apresentados, incluindo o Reusable Asset Specification e os trabalhos anteriores para solucionar suas lacunas. No capítulo 3, são apresentadas as lacunas do RAS e o Software Profile RAS é detalhado por suas extensões para a classificação dos ativos, a solução, o uso e os ativos relacionados. O Lavoi é descrito no Capítulo 4. No capítulo 5, é descrita a avaliação do SW-RAS através da comparação com os trabalhos anteriores de extensão do RAS, além de apresentar um estudo de caso realizado com a utilização do Lavoi em um ambiente real de desenvolvimento de software, junto de uma pesquisa com os usuários do Lavoi. As conclusões do trabalho são discutidas no capítulo 6.

2 MODELOS DE ATIVOS REUTILIZÁVEIS DE SOFTWARE

Esse capítulo apresenta os conceitos gerais (seção 2.1), os modelos de ativos reutilizáveis de software da literatura (seção 2.2), junto do Reusable Asset Specification (RAS) (seção 2.9) e dos trabalhos anteriores de extensão do RAS (seção 2.10).

2.1 Conceitos Gerais

Existem dois níveis de ativos reutilizáveis, o vertical e o horizontal (ALMEIDA et al., 2007). Ativos verticais são desenvolvidos especificamente para domínios de aplicação, ou seja, para famílias de aplicações com características comuns, por exemplo o domínio financeiro ou o domínio da medicina. Já os ativos horizontais podem ser reutilizados independentemente do domínio de aplicação, pois são elementos arquiteturais recorrentes, tais como objetos de interfaces de usuário, bibliotecas de acesso ao banco de dados, serviços de autenticação, frameworks para desenvolvimento de sistemas, padrões de projeto, entre diversos outros.

Além dos níveis horizontal e vertical, um ativo pode ser descrito em três dimensões chaves: granularidade, variabilidade e articulação (RAS, 2005), ilustradas na Figura 2.1. A dimensão da granularidade descreve quantos problemas em particular, ou alternativas de solução, um ativo empacotado trata. Quão mais próximo de uma coleção é sua granularidade, mais problemas o ativo resolve ou mais alternativas de solução são dadas, e maior é o seu tamanho e sua complexidade. Ezran (2002) lista exemplos de níveis de granularidade: uma função ou procedimento, uma classe, um grupo de classes, um subsistema, um framework ou um serviço, uma aplicação ou um produto.

A segunda dimensão, a variabilidade, descreve em quantas formas significativas o ativo pode variar. Em um dos extremos dessa dimensão o ativo é invariável, chamado de ativo caixa-preta (*black-box*), visto que sua estrutura interna não pode ser vista, nem modificada, por exemplo um componente binário. No outro extremo da dimensão estão os ativos caixa-branca (*white-box*), possuindo a variabilidade completa, sendo criados para que os consumidores alterem sua implementação, por exemplo padrões de projeto, requisitos, modelos ou códigos-fontes. No ponto médio dessa dimensão estão os ativos caixa-transparente (*clear-box*) e os ativos caixa-cinza (*gray-box*). Ativos caixa-transparente não podem ser modificados, porém expõem seus detalhes de implementação para serem melhor entendidos internamente. Ativos caixa-cinza expõem sua implementação e permitem modificações somente em alguns de seus artefatos com pontos de variabilidade, habitualmente por parâmetros, como exemplo um framework.

A terceira dimensão, a articulação, é sobre o nível de completude dos artefatos da

solução. Ativos com baixo nível de articulação contêm artefatos que especificam a solução porém não a provêm. Quanto mais artefatos especificados e implementados para uma solução, maior é o grau da articulação, podendo ser através de requisitos, modelos, casos de uso, artefatos de teste, entre diversos outros.

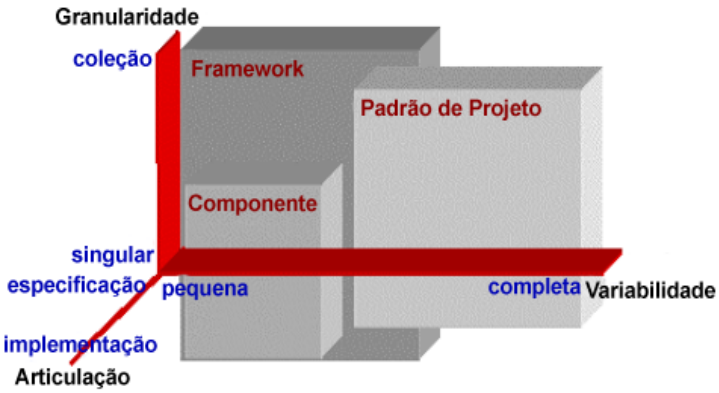


Figura 2.1: Dimensões dos ativos de granularidade, variabilidade e articulação – adaptada e traduzida de (OMG, 2005).

De acordo com Ezran (2002), ativos são vivos, possuindo estados em um ciclo de vida, pois farão parte de diversas aplicações e serão gerenciados por um longo período de tempo. A Figura 2.2 ilustra diversos estados dos ativos reutilizáveis de software. Depois de especificado, o ativo será produzido e então publicado no repositório (IBM, 2008). Quando estiver pronto e ser revisado, alguns ajustes poderão ser realizados até que ele então seja certificado e classificado. Após isso, o ativo estará disponível para outras pessoas da organização o consumirem. Esses consumidores estarão habilitados a baixar o ativo, bem como a prover comentários e pareceres sobre o ativo. Ao longo do tempo, o ativo pode ser atualizado, descontinuado ou removido de seu repositório.

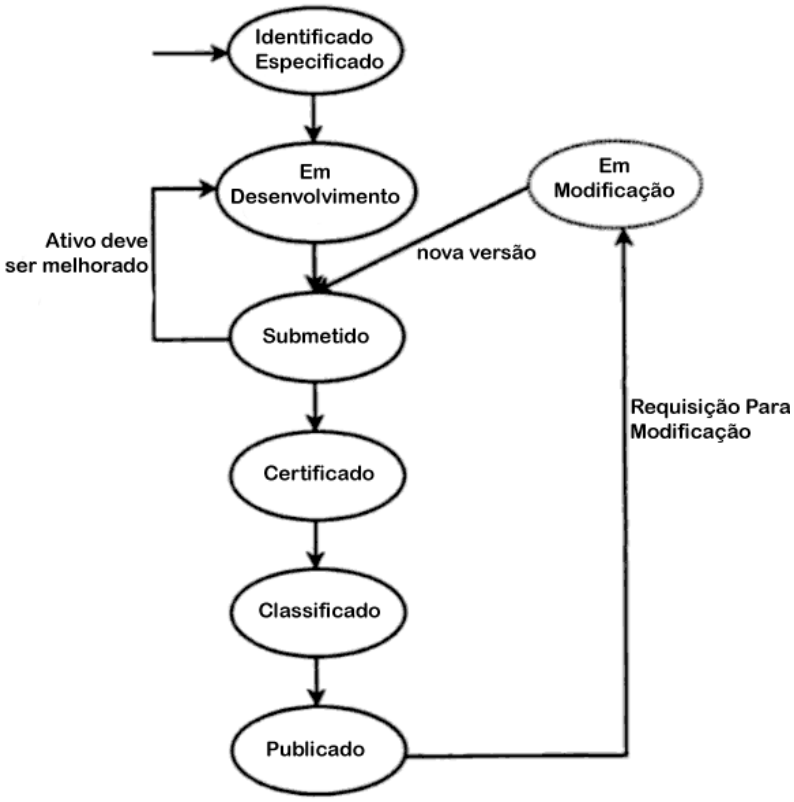


Figura 2.2: Estados em um ativo reutilizável de software – adaptada e traduzida de (EZRAN, 2002).

Um repositório é um local onde ativos reutilizáveis de software são armazenados, com mecanismos de pesquisa e recuperação (EZRAN, 2002; ALMEIDA et al., 2007; MILI, 1998; IBM, 2008). Repositórios de ativos reutilizáveis também são denominados de bibliotecas ou catálogos, ou então de repositórios de reutilização de software. O repositório gerencia o ciclo de vida dos ativos com foco no processo de reutilização de software da organização, sendo que todos na organização devem estar cientes que ele contém importante conhecimento especializado, estando aptos para acessá-lo e usá-lo facilmente. Os repositórios de ativos reutilizáveis podem ser especializados para suportar contextos especializados, tais como para artefatos relacionados ao projeto e ao desenvolvimento, de artefatos executáveis, ou de ativos relacionados a um certo domínio (IBM, 2008).

O investimento no desenvolvimento antecipado de artefatos reutilizáveis e a infraestrutura para torná-los disponíveis e fáceis de encontrar faz parte de uma combinação de fatores que habilita o reúso de software em uma organização (SOJER, 2011). Shiva (2007) cita que o maior desafio sobre o reúso de software baseado em componentes é gerenciar um grande número de componentes reutilizáveis eficientemente para permitir rápida alocação e recuperação. Além disso, Shiva enuncia que ainda não foi acordada a melhor maneira para estruturar os repositórios e implementar sua interface de pesquisa. Para Sherif (2003), problemas com ferramentas de suporte são responsáveis por um descontentamento para a adoção de reúso de software, pela dificuldade em encontrar um ativo reutilizável, podendo ser causada pela estrutura do repositório. A estrutura dos repositórios é um fator essencial para se obter bons resultados na recuperação dos ativos (SHIVA, 2007), porém ainda é um problema encontrar, entender e adaptar componentes reutilizáveis (SANDHU et al., 2010). Além

disso, a descrição imprecisa ou inexistente é uma barreira ao reúso do projeto, da implementação e da documentação de ativos reutilizáveis, porque a dificuldade aumenta ao buscar e selecionar ativos reutilizáveis quando os dados sobre os ativos não contém informações úteis e necessárias (SHERIF, 2003).

Um repositório auxilia no processo de reutilização de software por facilitar a catalogação e a pesquisa dos ativos produzidos ou adquiridos, utilizando uma padronização para a estrutura das informações e dos artefatos reutilizáveis. Nos processos da Engenharia de Software Baseada em Componentes (CBSE, do inglês) (Figura 2.3), o repositório auxilia na catalogação dos ativos desenvolvidos através do processo CBSE Para Reutilização, na busca dos ativos a serem reutilizados no processo CBSE Com Reutilização, no suporte da Certificação de Componentes, na busca de componentes em fontes externas para o processo de Aquisição de Componentes, e na Gerência de Componentes garantindo que os componentes estão propriamente catalogados, armazenados e disponíveis para o reúso (SOMMERVILLE, 2011). Na Engenharia de Domínio, o repositório proporciona a catalogação e a disseminação dos ativos que foram identificados e construídos para serem reutilizados em domínios de aplicações específicos (PRESSMAN, 2011).

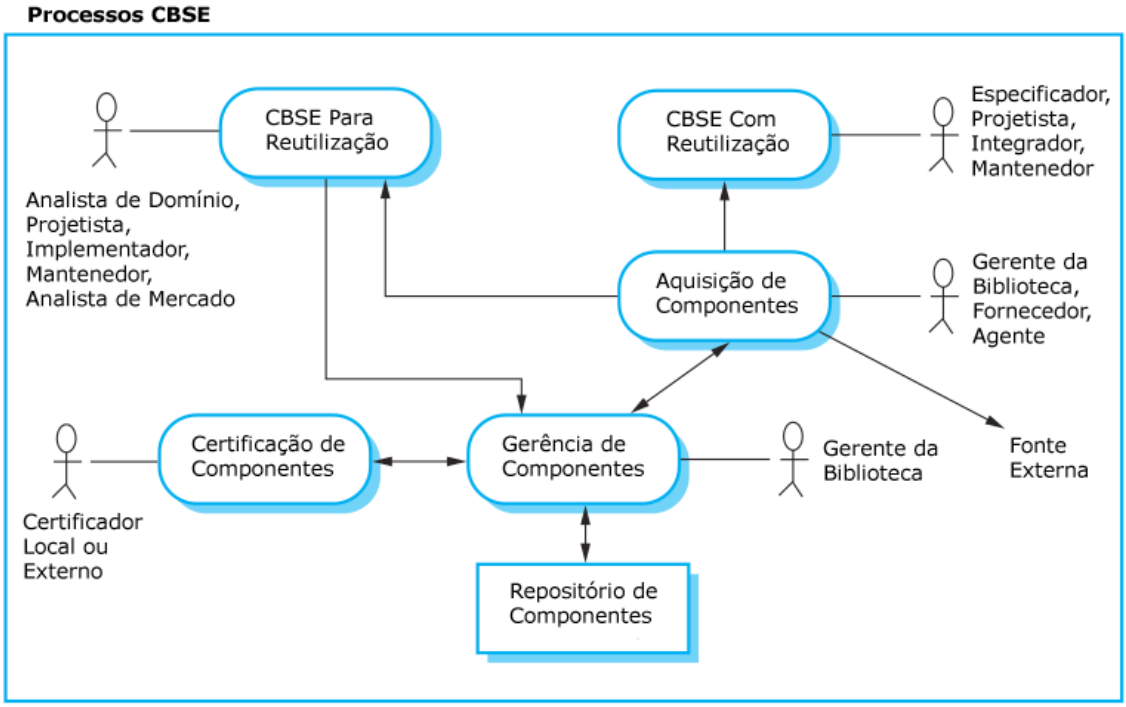


Figura 2.3: Repositório de componentes entre os processos da Engenharia de Software Baseada em Componentes – adaptada e traduzida de (SOMMERVILLE, 2011).

Um modelo de ativos reutilizáveis de software especifica, estrutura e padroniza as informações e os artefatos relevantes nos ativos para o empacotamento, devendo definir as diversas informações sobre a classificação, a solução, o uso e os ativos relacionados. Essa definição é importante porque influencia diretamente na estrutura do repositório de reutilização de software para o armazenamento, a classificação, o gerenciamento e o uso de tais ativos. Um modelo também serve como padronização para a interoperabilidade de repositórios para que os usuários possam usufruir dos ativos em diferentes locais com uma estrutura padronizada. Além disso, os dados do ciclo de vida do ativo e de sua utilização no processo de reutilização de software também devem estar contemplados

pelo modelo de ativos, por serem informações relevantes.

Através da estrutura do modelo de ativos, o empacotamento é definido para que todas as informações e artefatos úteis e importantes estejam contempladas na própria estrutura do ativo, e não contidas apenas em ferramentas, desatualizadas, perdidas ou nem sequer descritas. Todo pacote de ativo reutilizável deve conter ao menos um artefato reutilizável e uma descrição com suas informações através de metadados, por muitas vezes se utilizando um arquivo XML (OMG, 2005). Os ativos podem ser empacotados em apenas um arquivo, ou desempacotados em diversos arquivos. Os artefatos podem permanecer em seus locais de origem ou serem movidos para um local específico quando empacotados. De acordo com Ezran (2002), esse pacote deve conter as informações:

- de classificação para facilitar sua recuperação;
- de descrições para facilitar o entendimento do problema e da solução que o ativo provê;
- de documentação para entender como utilizá-lo;
- de qualificação e de teste para facilitar sua avaliação e sua verificação;
- da sua origem para facilitar a obtenção de informações adicionais

2.2 Revisão de Modelos de Ativos Reutilizáveis de Software

Com o objetivo de identificar os modelos de ativos reutilizáveis de software existentes na literatura e a adoção de cada modelo por repositórios de reutilização de software, além dos trabalhos relacionados ao Reusable Asset Specification, realizou-se uma revisão bibliográfica nas bibliotecas digitais IEEE Xplore Digital Library, ACM Digital Library, SpringerLink e Google Scholar. Para isso, foram utilizadas as palavras-chaves, com seus sinônimos e palavras-chaves relacionadas:

- *Reusable Software Asset Model; Reusable Asset Model; Asset Data Model; Software Data Model; Reusable Software Asset;*
- *Software Reuse Repository; Reuse Repository; Software Repository; Software Reuse Library; Software Reuse System; Reusable Software Asset Repository; Software Asset Repository; Reusable Asset Repository;*
- *Reusable Asset Specification; RAS; OMG RAS;*

A relação entre os modelos de ativos reutilizáveis de software e os repositórios encontrados na revisão bibliográfica encontra-se na Tabela 2.1, cronologicamente por suas publicações. O hífen "-" simboliza que o repositório não adotou um modelo de ativos reutilizáveis explicitamente (primeira coluna da tabela), que o modelo proposto não possui um repositório que o adota (segunda coluna), ou que não há a disponibilidade da ferramenta publicamente para uso (terceira coluna). Como critério de inclusão na revisão bibliográfica, os repositórios considerados nesse trabalho são os de reutilização de software, construídos para ativos de software. Como critério de exclusão na revisão bibliográfica, não são consideradas as ferramentas não específicas, nem as de gerenciamento ou versionamento de conteúdo ou de documentos em geral.

A maior parte desses repositórios (14 de 23) não utiliza um modelo de ativos como padrão para sua estrutura, assim como alguns modelos (2 de 7) são propostos sem serem utilizados em repositórios. Para cada repositório, a disponibilidade para uso de sua ferramenta é apresentada. Apenas duas ferramentas de repositório estão disponíveis publicamente para uso, as quais utilizam o RAS sem estendê-lo. Uma é proprietária (RAM) e a outra é uma solução livre, porém específica para artefatos de desenvolvimento de software (Ras4Nexus).

São sete os modelos identificados na literatura através dessa revisão bibliográfica: Common Data Model, Uniform Data Model, JB Component Library Data Model, Asset Model, modelo do Reuse Repository, Software Packaging Domain Model e Reusable Asset Specification (RAS). Essa lista de modelos e repositórios apresenta uma grande tendência à adoção do RAS logo após a sua publicação em 2005, sendo que a maioria desses trabalhos relacionados ao RAS (7 de 10) entendem como necessário estendê-lo para resolver suas lacunas, representados com o texto "Extensão RAS". Dentre os trabalhos que estendem o RAS, poucos foram adotados por repositórios e validados pelo seu uso (2 de 7), que são o Asset Management Tool e o OpenCom.

Nas próximas seções desse capítulo são apresentados os modelos de ativos reutilizáveis. Após serão descritos os Profiles do RAS e suas extensões anteriores propostas na literatura.

Tabela 2.1: Relação entre modelos de ativos reutilizáveis de software e repositórios de reutilização de software na literatura.

Modelos de Ativos Reutilizáveis de Software	**Repositórios de Reutilização de Software**	**Disponibilidade da Ferramenta**	**Referência**
-	CATALOG	-	(FRAKES, 1986)
Common Data Model	STARS ALOAF	-	(SOLDERITSCH, 1992)
-	ESTRO	-	(MACCHINI, 1992)
Uniform Data Model	-	-	(HOBBS, 1993)
-	REBOOT	-	(SINDRE, 1995)
-	Software Thesaurus	-	(LLORENS, 1996)
-	Memphis	-	(WERNER et al., 1997)
JB Component Library Data Model	JBCL	-	(KEQIN et al., 1997)
-	Odyssey	-	(BRAGA, 1999)
-	SEA	-	(SILVA, 2000)
-	CodeBroker	-	(YE, 2001)
Asset Model	-	-	(EZRAN, 2002)
-	(Repositório sem denominação)	-	(SCHIROKY, 2002)
-	FrameworkDoc	-	(LACERDA, 2005)
Reusable Asset Specification (RAS)	-	-	(OMG, 2005)
(Modelo sem denominação)	Reuse Repository	-	(BURÉGIO, 2006)
Extensão RAS (Metamodel of Domain Architecture)	-	-	(MOON, 2006)
-	Brechó	-	(WERNER et al., 2007)
RAS Profile	IBM Rational Asset Manager (RAM)	Sim (proprietária)	(IBM, 2007)
Extensão RAS (Multi Phased Component)	-	-	(PARK, 2007)

Extensão RAS (X-ARM)	-	-	(SCHUENCK, 2007)
Extensão RAS (Asset Management Tool)	AMT	-	(WANG et al., 2008)
-	ReUse	-	(MARTINS, 2008)
-	Estendendo ReUse	-	(FRANCESCHINI, 2008)
Extensão RAS (Support User Context)	-	-	(HADJI, 2008)
Extensão RAS (OpenCom)	OpenCom	-	(HONG-MIN, 2009)
RAS Profile	RASPUTIN	-	(ROSA, 2009)
RAS Profile	Ras4Nexus	Sim (software livre)	(ALMEIDA, 2009)
-	Component Repository Tool	-	(PALUDO et al., 2011)
Software Packaging Domain Model	NASA RES	-	(NASA, 2011) e (MARSHALL, 2010)
Extensão RAS (Reuse Assistants)	-	-	(BASSO, 2013)

2.3 Common Data Model

O modelo de ativos Common Data Model (CDM) é proposto pelo repositório Asset Library Open Architeture Framework (ALOAF) (SOLDERITSCH, 1992) para definir um conjunto de serviços para transferência de ativos e suas descrições entre bibliotecas (repositórios). O CDM foi desenvolvido baseado nas experiências dos desenvolvedores desse sistema, utilizando modelos de dados e padrões de modelos baseados em troca de informações, definindo uma configuração mínima para funcionamento dos serviços do repositório desenvolvido.

Esse modelo consiste em uma hierarquia simples de classes com atributos e relacionamentos (Figura 2.4). Os ativos são representados por instâncias das classes do CDM. Object é a superclasse, e suas subclasses herdam os atributos de identificação (Identifier) e nome da classe (Class_Name). Os atributos em itálico representam os relacionamentos, tais como Asset.Is_Composed_Of [File] ou File.Is_Part_Of [Asset]. O ativo é representado pela classe Asset, com atributos de nome (Name), nome alternativo (Alternate Name), versão (Version), data de liberação (Release Date), descrição (Description) e restrições (Restrictions Apply), com os relacionamentos de composição de arquivos (Is_Composed_Of [File]), ascendência (Is_Ancestor_Of [Asset]) e descendência de ativos (Is_Descendant_Of [Asset]), dependência de ativos (Requires [Asset]), criação por organizações (Was_Created_By [Organization]) e endendimento por pessoas (Is_Understood_By [Person]).

A classe File representa o conteúdo do ativo, com o atributo de nome (Name) e o relacionamento de parte de um ativo (Is_Part_Of [Asset]). As classes de organização (Organization) e pessoa (Person) informam os criadores ou os contatos responsáveis dos ativos. A organização possui os atributos de nome (Name), nome substituto (Alternate Name), endereço (Address), telefone (Telephone Number) e os relacionamentos dos ativos criados (Created [Asset]). A pessoa possui nome (Name), endereço (Address), telefone (Telephone Number), e-mail (Electronic_Mails_Address) e os relacionamentos com os ativos em que pode ser é contatada (Is_Contact_For [Asset]). Um ativo tipicamente é representado por um único objeto Asset, um ou mais objetos Organization e Person, e um ou mais objetos File com seus relacionamentos.

O CDM não mostra a obrigatoriedade das classes, dos atributos e dos relacionamentos em seu modelo, além de não informar os tipos de cada atributo.

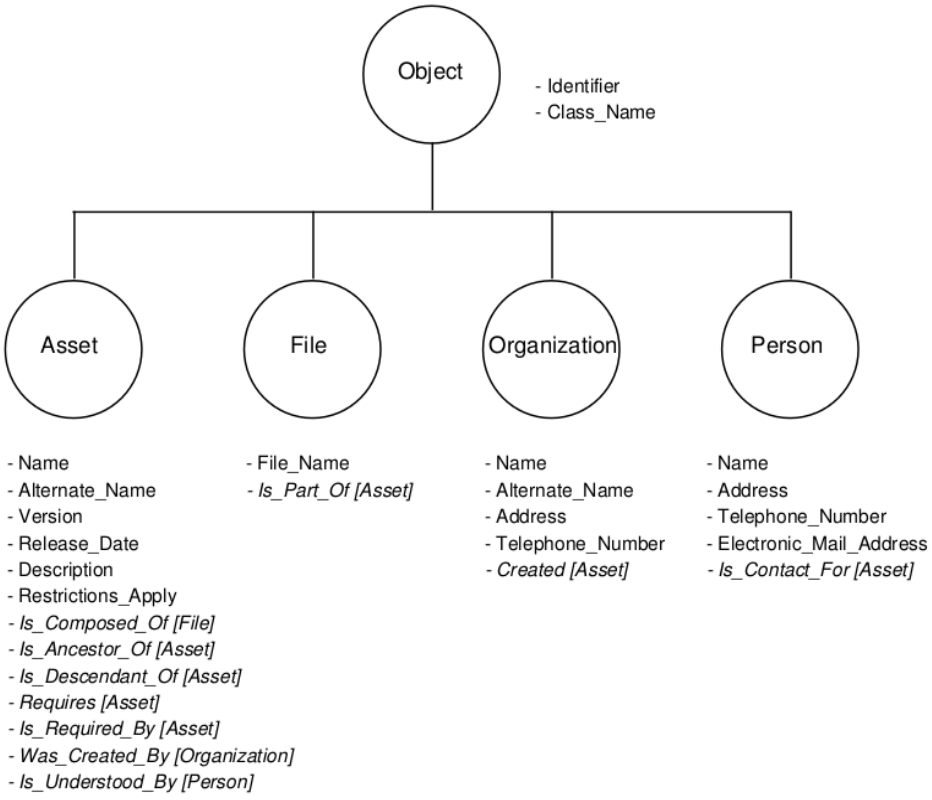


Figura 2.4: Hierarquia de classes do Common Data Model (SOLDERITSCH, 1992).

2.4 Uniform Data Model

O modelo Uniform Data Model (UDM) (HOBBS, 1993), desenvolvido pelo Reuse Library Interoperability Group (RIG), define a estrutura dos ativos que as bibliotecas (ou repositórios) devem intercambiar para suportar interoperabilidade. A interoperabilidade permite que os usuários possam acessar diversos ativos reutilizáveis em diferentes repositórios com uma estrutura padronizada de ativos. Esse modelo possui um conjunto maior de características que o CDM, pois estende um modelo mais básico de criação do mesmo grupo, denominado Basic Interoperability Data Model (BIDM), o qual utiliza conceitos do modelo CDM.

A Figura 2.5 exibe o modelo do UDM, dividido entre as classes RIGObject, Asset, Element, Library e Organization. Sua estrutura é semelhante ao CDM, com o RIGObject similar ao Object (CDM), assim como Element ao File (CDM), e Organization. A classe Element modela os artefatos de um ativo genericamente, tais como documentos, códigos-fontes e casos de teste. A classe Library informa os repositórios necessários para o intercâmbio de ativos. A classe Organization encapsula entidades como pessoas, companhias e comitês. Para a transferência de um ativo empacotado, a biblioteca que envia um ativo deve preencher todos os atributos e relacionamentos obrigatórios. A classe Asset traz novos atributos em relação ao CDM, tais como de resumo de descrição (Abstract), data de aceitação (AcceptanceDate), observância a padrões (ComplianceToStandards), custo (Cost), domínio (Domain), linguagem (Language), segurança (SecurityClassification), ambiente alvo (TargetEnvironment), garantias (Warranties), entre outros. A classe de repositório (Library) informa os métodos de classificação no atributo LibraryClassificationMethod,

as métricas (LibraryMetrics) e restrições (LibraryRestrictions).

O UDM não mostra a obrigatoriedade das classes, dos atributos e dos relacionamentos, nem as restrições semânticas em seu modelo, além de não informar os tipos de dados de cada atributo.

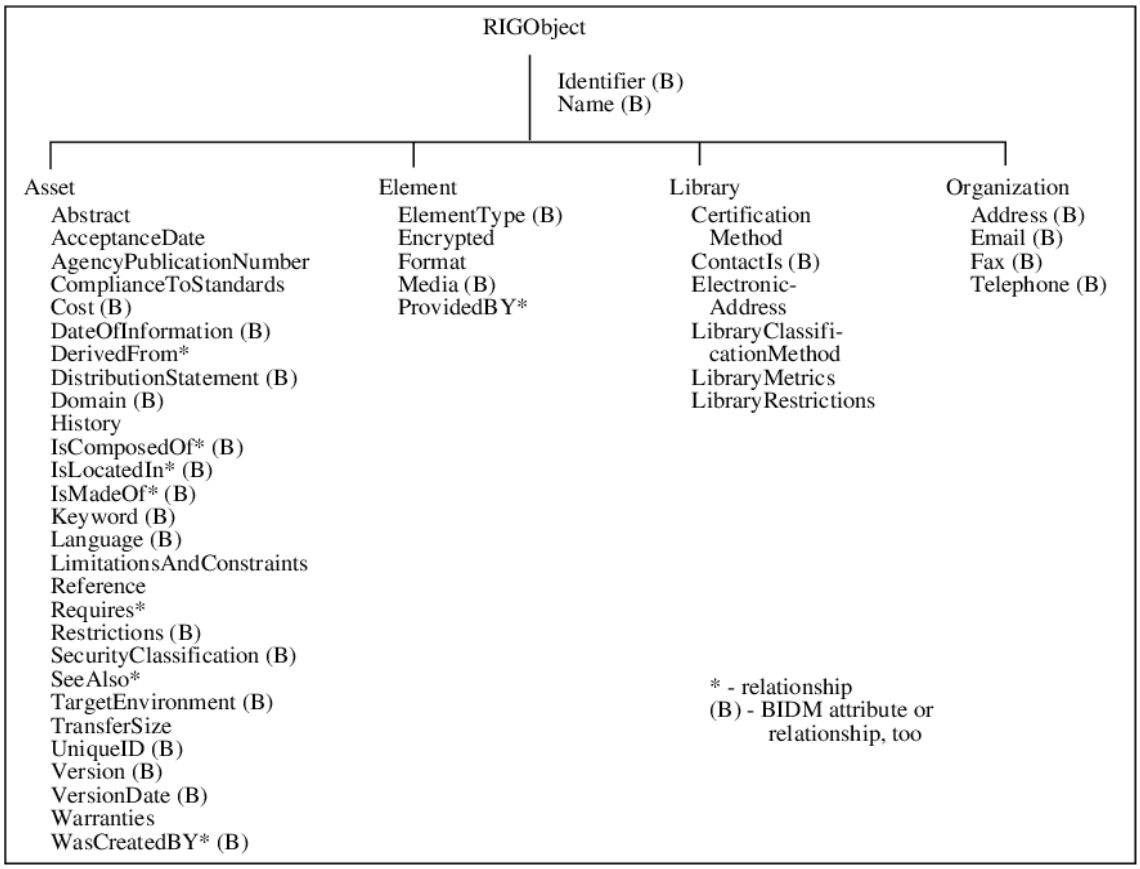


Figura 2.5: Uniform Data Model (HOBBS, 1993).

2.5 JB Component Library Data Model

O modelo JB Component Library Data Model (KEQIN et al., 1997) é desenvolvido como modelo para o repositório Jade Bird Component Library System (JBCL), tendo como base o UDM. Seu modelo é exibido na Figura 2.6, sendo a entidade central a de componente (Component), contendo relacionamentos dados pelo Relate. O uso (Use) é dado por usuários (User) e seus feedbacks (Feedback). Cada componente contém o seu conteúdo (Component Content), seu resumo (Component Abstract) e sua descrição numa linguagem de descrição específica (Component CDL Description). Sua classificação é realizada por facetas (Facet), as quais possuem termos de pesquisa (Term). O fornecedor do ativo é atribuído na entidade Provider.

Component possui alguns atributos não ilustrados na imagem do modelo, os quais são: nome, data de liberação, custo, tamanho, versão, histórico, forma (form), confirmação de distribuição (distribution statement), classificação de segurança e palavra-chave. Além disso, existem algumas relações de componentes, tais como a de refinamento (Refinement), que indica os artefatos produzidos a partir daquele componente no ciclo de vida do software. Outro relacionamento é o de versão, que indica todas as versões de evolução do componente. Os relacionamentos de ativos são

realizados através da entidade Relate, com os relacionamentos de similaridade, de cooperação e de herança.

O modelo define que as facetas de pesquisa seguem cinco características de componentes: ambiente de aplicação (sistema operacional, banco de dados, hardware, rede, compilador); domínio de aplicação; funcionalidades; nível de abstração (análise, projeto, código, teste, etc); e representação (linguagem de programação ou os diagramas utilizados na análise ou no projeto). Essas características são ortogonais e independentes.

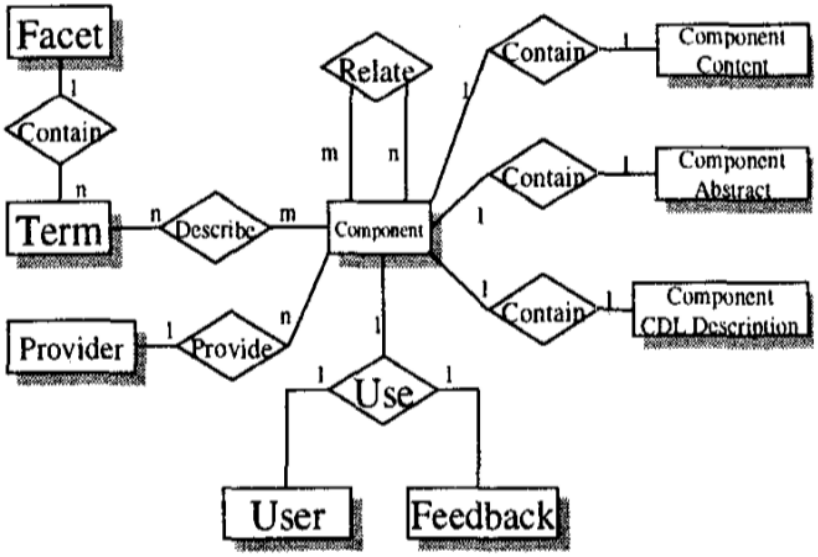


Figura 2.6: JB Component Library Data Model (KEQIN et al., 1997).

2.6 Asset Model

O Asset Model é proposto por Ezran (2002) com o propósito de definir a estrutura dos ativos em dois tipos de informação: seu corpo (Body) e sua descrição (Description), ilustrados na Figura 2.7. O corpo do ativo possui os produtos de trabalho reutilizáveis e a descrição contém a informação necessária para o processo de reúso. O ativo, descrito pela classe Asset, possui os atributos de identificação (Identifier), de origem (Origin) informando se é criado ou comprado, de granularidade (Granularity), e do uso em nível horizontal ou vertical. Os relacionamentos de ativos são de dois tipos: composição (is composed of) e uso (uses).

A descrição do ativo contém a informação administrativa (Administrative Information) com o autor e a data de criação; a informação de ambiente (Environment Information) com os ambientes de desenvolvimento (Development Environment), de execução (Execution Environment) e de testes (Test Environment); a informação de classificação (Classification Information) com as facetas (Facet) e palavras-chaves (Keywords); a informação de qualificação (Qualification Information) com os níveis de documentação (Documentation Level) e de confiança (Reliability Level); e o histórico de uso (Asset History) com seu contexto (Context) e resultado (Result).

O corpo do ativo descreve os produtos de trabalho tais como: modelo de análise (Analysis Model), modelo de projeto (Design Model), código-fonte (Source Code), documentação de usuário (User Documentation), código executável (Executable Code),

relatório de teste (Test Report), cenários de teste (Test Scenarios) e programa de teste (Test Program).

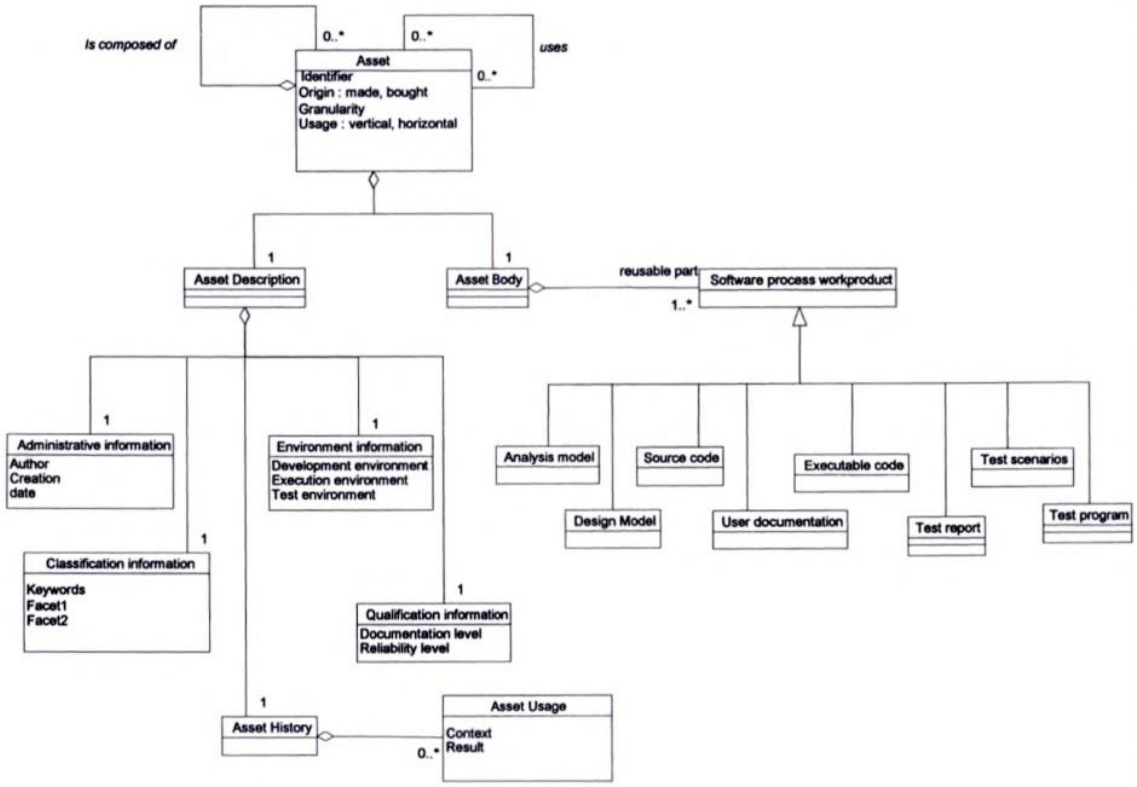


Figura 2.7: Asset Model (EZRAN, 2002).

2.7 Modelo do Reuse Repository

O modelo utilizado no desenvolvimento do repositório de reutilização denominado Reuse Repository (BURÉGIO, 2006) é ilustrado na Figura 2.8. Esse modelo referencia o RAS (seção 2.9), utilizando-o apenas para definir as classes de ativo (Asset) e de artefatos (Artifact). Esse modelo traz como classes: tipo de ativo (AssetType); catálogo (Catalog); um sistema de usuários com grupos (User e OrganizationalGroup), permissões (PermissionType) e papéis (Role) com suas funcionalidades (Functionality); versões relacionadas do ativo (AssetVersion e RelatedVersion); classificação por categorias (Classifier e Category); separação das referências dos artefatos em locais (LocalArtifact) e remotas (Remote Artifact); feedback de usuários (Feedback); certificação do ativo (Certification); métricas de custo, tempo e linhas de código (Metrics); e relatos de defeitos por usuários (Bugs).

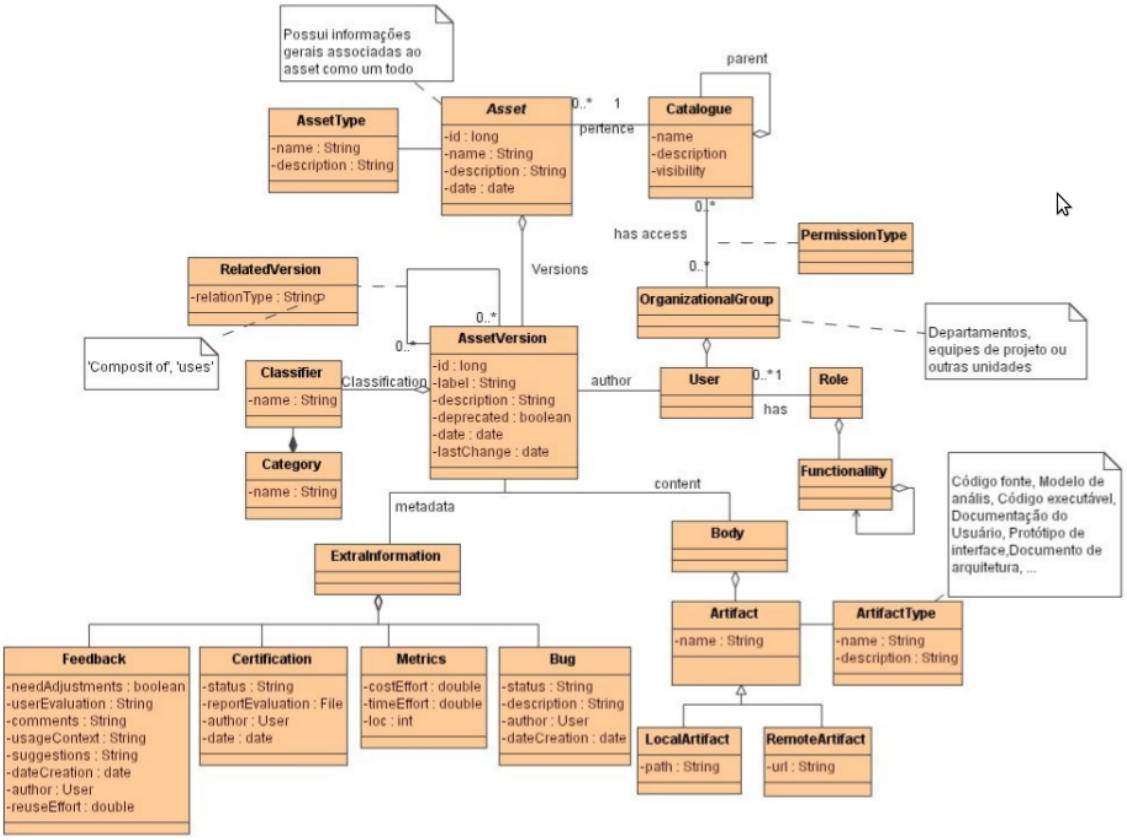


Figura 2.8: Modelo de ativos do Reuse Repository (BURÉGIO, 2006).

2.8 Software Packaging Domain Model

Com o objetivo de identificar, descrever e prescrever os atributos para o empacotamento de ativos reutilizáveis de software, o Software Reuse Working Group (WG) da NASA Earth Science Data Systems (ESDS) desenvolveu o Software Packaging Domain Model (NASA, 2011), ilustrado na Figura 2.9. Esse modelo de ativos é simplificado, possuindo a classe central do ativo (Software Asset) relacionada às de: versionamento (Versions), ferramentas de construção (Build Tools), dependências de outros softwares para execução ou configuração (Dependencies), métodos de empacotamento (Packaging Methods), licenças de software (License), distribuidor do software (Distributors), necessidades de manutenção (Maintenance Needs) e ciclo de vida (Lifecycle).

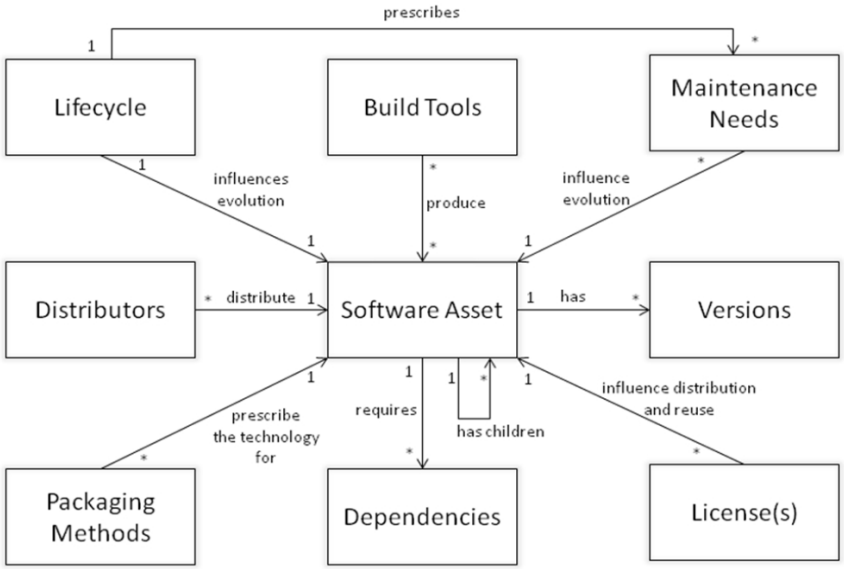


Figura 2.9: Software Packaging Domain Model (NASA, 2011).

2.9 Reusable Asset Specification

O escopo do Reusable Asset Specification (RAS) é prover um conjunto de diretrizes e recomendações sobre a estrutura, o conteúdo e as descrições de ativos reutilizáveis de software (OMG, 2005). O RAS foi desenvolvido pelo Object Management Group (OMG) em parceria com grandes empresas de software, tais como IBM e Blueprint Technologies. Foca-se na padronização consistente do empacotamento dos ativos, com o objetivo de reduzir o conflito das transações na reutilização de software. Desenvolvido em diagramas UML, esse modelo identifica categorias para os ativos, com tipos e Profiles com suas as diretrizes.

Essa especificação é descrita em duas categorias principais: o Core RAS e seus Profiles. O Core RAS representa os elementos fundamentais, e os Profiles descrevem extensões para esses elementos essenciais, nunca alterando a definição ou a semântica do Core RAS. O Core RAS não pode ser instanciado, e sim um Profile em particular. Um Profile pode estender o Core RAS diretamente ou então estender um outro Profile (Figura 2.10). A realização do Core RAS (versão 2.1) é o Default Profile (versão 2.2), o qual é estendido pelo Default Component Profile (versão 2.2) e pelo Default Web Service Profile (versão 2.2), ilustrada na Figura 2.11. Desse modo, o Default Profile estrutura um ativo em geral com suas características básicas, o Default Component Profile estrutura ativos de componentes e o Default Web Service Profile estrutura ativos de web services.

As principais partes do Core RAS são ilustradas na Figura 2.12. O ativo (Asset) define a descrição, o estado, a versão e o Profile utilizado, junto das categorias que representam sua classificação (Classification), sua solução com artefatos (Solution), seu uso com instruções e atividades (Usage), e seus ativos relacionados (Related Assets). A Figura 2.13 exibe o modelo de domínio do Core RAS com tais categorias. O RAS especifica para o Core RAS e seus Profiles os modelo dos ativos, detalhando suas classes, junto da obrigatoriedade de classes e atributos e das restrições semânticas que os diagramas não podem descrever.

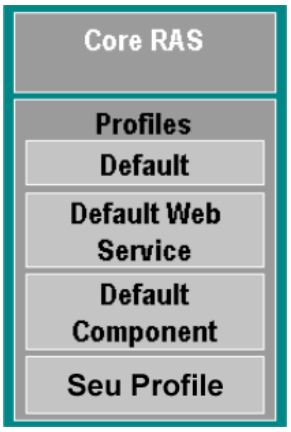


Figura 2.10: Core RAS e as extensões dos Profiles – Adaptada e traduzida de (OMG, 2005).

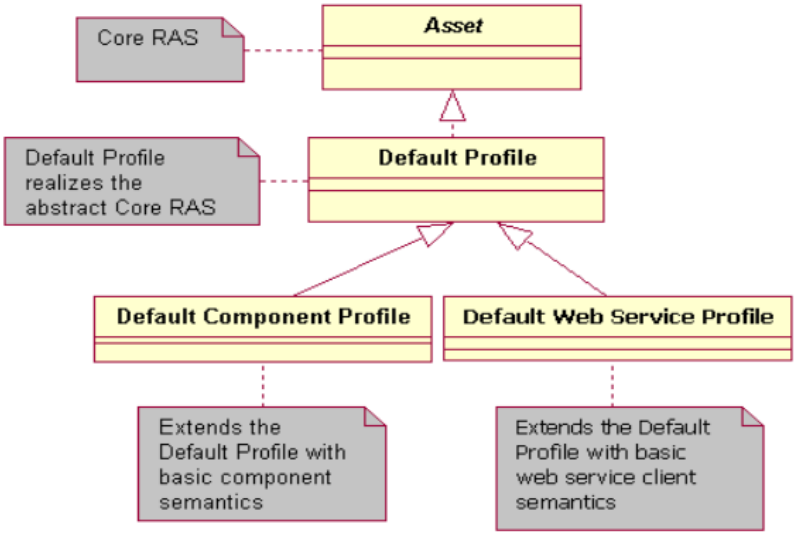


Figura 2.11: Extensões entre Profiles RAS (OMG, 2005).

Figura 2.12: Principais seções no Core RAS – Adaptada e traduzida de (OMG, 2005).

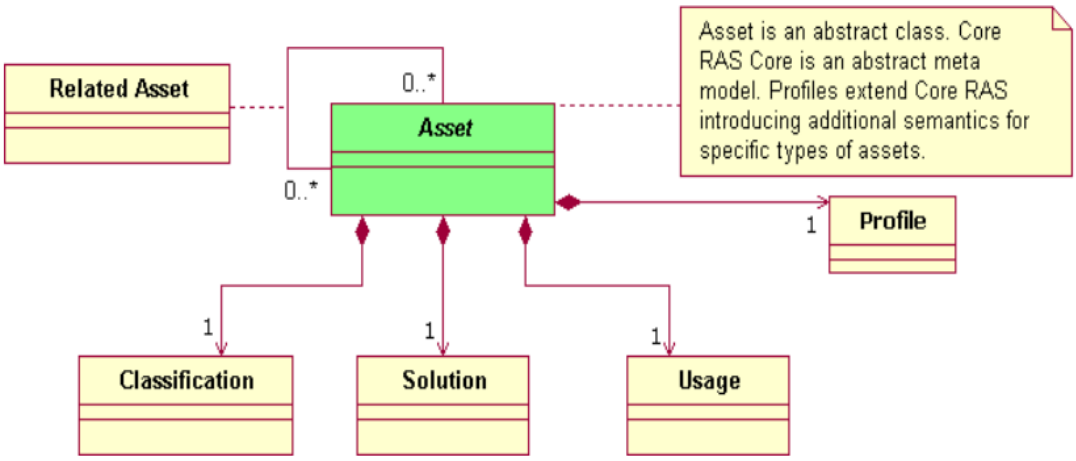


Figura 2.13: Principais seções do modelo de domínio do Core RAS (OMG, 2005).

2.9.1 Default Profile RAS

O Default Profile é a realização do Core RAS, contendo a mesma estrutura. A Figura 2.14 exibe as classes com seus atributos e relacionamentos do Default Profile (Core RAS). Os relacionamentos de agregação entre as classes declaram os elementos donos e os elementos contêiners. O ativo (Asset) possui atributos de nome (name), identificador (id), data (date), estado (state), versão (version), direitos de acesso (access-rights) e uma descrição curta (short-description). Relacionadas ao ativo estão as categorias da classificação (Classification), da solução (Solution), do uso (Usage), dos ativos relacionados (Related Asset) e do Profile utilizado (Profile). A categoria da classificação contém o contexto do ativo e os descritores. A categoria da solução possui os artefatos (Artifact) com tipos (ArtifactType), contextos (ArtifactContext), dependências (ArtifactDependency) e pontos de variabilidade (VariabilityPoint). O atributo Artifact.reference representa a localização do artefato através de uma URI (Universal Resource Identifier). A categoria de uso possui uma referência a um artefato, possuindo também contextos de uso e atividades a serem realizadas para o ativo e artefatos. A categoria de ativos relacionados possui os tipos de relacionamento e as referências a outros ativos relacionados. A categoria de Profile contém o Profile que o ativo utiliza para ser estruturado e seus os Profiles relacionados.

As classes obrigatórias desse Profile são a Asset e a Profile. A classe Asset possui os atributos obrigatórios name e id, e a Profile possui name, id-history, version-major e version-minor. São as principais restrições semânticas: o ativo deve possuir ao menos um artefato (com nome e referência); todo arquivo empacotado no ativo deve estar associado a um artefato ao menos; um Profile novo pode adicionar regras semânticas, mas nunca reduzi-las; atributos em classes já existentes podem ser adicionados em novos Profiles RAS; as restrições nos atributos existentes não podem ser reduzidas em novos Profiles RAS.

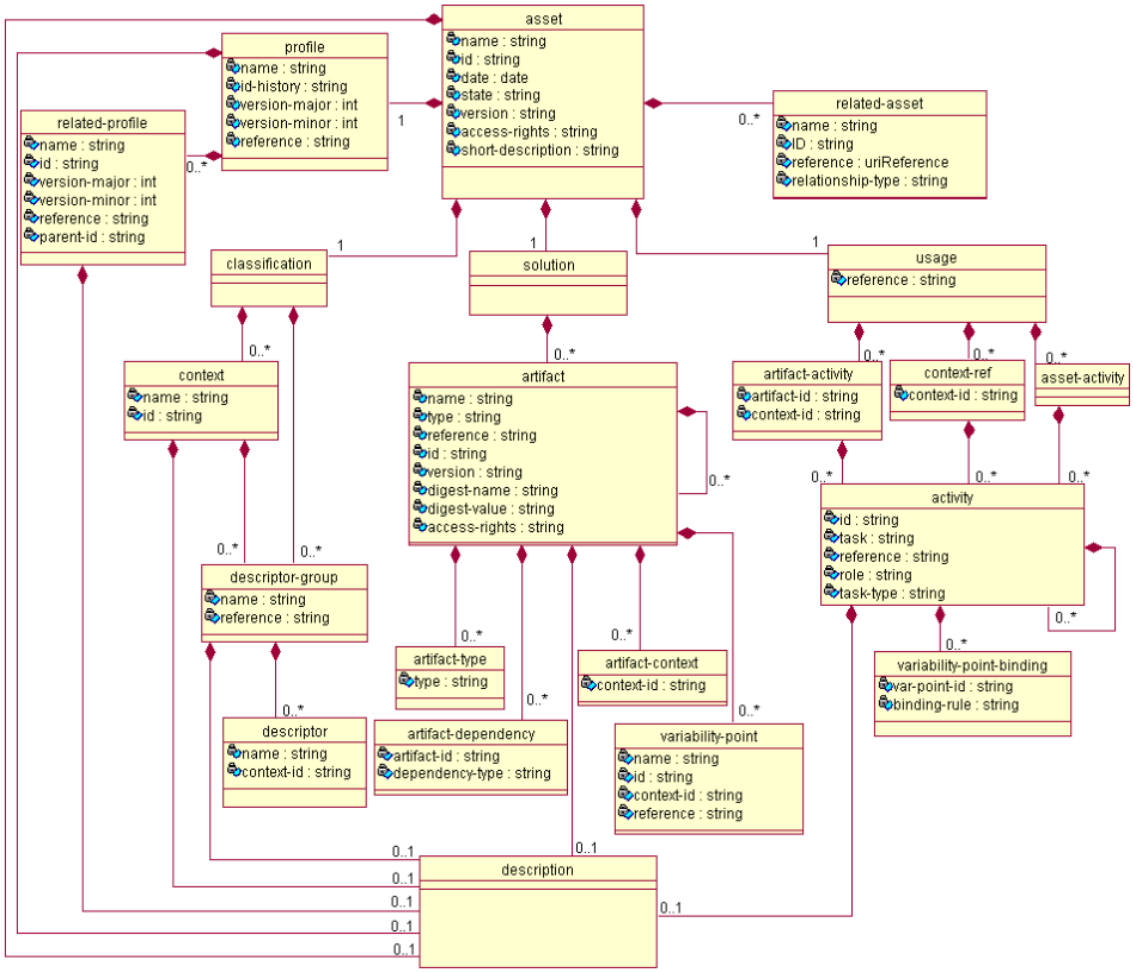


Figura 2.14: Modelo de ativos do Default Profile (Core RAS) (OMG, 2005).

2.9.2 Default Component Profile RAS

O Default Component Profile estende o Default Profile, herdando todas suas classes, atributos, restrições e regras semânticas do Core RAS. Esse Profile para componentes estende apenas a categoria de solução dos ativos, como se pode observar na Figura 2.15. Na solução são adicionadas as fases do ciclo de vida de software: a análise de requisitos (Requirements), o projeto (Design), a implementação (Implementation) e o teste (Test) dos ativos. Casos de uso (UseCase) são relacionados na análise de requisitos. Diagramas (Diagram), modelos (Model) e artefatos (Artifact) são associados nas fases de análise de requisitos, de projeto e de testes. Além disso, o projeto possui as especificações de interfaces (InterfaceSpec), com operações (Operation) possuindo condições (Condition) e parâmetros (Parameter), e informações sobre os modelos utilizados (InformationModel).

Além das restrições herdadas do Default Profile, as principais restrições semânticas adicionadas são: os artefatos de análise de requisitos, projeto, implementação e testes devem ser postos em seus locais devidos, ou seja, em Requirements, Design, Implementation e Test, respectivamente; e cada especificação de interface deve conter ao menos uma operação relacionada.

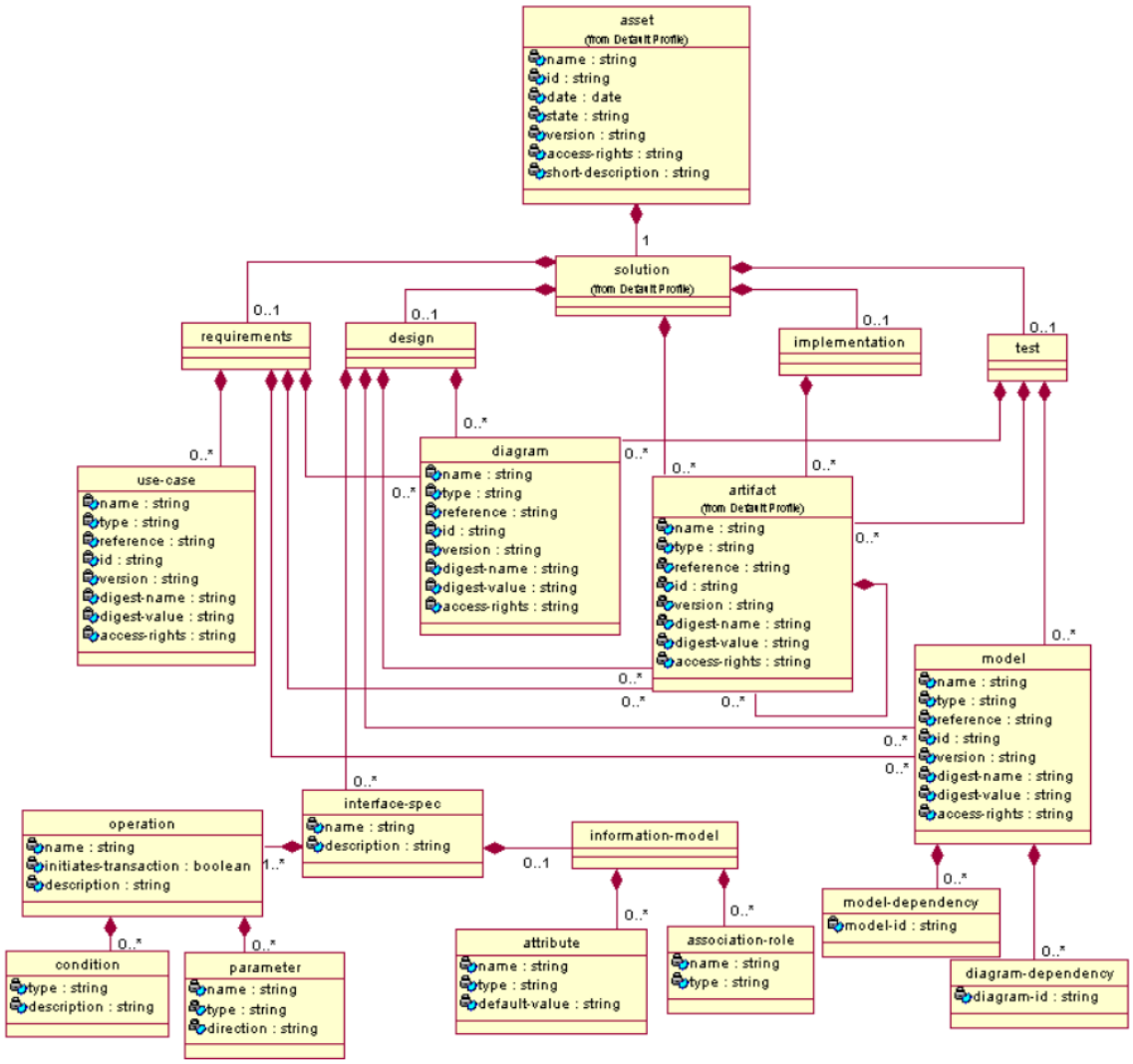


Figura 2.15: Default Component Profile e suas extensões na solução dos ativos (OMG, 2005).

2.9.3 Default Web Service Profile RAS

O Default Web Service Profile modela os ativos na forma de web services, derivando do Default Profile. Semelhantemente ao Profile de componentes, possui requisitos, casos de usos, projeto, especificação de interfaces, diagramas, modelos, implementação e teste (Figura 2.16). O que há de novidade é a classe WSDL para representar web services, a qual possui apenas o atributo reference (URI). As classes Implementation e WSDL são as obrigatórias. Além disso, a classe de especificação de interfaces (InterfaceSpec) possui o atributo de nome do WSDL (wsdl-name).

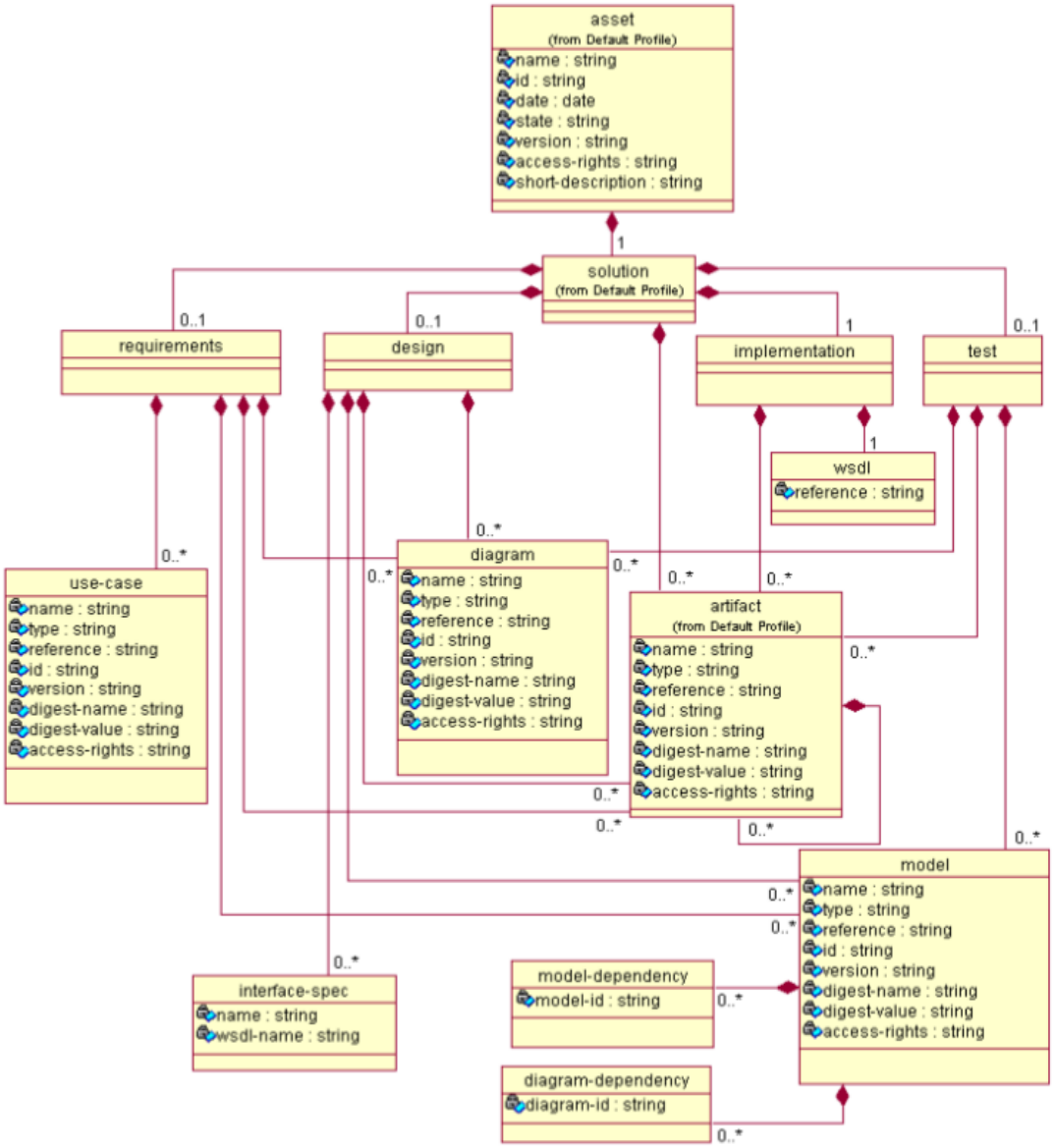


Figura 2.16: Default Web Service Profile e suas extensões na solução dos ativos (OMG, 2005).

2.10 Trabalhos Anteriores de Extensão do RAS

O RAS proporciona através de sua especificação que seus Profiles sejam estendidos. Por ser extensível, oportunidades de melhoria e solução de suas lacunas ficam em aberto, podendo-se condizer mais seu modelo com o reúso de software. Para isso, algumas extensões com objetivos específicos são propostas na literatura e são descritas a seguir.

2.10.1 Metamodel of Domain Architecture

Estendendo o Default Profile (Core RAS), o Metamodel of Domain Architecture (MOON, 2006) estabelece um esquema com o propósito de representar a arquitetura de domínio, como modelo de referência para um conjunto de famílias que compartilham funcionalidades similares, comportamentos e estruturas. Nessa extensão (Figura 2.17), a arquitetura de domínio (Domain Architecture) é associada à solução com: componentes

de domínio (Domain Component); interfaces das operações do componente de domínio (Interface e Operation); modelos do comportamento do domínio (Domain Behavior Model); especificação do componente de domínio (Domain Component Specification); modelos da estrutura de domínio (Domain Structure Model); e a variabilidade no componente de domínio, nas interfaces e operações (-vp:CV_property). Esse modelo não especifica os atributos de suas classes e não informa suas regras semânticas, nem a obrigatoriedade de suas classes e atributos.

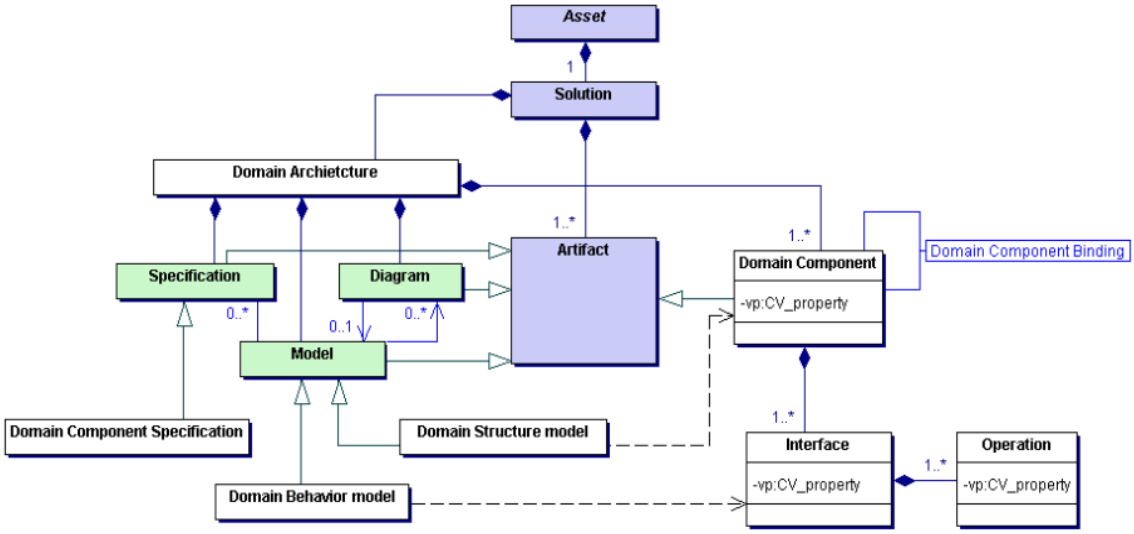


Figura 2.17: Extensão do RAS no Metamodel of Domain Architecture (MOON, 2006).

2.10.2 Multi Phased Component

O Multi Phased Component (MPC) (PARK, 2007) é uma extensão do Default Component Profile do RAS com o propósito de adicionar fases do ciclo de vida de software à solução do ativo (Figura 2.18). Traz em seu modelo a arquitetura (Architecture), a análise (Analysis), requisitos funcionais (Functional Requirement) e não-funcionais (Nonfunctional Requirement), atributos de qualidade (QualityAttribute) e parâmetros condicionais no ponto de variabilidade dos artefatos (VariabilityPoint). Não especifica a obrigatoriedade das novas classes e dos novos atributos, nem apresenta suas restrições semânticas. Além disso, não detalha quais são os atributos de qualidade, nem quais são as diferenças entre as classes de arquitetura (Architecture) e projeto (Design - já existente no RAS), nem entre as classes de análise (Analysis) e requisitos (Requirement - já existente no RAS).

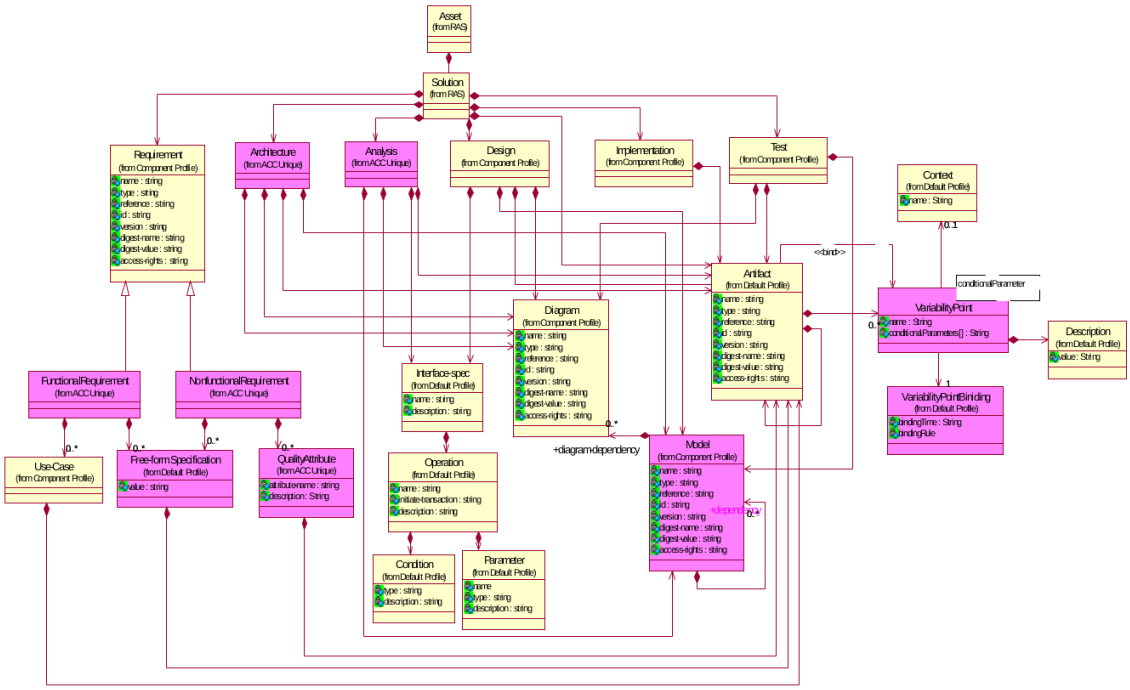


Figura 2.18: Extensão da categoria de solução do RAS do Multi Phased Component (PARK, 2007).

2.10.3 X-ARM Model Profile

O XML-based Asset Representation Model (X-ARM) (SCHUENCK, 2007) visa ao suporte de certificação de componentes reutilizáveis de software, estendendo o Default Profile RAS. O X-ARM possibilita que o modelo de certificação seja especificado com itens (CertificationItem) por suas chaves (atributo key), com a obrigatoriedade de preenchimento desses itens (atributo mandatory), junto de seus valores permitidos (AllowedValue) (Figura 2.19). Para descrever a certificação (CertificationInfo), diversas entidades (ProducerCertification) podem certificar o componente, e cada entidade pode utilizar um modelo próprio de certificação (Figura 2.20). Toda certificação (Certification) contém uma assinatura digital (Signature) para garantir a identidade do certificador.

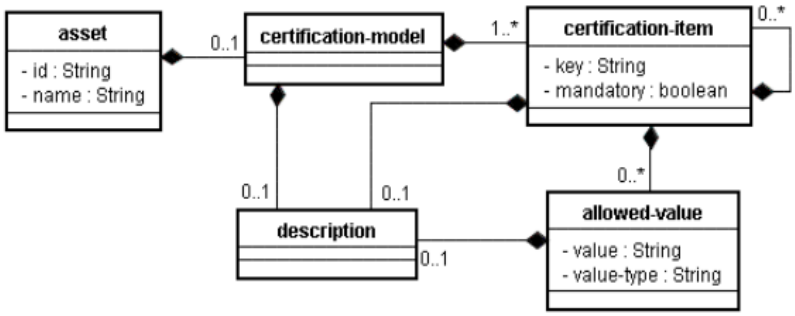


Figura 2.19: Especificação da extensão para modelos de certificação (SCHUENCK, 2007).

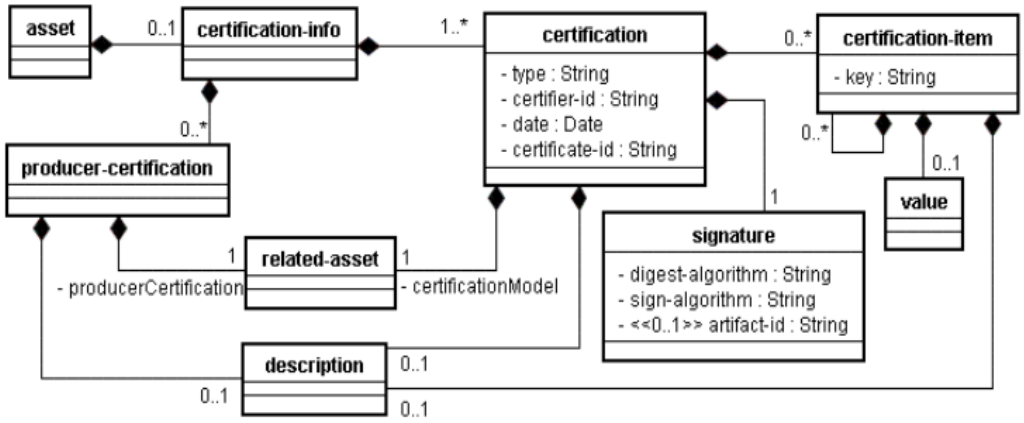


Figura 2.20: Especificação da extensão para descrições de certificação (SCHUENCK, 2007).

2.10.4 Extensão no Asset Management Tool

O Asset Management Tool (AMT) (WANG et al., 2008) cria seu Profile a partir do Default Component Profile para a solução dos ativos. A Figura 2.21 apresenta a extensão da categoria da solução, com as classes novas com um sinal de asterisco (*): classe de domínio (Domain), de arquitetura (Architecture), de análise (Analysis), de padrões (Pattern) e casos de teste (Test-Case). Sem um maior esclarecimento, liga as classes de domínio e de padrões diretamente à solução, em vez de associá-las às fases de requisitos ou de projeto. Não exibido graficamente no modelo, o AMT estende também a categoria de classificação com as classes de tipos funcionais (Functional Type), linguagem de programação (Programming Language), sistemas operacionais (Operational System), estilo de encapsulamento (Encapsulation Style), compatibilidade de contêiner (Container Compatibility) e internacionalização (Internationalization). Não há informações sobre a obrigatoriedade das classes, nem sobre quais atributos existem nas classes. Também não são informadas suas restrições semânticas.

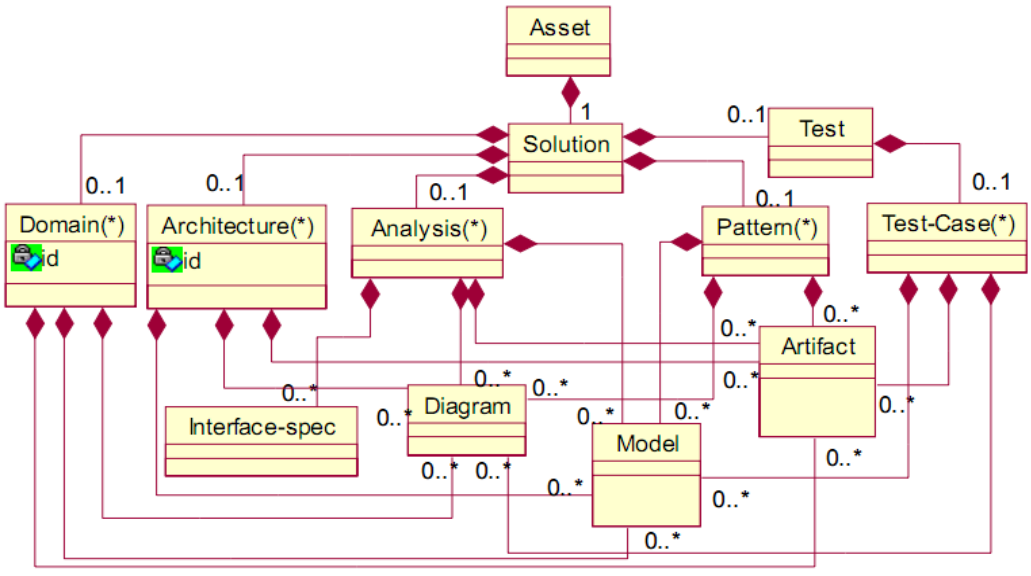


Figura 2.21: Extensão da categoria de solução do RAS do Asset Management Tool (WANG et al., 2008).

2.10.5 Representation Model for Reusable Assets to Support User Context

Com foco no contexto de usuário, o trabalho de Hadji (2008) propõe extensões no RAS para os graus de acessibilidade para utilização do ativo e a experiência em reúso de software do usuário (Figura 2.22). Além do foco no usuário, traz extensões para a classificação, requisitos não-funcionais, certificação e modelo econômico. O grau de acessibilidade preocupa-se com a usabilidade do ativo, com o nível de complexidade e destreza necessário do usuário consumidor para manipular tal ativo. A experiência em reúso provê informação sobre experiências prévias dos usuários em reúso de ativos, com conselhos para utilizar propriamente o ativo, ou seja, comentários de usuários sobre o uso. A classificação se dá pelos domínios de aplicação. A certificação identifica se o ativo é certificado ou não. O modelo econômico especifica as restrições da utilidade econômica dos ativos. Essa extensão não informa as classes e seus atributos, nem a obrigatoriedade deles, nem suas regras semânticas.

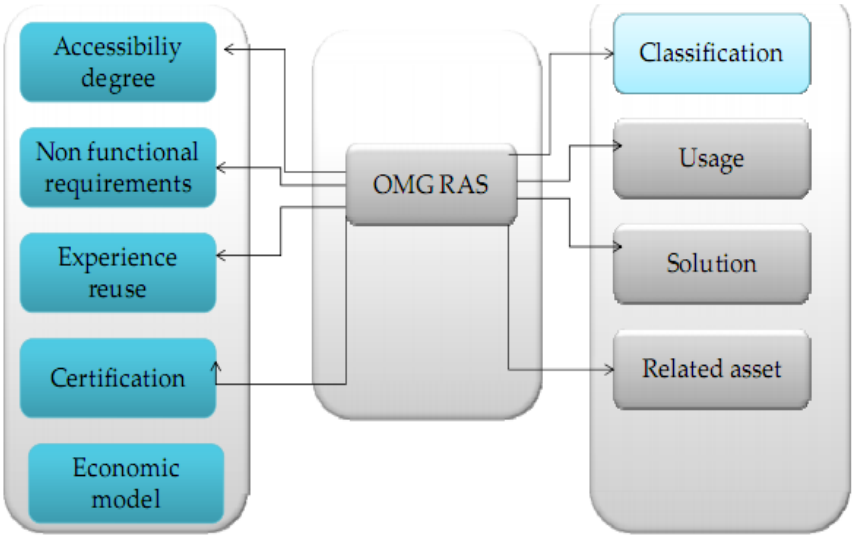


Figura 2.22: Extensão do RAS suportando contextos de usuários (HADJI, 2008).

2.10.6 Extensão no OpenCom

Com o objetivo de aprimorar a reutilização de software open source, o repositório OpenCom (HONG-MIN, 2009) cria uma extensão do RAS através do Default Component Profile (Figura 2.23). A extensão é dada nas categorias de classificação, de solução e de ativos relacionados. Na classificação, as licenças de software (License) são relacionadas. Na solução são adicionadas as classes de arquitetura (Architecture); modelo de dados (DataModel); ponto de variabilidade (VariabilityPoint); variação (Variant); e atributos de qualidade (QualityAttribute), exceções (Exception) e restrições (Constraint) relacionadas à especificação de interface (Interface). Na categoria de ativos relacionados é associada a classe de comunidade (Community).

Não há informações sobre quais são atributos nas classes e suas obrigatoriedades, nem sobre suas restrições semânticas. Não existe uma explicação do porquê que a classe de comunidade está nos ativos relacionados, nem uma razão da diferença semântica da classe de arquitetura proposta para a classe de projeto (Desing) já existente no RAS. Além disso, não é explicado o motivo dos atributos de qualidade estarem na especificação da interface, nem o motivo do modelo de dados estar ligado diretamente na solução, em vez de estar ligado às fases de projeto ou implementação do ativo

existentes no Default Component Profile. A classe Variant não tem uma explicação do que ela representa, nem a razão da classe VariabilityPoint ser uma extensão, visto que já faz parte do Default Profile.

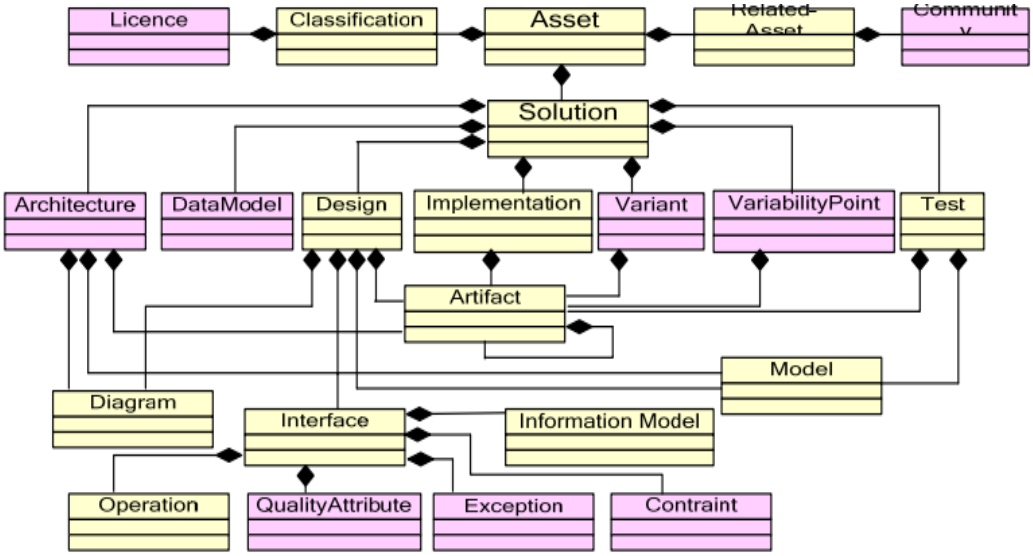


Figura 2.23: Extensão do RAS no repositório OpenCom (HONG-MIN, 2009).

2.10.7 Common Representation for Reuse Assistants

O modelo do Common Representation for Reuse Assistants (BASSO, 2013) define extensões para a execução e o deploy de artefatos através de tarefas de reúso automatizadas, especificadas em um modelo, denominadas de RA (Reuse Assistants). A Figura 2.24 ilustra a extensão para suporte de tarefas para execução de artefatos (ExecutableTask), guiadas pelo RA (ReuseAssistant) e pelo processo de reúso (ReuseProcess). Para os artefatos, define os gêneros (ArtifactKind) de API, documentação, caso de teste, requisitos, código-fonte, descritor de deploy e tutorial. A Figura 2.25 ilustra a extensão para suporte de tarefas de deploy (DeployInfo), com informações de download dos artefatos (DownloadInfo) e seus repositórios (Repository e Path).

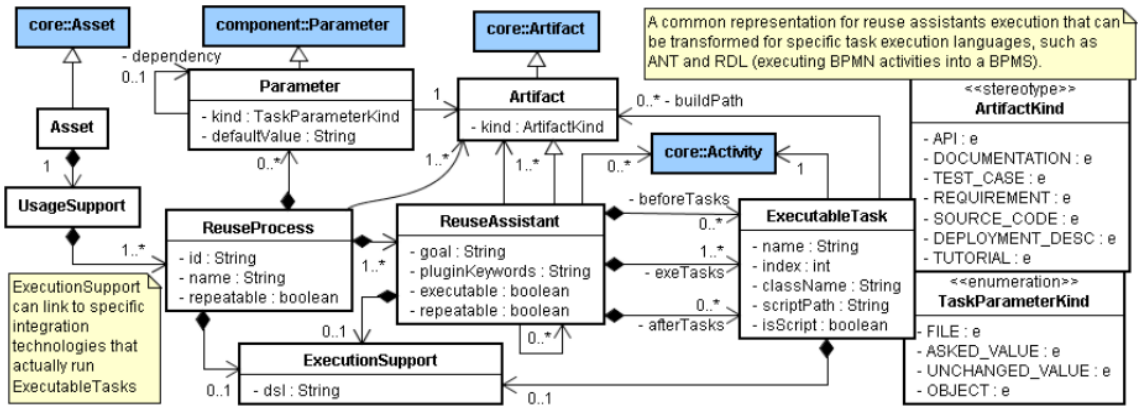


Figura 2.24: Extensão do RAS para suportar a execução de artefatos como tarefa de reúso (BASSO, 2013).

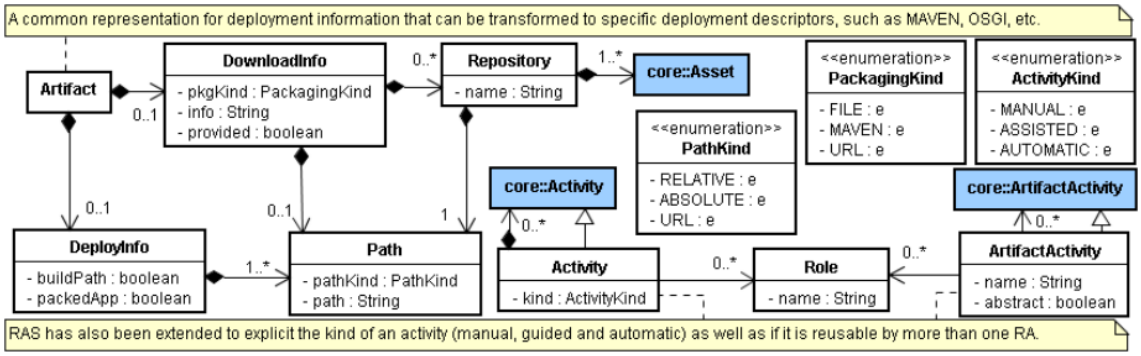


Figura 2.25: Extensão do RAS para suportar o deploy dos artefatos como tarefa de reúso (BASSO, 2013).

3 SOFTWARE PROFILE RAS

Esse capítulo lista as lacunas do Reusable Asset Specification e apresenta o trabalho proposto para solução de tais lacunas, o Software Profile RAS.

Mesmo que o Reusable Asset Specification seja uma padronização *de facto* como modelo de ativos reutilizáveis, ele ainda necessita ser complementado para se tornar adotado na prática. Algumas de suas lacunas são relacionadas ao conteúdo do ativo que fica apenas na ferramenta do repositório e é perdido, ou nem sequer é descrito, não sendo empacotado no ativo. Outros modelos de ativos abordam itens relevantes que o RAS não possui, também considerados como lacunas, mesmo que nem toda lacuna seja uma limitação e sim uma oportunidade para um determinado contexto. Além disso, os trabalhos de extensões do RAS reconhecem lacunas existentes em focos específicos. Outras lacunas são relacionadas às informações e aos artefatos relacionados ao ciclo de vida do software. Tais lacunas relevantes do RAS são numeradas a seguir:

1. Qualidade do ativo. A gerência da qualidade dos ativos é uma das principais carências do RAS (VOTH, 2004), não proporcionando que atributos de qualidade do ativo sejam avaliados. No repositório, isso faz com que os usuários não possam obter uma classificação pela qualidade dos ativos, não podendo escolher os ativos de maior confiança ou ter mais atenção em ativos de baixa qualidade.

2. Tipo do ativo. Não existe a informação sobre os tipos dos ativos, por isso o RAS propõe um Profile para cada tipo de ativo. O tipo do ativo é o primeiro nível de organização em um repositório (IBM, 2008), então ter um Profile para cada tipo de ativo pode tornar a estrutura do repositório complexa. Visto que cada tipo de ativo pode possuir informações e artefatos específicos, isso faz com que os Profiles sejam muito prescritivos para cada tipo de ativo, engessando o uso novas formas de ativos pelo fato de ter que gerar novos Profiles. Esse problema já ocorre nos Profiles do RAS de componentes e de web services que estendem o Default Profile. Esses dois Profiles têm várias características em comum em suas extensões, contendo a maioria das suas classes iguais, porém um difere do outro em pouquíssimos itens (vide seção 2.9). As regras semânticas contempladas pelo RAS podem abranger as regras dos tipos de ativos.

3. Reusabilidade do ativo. A reusabilidade, ou seja, a quantia de consumos do ativo, não é contabilizada para influenciar na relevância da classificação do ativo. Assim como a qualidade do ativo, a reusabilidade auxilia os consumidores a escolherem os ativos pela efetividade da quantia de consumos já realizadas por outros usuários (WASHIZAKI, 2003).

4. Domínio de aplicação do ativo. A ausência da classificação por domínios de aplicação, não permitindo que os ativos sejam descritos de acordo sua linha de negócio (FRAKES, 2005).

5. Organizações e projetos do ativo. A indefinição da classificação dos ativos por projetos e organizações, não estruturando o desenvolvimento distribuído entre instituições (FRAKES, 2005), bem como o controle de permissões nos ativos.

6. Licença de software do ativo. Não há a informação sobre tipos de licença de software nos ativos e artefatos, a qual permite observar as questões legais e a compatibilidade entre as licenças. De fato, não proteger legalmente os direitos de quem produz e de quem reutiliza é um problema que obstrui o reúso de software (FRAKES, 1990).

7. Custos e esforços de desenvolvimento do ativo. As informações de custos e esforços de desenvolvimento do ativo para sua a gerência não existem no RAS. Desse modo, o Retorno no Investimento (ROI) de cada ativo não pode ser avaliado, gerenciando custos e esforços tais como o valor monetário, ou a relação entre o tempo estimado e o tempo real de desenvolvimento (RAMACHANDRAN, 2005).

8. Classificação colaborativa do ativo. A classificação do RAS é estática e não permite que os usuários possam colaborar com a classificação do ativo com o livre preenchimento de tags (ou palavras-chaves), assim possibilitando emergir uma taxonomia mais dinâmica e mais consistente com o vocabulário e o conhecimento dos usuários através da Folksonomia (WAL, 2007).

9. Requisitos funcionais e não-funcionais no ativo. O RAS não distingue os requisitos funcionais dos requisitos não-funcionais, podendo causar uma perda na especificação, na busca e no entendimento dos ativos de software (SOMMERVILLE, 2011).

10. Padrões de projeto no ativo. A especificação de padrões de projeto está ausente na solução do RAS. Os usuários poderiam entender melhor a estrutura projetada do problema solucionado com os padrões de projetos especificados (PRESSMAN, 2011).

11. Interfaces de usuários no ativo. Falta uma forma especializada para reutilizar interfaces de usuários, um aspecto muitas vezes desprezado, mas que afeta o tempo e esforço do desenvolvimento (SUTCLIFFE, 1999).

12. Serviços no ativo. Os serviços (web services) poderiam ser melhor documentados pois são ativos potencialmente reutilizáveis, porém o Profile de serviços contém apenas uma referência para a descrição do serviço, além de conter divergências relacionadas ao Profile de componentes (vide subseção 2.9.3).

13. Códigos-fontes e linguagens de programação no ativo. A solução dos ativos no RAS não especializa os artefatos para códigos-fontes e as linguagens de programação utilizadas. Códigos-fontes são altamente reutilizáveis, e a linguagem de programação influencia na construção de um módulo reutilizável de software (AGRESTI, 2011), influenciando na reusabilidade dos ativos.

14. Testes do ativo. O RAS também não distingue os artefatos de testes e seus tipos, assim como os métodos de testes aplicados na verificação e validação dos ativos (vide subseção 2.9.2).

15. Perfis de usuários do ativo. A indefinição de usuários e perfis de usuários para o uso dos ativos, perfis tais como de consumidores, produtores e certificadores, dificultando a rastreabilidade das ações realizadas e dos usuários relacionados aos ativos. Desse modo, a comunicação entre os usuários com seus pares (SOFTEX, 2011a) é dificultada, assim como as notificações automáticas pela ferramenta de repositório e o controle das permissões de acesso em níveis de visualização, consumo e edição para cada usuário.

16. Certificação do ativo. O RAS não provê suporte à certificação dos ativos. Assim, não tem como saber por quem e quando foi certificado o ativo, dificultando a comunicação entre os certificadores e os produtores dos ativos (SCHUENCK, 2007).

17. Guias de usuários no ativo. Falta uma documentação específica para a utilização do ativo, na forma de guias de usuários, sendo considerados potenciais para o reúso (AGRESTI, 2011).

18. Comentários de usuários no ativo. Os comentários de usuários não são previstos, não documentando no ativo o histórico dos relatos de defeitos, dúvidas, sugestões ou observações dos usuários consumidores, produtores ou certificadores.

19. Relacionamento de ativos por versão anterior. Não há o tipo de relacionamento de versão anterior, impedindo a rastreabilidade de versões, considerada um requisito importante para um repositório de reúso eficiente (ALMEIDA et al., 2007). Além do entendimento sobre a evolução dos ativos, a rastreabilidade de versões permite que novas versões dos ativos sejam notificadas para os consumidores das versões anteriores.

20. Arquitetura de domínio do ativo. A arquitetura de domínio que permite identificar a similaridade entre sistemas de software para o reúso (SHIVA, 2007), não sendo abordada no RAS.

21. Experiência em reúso de software do usuário e nível de acessibilidade do ativo. Com foco no usuário consumidor, a experiência dele em reutilização de software não consta no RAS, bem como não abrange o nível de acessibilidade de uso ativo (HADJI, 2008), ou seja, a experiência necessária para reutilizá-lo em conformidade.

22. Repositório onde se encontra o ativo. Relacionado à interoperabilidade entre repositórios (HOBBS, 1993), o RAS não especifica o repositório onde se encontra o ativo. Descrever essa localização permite que os ativos sejam classificados por repositórios (bibliotecas, ou catálogos). Isso permite também que cada instituição mantenha o seu próprio repositório para o desenvolvimento distribuído.

23. Automatização da execução do ativo e do deploy de seus artefatos. A falta de descrição para automatização da execução do ativo e do deploy de seus artefatos (BASSO, 2013), repassando para ferramentas essas tarefas que seriam realizadas

manualmente.

O Software Profile RAS (SW-RAS) estende o Profile de componentes do Reusable Asset Specification (o Default Component Profile), adicionando informações e artefatos necessários para solucionar várias de suas lacunas. Seu Profile é compatível com o RAS, visando a ser enxuto (simples) e suficiente.

O SW-RAS propõe diversas extensões para as categorias da classificação (seção 3.1), da solução para o problema que o ativo resolve (seção 3.2), da forma de uso e reúso (seção 3.3) e dos ativos relacionados (seção 3.4). A seção 3.5 lista a obrigatoriedade das classes e atributos. A seção 3.6 descreve as restrições semânticas. A seção 3.7 exibe a relação entre as extensões propostas e as lacunas do RAS. Cada Profile é representado nos diagramas pelas cores: amarela para o Default Profile, azul para o Default Component Profile e verde para o Software Profile. Em cada descrição das extensões propostas, são apresentadas a intenção, a motivação e a aplicabilidade, a estrutura proposta e exemplos.

3.1 Extensões na Classificação dos Ativos

As extensões na classificação do SW-RAS são ilustradas em sua forma reduzida na Figura 3.1 e serão descritas nas próximas subseções. As classes em amarelo pertencem ao Default Profile e são estendidas pelas classes do Software Profile RAS (em verde). Não há classes do Default Component Profile na classificação dos ativos, por essa razão ele não é exibido nessas extensões. Os diagramas mostram apenas as classes e os relacionamentos associados ao SW-RAS. São as extensões na classificação com suas classes no modelo:

- Classificação pelo Tipo
  - Classe **AssetType**
- Classificação pela Avaliação da Qualidade
  - Classe **Ranking**
- Classificação pela Reusabilidade
  - Classe **Ranking**
- Classificação por Domínios de Aplicação
  - Classes **ApplicationDomain** e **ApplicationSubdomain**
- Classificação por Organizações e Projetos
  - Classes **Organization** e **Project**
- Classificação por Licenças de Software
  - Classe **SoftwareLicenseType**
- Classificação por Custos e Esforços
  - Classes **Effort** e **EffortType**

- Classificação por Social Tagging
  - Classe **Tag**

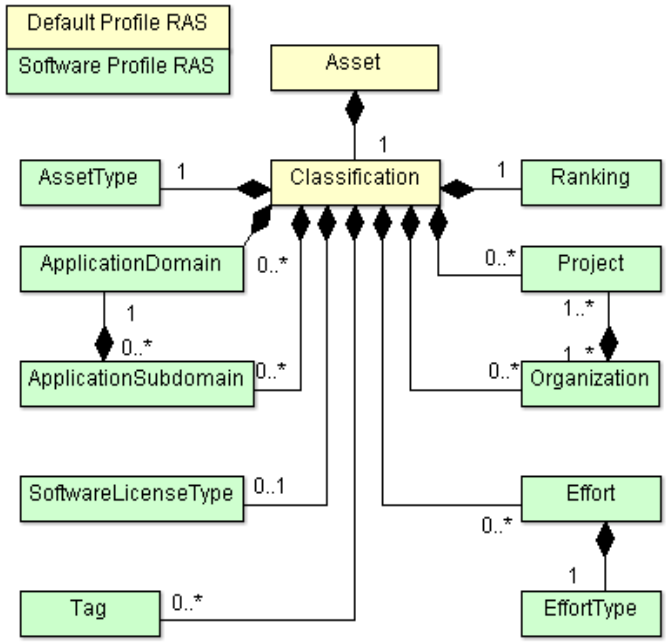


Figura 3.1: Representação reduzida das extensões na classificação dos ativos no SW-RAS.

3.1.1 Classificação pelo Tipo

3.1.1.1 Intenção

Classificar os ativos por seus tipos.

3.1.1.2 Motivação e Aplicabilidade

O tipo de ativo é o primeiro nível de organização para ativos dentro de um repositório de ativos reutilizáveis (IBM, 2008). Os tipos contêm informações que auxiliam o usuário a armazenar e encontrar ativos, facilitando sua administração e o controle das informações e dos artefatos que devem estar empacotadas no ativo.

Para o RAS, um ativo é definido por um Profile, e esse Profile descreve apenas um tipo de ativo (RAS, 2005). Cada Profile pode conter diferentes versões, hierarquias e Profiles relacionados, o que pode dificultar entendimento das características dos tipos. A informação do tipo permite abranger diversos tipos de ativos em apenas um Profile, simplificando o entendimento dos tipos dos ativos.

3.1.1.3 Estrutura Proposta

O tipo do ativo é associado na classificação pela classe AssetType (Figura 3.2), com o relacionamento obrigatório. O atributo AssetType.name armazena os tipos de ativos que podem ser associados.

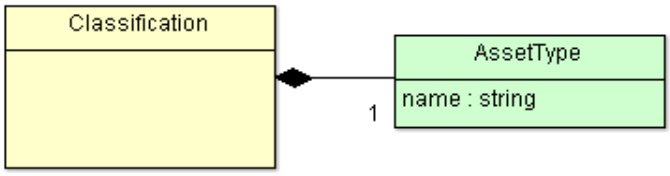


Figura 3.2: Extensão da classificação pelo tipo de ativo.

3.1.1.4 Exemplos

a) Tipos de ativos. Os ativos podem ser de diversos tipos, tais como componentes, serviços, frameworks, padrões de projeto (GAMMA et al., 1995), padrões de workflow (THOM, 2007), padrões arquiteturais, padrões de interação com o usuário e padrões de casos de uso (MOREIRA, 2007), entre outros ativos com artefatos reutilizáveis na construção de software.

b) Restrições para os tipos. Algumas regras podem ser descritas para cada tipo. Por exemplo, para um ativo com o tipo de serviço, deve haver ao menos um artefato de WSDL, o qual descreve a funcionalidade de um web service. Outro exemplo é para um ativo do tipo de interface de usuário, devendo conter um artefato de interface de usuário.

3.1.2 Classificação pela Avaliação da Qualidade

3.1.2.1 Intenção

Classificar o ativo pela sua qualidade através da média das avaliações realizadas pelos usuários, auxiliando no ranking dos ativos em repositórios e na decisão dos usuários em consumir os ativos.

3.1.2.2 Motivação e Aplicabilidade

O SW-RAS utiliza a ISO/IEC 25010 (ISO/IEC, 2011) para a avaliação da qualidade dos ativos, permitindo que os ativos sejam avaliados tanto interna quanto externamente em uma completude padronizada de características. A ISO/IEC 25010 é uma norma ISO para qualidade de produto de software, sendo uma evolução da norma ISO/IEC 9126 (ISO/IEC, 2001). Divide as características de qualidade em duas partes: qualidade no uso e qualidade do produto. Diversas subcaracterísticas são descritas para auxiliar na avaliação e no entendimento da característica em questão.

A qualidade no uso é o nível atribuído pelos usuários do produto em seus contextos para avaliar as propriedades externas do produto (Figura 3.3 – em inglês por não haver ainda uma tradução oficial da sua especificação). Essas propriedades são categorizadas em cinco características:

- Efetividade (*Effectiveness*)
- Eficiência (*Efficiency*)
- Satisfação (*Satisfaction*)
- Isenção de riscos (*Freedom from risk*)
- Cobertura do contexto (*Context coverage*)

Já a qualidade do produto categoriza as propriedades internas do produto (Figura 3.4

– em inglês por não haver ainda uma tradução oficial da sua especificação), as quais são divididas em oito características:

- Adequação funcional (*Functional suitability*)
- Confiabilidade (*Reliability*)
- Eficiência de desempenho (*Performance efficiency*)
- Usabilidade (*Usability*)
- Segurança (*Security*)
- Compatibilidade (*Compatibility*)
- Manutenibilidade (*Maintainability*)
- Portabilidade (*Portability*)

Com base nessa norma, os papéis do processo de reúso (consumidores e certificadores dos ativos – ver subseção 3.2.1) podem avaliar a qualidade do ativo em questão. O consumidor do ativo avalia as características da qualidade no uso (externas) e o certificador do ativo avalia as características da qualidade do produto (internas). A média de todas avaliações do ativo auxilia na sua classificação (ranking) no repositório, podendo ser consultada e ser utilizada como métrica para gerenciar a qualidade do repositório de ativos reutilizáveis.

Effectiveness
Efficiency
Satisfaction
Usefulness
Trust
Pleasure
Comfort
Freedom from risk
Economic risk mitigation
Health and safety risk mitigation
Environmental risk mitigation
Context coverage
Context completeness
Flexibility

Figura 3.3: Qualidade no uso da ISO/IEC 25010:2011 – suas cinco características e subcaracterísticas (ISO/IEC, 2011).

(Sub)Characteristic
Functional suitability
Functional completeness
Functional correctness
Functional appropriateness
Performance efficiency
Time behaviour
Resource utilization
Capacity
Compatibility
Co-existence
Interoperability
Usability
Appropriateness recognizability
Learnability
Operability
User error protection
User interface aesthetics
Accessibility

Reliability
Maturity
Availability
Fault tolerance
Recoverability
Security
Confidentiality
Integrity
Non-repudiation
Accountability
Authenticity
Maintainability
Modularity
Reusability
Analysability
Modifiability
Testability
Portability
Adaptability
Installability
Replaceability

Figura 3.4: Qualidade do produto da ISO/IEC 25010:2011 – suas oito características e subcaracterísticas (ISO/IEC, 2011).

3.1.2.3 Estrutura Proposta

As avaliações de qualidade pelos usuários são realizadas por instâncias da classe QualityEvaluation (Figura 3.5). Essa classe é estendida pela SoftwareProductQuality e pela QualityInUse, representando a avaliação da qualidade do produto e da qualidade no uso, respectivamente. Essas duas classes são associadas ao uso dos ativos (classe Usage), que é discutido mais adiante na seção 3.3.

O atributo QualityEvaluation.generalScore permite que o usuário também avalie uma nota geral para o ativo, além das características de qualidade. Os usuários podem também comentar textualmente na avaliação do ativo, pelo atributo QualityEvaluation.comment. A média de todas as avaliações é armazenada no atributo derivado Ranking.averageScore. O atributo QualityEvaluation.date representa a data em que foi realizada a avaliação. O atributo derivado Ranking.reuseCounter é utilizado na classificação pela Reusabilidade (seção 3.1.3).

O intervalo de valores para as avaliações de cada característica é sugerido que seja entre zero e cinco. Zero representa a baixa qualidade, e cinco a qualidade total.

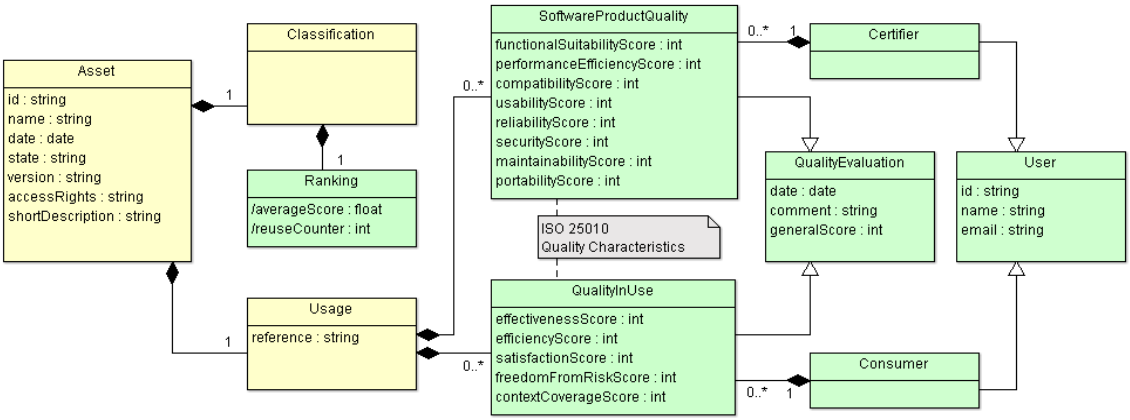


Figura 3.5: Extensão da classificação para a qualidade através de avaliações de usuários consumidores e certificadores, baseadas na norma de qualidade da ISO/IEC 25010:2011.

3.1.2.4 Exemplos

a) Ativo com alta satisfação (*satisfationScore*). Um ativo de um framework bem avaliado em sua satisfação por diversos usuários é um indicativo que é bem-aceito e bastante útil para seus consumidores.

b) Ativo com baixo desempenho (*performanceEfficiencyScore*). Um ativo de serviço avaliado pelo certificador com a eficiência do desempenho baixa é um indicativo que o serviço não deve ser utilizado em aplicações com muitas requisições.

c) Ativo com alto risco (*freedomFromRiskScore*). Um componente que tenha uma má avaliação na isenção de riscos é um indicativo que necessita de mais testes de integração com as aplicações, ou até mesmo a sua reestruturação.

d) Ranking no repositório. Os ativos melhores avaliados ficarão no topo dos resultados das buscas no repositório, utilizando o atributo Classification.averageScore em ordem descendente. Isso permite que os usuários encontrem primeiramente os ativos com maior qualidade, e os ativos com as piores avaliações estarão no final dos resultados, com baixa relevância.

3.1.3 Classificação pela Reusabilidade

3.1.3.1 Intenção

Classificar o ativo pela reusabilidade do ativo, ou seja, pela quantidade de reutilizações realizadas pelos usuários consumidores, auxiliando no ranking dos ativos em seus repositórios e no conhecimento da utilidade de cada ativo.

3.1.3.2 Motivação e Aplicabilidade

Washizaki (2003) aponta que é necessário medir a reusabilidade de componentes para reconhecer a efetividade do reúso de cada componente. Essa efetividade auxilia na análise de quais ativos são muito ou pouco reutilizados, necessária para a manutenção periódica do repositório de ativos reutilizáveis. Além disso, os usuários poderão decidir sobre o consumo do ativo através da reusabilidade do ativo.

A reusabilidade dos ativos pode ser um fator para continuar ou descontinuar um

ativo. O processo de Gerência de Reutilização do MPS-BR (SOFTEX, 2011a) tem como propósito gerenciar o ciclo de vida dos ativos de uma organização, e um de seus resultados esperados está em documentar uma estratégia de gerenciamento de ativos contemplando critérios para a descontinuidade desses ativos. Esses critérios estabelecem os ativos não mais utilizados para que sejam descontinuados ou removidos do repositório, podendo essa análise ser automatizada pelo sistema do repositório.

3.1.3.3 Estrutura Proposta

A quantidade de reutilizações é armazenada no atributo derivado Ranking.reuseCounter (Figura 3.6). Cada reutilização (ou consumo) é armazenada na classe Consumption, a qual contém a data (atributo date) e o relacionamento com seu usuário consumidor (Consumer).

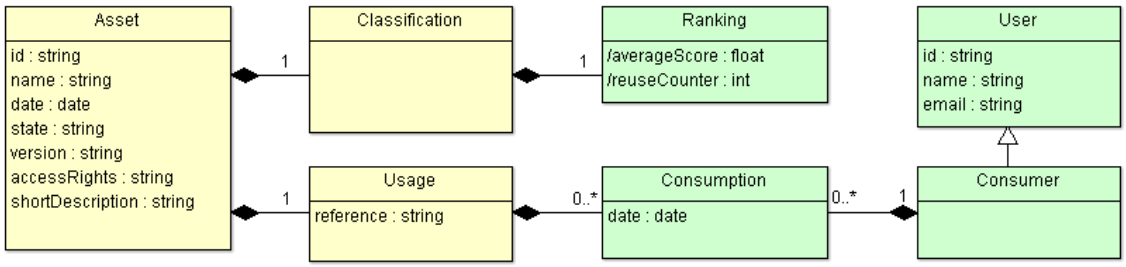


Figura 3.6: Extensão da classificação pela quantidade de reutilizações do ativo.

3.1.3.4 Exemplos

a) Ranking de reusabilidade dos ativos. Para obter um ranking dos ativos mais reutilizados, basta ordenar por ordem decrescente pelo atributo da reusabilidade dos ativos.

b) Descontinuidade de um ativo. Um relatório pode ser obtido periodicamente no repositório de ativos reutilizáveis para descrever os ativos que não foram reutilizados nos últimos meses, e, por essa razão, descontinuá-los, ou até removê-los do repositório.

c) Métrica de reusabilidade do repositório de ativos reutilizáveis. Essa métrica pode ser obtida pela média e desvio padrão do consumo dos ativos, consultando-se a efetividade do repositório para o reúso de software na organização.

3.1.4 Classificação por Domínios de Aplicação

3.1.4.1 Intenção

Classificar os ativos pelos seus domínios de aplicação e subdomínios, ativos os quais foram desenvolvidos para determinadas famílias de aplicações.

3.1.4.2 Motivação e Aplicabilidade

O domínio de aplicação é um fator chave para o reúso de acordo com Sommerville (2011), ele cita que diversas técnicas desenvolvidas para suportar o reúso de software exploram a similaridade de aplicações, tendo um potencial para reúso, e que, se o desenvolvimento de software for em um domínio, isso deve sempre ser considerado como uma opção.

Na reutilização de software, existem dois processos básicos: o Desenvolvimento com Reutilização e o Desenvolvimento Para Reutilização (SOFTEX, 2011b). O primeiro processo utiliza os ativos desenvolvidos pelo segundo processo para construir novas aplicações de software. O segundo processo, o Desenvolvimento Para Reutilização, utiliza as técnicas de Engenharia de Domínio para desenvolver ativos de domínio, denominados assim por serem desenvolvidos com foco em domínios e subdomínios específicos. A Engenharia de Domínio (SHIVA, 2007) captura as semelhanças e variabilidades em um conjunto de sistemas de software e as usa para construir ativos reutilizáveis. Sistemas de software no mesmo domínio têm padrões arquiteturais similares que podem ser descritos com uma arquitetura genérica (arquitetura de domínio). A arquitetura de domínio é então refinada para ajustar uma aplicação individual no domínio criando a arquitetura da aplicação.

3.1.4.3 Estrutura Proposta

As classes de domínio de aplicação (ApplicationDomain) e de subdomínio de aplicação (ApplicationSubdomain) são adicionadas na classificação (Classification) do ativo (Figura 3.7). Possuem seus relacionamentos de multiplicidade 0..*, deixando livre a quantidade de domínios e subdomínios a serem descritos no ativo.

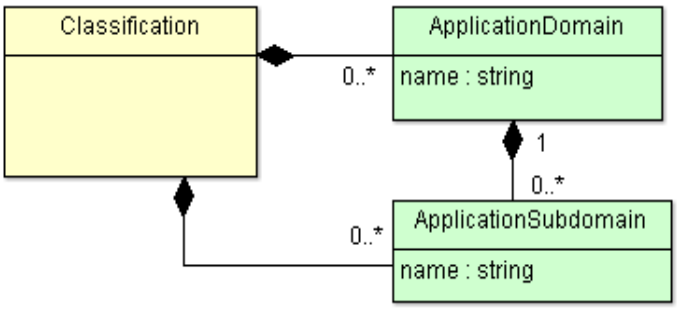


Figura 3.7: Extensão da classificação com domínios e subdomínios de aplicação.

3.1.4.4 Exemplos

Um exemplo de um ativo do domínio jurídico pode ser um serviço que consulta as informações de processos jurídicos públicos, contendo o subdomínio de processos jurídicos. Outro exemplo de ativo, mas no domínio da Geologia e subdomínio de prospecção de petróleo, pode ser um componente com funções de análise de rocha com conhecimento especializado para o desenvolvimento de aplicações de prospecção de petróleo.

Os valores para a taxonomia de domínios e subdomínios a serem utilizados dependerá do contexto do desenvolvimento do ativo. Um exemplo base é a Computing Classification System da ACM (2012), a qual é utilizada para o domínio da computação e é divida nos subdomínios:

- General and reference
- Hardware
- Computer systems organization
- Networks

- Software and its engineering
- Theory of computation
- Mathematics of computing
- Information systems
- Security and privacy
- Human-centered computing
- Computing methodologies
- Applied computing
- Social and professional topics
- Proper nouns: People, technologies and companies

Outros exemplos de taxonomias podem ser da Elsevier (2013), a ANZSRC (Australian and New Zealand Standard Research Classification) (AUSSTATS, 2008) e o JACS (Joint Academic Coding System) (HESA, 2008). Essas taxonomias são mais amplas, descrevendo os domínios com os respectivos subdomínios tais como os de astronomia, computação, defesa, economia e finanças, educação, química, matemática, medicina e sociedade.

3.1.5 Classificação por Organizações e Projetos

3.1.5.1 Intenção

Classificar o ativo pelas organizações e projetos em que foi desenvolvido, visando ao desenvolvimento distribuído de ativos e às permissões de acesso.

3.1.5.2 Motivação e Aplicabilidade

Existem dois enfoques para estabelecer um programa de reutilização em uma organização, o desenvolvimento centralizado e o desenvolvimento distribuído de ativos (FRAKES, 2005). O desenvolvimento centralizado é baseado em uma unidade dedicada para desenvolver, distribuir, manter e, frequentemente, prover treinamento sobre os ativos reutilizáveis. O custo dessa unidade requer um investimento longo, necessitando de um forte apoio gerencial, e é amortizado pelos projetos na linha de produto a qual utiliza tais ativos desenvolvidos.

Já para o desenvolvimento distribuído, o programa de reutilização é implementado colaborativamente pelos projetos na mesma linha de produto, havendo diversas unidades de desenvolvimento de ativos. Cada projeto tem a responsabilidade de contribuir com ativos reutilizáveis para outros projetos, entretanto, o desenvolvimento e o suporte são distribuídos entre os projetos. Por não haver necessidade de criar uma unidade separada de desenvolvimento, os custos são distribuídos entre os projetos, não sendo necessário um investimento de antemão. Por questões de privacidade dos ativos no desenvolvimento distribuído, o controle de acesso pode ser necessário para que apenas os projetos e as organizações relacionadas possam acessar os ativos para leitura, reutilização ou alteração.

3.1.5.3 Estrutura Proposta

O ativo pode ser associado a diversos projetos e organizações (Figura 3.8). Cada organização é relacionada em um ou mais projetos. Os usuários são associados a organizações e a projetos. Desse modo, é possível classificar e identificar os ativos de cada projeto e de cada organização, possibilitando o controle de acesso de cada ativo aos usuários.

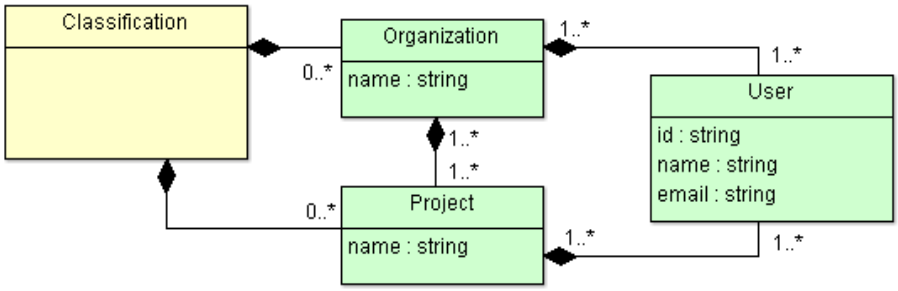


Figura 3.8: Extensão da classificação para organizações e projetos.

Para armazenar o controle de acesso do ativo, o RAS define no Default Profile o atributo Asset.accessRights, que pode ser qualquer string que represente as permissões dos usuários para interagir com o ativo tais como leitura ou reutilização. Entretanto, o RAS não define o formato das informações contidas nessa string, então o SW-RAS estende o escopo de utilização para consumidores (leitura e reutilização) e autores (edição) dentre equipes e organizações distribuídas. A estrutura é compatível também com o desenvolvimento centralizado, basta associar os usuários a uma única organização e a uma única equipe padrão.

O formato proposto pelo SW-RAS é baseado na notação simbólica de permissões de arquivos de sistemas operacionais Unix, possuindo três tríades de permissões, gerando o formato: "`RWXRWXRWX`". Cada tríade é formada por três caracteres: `RWX`, nessa sequência. A primeira tríade é para os usuários das equipes relacionadas ao ativo, a segunda para os usuários das organizações que desenvolvem o ativo, e a terceira tríade para os usuários em geral. O `R` representa o acesso de leitura (Reading) das informações do ativo, o `W` de escrita (Writing) nas informações do ativo, e o `X` de execução (eXecution) para copiar o ativo empacotado, adaptá-lo e executá-lo, ou seja, consumi-lo. O caractere "`-`" (hífen) simboliza que não há permissão respectiva à posição do caractere.

3.1.5.4 Exemplos

a) Desenvolvimento distribuído. Duas organizações compartilham o mesmo repositório de ativos reutilizáveis, e desenvolvem colaborativamente um ativo que poderá ser reutilizado por todos os usuários, porém apenas a equipe desse projeto (dividida entre essas duas organizações) pode editá-lo. Então remove-se as permissões de edição para todos que não forem da equipe do projeto, ficando com o formato "`RWXR-XR-X`".

b) Repositório de software livre. Um repositório de ativos de software livre é disponibilizado na Internet, onde todos podem requisitar para fazer parte das equipes e colaborar com as especificações, com o código e com a documentação dos ativos. Portanto, por padrão, todos os ativos terão o controle de acesso liberado de leitura (`R`) e

consumo (X) "RWX---R-X". As permissões de organização não serão utilizadas nesse caso.

3.1.6 Classificação por Licenças de Software

3.1.6.1 Intenção

Classificar os ativos pelas licenças de software, oportunizando a análise da compatibilidade de tais licenças por questões legais.

3.1.6.2 Motivação e Aplicabilidade

Em diversas áreas de aplicações de software, frequentemente é mais efetivo em custo adquirir em vez de desenvolver o software (PRESSMAN, 2011), então as licenças de software são importantes na escolha de tais componentes de software. Deve-se estar atento nos diferentes tipos de licenças e entender como o componente é licenciado antes de utilizá-lo (SOMMERVILLE, 2011), para assim então decidir de usar o componente em um sistema ou em outro em diferentes maneiras.

Essa atenção dada às licenças de software é necessária porque os problemas legais impedem o reúso (FRAKES, 1990), por ter que proteger legalmente os direitos dos criadores e consumidores de software reutilizável, e também no ponto de vista de suas responsabilidades (FRAKES, 1994). Desse modo, empacotar a licença de software para a reutilização é relevante e influencia diretamente na reusabilidade dos ativos (NASA, 2011). Componentes de software licenciados geralmente constituem uma parte significante do produto (CHAVEZ, 1998), e antes de serem reutilizados, os desenvolvedores devem informar peculiaridades que caracterizam o licenciamento de componentes de software.

Outros estudos sobre licenças de software e questões legais na reutilização de software pode ser encontrados em (SAMETINGER, 1997), (IBM, 2008) e (DAVIES, 2011).

3.1.6.3 Estrutura Proposta

O ativo pode conter apenas um tipo de licença de software em sua classificação, sendo designada pela classe Software LicenseType (Figura 3.9). O atributo name dessa classe armazena as diversas licenças de software.

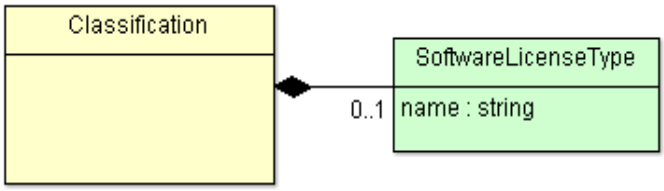


Figura 3.9: Extensão da classificação para licenças de software.

3.1.6.4 Exemplos

Ao desenvolver um sistema que utiliza ativos de software livre, deve-se também analisar a compatibilidade entre as licenças dos componentes utilizados com o próprio sistema. Como base para essa análise, pode-se utilizar a lista de licenças descritas em (FSF, 2013). Essa lista é bastante completa para software livre, separando as licenças

compatíveis das não compatíveis com a licença GNU GPL (GNU General Public License), contendo inclusive uma lista de licenças que não são para software livre. Alguns exemplos dessa lista, compatíveis com a GNU GPL, são:

- GNU General Public License (GPL) version 2
- GNU General Public License (GPL) version 3
- GNU Affero General Public License (AGPL) version 3
- GNU Lesser General Public License (LGPL) version 3
- GNU Lesser General Public License (LGPL) version 2.1
- Apache License version 2.0
- Artistic License 2.0
- Clarified Artistic License
- Educational Community License 2.0
- FreeBSD License
- Intel Open Source License
- MIT
- Mozilla Public License (MPL) version 2.0
- Public Domain
- Standard ML of New Jersey
- The Clear BSD
- W3C Software Notice and License

3.1.7 Classificação por Custos e Esforços

3.1.7.1 Intenção

Classificar os ativos pelos custos e esforços de desenvolvimento, servindo como base para o gerenciamento do ROI (Retorno no Investimento) do desenvolvimento para reúso.

3.1.7.2 Motivação e Aplicabilidade

Um dos diversos aspectos explorados em reúso de software está em definir o quanto o reúso (de diversos artefatos) reduz o custo de desenvolvimento (AGRESTI, 2011). Frakes (1990) aponta que a viabilidade econômica está dentre diversos problemas que impedem o reúso de software, e os custos relacionados com a criação e a administração do sistema devem ser considerados ao representar componentes reutilizáveis.

Gartner (2010) aponta que para julgar a efetividade do processo de reúso de software, as quantidades de tempo, de esforço e de custo associadas com a criação de

cada artefato devem ser quantificadas para gerir a organização. Junto disso, deve existir a quantidade de artefatos reutilizáveis ou serviços, e o número de vezes que cada artefato é utilizado ao longo do tempo, com a economia de custos e esforços.

Para Sherif (2003), uma das barreiras para a adoção do reúso é a falta de medidas quantitativas para avaliar os benefícios e custos, causando aversão da gerência ao reúso de software. Além disso, nem os custos nem os benefícios são precisamente conhecidos e modelos propostos não são fáceis de compreender. Para Ramachandran (2005) isso que ocorre é justificável, porque a principal responsabilidade de um gerente de projetos é entregar o sistema de software requerido no tempo e dentro do orçamento. Criar componentes reutilizáveis requer esforço adicional para ser gasto, o que não é um benefício imediato para o projeto. O gerente de projetos não pode dar maior prioridade para a produção de componentes reutilizáveis, porém o reúso de componentes de software é a chave para ganhos significantes na produtividade. Para alcançar esse potencial completo, é necessário focar-se no desenvolvimento para reúso, o qual é um processo de produção de potenciais componentes reutilizáveis. O custo extra do desenvolvimento de componentes reutilizáveis deve ser uma responsabilidade organizacional e não apenas de um projeto.

3.1.7.3 Estrutura Proposta

A classe Effort (Figura 3.10) representa os custos e os esforços do desenvolvimento do ativo. Seu relacionamento com a classe de Classificação permite que o ativo possa ser descrito em diversos tipos de esforços e custos (classe EffortType), tais como em tempo estimado de desenvolvimento, tempo real de desenvolvimento, quantidade monetária, pessoas envolvidas no desenvolvimento, linhas de código, entre outros.

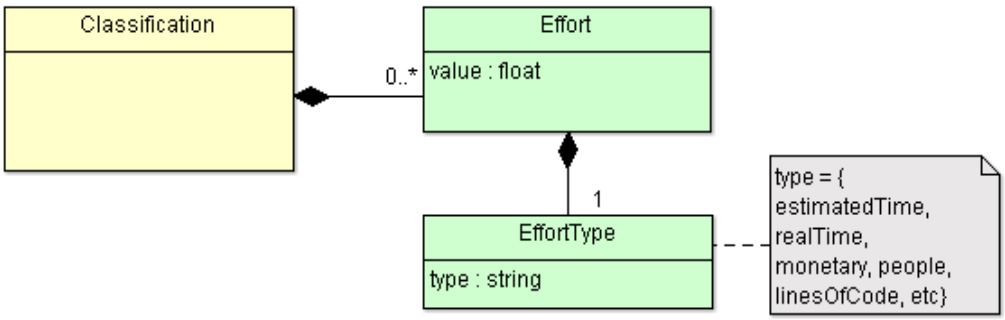


Figura 3.10: Extensão da classificação para custos e esforços.

A estrutura flexível da classe Effort com o tipo (EffortType) e valor (value) permite que outros custos e esforços possam ser especificados. Um desses exemplos é a porcentagem de cobertura de testes de unidade do código contido no ativo (FRAKES, 1996). Outras dessas métricas podem ser encontradas em Mills (1988).

3.1.7.4 Exemplos

a) Análise de custo/benefício. Utilizando-se o valor monetário de desenvolvimento de cada ativo, pode-se obter o ROI (Retorno no Investimento) de cada ativo, analisando-se a razão do valor monetário pela reusabilidade. Por exemplo, um framework custou R$7.000,00 para ser desenvolvido completamente tornando-se um ativo reutilizável na organização, e até o dado momento ele foi reutilizado vinte vezes em sistemas de software, então a efetividade de custo por projeto é de R$450,00.

b) Tempo de desenvolvimento dos ativos. O tempo estimado e o tempo real de desenvolvimento do ativo mostram a efetividade da estimativa de tempo para o desenvolvimento para reúso dos ativos. A partir disso, pode-se ajustar a estimativa de tempo para os próximos ativos reutilizáveis a serem desenvolvidos, permitindo analisar a sua viabilidade em tempo de desenvolvimento. Por exemplo, o tempo estimado para o desenvolvimento de um certo ativo foi de três semanas, porém levou-se cinco para ser desenvolvido, então a efetividade de estimativa desse ativo é 60% (5 semanas / 3 semanas). O histórico dessa efetividade pode auxiliar na estimativa de desenvolvimento de novos de ativos.

3.1.8 Classificação por Social Tagging

3.1.8.1 Intenção

Classificar os ativos através do livre preenchimento de tags pelos usuários.

3.1.8.2 Motivação e Aplicabilidade

Social Tagging permite que tags (palavras-chaves) sejam descritas livremente por indivíduos, em um ambiente social e aberto. A Folksonomia, termo cunhado por Wal (2007) pela combinação de *folks* (pessoas) com taxonomia, é o resultado dessas tags que gera uma base de informação para classificação com o vocabulário e significado para esses próprios indivíduos. As tags complementam a descrição dos ativos reutilizáveis de software pelo fato de não terem uma estrutura específica.

De um modo análogo como o Delicious[1] classifica diversos websites por tags, os ativos podem ser descritos por tags pelos seus usuários. Pode-se então classificar os ativos que são similares por conter as mesmas tags, ou então classificar os usuários que têm interesses similares por descreverem as mesmas tags, e fazer disso sugestões de ativos relacionados ou encontrar usuários com interesses similares. Os ativos em um repositório podem ser encontrados a partir da pesquisa de suas tags, e pesquisas baseadas em relevância podem dar mais peso para o campo de tag.

A nuvem de tags (tag cloud) também pode auxiliar na descoberta dos ativos em um repositório. Uma nuvem de tags é uma apresentação visual de um conjunto de tags selecionadas por uma razão, e seus atributos de texto (tais como tamanho, peso ou cor) são utilizados para representar características dos termos associados (RIVADENEIRA et al., 2007). A nuvem de tags pode representar o conjunto completo de documentos (ou ativos) ou apenas características de cada documento (ou ativo) (XEXEO, 2009). A Figura 3.11 ilustra um exemplo de nuvem de tags gerada de uma coleção de documentos, que poderia ser vista como uma nuvem de tags para ativos reutilizáveis de software.

1 http://delicious.com

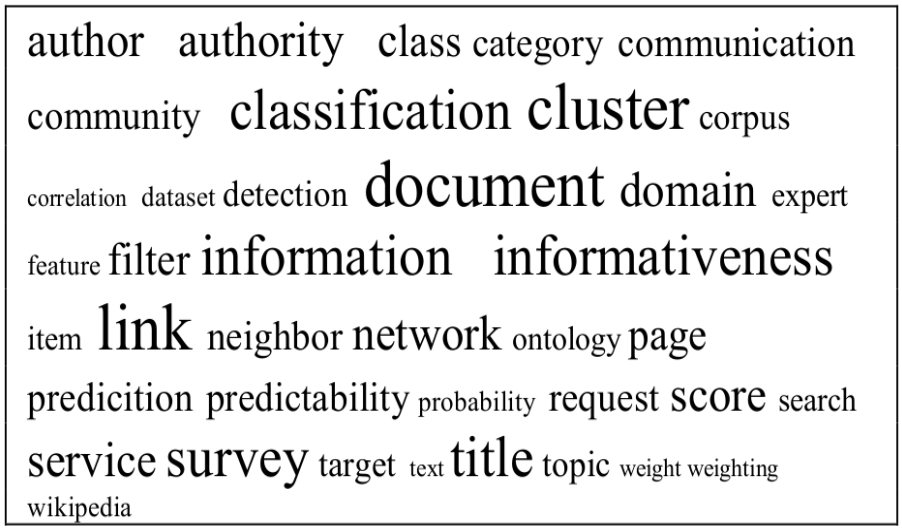


Figura 3.11: Exemplo de nuvem de tags gerada de uma coleção de documentos (XEXEO, 2009).

3.1.8.3 Estrutura Proposta

Diversas tags (classe Tag) podem ser associadas livremente na classificação do ativo por usuários (Figura 3.12). Cada tag é constituída do texto que o usuário descreve (atributo Tag.name), junto da data em que foi associada essa tag (atributo Tag.date) e do usuário que a preencheu (User). Desse modo, os usuários podem descrever diversas tags para cada ativo.

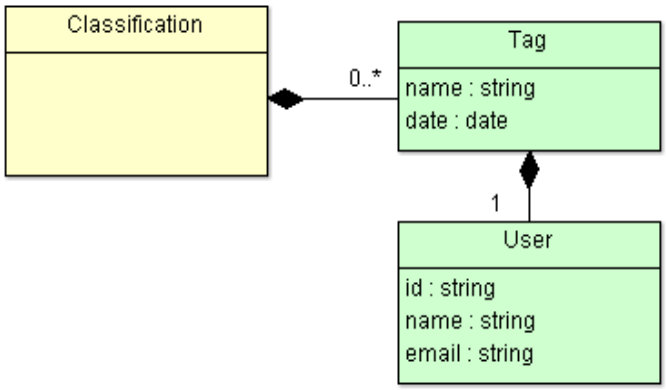


Figura 3.12: Extensão da classificação para social tagging e Folksonomia.

3.1.8.4 Exemplos

a) Navegação por tags no repositório. Considerando-se dois ativos em um repositório com a mesma tag denominada “PDF”. O primeiro ativo é uma API que extrai o texto e os metadados de documentos PDF (Portable Document Format). O segundo ativo é um gerador de relatórios para documentos PDF. Os dois ativos estão relacionados por essa tag “PDF”, uma informação não estruturada. Ao consultar um dos ativos, o usuário verá a tag e poderá clicar nela para pesquisar outros ativos com a mesma tag, ou então já serem listados seus ativos relacionados pelas tags antecipadamente na consulta do ativo.

b) Categorização por tags no repositório. Um conjunto específico de ativos pode ser identificado por uma tag específica. Por exemplo, para identificar os ativos que foram

desenvolvidos a partir de um programa de reutilização chamado “ProjReuso”, os produtores dos ativos podem utilizar a mesma tag “Proj_Reuso”.

3.2 Extensões na Solução dos Ativos

O SW-RAS estende a categoria de solução do ativo com informações e artefatos do ciclo de vida de software. A Figura 3.13 ilustra as extensões na solução na forma reduzida, que são descritas nas próximas subseções. As classes do Default Profile estão em amarelo e as classes do Default Component Profile em azul, sendo extendidas pelas classes do SW-RAS (em verde). Os diagramas exibem apenas as classes e os relacionamentos associados ao SW-RAS. São as extensões na solução com suas classes no modelo:

- Solução com Requisitos Funcionais
  - Classe **FunctionalRequirement**
- Solução com Requisitos Não-funcionais
  - Classes **NonFunctionalRequirement**, **i18n** e **OperationalSystem**
- Solução com Padrões de Projeto
  - Classe **DesignPattern**
- Solução com Interfaces de Usuários e seus Níveis de Abstração
  - Classe **UserInterface** e **UIAbstractionLevel**
- Solução com Serviços
  - Classe **WSDL**
- Solução com Códigos-fontes e Linguagens de Programação
  - Classe **SourceCode** e **ProgrammingLanguage**
- Solução com Artefatos e Métodos de Testes
  - Classe **TestArtifact**, **TestType** e **TestMethod**

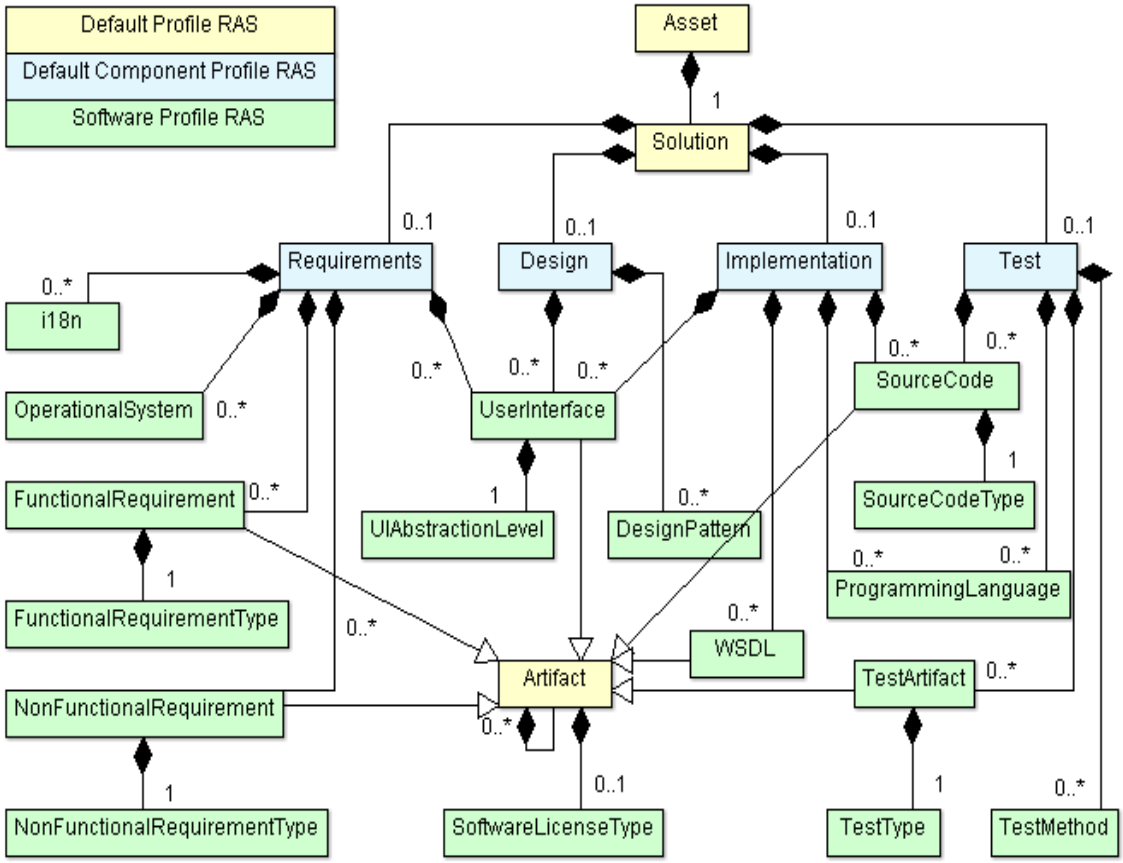


Figura 3.13: Representação reduzida das extensões da solução dos ativos contidas no SW-RAS.

3.2.1 Solução com Requisitos Funcionais e Requisitos Não-funcionais

3.2.1.1 Intenção

Especializar os artefatos de requisitos de software para requisitos funcionais (de usuário) e requisitos não-funcionais (de sistema).

3.2.1.2 Motivação e Aplicabilidade

Requisitos de software são a base para cada projeto (HULL, 2005), definindo os usuários interessados (usuários, clientes, fornecedores, desenvolvedores e pessoas de negócios) em um novo sistema necessário a eles, definindo também o que o sistema deve fazer para satisfazer esse necessário. Desse modo, os requisitos são importantes também para descrever efetivamente os ativos reutilizáveis de software, visto que também são desenvolvidos através de projetos.

Os requisitos são descritos através de técnicas que a Engenharia de Requisitos provê, e muitos dos problemas ocorridos no processo de Engenharia de Requisitos são pela falha na separação nítida entre diferentes níveis de descrição (SOMMERVILLE, 2011). Os requisitos de usuário são de alto nível de abstração, sendo afirmações em linguagem natural e em diagramas de quais serviços e restrições o sistema deve prover aos usuários. Artefatos de requisitos funcionais podem ser na forma de protótipos, casos de uso, estórias de usuário, etc. Já os requisitos de sistema fazem a descrição detalhada do que o sistema deve fazer, entre funções, serviços e restrições operacionais do sistema.

Artefatos de requisitos não-funcionais podem especificar em relação à usabilidade, à internacionalização, ao desempenho, aos sistemas operacionais, à legislação, à segurança, entre outras formas. A Figura 3.14 ilustra mais dessas formas, com os tipos e subtipos não-funcionais para os requisitos de sistema descritos por Sommerville (2011).

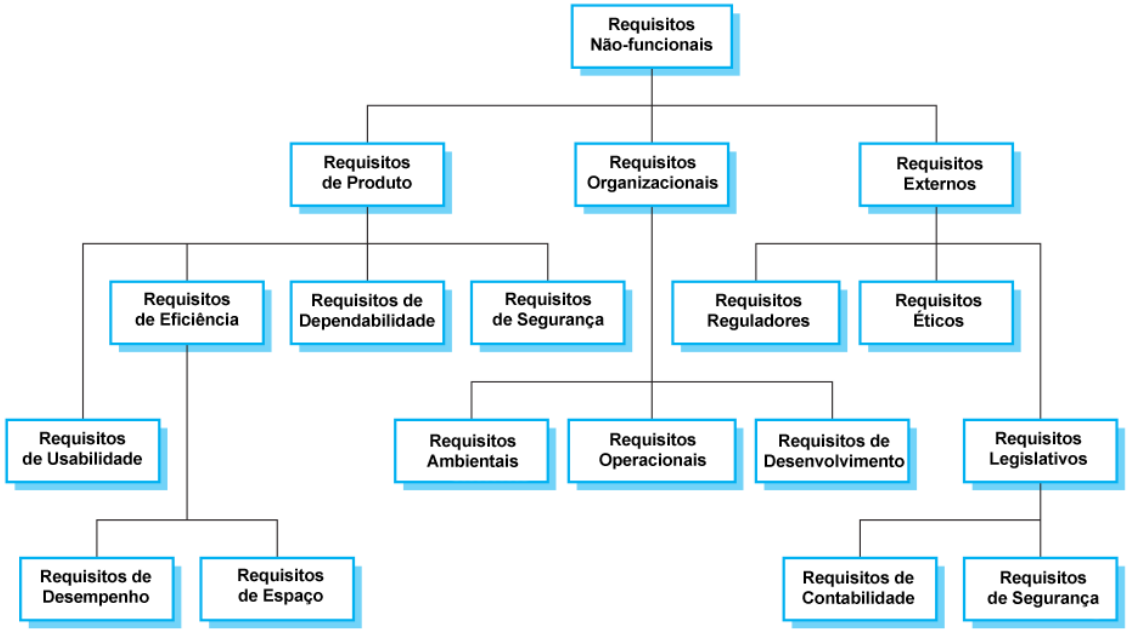


Figura 3.14: Tipos e subtipos para os requisitos não-funcionais de sistema – adaptada e traduzida de (SOMMERVILLE, 2011).

3.2.1.3 Estrutura Proposta

Os artefatos de requisitos funcionais (FunctionalRequirement) e os artefatos de requisitos não-funcionais (NonFunctionalRequirement) são relacionados na classe de requisitos do RAS (Requirements), ilustrado na Figura 3.15. Os tipos dos requisitos funcionais são designados pela classe FunctionalRequirementType e os tipos dos requisitos não-funcionais pela classe NonFunctionalRequirementType, além do sistema operacional (OperationalSystem) e de internacionalização (i18n).

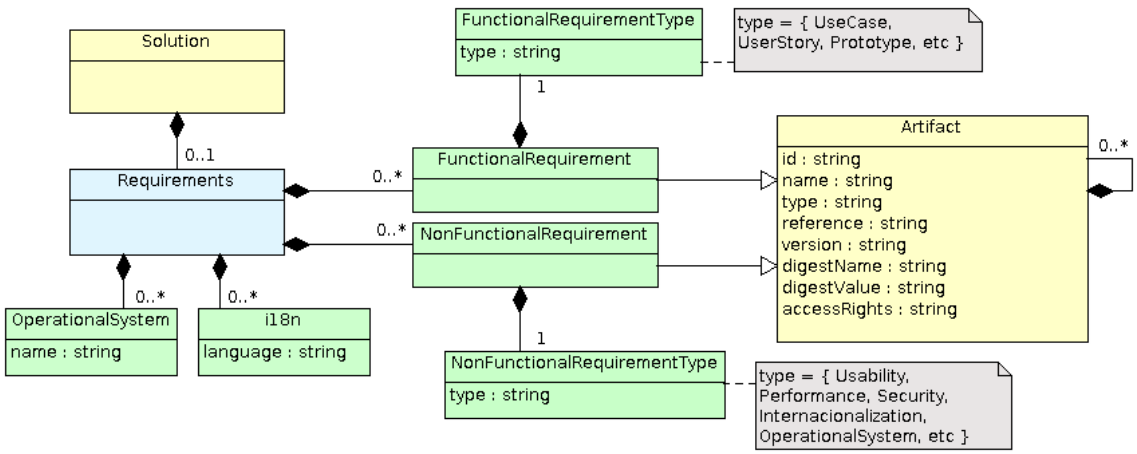


Figura 3.15: Extensão da solução para requisitos funcionais e requisitos não-funcionais.

3.2.1.4 Exemplos

a) Requisito funcional de caso de uso. Um ativo do tipo serviço pode conter um diagrama de caso de uso que especifica a funcionalidade principal que ele possui para outros sistemas o utilizarem.

b) Requisito funcional de componentes. Um diagrama UML de componentes pode especificar a sua estrutura interna com as interfaces das funcionalidades disponíveis pelo componente.

c) Requisito não-funcional de desempenho. Uma forma de requisito de desempenho para um ativo do tipo serviço pode ser a quantidade de transações por segundo que ele suporta, por exemplo, 150 requisições no mínimo e 300 no máximo, criando-se uma fila de requisições quando ultrapassar de 300 requisições.

d) Requisito não-funcional de dependabilidade. Um requisito de dependabilidade para um sistema de permissões de usuários pode ser a disponibilidade em porcentagem de tempo que um ativo de serviço está funcionando sem erros, por exemplo, em 99,9% do tempo deve estar disponível.

3.2.2 Solução com Padrões de Projeto

3.2.2.1 Intenção

Identificar as informações de padrões de projeto contidos na solução dos ativos.

3.2.2.2 Motivação e Aplicabilidade

Padrões de projeto permitem reutilizar a estrutura de diversas classes e suas colaborações em uma categoria de problemas, descrevendo uma estrutura que soluciona um problema de projeto particular dentro de um contexto específico (PRESSMAN, 2011). Para Fowler (2003), um problema importante está em como falar sobre assuntos de projeto em uma maneira relativamente encapsulada, e padrões de projeto podem auxiliar nisso por identificar soluções comuns para problemas recorrentes. Outro valor visto por Fowler, é o de que os padrões de projeto auxiliam especialistas com um vocabulário padrão, podendo-se utilizá-los com uma boa chance de entendimento do significado pelo projetista sem muita explicação extra, facilitando discutir a fase de projeto de software.

3.2.2.3 Estrutura Proposta

Os padrões de projetos (DesignPattern) são associados ao projeto do ativo do RAS (Design), podendo haver diversos padrões relacionados (Figura 3.16).

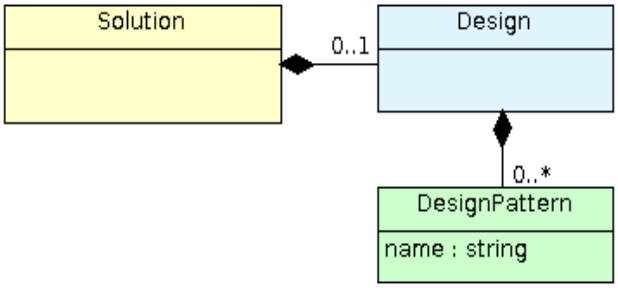


Figura 3.16. Extensão da solução do ativo para padrões de projeto.

3.2.2.4 Exemplos

Exemplos de padrões de projeto podem ser desde a categoria clássica de padrões GoF (GAMMA et al., 1995), J2EE, ou outros tipos fazendo referência a catálogos diferentes de padrões. Os padrões GoF dividem-se em propósitos de criação, estruturais e comportamentais. Um exemplo de um padrão estrutural é o Composite, o qual é utilizado para representar hierarquias parte-todo de objetos, e pode ser amplamente utilizado na construção de interfaces de usuários reutilizáveis. A Figura 3.17 exibe um exemplifica o Composite para componentes gráficos. Um outro exemplo é o padrão estrutural Façade, o qual pode ser utilizado para compor uma interface simplificada para um componente reutilizável que encapsula um subsistema complexo de mapeamento de memória para sistemas operacionais (Figura 3.18). Nesse exemplo, Domain é o Façade, representando um espaço de endereço, o qual encapsula um subsistema de gerenciamento de memória o qual um processo de sistema operacional (Process) pode requisitar funcionalidades.

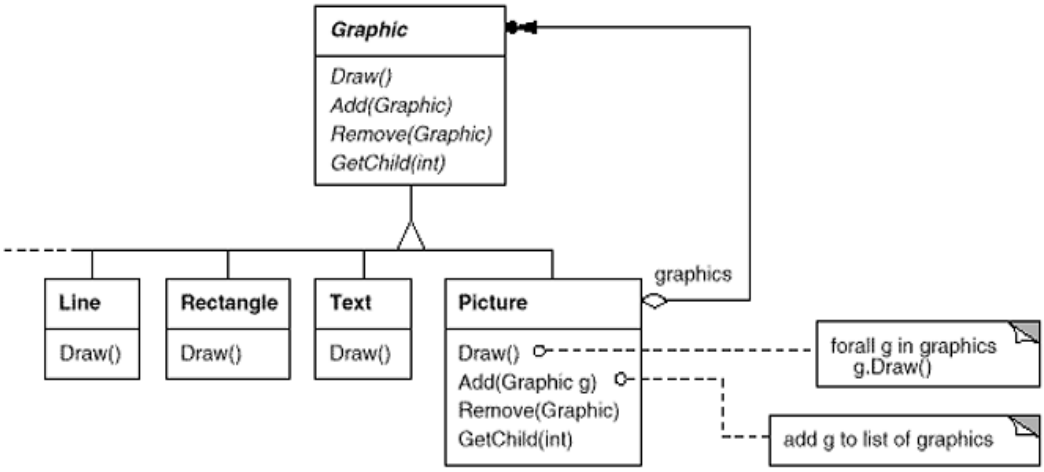


Figura 3.17: Exemplo do padrão de projeto estrutural Composite para compor componentes de interfaces de usuários reutilizáveis (GAMMA et al., 1995).

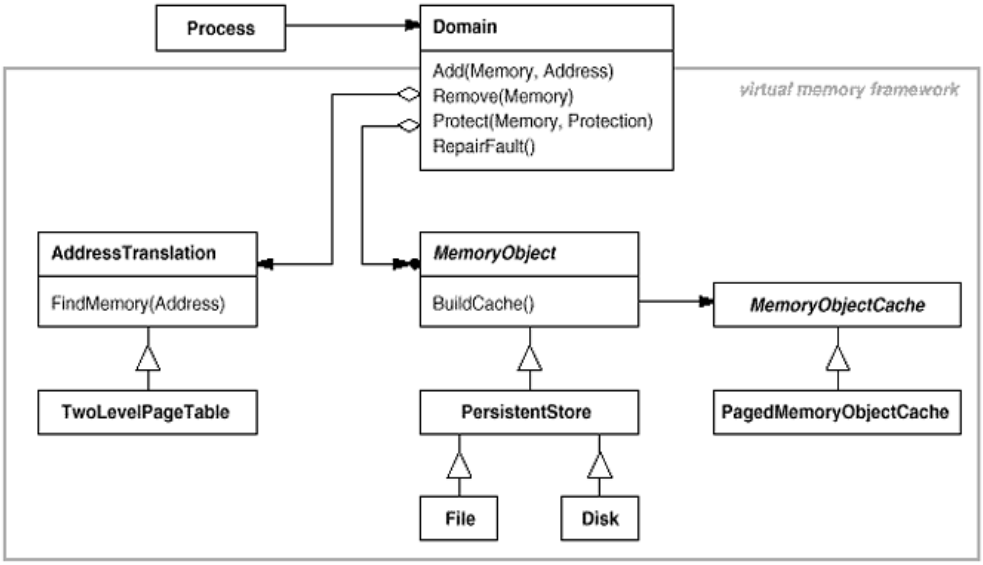


Figura 3.18: Exemplo do padrão de projeto estrutural Façade encapsulando um subsistema de endereçamento de memória de sistemas operacionais (GAMMA et al., 1995).

3.2.3 Solução com Interfaces de Usuário e seus Níveis de Abstração

3.2.3.1 Intenção

Especificar como solução os artefatos de interfaces de usuário e a informação de seus níveis de abstração de reúso.

3.2.3.2 Motivação e Aplicabilidade

Os componentes de interface constituem a forma mais utilizada de reúso em interfaces de usuário (MOREIRA, 2007). Comumente, as ferramentas IDEs disponibilizam uma paleta de objetos de interface de usuário (widgets) com tais componentes de interface na forma de botões, menus, caixas de texto, caixas de listagem, entre outros.

As interfaces de usuários (IU) podem ser reutilizadas em diversos níveis de abstração, a partir dos requisitos os quais elas estejam associadas. O Cameleon Reference Framework Estendido (MOREIRA, 2007) define seis desses níveis de abstração para os artefatos de interfaces de usuários (Figura 3.19), do nível mais abstrato ao nível mais concreto:

- Task & Concepts (T&C): descreve várias tarefas e conceitos do domínio para o desempenho das tarefas na IU.
- Abstract UI (AUI): define containers abstratos e componentes individuais, sendo uma expressão canônica da renderização dos conceitos de domínio e de tarefas independente do ambiente de interação da IU.
- Sketched UI (SUI): são artefatos de esboço (protótipos, desenhos, screenshots) com uma descrição textual da IU modelada. Deve existir somente um artefato de esboço (SUI) por IU.
- Concrete UI (CUI): permite a especificação da aparência e do comportamento da IU através da descrição dos objetos de interação. É uma representação abstrata de uma FUI, sendo independente de plataforma computacional e dependente apenas do ambiente de interação.
- Final UI (FUI): é uma IU operacional, executada em uma plataforma computacional específica. São códigos-fontes que implementam a IU, podendo ser interpretada, compilada e executada na sua plataforma, tal como em Java Swing ou HTML.
- Executable (XUI): é um programa executável na plataforma que renderiza a IU.

Esses níveis constroem um modelo de IU, chamado de árvore de reificação de IU, com artefatos em seus níveis de abstração (amplitude vertical) e n ramificações para cada nível (amplitude vertical). Quanto maior a amplitude vertical de um ativo reutilizável de interface, mais chance de reúso ele terá nas fases do processo de desenvolvimento (Tabela 3.1). Quanto maior a amplitude horizontal, mais opções de reúso o ativo terá para cada uma dessas fases do processo de desenvolvimento. Cada nível mais concreto deve representar seu nível mais abstrato. A Figura 3.19 ilustra o processo de reificação (implementação), de abstração e de tradução entre dois contextos de interação A e B (MOREIRA, 2007).

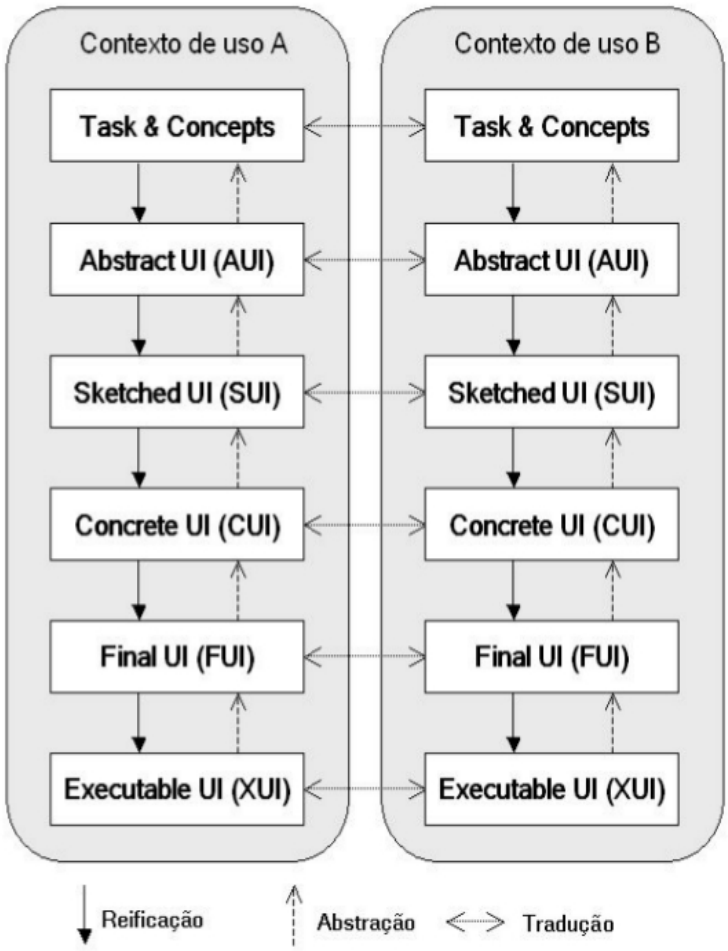


Figura 3.19: Os seis níveis de abstração do Cameleon Reference Framework Estendido com os relacionamentos de reificação, de abstração e de tradução entre dois contextos de interação A e B (MOREIRA, 2007).

Tabela 3.1. Níveis de abstração de reúso de interfaces de usuários para um requisito de caso de uso nas fases de desenvolvimento de sistemas.

Fase	**Número**	**Nível de reúso**	**Comentário**
Análise	0	Nenhum	Não há nenhum artefato que possa ser reusado.
	1	Objetivo	Somente o nome do caso de uso e, eventualmente, seus relacionamentos são descritos. Não há a descrição do fluxo de eventos (narrativa).
	2	Narrativa inicial	Contém o fluxo principal do caso de uso descrito, mas ainda não são detalhados os fluxos alternativos.
	3	Narrativa completa	O caso de uso está completamente descrito.
	4	AUI	Os conceitos que participam do caso de uso e os espaços abstratos de interação estão definidos.
	5	SUI	Protótipo da interface.
Projeto	6	CUI Básica	A CUI Básica serve como especificação inicial da IU.

			Pode-se definir IUS em ambientes de tecnologia de interação genéricos.
	7	CUI Intermediária	CUIs intermediárias permitem poupar parte do trabalho de especificação de IU.
	8	CUI Completa	A especificação da IU está completa. O reúso neste nível significa que não há necessidade de esforço de análise e projeto para a obtenção da IU desde caso de uso.
Construção	9	FUI	O código-fonte da IU na tecnologia alvo pode ser reusado. O esforço de desenvolvimento neste nível de reúso restringe-se a pequenos ajustes (customizações) no código.
	10	XUI	O reúso em nível de XUI elimina quaisquer esforços de desenvolvimento desta IU (portanto, com 100% de reúso) na medida em que é necessário apenas definir como executar a XUI no contexto do sistema.

Fonte: MOREIRA, 2007. p. 88 (adaptada).

3.2.3.3 Estrutura Proposta

Os artefatos de interface de usuário (UserInterface) estão presentes na análise de requisitos (Requirements), no projeto (Design) e na implementação (Implementation) dos ativos, ilustrado na Figura 3.20. Cada artefato de interface de usuário deve estar associado a um nível de abstração (UIAbstractionLevel) do Cameleon Reference Framework Estendido.

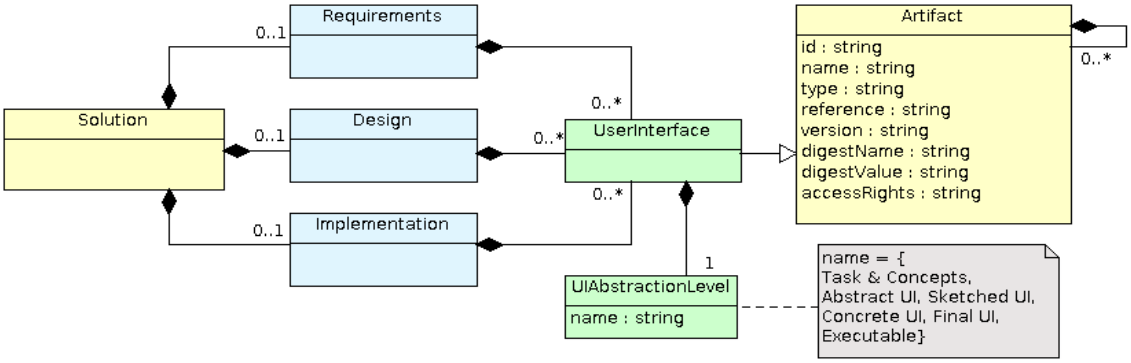


Figura 3.20: Extensão da solução dos ativos com artefatos de interfaces de usuários e seus níveis de abstrações do Cameleon Reference Framework Estendido.

3.2.3.4 Exemplos

Moreira (2007) exemplifica os seis níveis de abstração (Figura 3.21) do padrão de interação Seleção de Partes (Parts Selector) (Figura 3.22). O padrão é prototipado no nível Sketched UI (2) e implementado em duas FUIs: Web (1) e Java/Swing (6), com seu comportamento descrito por três CUIs: (3), (4) e (5). Desse modo, os consumidores do ativo podem optar entre a implementação Web ou Desktop (Java Swing), e compreender o comportamento das FUIs pelo protótipo (SUI) e pelas CUIs.

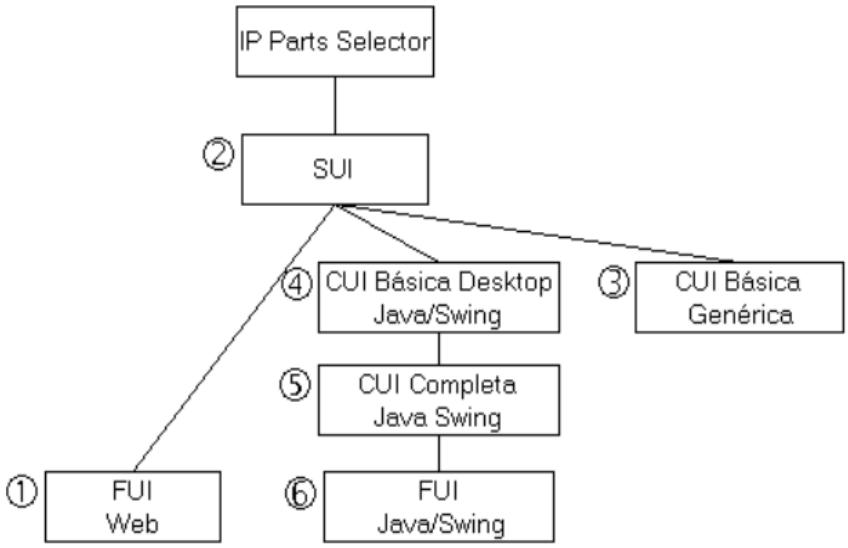


Figura 3.21: Níveis de abstração de um ativo reutilizável do padrão de interação com o usuário de Seleção de Partes (Parts Selector) (MOREIRA, 2007).

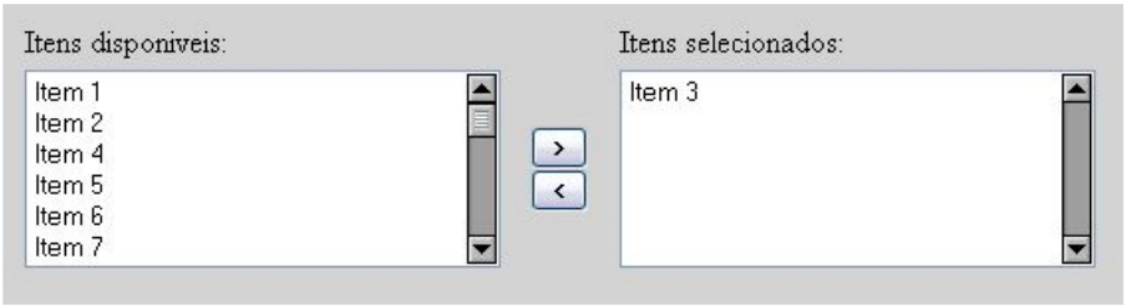


Figura 3.22: Padrão de interação com o usuário de Seleção de Partes (*Parts Selector*) (MOREIRA, 2007).

3.2.4 Solução com Serviços

3.2.4.1 Intenção

Especializar os artefatos de serviços na implementação da solução dos ativos.

3.2.4.2 Motivação e Aplicabilidade

Um serviço (web service) tem o benefício de ser reutilizado com baixo acoplamento, com neutralidade de plataforma, utilizando-se uma implementação padronizada. Em um plano de implementação de reúso numa organização, uma das fases previstas é a utilização dos serviços como ativos (IBM, 2008). Nessa fase, o foco está em utilizar SOA (Service-oriented Architecture) como um mecanismo de relevância e suporte ao reúso, sendo necessário também encontrar maneiras de reutilizar ativos de serviço disponibilizados por terceiros. SOA é projetada especialmente para prover flexibilidade e reúso (YU, 2009), permitindo a integração de sistemas, dados, aplicações e processos facilmente através da ligação de serviços, além de prover um ambiente de segurança e privacidade.

Machado (2010) defende que SOA e linhas de produto de software podem ser utilizados juntos com o propósito de aumentar e sistematizar o reúso durante o desenvolvimento, produzindo sistemas orientados a serviços mais rapidamente, mais

barato e customizável. As linhas de produto de software tratam de assuntos da engenharia de famílias de sistemas de software (KRUEGER, 2000), e têm como objetivo reduzir o esforço de produzir uma coleção de sistemas similares unindo as semelhanças de sistemas, formalmente gerenciando a variação desses sistemas, sendo os serviços reutilizáveis uma forma favorável ao reúso para esses sistemas.

3.2.4.3 Estrutura Proposta

A implementação da solução do ativo pode conter diversos artefatos WSDL de serviços (Figura 3.23), representados pela classe WSDL. Web Services Description Language (WSDL) é uma linguagem baseada em XML que descreve as funcionalidades oferecidas por um web service). Por estender Artifact, contém a referência (atributo reference) que descreve um URI (Universal Resource Identifier), que identifica a localização universal do serviço, podendo ser uma URL (Uniform Resource Locator) ou uma URN (Uniform Resource Name).

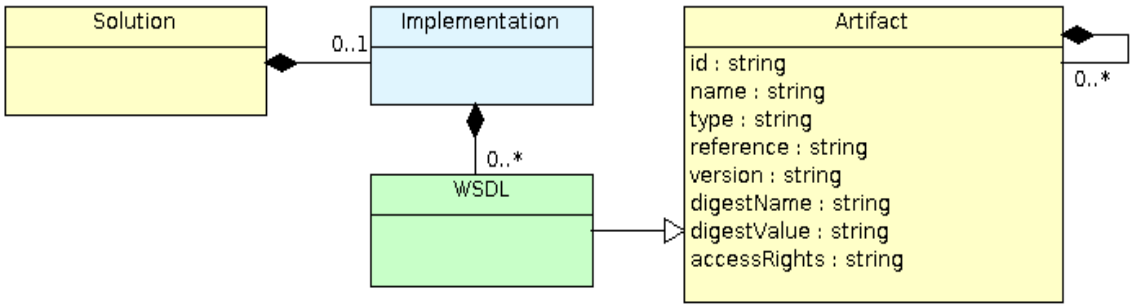


Figura 3.23: Extensão da solução para artefatos de serviços (web services).

3.2.4.4 Exemplos

Um exemplo de ativo reutilizável de serviço para a família de aplicações de geoprocessamento, pode ser uma validação se a coordenada geográfica pertence ao município informado. O artefato de WSDL conterá as informações para acessar o web service dessa função de validação.

3.2.5 Solução com Códigos-fontes e Linguagens de Programação

3.2.5.1 Intenção

Especificar os artefatos de código-fonte e a informação das linguagens de programação utilizadas na solução dos ativos.

3.2.5.2 Motivação e Aplicabilidade

De acordo com Agresti (2011), reutilizar o código de programação é uma técnica praticada amplamente para obter proveito de recursos no desenvolvimento de projetos, e, além disso, a linguagem de programação utilizada influencia na criação de um módulo reutilizável. Em uma pesquisa aplicada a desenvolvedores de software por Agresti, o tipo de artefato reutilizado que mais economizou o trabalho nos projetos (em 26%) foi o código-fonte entregue no sistema.

Além da reusabilidade considerável do código-fonte, armazenar o código-fonte do ativo de software é importante para suas futuras manutenções e evoluções. Sommerville (2011) aponta que existem custos e problemas associados com o reúso de software.

Sendo um desses problemas o da manutenção elevada dos componentes reutilizáveis. Isso ocorre caso o código-fonte do ativo reutilizado não esteja disponível.

3.2.5.3 Estrutura Proposta

Os artefatos de código-fonte (SourceCode) e as linguagens de programação (ProgrammingLanguage) e são adicionadas na implementação e no teste da solução dos ativos (Figura 3.24). Cada artefato de código-fonte deve conter o seu tipo (SourceCodeType), que pode ser desde uma função ou procedimento até um pacote ou aplicação. As linguagens de programação podem ser diversas, tais como C, C++, Java, PHP, ShellScript.

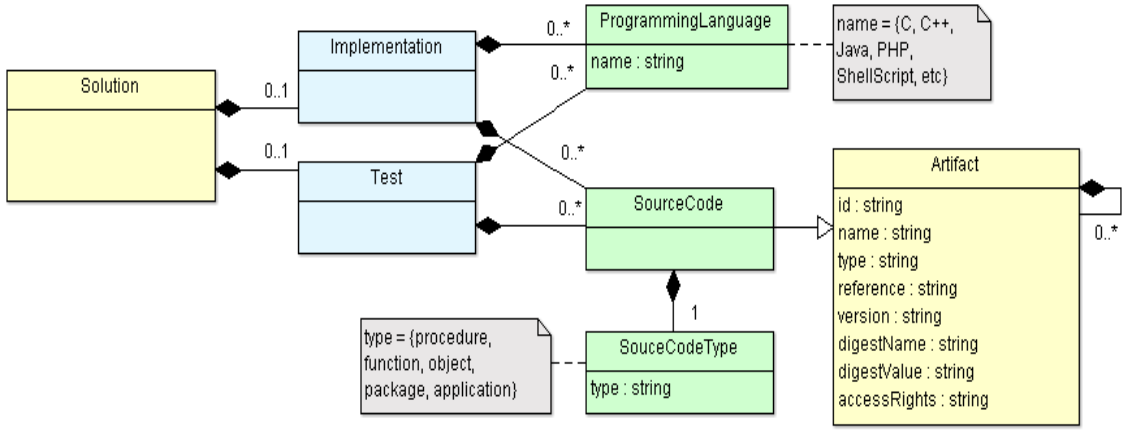


Figura 3.24: Extensão da solução para artefatos de códigos-fontes e linguagens de programação.

3.2.5.4 Exemplos

a) Componentes para autenticação e permissões. Para cada linguagem de programação utilizada no desenvolvimento em uma organização (por exemplo: C, Java e PHP), um componente pode ser desenvolvido para o acesso de um sistema centralizado de autenticação de usuários e permissões de ações. Cada componente possuirá sua linguagem de programação associada e seus artefatos de código-fonte.

b) Framework de desenvolvimento de sistemas Java. Um framework pode ser desenvolvido para reúso em sistemas Java em um ambiente de desenvolvimento de software, provendo diversas funcionalidades básicas para novos sistemas. Então, o ativo terá a linguagem de programação Java ligada à solução dele, juntamente ao código-fonte base para a reutilização nos novos sistemas.

3.2.6 Solução com Artefatos e Métodos de Testes

3.2.6.1 Intenção

Especializar os artefatos de teste e seus tipos, detalhando as informações dos métodos de teste aplicados no ativo.

3.2.6.2 Motivação e Aplicabilidade

Um dos principais enganos realizados na verificação de componentes reutilizáveis é o de que eles podem ser testados uma única vez e então reutilizados onde for (GAO et

al., 2003), porém, na realidade, um componente deve ser testado sempre que for reutilizado em um novo ambiente pelo fato de que o comportamento será diferente. Gao informa que, no processo de teste do Desenvolvimento Baseado em Componentes, a principal tarefa está em validar se o componente realiza a especificação e o projeto dele. Os testadores verificam o componente com métodos de teste de unidade, de teste de caixa-preta (black-box), de caixa branca (white-box), de desempenho, de segurança, de regressão, de carga, entre outros. O resultado são os artefatos de planos de teste, de casos de teste, de dados de teste, entre outros artefatos utilizados para a verificação do componente dentre diversos métodos.

Um caso famoso relacionado a testes de componentes é o desastre ocorrido com o foguete lançador Ariane 5 (LIONS, 1996), seus desenvolvedores reutilizaram alguns componentes da versão anterior (Ariane 4) para construi-lo, sem realizar um teste de integração, resultando as diversas consequências e falhas em uma explosão.

Além da importância do teste de integração de componentes em novos ambientes, um outro teste importante é o que ocorre na manutenção e na evolução dos sistemas. Um dos desafios da evolução dos componentes está em minimizar o esforço de selecionar e adaptar componentes de versões antigas durante o desenvolvimento de novos sistemas (JARZABEK, 2007). Esse esforço gera ganhos na produtividade, porém é necessário possuir os artefatos de testes para serem reexecutados no novo ambiente evoluído ou manutenido.

3.2.6.3 Estrutura Proposta

Os artefatos de teste (classe TestArtifact) com seus tipos (TestType), bem como os métodos de teste (TestMethod) aplicados, são associados no teste da solução do ativo (Test) (Figura 3.25). Os tipos dos artefatos de testes variam entre planos de teste, casos de teste, dados de teste, entre outros tipos. Os métodos de teste podem ser diversos, tais como teste de aceitação, de caixa-preta, funcional, de carga, de desempenho, de segurança, de usabilidade, de caixa-branca, entre outras técnicas.

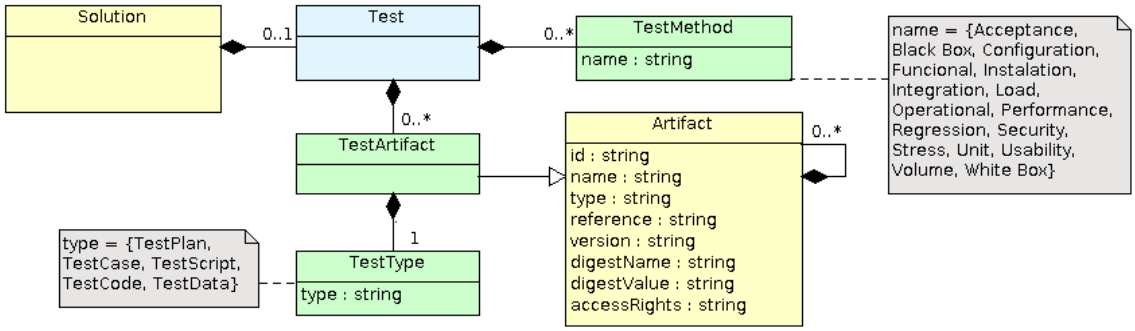


Figura 3.25: Extensão da solução para artefatos de testes com seus tipos e métodos de teste.

3.2.6.4 Exemplos

a) Testes de carga e de desempenho. Um ativo que empacota um serviço reutilizável pode contemplar artefatos de teste para os métodos de teste de carga e de desempenho, validando se o serviço suportará a quantia de demandas necessárias especificadas em seus requisitos não-funcionais.

b) Framework reutilizável para automatização de testes. Um ativo o qual representa

um framework com funcionalidades que facilitam a automatização dos testes de sistemas web, contendo os artefatos dos planos de testes bases para a automatização, junto com scripts, códigos e dados de teste.

3.3 Extensões no Uso dos Ativos

As extensões propostas no SW-RAS para o uso dos ativos abrangem os perfis de usuários para o ciclo de vida dos ativos, além da documentação para o uso dos ativos reutilizáveis de software, ilustrados na forma reduzida na Figura 3.26. No diagrama aparece apenas as classes e os relacionamentos relacionados ao SW-RAS. Da mesma forma que na classificação, o Default Component Profile não possui classes relativas ao uso dos ativos, por essa razão ele não é exibido nos diagramas das extensões do uso. São as extensões no uso com suas classes no modelo:

- Uso com Perfil de Usuário Consumidor
  - Classe **Consumer**
- Uso com Perfil de Usuário Produtor
  - Classes **Publisher** e **Author**
- Uso com Perfil de Usuário Certificador
  - Classe **Certifier**
- Uso com Guias de Usuário
  - Classe **UserGuide**
- Uso com Descrições de Uso
  - Classe **Description**
- Uso com Comentários de Usuários
  - Classe **UserComment**

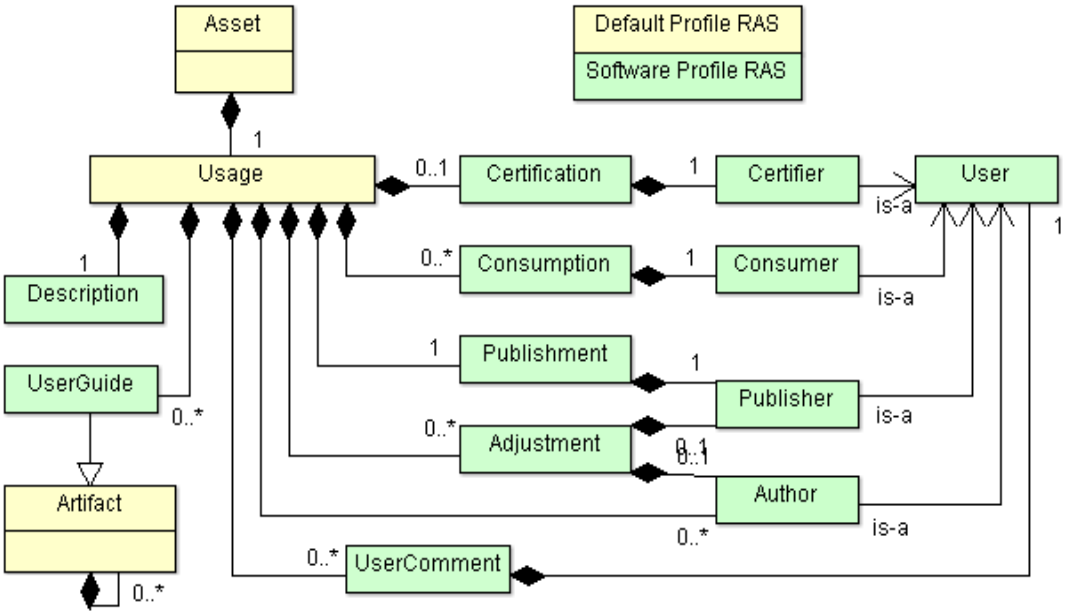


Figura 3.26: Representação reduzida das extensões do uso do ativo no RAS contidas no SW-RAS.

3.3.1 Uso com Perfis de Usuários Consumidores, Produtores e Certificadores

3.3.1.1 Intenção

Utilizar os perfis de usuários básicos para o processo de reúso de ativos, através da publicação, dos ajustes, da certificação e do consumo.

3.3.1.2 Motivação e Aplicabilidade

No processo de desenvolvimento Para Reutilização, os produtores projetam e desenvolvem ativos reutilizáveis para então os consumidores reutilizarem e desenvolverem novos sistemas, sendo essencial o relacionamento entre eles (SAMETINGER, 1997). O processo de Gerência de Reutilização do modelo de referência para Melhoria do Processo de Software MPS-BR define os papéis de usuários: o produtor, o consumidor e o gerente de ativos reutilizáveis (SOFTEX, 2011a). O gerente de ativos reutilizáveis é responsável pela gerência da biblioteca (repositório) de ativos reutilizáveis, supervisionando toda a utilização da biblioteca: o cadastro, a certificação e o controle de todo o ciclo de vida de tais ativos.

Um dos resultados esperados pelo processo de Gerência de Reutilização do MPS-BR é o registro dos dados de utilização dos ativos reutilizáveis contemplando sua produção, sua certificação e seus consumos. Essa informação permite notificar os consumidores e os produtores. Além disso, permite observar a utilização de cada ativo, podendo analisar a tendência ou comportamento específico de suas reutilizações. Esses dados facilitam também a notificação da descontinuação dos ativos ou da criação de novas versões.

3.3.1.3 Estrutura Proposta

O SW-RAS define quatro perfis de usuários, ilustrados na Figura 3.27, os quais são:

- Publicador (Publisher), que representa o usuário que cadastra o ativo em um repositório, o qual nem sempre é um dos autores. A data da publicação do ativo é

armazenada no atributo Publishment.date.

- Autor (Author), que representa os criadores do ativo e, por muitas vezes, também podem ser os publicadores de seus ativos criados. Os autores e o publicador do ativo podem ajustar o cadastro. Esse ajuste é armazenado na classe Adjustment, com o atributo date para a data de realização e o atributo changeDescription para o texto explicativo da mudança.
- Certificador (Certifier), que representa o responsável pela validação do ativo com a política de reutilização. A data da certificação é armazenada no atributo date da classe de certificação (Certification). O certificador é similar ao gerente do repositório definido no MPS-BR (SOFTEX, 2011a), podendo também descontinuar ou remover ativos. Para serem certificadores, os usuários são restringidos pelo atributo User.isCertifier.
- Consumidor (Consumer), que representa os usuários que reutilizam os ativos, e a data do consumo é armazenada no atributo Consumption.date.

Esse modelo de perfis de usuários é colaborativo, no qual qualquer usuário com acesso ao repositório pode publicar ou consumir os ativos reutilizáveis de software. Todos os perfis de usuários referenciam um usuário (User), então podem ser notificados ou contatados por possuírem o endereço de correio eletrônico (atributo User.email).

Tais perfis abrangem as ações de usuários para o ciclo de vida dos ativos. Essas ações são ilustradas no diagrama de caso de uso da Figura 3.28. Todos os usuários podem pesquisar ativos e adicionar comentários. O consumidor reutiliza ativos e os avalia pela qualidade no uso. O publicador cadastra o ativo e poderá ajustá-lo, além dos seus autores. O certificador realiza a certificação dos ativos, além de avaliá-los pela qualidade do produto, descontinuá-los ou removê-los do repositório.

O ciclo de vida de um ativo reutilizável é exemplificado na Figura 3.29. O ativo inicia com o estado “Em Definição” enquanto é desenvolvido. Assim que criado, então é publicado com o estado “Pronto para Reúso”. A certificação do ativo então é realizada, ou recusada até ser revisado e ajustado. O último estado do ativo é o Descontinuado, não estando mais disponível para reúso.

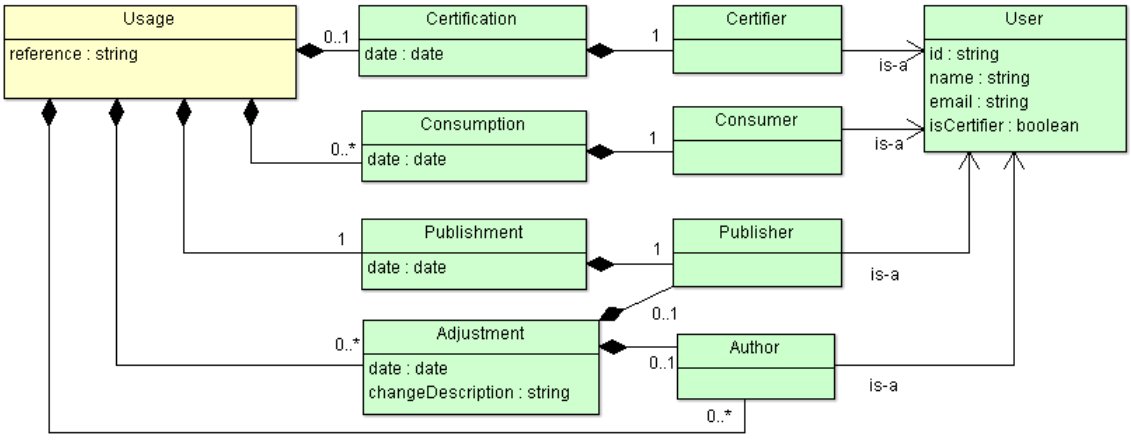


Figura 3.27: Extensão do uso para perfis de usuários certificadores, consumidores e produtores (autores e publicadores).

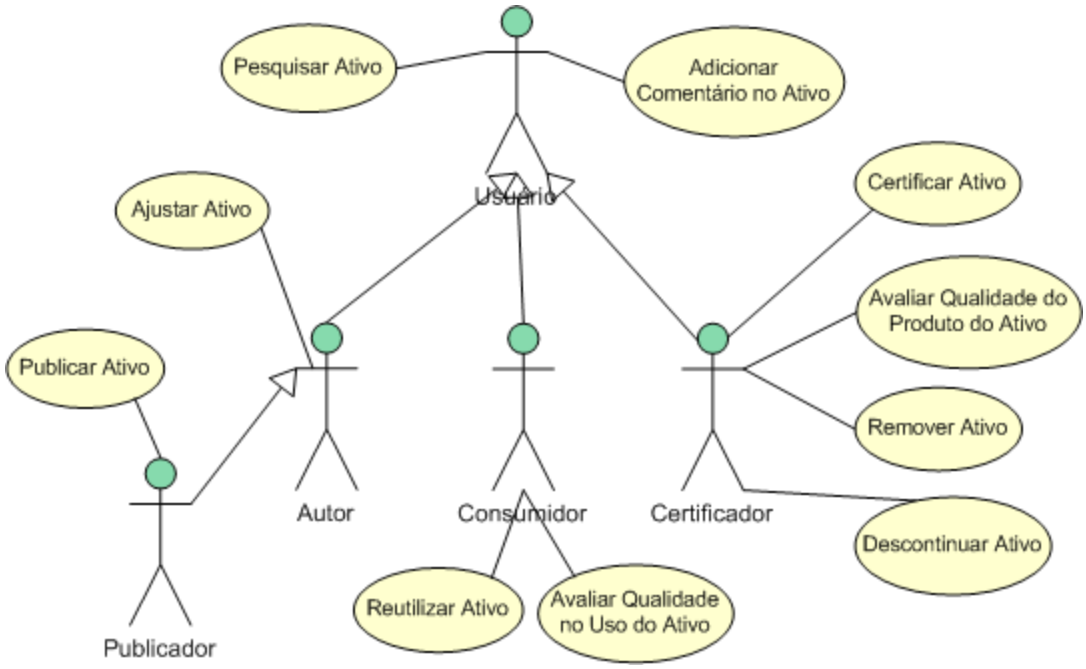


Figura 3.28: Diagrama de casos de uso das ações essenciais dos usuários e seus perfis.

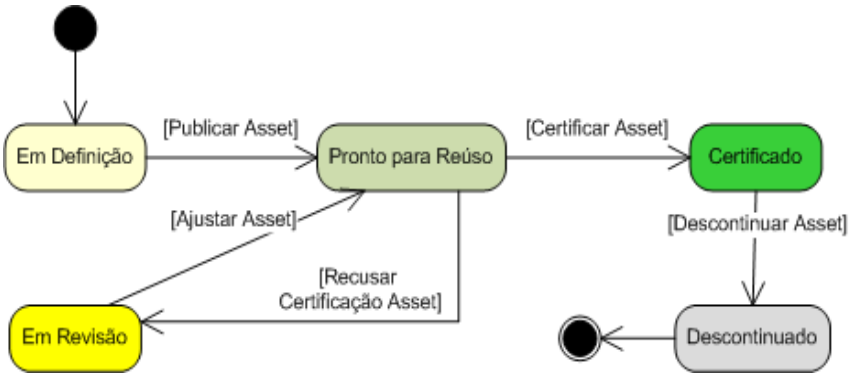


Figura 3.29: Diagrama de estados do ciclo de vida de um ativo.

3.3.1.4 Exemplos

a) Ciclo de vida de um ativo. Um usuário publicador P cadastra o ativo associando o usuário A como autor. Então, o usuário certificador C certifica o ativo, e os usuários produtores P e A recebem uma notificação por e-mail de que o ativo publicado foi certificado. Após a certificação, um usuário R consome o ativo. Após isso, o autor A atualiza o cadastro do ativo com um artefato modificado, então o usuário consumidor R recebe uma notificação automática por e-mail com a descrição da modificação. Após um longo tempo, o certificador recebe uma notificação automática do repositório de que o ativo está em desuso. Então o ativo é descontinuado no repositório pelo certificador, e os usuários produtores (R e A) e todos consumidores recebem uma notificação por e-mail que o ativo foi descontinuado.

b) Análise de tendência de reutilização. Através do atributo de data de consumo (Consumption.date) é possível gerenciar a tendência de reutilização de cada ativo ao longo do tempo. Com a tendência, pode-se prever sua reusabilidade e sua descontinuação.

3.3.2 Uso com Guias de Usuário, Descrições de Uso e Comentários de Usuários

3.3.2.1 Intenção

Acrescentar informações para o uso dos ativos com artefatos de guias de usuários, e informações por descrições textuais de uso e por comentários de usuários.

3.3.2.2 Motivação e Aplicabilidade

Guias de usuários para os ativos reutilizáveis de software orientam os consumidores sobre a reutilização, a integração e a aplicação do teste dos artefatos, complementando com as informações que os requisitos de software não contemplam. O objetivo desses guias é diferente do objetivo dos requisitos de software (HULL, 2005), pois os guias contêm uma escrita técnica para conduzir a utilização do software em questão, enquanto os requisitos informam as características do software. Na pesquisa realizada por Agresti (2011), os desenvolvedores de software tiveram a percepção de uma alta chance de reúso de guias de usuários e de descrições de sistemas, por possuir conteúdo consistente em tais documentos, tais como descrever o ambiente operacional e os modos de interação de usuário.

Como um meio de comunicação, os usuários podem comentar nos ativos de um modo livre, sendo útil para os produtores terem conhecimento de informações tais como o relato de defeitos, podendo esclarecer dúvidas dos consumidores e dos certificadores. Pode-se apresentar um histórico dos comentários para cada ativo, assim como um fórum de discussão.

3.3.2.3 Estrutura Proposta

O uso do ativo pode conter diversos artefatos de guias de usuários (UserGuide), além de possuir uma descrição textual obrigatória do uso do ativo (Description), ilustrados na Figura 3.30. Comentários de usuários podem ser associados no uso do ativo (UserComment), armazenando-se o texto no atributo comment e a data no atributo date.

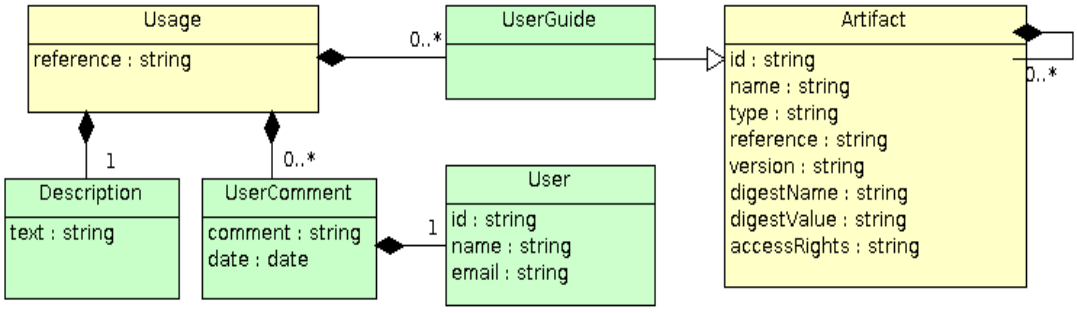


Figura 3.30: Extensões do uso com guias de usuários, descrição de uso e comentários de usuários.

3.3.2.4 Exemplos

a) Guia de usuário de um framework. O guia de usuário descreve as funções e a arquitetura que o framework disponibiliza para desenvolver novas aplicações em seu contexto, contendo instruções e tutoriais de como configurar o ambiente de desenvolvimento com o framework, facilitando o entendimento do seu funcionamento.

b) Comentários de usuários. Um consumidor encontra um defeito na execução do ativo e informa aos produtores pelo fórum de comentários do repositório, gerando uma notificação a todos os usuários relacionados ao ativo com o texto do comentário. Logo então, um dos autores pode informar que não era um defeito de código, e sim a falta de uma configuração específica descrita no guia de usuário, e que logo será atualizada e notificada a todos.

3.4 Extensão nos Ativos Relacionados

Apenas uma extensão é realizada nos ativos relacionados pelo SW-RAS, adicionando o tipo de versão anterior para os ativos relacionados, descrita a seguir.

3.4.1 Relacionamento de Ativos pela Versão Anterior

3.4.1.1 Intenção

Adicionar o tipo de versão anterior para os ativos relacionados, permitindo a rastreabilidade de versões anteriores em cada ativo.

3.4.1.2 Motivação e Aplicabilidade

O gerenciamento de versões é um dos requisitos para a construção de um repositório de reutilização padronizado (BURÉGIO et al., 2008), auxiliando a gerar bons resultados em seu uso. O repositório deve estar hábil em armazenar diferentes versões de ativos para que os desenvolvedores possam recuperar as versões anteriores de um ativo para suportar implementações alternativas, ligando todas as versões anteriores e novas de um ativo. Para o teste de componentes de software, Gao et al. (2003) indica que uma das razões da pobre testabilidade de componentes é a baixa rastreabilidade, dificultando o entendimento do comportamento de componentes e de relatório de erros durante a validação e integração de componentes.

Possuindo-se a informação de quais são as versões anteriores de um ativo e de quem o reutilizou, é possível notificar os consumidores de versões anteriores do ativo sobre a publicação de uma nova versão daquele ativo, informando todos os usuários interessados da nova versão (SOFTEX, 2011a). Além das notificações, o gerente do repositório pode facilmente encontrar as versões anteriores e descontinuá-las, caso a política do repositório seja de descontinuar os ativos anteriores quando houver uma nova versão.

3.4.1.3 Estrutura Proposta

O tipo de versão anterior (previousVersion) é adicionado aos tipos de ativos relacionados (RelatedAsset.relationshipType), ilustrado na Figura 3.31. Os tipos de relacionamento de ativos já definidos pelo RAS são os de agregação, similaridade, dependência e herança.

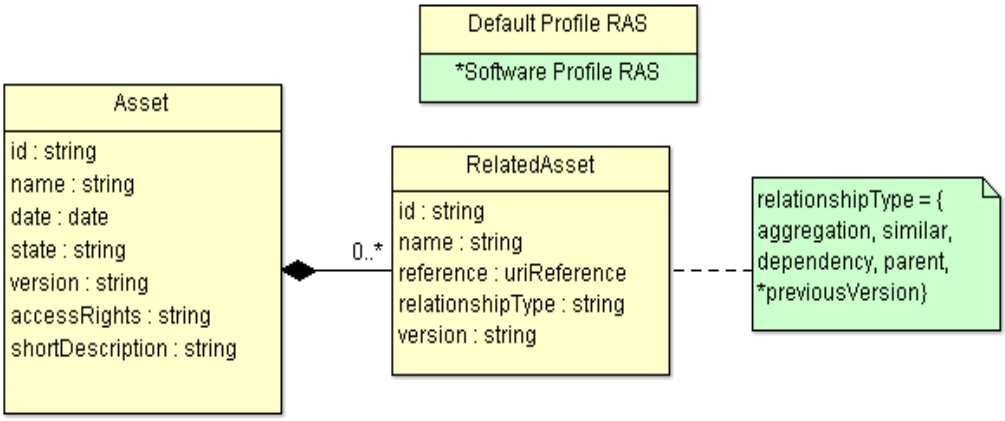


Figura 3.31: Extensão nos ativos relacionados com o tipo de versão anterior (previousVersion).

3.4.1.4 Exemplos

a) Notificação de nova versão do ativo. Ao ser publicada uma nova versão corretiva de segurança de um ativo em um repositório, todos os usuários consumidores das versões anteriores são notificados de que há uma nova versão importante do ativo.

b) Aviso de depreciação de ativo. Ao consultar um ativo com uma versão antiga, o usuário de um repositório pode ser avisado que uma nova versão já existe e é aconselhada, exibindo-se um link para a consulta dessa nova versão.

3.5 Obrigatoriedade de Classes e Atributos

As classes obrigatórias e as classes opcionais do Software Profile RAS, junto de seus atributos obrigatórios, são listadas na Tabela 3.2. Algumas classes são obrigatórias apenas quando outras classes forem instanciadas. Por exemplo, EffortType é obrigatória apenas quando Effort for instanciada.

Tabela 3.2: Classes obrigatórias e opcionais com seus atributos obrigatórios no SW-RAS.

Classe Obrigatória	**Atributo Obrigatório**	**Classe Opcional**	**Atributo Obrigatório**
AssetType	name	ApplicationDomain	name
Ranking	/averageScore	ApplicationSubdomain	name
Ranking	/reuseCounter	Effort	value
Tag	name	SoftwareLicenseType	name
Tag	date	Organization	name
EffortType	type	Project	name
User	id	QualityEvaluation	date

User	name	QualityEvaluation	comment
User	e-mail	QualityEvaluation	generalScore
User	isCertifier	SoftwareProductQuality	functionalSuitability Score
Publishment	date	SoftwareProductQuality	performanceEfficiency Score
Description	text	SoftwareProductQuality	compatibilityScore
FunctionalRequirement Type	type	SoftwareProductQuality	usabilityScore
NonFunctional RequirementType	type	SoftwareProductQuality	reliabilityScore
SouceCodeType	type	SoftwareProductQuality	securityScore
TestType	type	SoftwareProductQuality	maintainabilityScore
UIAbstractionLevel	name	SoftwareProductQuality	portabilityScore
		QualityInUse	effectivenessScore
		QualityInUse	efficiencyScore
		QualityInUse	satisfactionScore
		QualityInUse	freedomFromRiskScore
		QualityInUse	contextCoverageScore
		Consumption	date
		Certification	date
		Adjustment	date
		Adjustment	changeDescription
		UserComment	comment
		UserComment	date
		DesignPattern	name
		OperationalSystem	name
		i18n	language
		ProgrammingLanguage	name

		TestMethod	name

3.6 Restrições Semânticas

As restrições semânticas do SW-RAS são listadas a seguir, com o objetivo de definir as regras que os diagramas não podem cobrir. O SW-RAS é compatível com todas as restrições semânticas herdadas do Default Component Profile e do Default Profile (vide seção 2.9). São as restrições semânticas do Software Profile RAS:

- Ativos do tipo de serviço (web service) devem possuir ao menos um artefato WSDL (seção 3.1.1).
- Ativos do tipo de interfaces de usuários devem possuir ao menos um artefato UserInterface (seção 3.1.1).
- Ativos do tipo de domínio devem conter ao menos um domínio de aplicação (ApplicationDomain) relacionado (seções 3.1.1 e 3.1.4).
- A cada avaliação de qualidade realizada deve ser atualizado o valor do atributo derivado Ranking.averageScore com a média da média de cada avaliação do ativo. Quando não houver avaliações, o atributo deve ter o valor zero (0) (seção 3.1.2).
- A cada reutilização deve ser atualizado o atributo derivado Ranking.reuseCounter com a quantidade de consumos do ativo. Quando não houver consumos, o atributo deve ter o valor zero (0) (seção 3.1.2).
- Caso o ativo possua uma licença (livre ou não), ela deve ser especificada através da classe SoftwareLicenseType. Quando não houver, a licença implicitamente é proprietária. A lei de direitos autorais faz com que todos os trabalhos tenham por padrão, quando não especificado, o direito autoral não livre (FSF, 2013) (seção 3.1.6).
- Duas tags ao menos devem estar descritas na classificação do ativo (seção 3.1.8).
- A publicação do ativo deve estar representada em uma instância de Publishment (seção 3.3.1).
- A certificação do ativo deve estar representada em uma instância de Certification (seção 3.3.1).
- Cada consumo do ativo deve estar representado em uma instância da classe Consumption (seção 3.3.1).
- Qualquer usuário (User) pode ser associado aos perfis de consumidor (Consumer), autor (Author) e publicador (Publisher). Isso torna colaborativa a forma do reúso de software (seção 3.3.1).
- Apenas usuários com perfil de certificador (User.isCertifier em `true`) poderão ser associado na certificação (Certification) (seção 3.3.1).
- Cada alteração deve ser descrita em uma instância da classe de ajuste

(Adjustment), associada a um publicador (Publisher) ou a um autor (Author) do ativo (seção 3.3.1).

3.7 Relação com as Lacunas do RAS

A Tabela 3.3 apresenta a relação entre as extensões propostas no SW-RAS e as lacunas que são solucionadas por tais extensões. As lacunas #20, #21, #22 e #23 não são abordadas nesse Profile com o objetivo de restringir o escopo do Profile.

Tabela 3.3: Relação entre as extensões no Software Profile RAS e as lacunas do RAS.

Extensões do Software Profile RAS	**Lacunas do RAS**
Classificação pela Avaliação da Qualidade	#1
Classificação pelo Tipo	#2
Classificação pela Reusabilidade	#3
Classificação por Domínios de Aplicação	#4
Classificação por Organizações e Projetos	#5
Classificação por Licenças de Software	#6
Classificação por Custos e Esforços	#7
Classificação por Social Tagging	#8
Solução com Requisitos Funcionais e Não-funcionais	#9
Solução com Padrões de Projeto	#10
Solução com Interfaces de Usuários e seus Níveis de Abstração	#11
Solução com Serviços	#2, #12
Solução com Códigos-fontes e Linguagens de Programação	#13
Solução com Artefatos e Métodos de Testes	#14
Uso com Perfis de Usuários Consumidores, Produtores e Certificadores	#15 e #16
Uso com Guias de Usuário, Descrições de Uso e Comentários de Usuários	#17 e #18
Ativos Relacionados pela Versão Anterior	#19
(Não abordadas)	#20, #21, #22 e #23

4 LAVOI, UM REPOSITÓRIO ESTRUTURADO PELO SOFTWARE PROFILE RAS

Esse capítulo apresenta o repositório de ativos reutilizáveis Lavoi, desenvolvido com a estrutura do SW-RAS, e descreve seus mecanismos de busca e de armazenamento.

O Lavoi[2] é um sistema de repositório de ativos reutilizáveis de software implementado com base nas extensões do Software Profile RAS. Proporciona mecanismos de pesquisa e de armazenamento de ativos, junto de notificações aos usuários. O Lavoi está funcional e sendo utilizado num contexto real em uma companhia pública de TI (vide Capítulo 5). Sua construção foi útil para consolidar e refinar o Profile proposto. Sua arquitetura Java web é escalável, suportando o armazenamento e a busca de ativos em larga escala. Os usuários acessam o sistema do Lavoi por uma interface web (Figura 4.1), o qual é executado em um servidor de aplicação Java, armazenando seus ativos em um sistema gerenciador de banco de dados e indexando as informações dos ativos em uma plataforma de índice de pesquisa.

É disponibilizado com licença de software livre (GPLv3), seu código-fonte está no repositório de projetos open source GitHub, em https://github.com/dsmoura/lavoi. Por ser livre, pode ser baixado, utilizado, adaptado e evoluído por qualquer pessoa ou entidade, sendo desenvolvido com tecnologias livres e atuais na plataforma Java Enterprise Edition. Foi utilizado o Framework MVC JavaServer Faces (JSF)[3] com AJAX para a construção da interface web dinâmica com o usuário. Para a persistência dos dados foi utilizada a tecnologia Java Persistence API (JPA)[4], que é compatível com diversos sistemas gerenciadores de banco de dados. O servidor de aplicação Java pode ser qualquer um compatível com JavaEE 5: Apache Tomcat[5], Jboss Application Server[6], entre outros. A plataforma de índice de pesquisa utilizada é o Apache Solr[7], baseado em Apache Lucene[8].

---

2 O nome homenageia Antoine Lavoisier, pai da química moderna, que enunciou o Princípio da Conservação da Matéria “Nada se perde, nada se cria e tudo se transforma.”, que informalmente traduz o que seria o ideal para o reúso de software.

3 http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html
4 http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html
5 http://tomcat.apache.org/
6 http://www.jboss.org/
7 http://lucene.apache.org/solr/
8 http://lucene.apache.org/core/

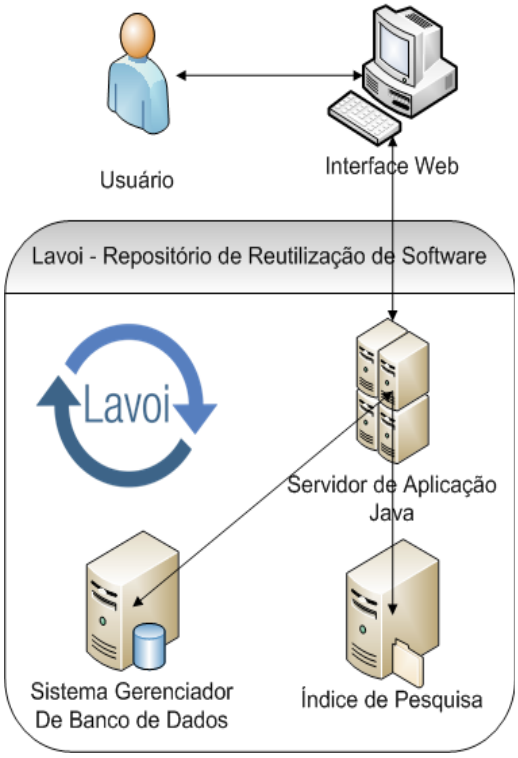


Figura 4.1: Arquitetura Web do Lavoi.

As capturas de telas do Lavoi correspondem a sua utilização descrita no Capítulo 5. A tela principal do Lavoi (Figura 4.2) exibe os itens:

- Search Assets – Pesquisar ativos (à esquerda e no menu)
- Publish Asset – Publicar ativos (à esquerda e no menu)
- My Consumed Assets – Meus ativos consumidos (à esquerda)
- My Published Assets – Meus ativos publicados (à esquerda)
- Evaluate My Consumed Assets – Avaliar meus ativos consumidos (à esquerda)
- Search in Repository – Pesquisa rápida (no topo direito)
- Repository Status – Quantidade de ativos publicados no Lavoi (no centro)
- Last Published – Últimos ativos publicados (à direita)
- Top Reused – Ativos mais reutilizados (à direita)
- Top Rated – Ativos melhor avaliados (à direita)
- Last Certified – Últimos ativos certificados (à direita)
- Assets Tag Cloud – Nuvem de tags com as cinquenta tags mais frequentes (abaixo e no menu)

Os mecanismos de pesquisa de ativos do Lavoi são quatro:

- Busca textual simples com relevância: Busca textos simples sem referenciar campos de pesquisa. É uma generalização da busca textual campo a campo com

relevância, pois busca em todos os campos com os pesos da relevância de cada atributo do ativo.

- Busca textual campo a campo com relevância: busca o texto simples em todos os campos de pesquisa com os pesos da relevância de cada atributo do ativo. A Figura 4.3 exemplifica a combinação das duas buscas textuais, a simples e a campo e a campo.

- Nuvem de tags: expõe lado a lado o tamanho das tags de um modo proporcional à frequência de cada tag no repositório (vide seção 3.1.8), gerando uma visualização do repositório baseada na relevância da frequência das tags (Figura 4.4).

- Navegação por tags: é utilizada quando se abre o sumário do ativo e clica-se na sua tag, então é acionado o mecanismo de busca campo a campo buscando os ativos que contêm a tag selecionada. Os links de tags são ilustrados na Figura 4.7, na categoria de classificação (Classification).

A relevância de cada ativo é leva em conta quatro itens

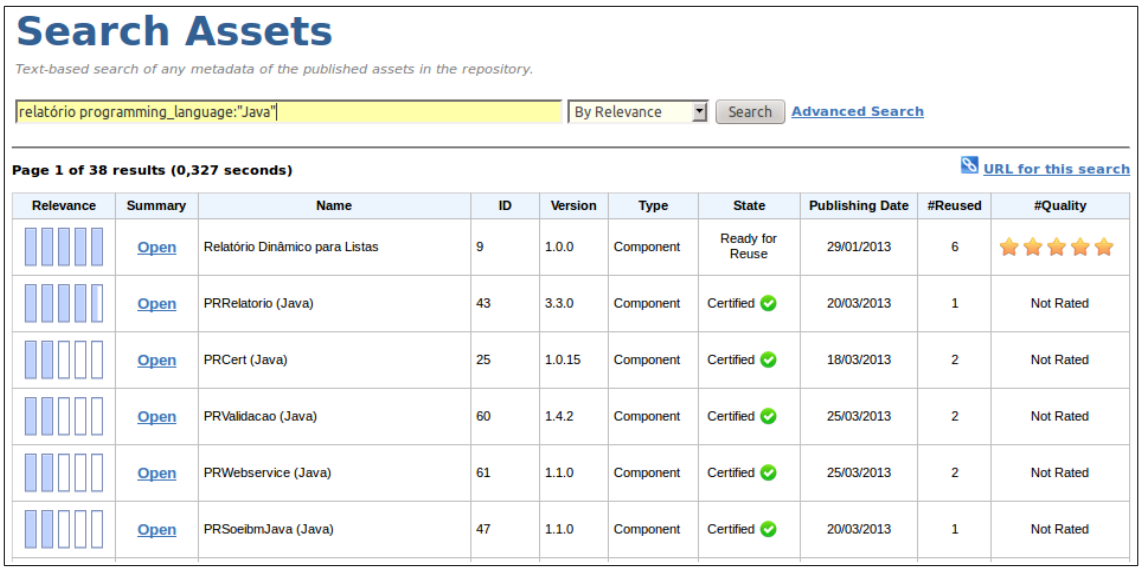
- A média de qualidade das avaliações de usuários, obtida pelo atributo Asset.Classification.Ranking.averageScore (vide subseção 3.1.2).

- A reusabilidade (quantia de consumos), obtida pelo atributo Asset.Classification.Ranking.reuseCounter (vide subseção 3.1.3)

- O estado, obtido no atributo Asset.state (vide subseção 2.9.1 e Figura 3.29).

- O peso de relevância semântica que cada atributo pesquisado tem no ativo. Quando o texto é encontrado dentro do atributo, o resultado é multiplicado pelo seu peso respectivo. Os pesos dos campos mais importantes são os: ID (peso 20), nome (peso 15), tag (peso 5), linguagem de programação (peso 3), domínio e subdomínio de aplicação (peso 3), tipo de ativo (peso 2), ID e nome de ativo relacionado (peso 2).

Figura 4.2: Tela principal do Lavoi.



Relevance	Summary	Name	ID	Version	Type	State	Publishing Date	#Reused	#Quality
	Open	Relatório Dinâmico para Listas	9	1.0.0	Component	Ready for Reuse	29/01/2013	6	★★★★★
	Open	PRRelatorio (Java)	43	3.3.0	Component	Certified	20/03/2013	1	Not Rated
	Open	PRCert (Java)	25	1.0.15	Component	Certified	18/03/2013	2	Not Rated
	Open	PRValidacao (Java)	60	1.4.2	Component	Certified	25/03/2013	2	Not Rated
	Open	PRWebservice (Java)	61	1.1.0	Component	Certified	25/03/2013	2	Not Rated
	Open	PRSoeibmJava (Java)	47	1.1.0	Component	Certified	20/03/2013	1	Not Rated

Figura 4.3: Search Assets – Tela de busca textual simples (palavra “relatório”) junto da busca textual campo a campo (campo de linguagem de programação java) do Lavoi.

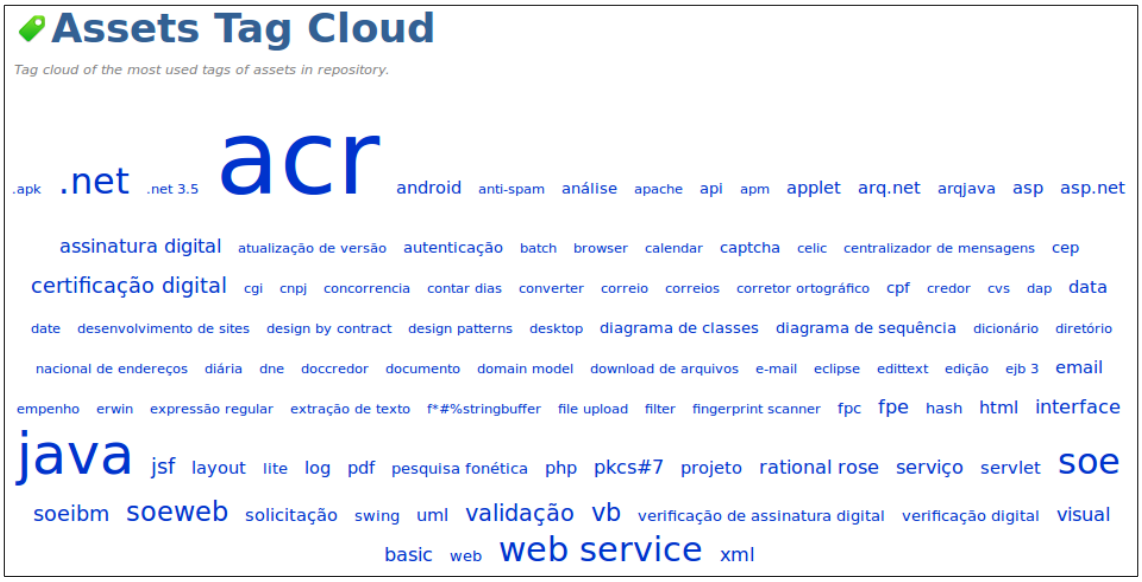


Figura 4.4: Assets Tag Cloud – Nuvem de tags dos ativos do Lavoi.

O mecanismo de armazenamento orienta o cadastro e a edição do ativo através de um wizard (Figura 4.5), cada passo representa uma seção da estrutura do SW-RAS. Os passos são os de:

- Descrição geral (Description).
- Classificação (Classification).
- Esforços e custos (Effort).
- Análise de requisitos (Requirements e Analysis).
- Projeto (Design).
- Implementação (Implementation).
- Teste (Test).
- Uso (Usage).
- Ativos relacionados (Related Assets). São sugeridos os ativos relacionados (já publicados) com relevância baseada nos campos já preenchidos no cadastro (Figura 4.6)
- Publicação (Publish and Final).

Os arquivos dos diversos tipos de artefatos podem ser carregados através de uma opção de upload do Lavoi, além de poderem ser referenciados pelo local onde se encontram através da URI. Ao efetivar a publicação, o Lavoi notifica a existência de uma nova versão aos consumidores das versões anteriores do ativo, caso existam. O ativo recém-publicado aparecerá numa lista de ativos a serem revisados e certificados.

Ao consultar-se um ativo, o sumário é apresentado com todo o conteúdo de informações e de artefatos empacotados (Figura 4.7). Os dados são apresentados na sequência do wizard de cadastro. O botão de download baixa o ativo empacotado em um arquivo compactado de extensão “.zip”, contendo um arquivo XML das informações

e todos os arquivos dos artefatos, de acordo com a especificação de empacotamento do RAS.

Os arquivos que foram carregados por upload são empacotados dentro do diretório específico da categoria do ativo. Na Figura 4.8 é exibido um exemplo de conteúdo de um ativo simples de código Java. O arquivo compactado possui o nome do ativo (CpfEditText Android), contendo o XML das informações sobre o ativo (`rasset.xml`) e um código-fonte no diretório de código-fonte da implementação da solução (`\Solution\Implementation\Source Code\`). Um exemplo de um arquivo `rasset.xml` é exibido na Figura 4.9.

As avaliações de qualidade no uso e de qualidade do produto descritas na seção 3.1.2 são suportadas pelo Lavoi (Figura 4.10). Ao serem avaliadas as características de qualidade da ISO/IEC 25010, o Lavoi já vai contabilizando e informando a média em porcentagem da avaliação. Subcaracterísticas são exibidas ao lado de cada característica para o usuário ter um melhor entendimento do que está sendo avaliado.

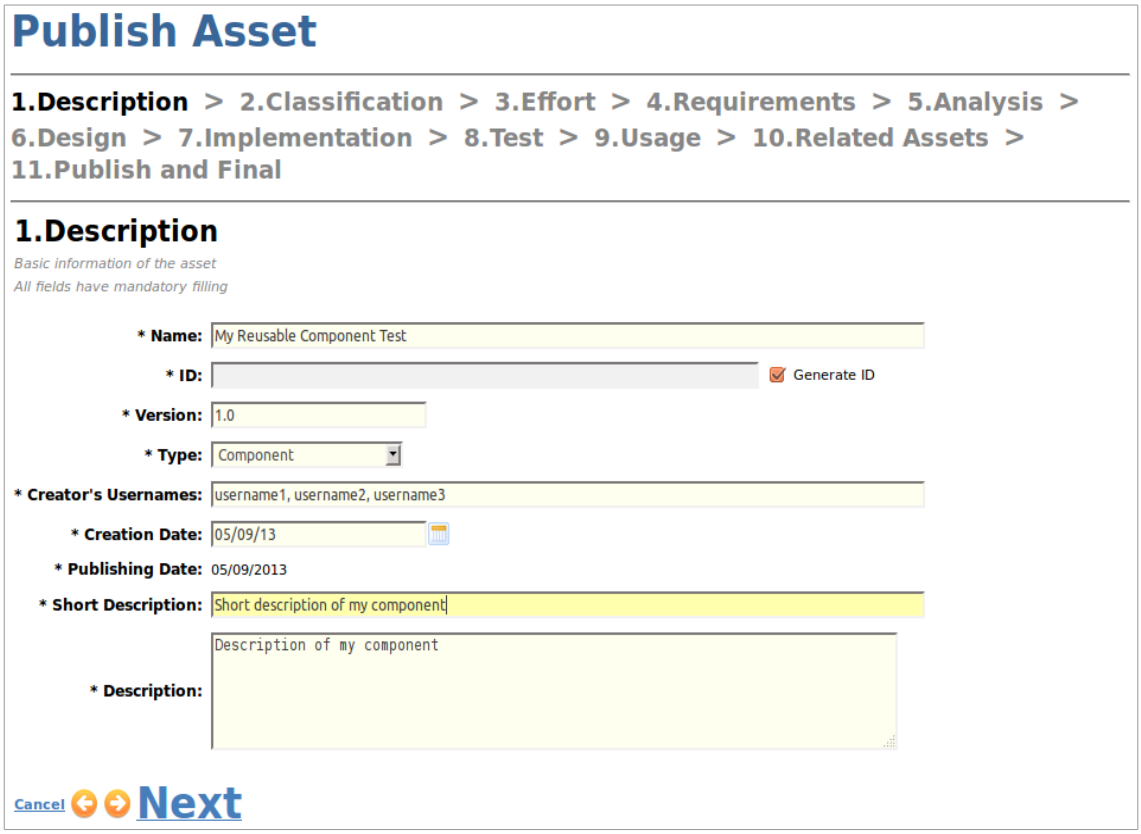


Figura 4.5: Publish Asset – Tela do wizard de cadastro dos ativos, no primeiro passo para as descrições gerais do ativo.

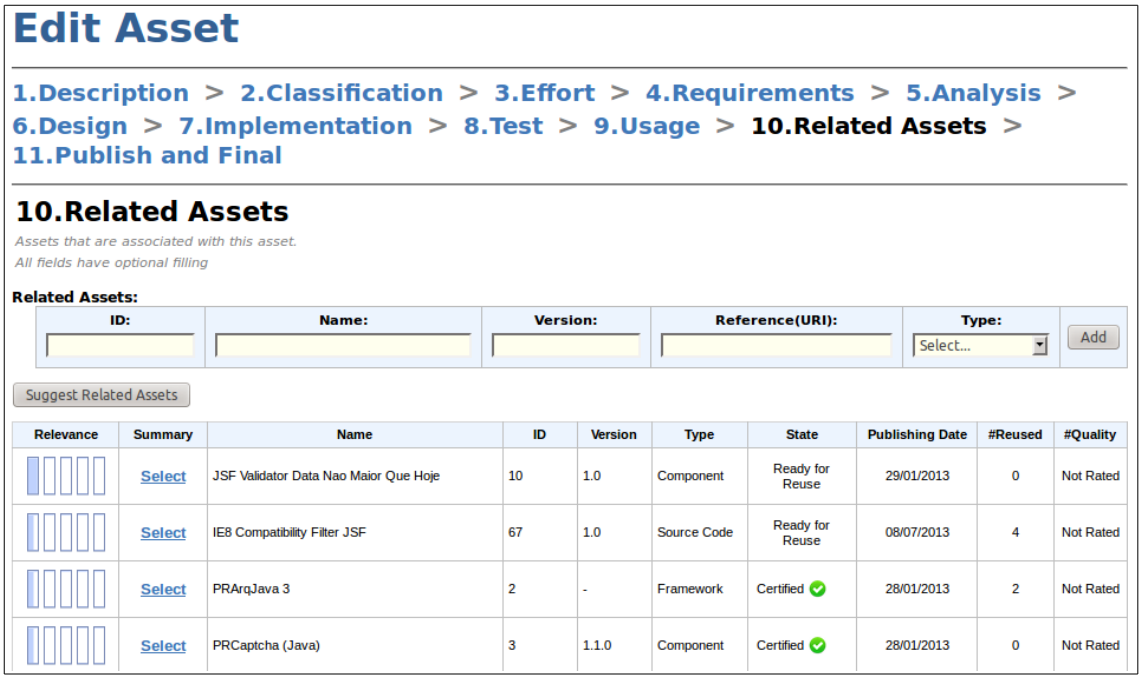


Figura 4.6: Edit Asset – Passo do wizard de edição de ativos para os ativos relacionados, com sugestão automática de ativos utilizando o mecanismo de busca do repositório.

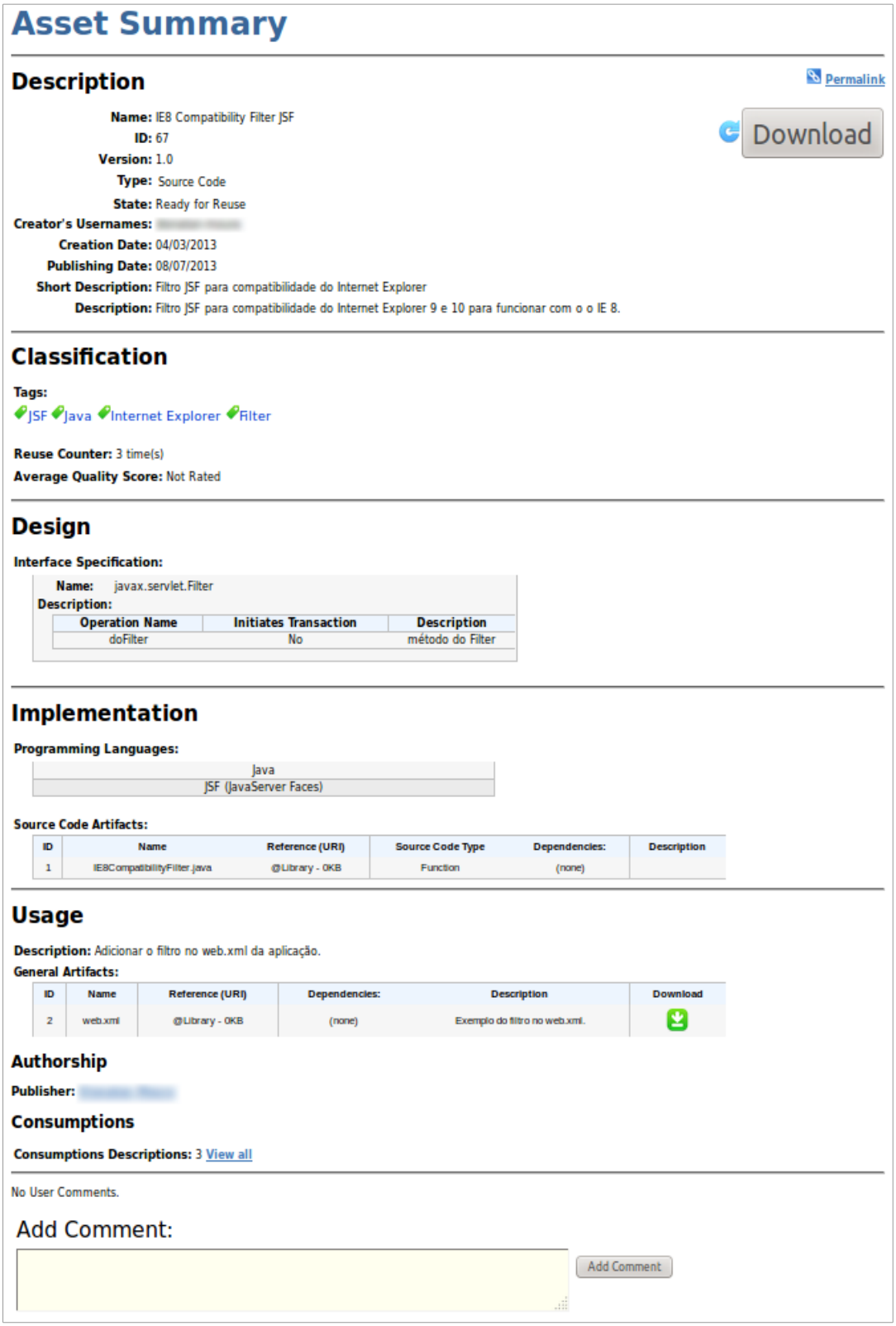


Figura 4.7: Asset Summary – Exemplo do sumário das informações e dos artefatos de um ativo no Lavoi.

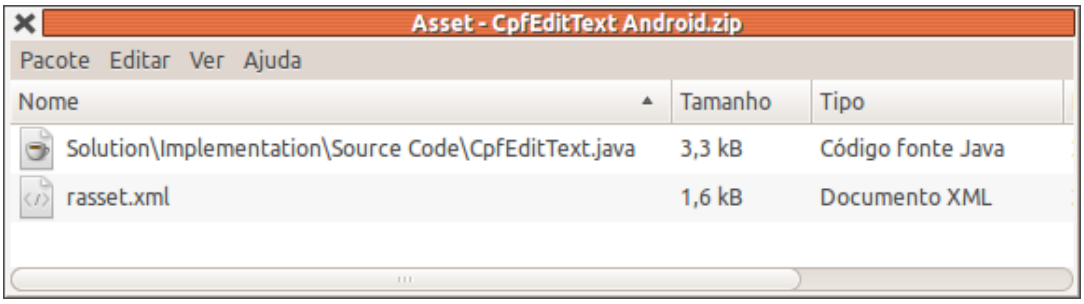


Figura 4.8: Exemplo do conteúdo de um ativo empacotado.

```
<?xml version="1.0" encoding="UTF-8"?>
<Asset name="CpfEditText Android" id="14" state="Ready for Reuse"
       version="1.0" date="[redacted]" access_rights="------R-X"
       short_description="Campo CPF para Android">
  <description>Campo CPF para Android, aceitando apenas dígitos numéricos.</description>
  <classification>
    <type name="Component" />
    <tag name="Android" />
    <tag name="CPF" />
  </classification>
  <solution>
    <effort />
    <requirements />
    <design />
    <implementation>
      <programming_language name="Android" />
      <programming_language name="Java" />
      <source_code id="1" name="CpfEditText.java" source_code_type="Object"
                   reference="Solution\Implementation\Source Code\CpfEditText.java">
        <description>EditText de CPF.</description>
      </source_code>
    </implementation>
    <test />
  </solution>
  <usage>
    <description>[...]</description>
    <publishment date="[redacted]">
      <user id="4" name="[redacted]" email="[redacted]" />
    </publishment>
    <consumption date="[redacted]">
      <user id="11" name="[redacted]" email="[redacted]" />
    </consumption>
  </usage>
  <related_assets>
    <related_asset id="13" name="CepEditText Android"
                   version="1.0" reference="@Library" relationshipType="Similar" />
  </related_assets>
</Asset>
```

Figura 4.9: Exemplo do arquivo XML para a descrição das informações de um ativo empacotado.

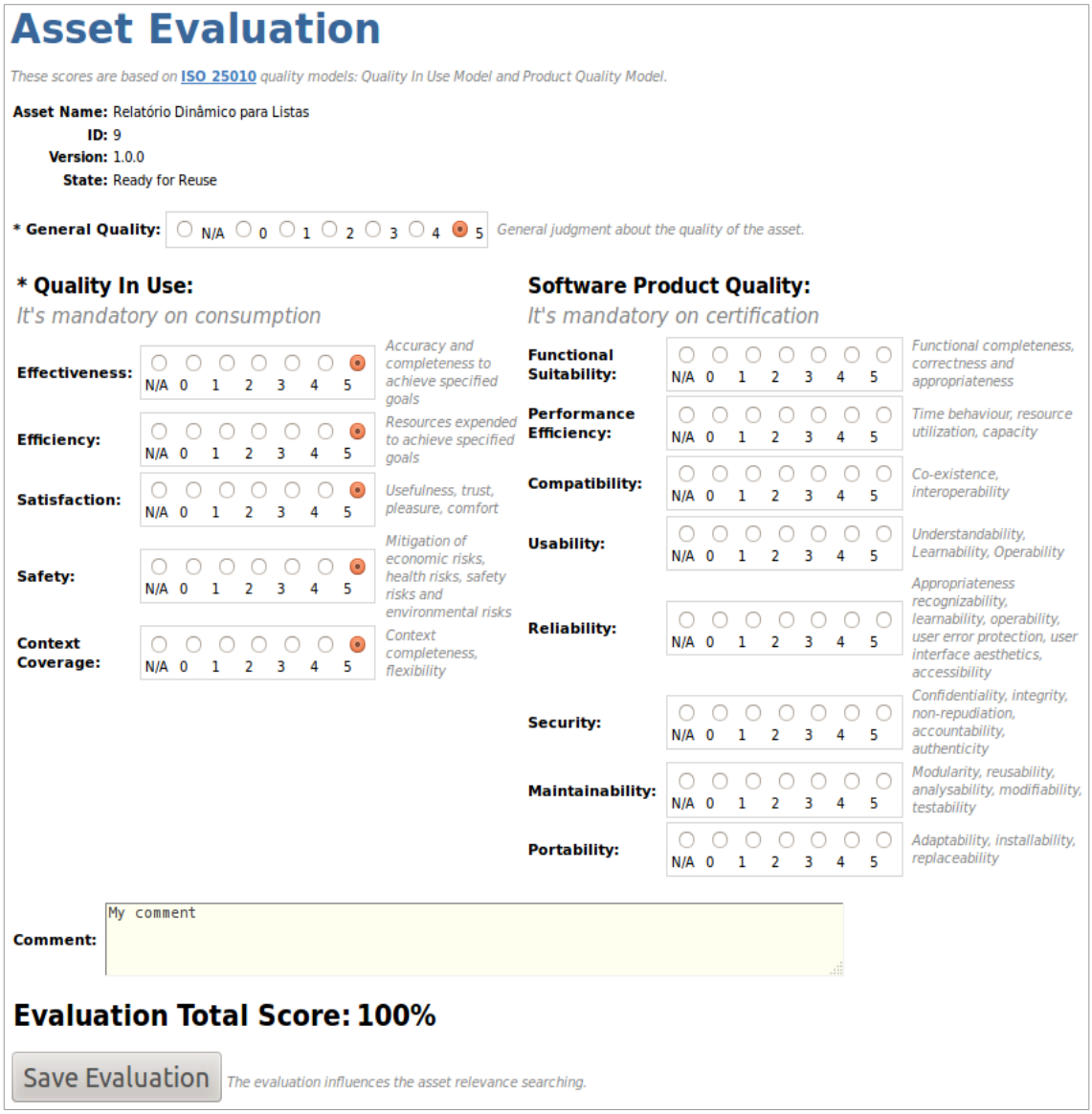


Figura 4.10: Asset Evaluation – Tela de avaliação da qualidade dos ativos, com os atributos da ISO/IEC 25010.

5 AVALIAÇÃO

A avaliação das extensões propostas do Software Profile RAS é realizada de três formas. Primeiramente, na seção 5.1, é realizada uma comparação entre o SW-RAS e os outros trabalhos anteriores de extensões do RAS da literatura. A seguir, na seção 5.2, é descrita a experimentação por um estudo de caso com o uso das extensões do SW-RAS através da utilização do Lavoi, ocorrendo num ambiente real de desenvolvimento e reúso de software. Logo após, na seção 5.3, é apresentada uma pesquisa de campo realizada com os usuários do Lavoi.

5.1 Comparação do Software Profile com Extensões Anteriores

Um comparativo entre as extensões do Software Profile RAS e as extensões dos trabalhos anteriores propostos na literatura é encontrado na Tabela 5.1, listando as informações e os artefatos relacionados a cada extensão, junto das respectivas lacunas do RAS. Os trabalhos anteriores de extensão do RAS são encontrados na seção 2.10. As lacunas do RAS são as descritas no Capítulo 3. Nessa tabela, o hífen "-" representa que não foi identificada uma lacuna ao menos relevante para a extensão proposta.

O Software Profile é o único trabalho que estende todas as categorias do ativo, ou seja, a classificação, a solução, o uso e os ativos relacionados, abordando a maioria das extensões para as lacunas identificadas do RAS. Essa comparação exibe que o SW-RAS traz diversas novas extensões ao contexto do RAS, representados com um sinal de adição "+" na Tabela 5.1. O SW-RAS propõe como diferencial:

- Classificação pelo tipo do ativo (subseção 3.1.1).
- Classificação com relevância pela qualidade avaliada pelos usuários (subseção 3.1.2)
- Classificação com relevância pela reusabilidade (subseção 3.1.3)
- Classificação por custos e esforços de desenvolvimento (subseção 3.1.7)
- Classificação por social tagging (tags) (subseção 3.1.8)
- Solução com artefatos de interfaces de usuários (subseção 3.2.3)
- Solução com artefatos de serviços (web services) (subseção 3.2.4)
- Solução com artefatos de testes e métodos aplicados aos ativos (subseção 3.2.6)
- Uso com perfis de usuários publicadores, autores e consumidores (subseção

3.3.1)

- Comentários de usuários, para a comunicação de dúvidas, sugestões e defeitos pelos usuários (subseção 3.3.2)
- Uso com artefatos de guias de usuários (subseção 3.3.2)
- Uso com descrições de uso dos ativos (subseção 3.3.2)
- Uso com descrições de ajustes realizados nos ativos pelos produtores (subseção 3.3.1)
- Ativos relacionados pela versão anterior (subseção 3.4.1)

As lacunas do RAS identificadas que não são solucionadas pelo Software Profile RAS são as relacionadas:

- À arquitetura e ao modelo de domínio (lacuna #20).
- Ao grau de acessibilidade do ativo relacionado ao nível de experiência do usuário (lacuna #21).
- À automatização das tarefas de deploy de artefatos e de execução dos ativos (lacuna #23).

Tabela 5.1: Comparação entre as lacunas do RAS e as extensões do SW-RAS e dos outros trabalhos de extensões no RAS.

Extensões Propostas	Lacuna do RAS	SW-RAS	Domain Arch.	MPC	X-ARM	AMT	User Context	OpenCom	Reuse Assistants
Análise	-	*		x		x			
Arquitetura	-	**		x		x		x	
Arquitetura de Domínio	#20		x						
Artefatos de Teste	#14	x				*4			
Atributos de Qualidade	#1	x		x				x	
Avaliação de Qualidade por Usuários	#1	+							
Bugs / Relato de Defeitos	#18	+							
Casos de Teste	#14	***				x			x
Certificação	#16	x			x		x		
Códigos-Fontes	#13	x							x
Comentários dos Usuários	#18	+							
Descrições de Ajustes	#15	+							
Deploy dos Artefatos	#23								x
Domínios de Aplicação	#4	x				x	x		
Esforços e Custos de Desenvolvimento	#7	+							
Execução do Ativo	#23								x
Grau de Acessibilidade	#21						x		
Guias de Usuário / Descrição do Uso	#17	+							
Interfaces de Usuários	#11	+							
Internacionalização	#9	x				x			
Licenças de Software	#6	x						x	
Linguagens de Programação	#13	x				x			
Métodos de Teste	#14	+							
Modelos de Domínio	#20		x						
Modelos de Dados	-	****						x	
Nível de Experiência do Usuário	#21						x		
Organizações e Projetos / Comunidades	#5	x						x	
Padrões de Projeto	#10	x				x			

Perfil de Publicador	#15	+							
Perfil de Usuário Autor	#15	+							
Perfil de Usuário Certificador	#15	x			x				
Perfil de Usuário Consumidor	#15	+							
Ponto de Variabilidade para Artefatos	-			x				x	
Relacionamento de Versão Anterior	#19	+							
Relevância por Qualidade	#1	+							
Relevância por Reusabilidade	#3	+							
Requisitos Funcionais	#9	x		x		x			
Requisitos Não-funcionais	#9	x					x		
Restrições Econômicas	-						x		
Serviços (web services)	#12	+							
Sistemas Operacionais	#9	x				x			
Tags (ou Palavras-chaves)	#8	+							
Tipos de Artefatos de Teste	#14	+							
Tipos de Ativos	#2	+							

+ Informação ou artefato exclusivo no Software Profile RAS.

* A análise de requisitos pode ser descrita na classe de requisitos do próprio Default Component Profile (Asset.Solution.Requirements).

** A arquitetura pode ser descrita na classe de projeto do próprio Default Component Profile (Asset.Solution.Design).

*** Os casos de testes são agrupados no tipo de artefatos de teste do Software Profile (Asset.Solution.Test.TestArtifact.Type).

**** Os modelos de dados podem ser mapeados pela classe de modelo do Default Component Profile (Asset.Solution.Requirements.Model ou Asset.Solution.Design.Model).

5.2 Utilização do Lavoi em um Ambiente Real

Com o objetivo de avaliar empiricamente o uso das extensões do Software Profile RAS, o repositório de ativos reutilizáveis Lavoi está sendo utilizado no ambiente de desenvolvimento de sistemas de uma grande companhia pública de TI. Essa companhia desenvolve e mantém os sistemas de software de diversos órgãos estaduais do Rio Grande do Sul, possuindo aproximadamente quinhentas e cinquenta pessoas relacionadas diretamente ao desenvolvimento de software. Ao longo dos anos, na forma de Desenvolvimento Para Reutilização, a divisão de tecnologia da organização desenvolveu diversos ativos reutilizáveis para os setores de desenvolvimento de software criarem sistemas e websites com base nesses ativos, na forma de componentes, frameworks, APIs reutilizáveis e web services.

No cenário anterior ao Lavoi, os ativos eram cadastrados em uma página na intranet e seus artefatos armazenados em um diretório específico num servidor de arquivos. A publicação era de forma ad hoc, sem informações do uso dos ativos, tendo uma padronização pobre de descrição. Não havia um meio de pesquisar ativos, nem gerenciá-los. Inexistia o histórico de consumo, nem um mecanismo de comunicação entre os produtores e consumidores dos ativos. Também não havia uma forma de contribuição colaborativa entre os engenheiros de software, pelo fato de que apenas a divisão de tecnologia podia cadastrar tais ativos.

No contexto atual, o repositório Lavoi está no ambiente de produção com acesso a toda organização. Os ativos reutilizáveis, anteriormente dispostos de uma forma ad hoc, foram migrados para o Lavoi, sendo refinados e publicados. Muitos desses ativos já são homologados e certificados por terem sido desenvolvidos e reutilizados ao longo dos anos por diversos sistemas em produção. Alguns ativos colaborativos já estão sendo

publicados pelos engenheiros de software interessados. Para incentivar e fortalecer a cultura de reúso de software, um programa está sendo desenvolvido, incluindo o treinamento em reúso de software e sobre o Lavoi. O repositório tornou-se também o catálogo de web services reutilizáveis. Além disso, será o catálogo dos serviços disponibilizados pela Arquitetura Orientada a Serviços (SOA), fazendo parte de um projeto estratégico que está em fase de prospecção e implantação.

5.2.1 Infraestrutura Utilizada

Pelo fato de ter sido desenvolvido com tecnologias livres, a infraestrutura para a implantação do Lavoi teve custo zero em licenças de software. O servidor de aplicação utilizado é o JBoss Application Server[9], software livre, compatível com a arquitetura web do Java Enterprise Edition, possuindo duas instâncias de servidor de produção balanceadas pelo aplicativo mod_cluster da Jboss Community[10]. O sistema gerenciador de banco de dados relacional utilizado é o PostgreSQL[11], também software livre. É acessado com acesso seguro por HTTPS, e a autenticação de usuários é integrada ao LDAP da organização, ou seja, todos os usuários podem fazer login com seus usuários e senhas já existentes.

5.2.2 Refinamentos Realizados

Algumas necessidades de refinamentos foram levantadas para o SW-RAS e o Lavoi durante a implantação e o uso. O primeiro refinamento para a implantação foi em tornar opcional a associação de organizações e projetos ao desenvolvimento do ativo (de multiplicidade 1..* para 0..*), visto que o desenvolvimento dos ativos é centralizado, e não distribuído entre organizações. O segundo refinamento foi de tornar a licença de software também opcional (de multiplicidade 1 para 0..1), visto que os ativos são desenvolvidos pela organização para ela própria reutilizar.

Durante o início da utilização do Lavoi, foi necessário adicionar o perfil para autores no SW-RAS. Desse modo, os publicadores e os autores têm permissão de editar seus ativos. Antes disso, apenas quem publicou o ativo poderia editá-lo. Além disso, foi necessário adicionar artefatos de WSDL para a migração do cadastro de web services e para o projeto de SOA.

5.2.3 Análise dos Dados

Com propósito de avaliação e investigação empírica desse trabalho, a estratégia de experimentação é a de estudo de caso (WOHLIN et al., 2012). O objetivo é investigar as evidências de uso das extensões propostas no Software Profile RAS através da livre utilização do repositório de ativos reutilizáveis Lavoi no contexto citado por um período de seis meses. O estudo de caso foi planejado para que todos os ativos reutilizáveis já desenvolvidos pela empresa fossem cadastrados com a estrutura do SW-RAS no Lavoi, obtendo-se dados sobre sua livre utilização. Os dados do SW-RAS foram coletados da base de dados do Lavoi e a análise é realizada para cada extensão proposta. Ao final, é realizada uma discussão sobre as evidências de uso das extensões.

9 https://www.jboss.org/jbossas/
10 http://www.jboss.org/mod_cluster
11 http://www.postgresql.org/

5.2.3.1 Análise dos Dados dos Ativos

A quantidade de ativos reutilizáveis de software publicados no Lavoi é de setenta e três (73). Dentre todos ativos, cinquenta e três (53) são certificados, dezenove (19) ativos estão prontos para reúso e apenas um está no estado em definição, ilustrados na Figura 5.1.

Figura 5.1: Estados dos ativos publicados no Lavoi.

5.2.3.2 Análise dos Dados da Classificação

Os dados relacionados às extensões da classificação do SW-RAS, descritas na seção 3.1, são apresentados a seguir.

5.2.3.2.1 Dados de Tipos de Ativos

Os tipos que classificam os ativos, exibidos na Figura 5.2, são em sua maioria componentes, com quarenta e nove (49) ao total. Em menores quantidades, seis ativos são de serviços (web services), seis ativos de códigos-fontes, três de frameworks, dois de padrão de caso de uso e um de padrão de domínio. Dentre os seis outros tipos de ativos não previstos em uma lista pré-definida estão: dois modelos de especificação de projeto, dois templates de interface web, um de regras de negócio e outro de template (Figura 5.3). As informações sobre essa extensão se encontram na subseção 3.1.1.

Figura 5.2: Tipos dos ativos publicados.

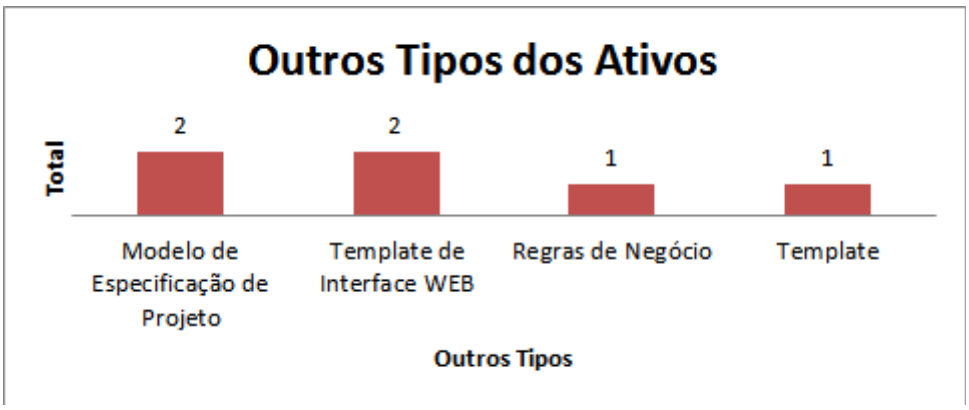


Figura 5.3: Outros tipos de ativos (Other) publicados manualmente descritos pelos publicadores.

5.2.3.2.2 Dados de Avaliações da Qualidade

Dois ativos ao todo foram avaliados e classificados por suas qualidades, cujos identificadores (ID) são 9 e 55. O ativo de ID 9 teve sua média de avaliação em 100%, enquanto o ativo de ID 55 em 33,33% (Figura 5.4). Ao total, foram realizadas três avaliações da qualidade no uso e apenas uma avaliação da qualidade do produto dentre esses ativos. Como avaliação geral, a pontuação foi a máxima (5 pontos) para esses dois ativos (Figura 5.5).

Das três avaliações da qualidade no uso, duas avaliações foram feitas para o ativo de ID 9 e uma para o de ID 55 (Figura 5.6). A pontuação do ativo de ID 9 foi alta, enquanto para o ativo de ID 55 foi baixa. Na única avaliação da qualidade do produto, a pontuação do ativo de ID 55 foi alta em todas suas características (Figura 5.7). As informações sobre essa extensão se encontram na subseção 3.1.2.

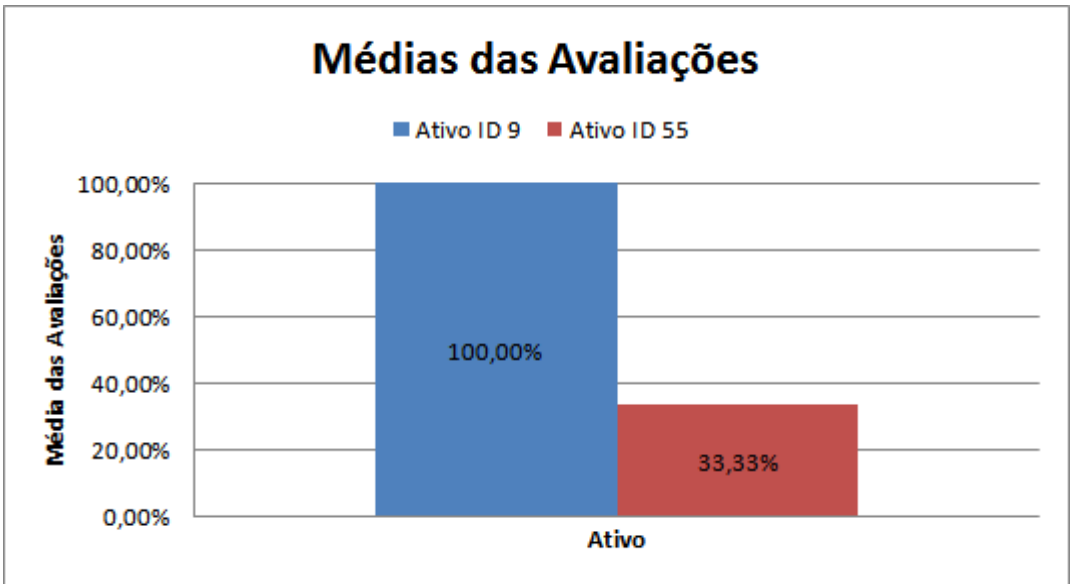


Figura 5.4: Média das avaliações realizadas para cada ativo.

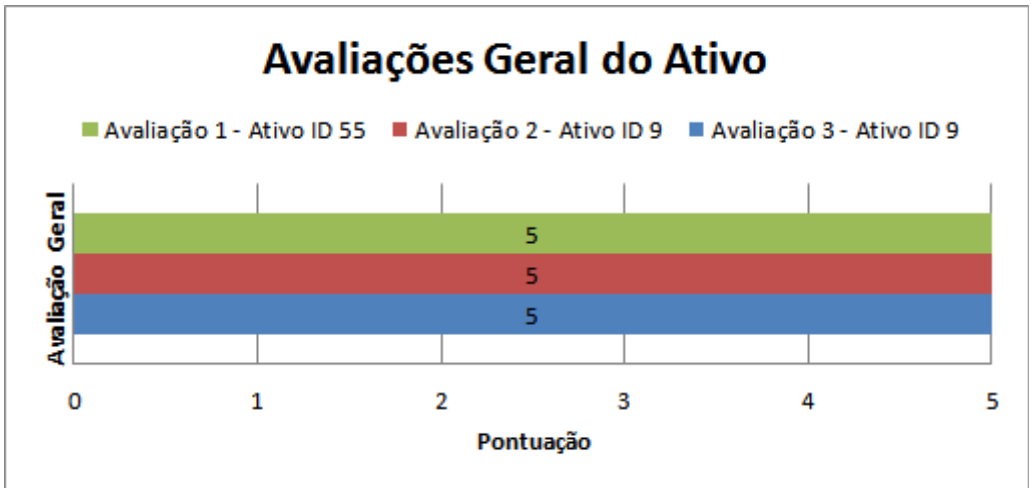


Figura 5.5: Avaliação da pontuação geral dos ativos.

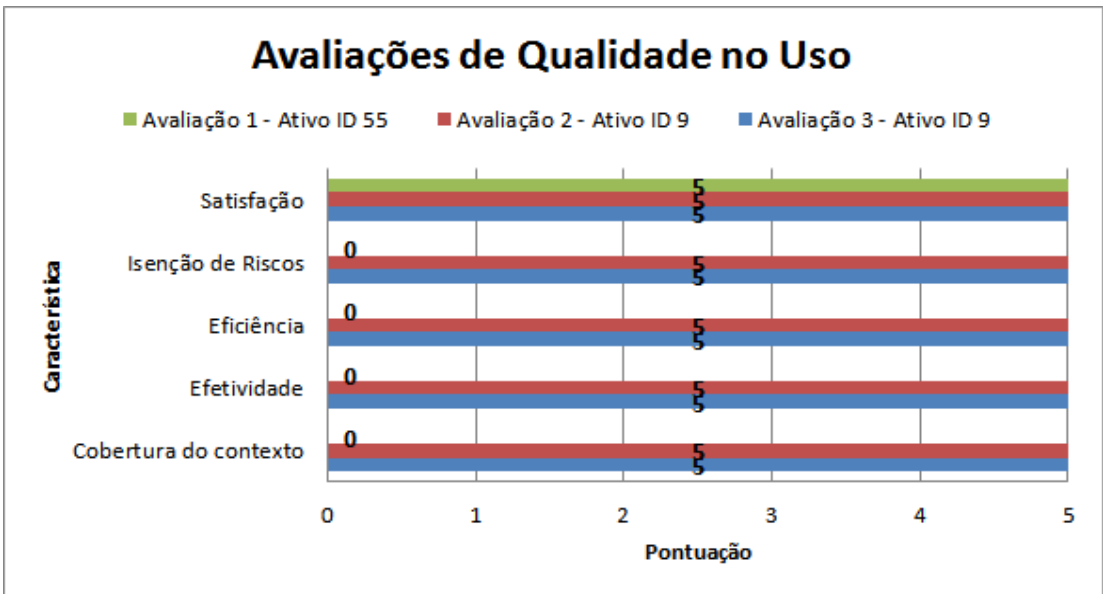


Figura 5.6: Avaliações realizadas da qualidade no uso dos ativos.

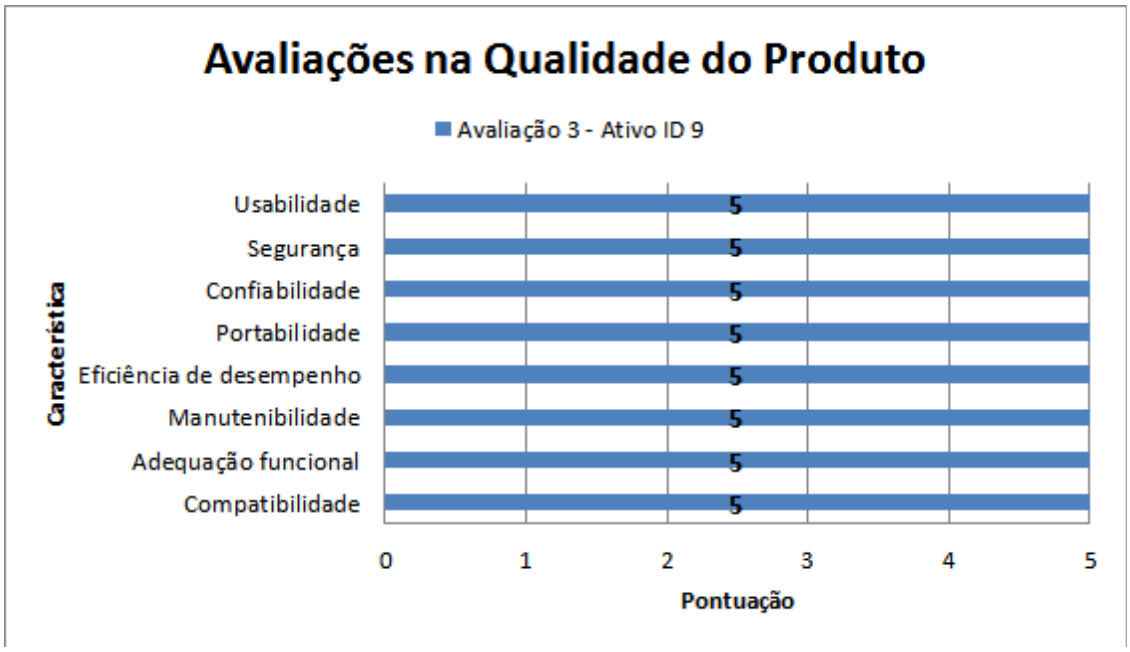


Figura 5.7: Avaliações realizadas da qualidade do produto dos ativos.

5.2.3.2.3 Dados de Reusabilidade

O total de ativos classificados pela sua reusabilidade é de vinte e oito (28), os quais foram consumidos ao menos uma vez. A Figura 5.8 exibe o histograma da frequência da reusabilidade dos ativos: um ativo foi consumido seis vezes, outros dois foram consumidos quatro vezes, nove ativos foram consumidos duas vezes, e dezesseis (16) ativos foram consumidos apenas uma vez. As informações sobre essa extensão se encontram na subseção 3.1.3.

Figura 5.8: Histograma da frequência de consumos por ativo.

5.2.3.2.4 Dados de Domínios de Aplicação

Seis dos ativos foram classificados entre três domínios e subdomínios de aplicação. Quatro desses ativos pertencem ao domínio Financeiro e ao subdomínio de Finanças Públicas, um desses ativos pertence ao domínio de Governo e subdomínio de Expedientes e Processos, e o outro desses ativos pertence ao domínio de Segurança pública e subdomínio Penitenciário (Figura 5.9). As informações sobre essa extensão se encontram na subseção 3.1.4.

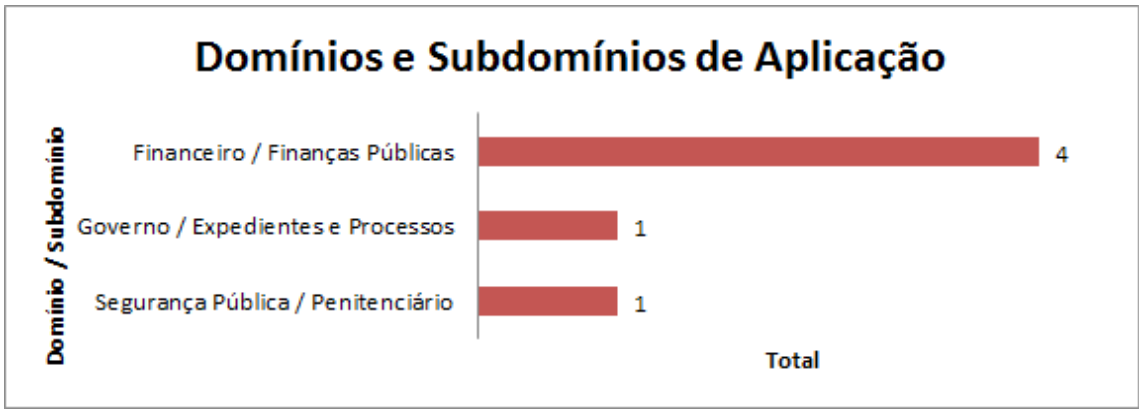


Figura 5.9: Domínios e subdomínios de aplicação dos ativos publicados no Lavoi.

5.2.3.2.5 Dados de Organizações e Projetos

Nenhuma organização ou projeto foi classificado nos ativos. Isso ocorre pela razão de que o desenvolvimento é centralizado na empresa, não sendo distribuído entre outras organizações ou projetos. As informações sobre essa extensão se encontram na subseção 3.1.5.

5.2.3.2.6 Dados de Licenças de Software

Nenhuma licença de software foi atribuída à classificação dos ativos. Isso ocorre pelo fato de que são desenvolvidos para a própria organização, tendo suas licenças proprietárias por padrão. As informações sobre essa extensão se encontram na subseção 3.1.6.

5.2.3.2.7 Dados de Esforços e Custos

Foram seis os ativos que foram classificados por seus esforços e custos de desenvolvimento. Cinco desses ativos foram desenvolvidos por apenas um desenvolvedor, e somente um ativo teve seis desenvolvedores, tendo uma média de 1,83 desenvolvedores por ativo (Figura 5.10). Para os tempos de desenvolvimento estimados e reais dos ativos, a média de tempo estimado é de 6,5 horas por ativo e a média do tempo real de desenvolvimento é de 6,6 horas por ativo (Figura 5.11). Em relação à quantidade de linhas de código, apenas dois ativos foram classificados com 94 e 20.000 linhas de código (Figura 5.12). Nenhum ativo teve o seu valor monetário informado, pelo fato de que a organização não contabiliza monetariamente o desenvolvimento. As informações sobre essa extensão se encontram na subseção 3.1.7.

Figura 5.10: Total dos esforços de desenvolvedores classificados nos ativos.

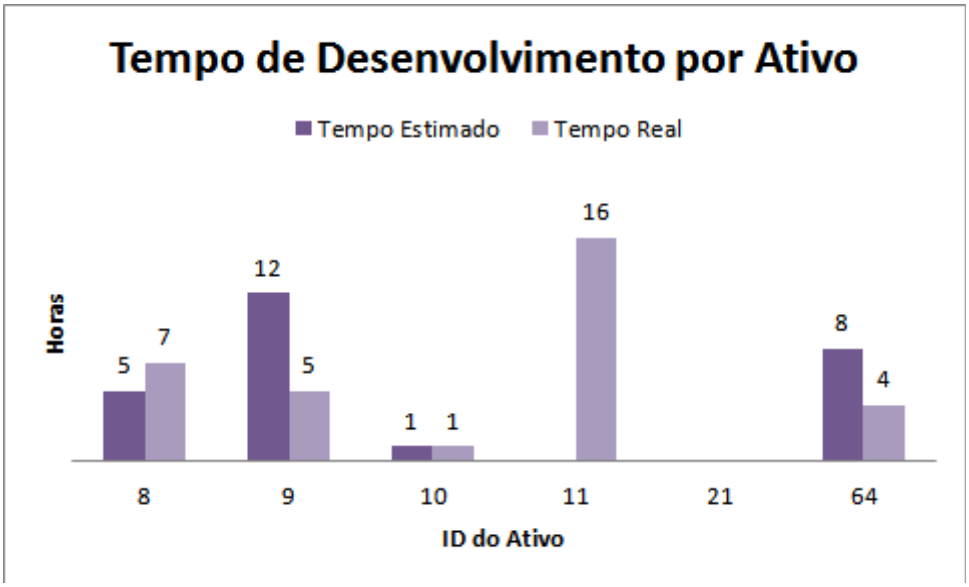


Figura 5.11: Custos dos tempos de desenvolvimento classificados nos ativos.

Figura 5.12: Total de linhas de código classificados nos ativos.

5.2.3.2.8 Dados de Social Tagging

O total de tags descritas nos ativos foi de quatrocentos e seis (406). Esse número contabiliza as tags repetidas ou não. O histograma da quantia de ativos por tag na Figura 5.13 mostra que nove tags classificaram cinco ou mais ativos, e quatro dessas tags classificaram mais de dez ativos. A quantidade de tags isoladas é de cento e quarenta e seis (146), ou seja, que classificam apenas um ativo. Além disso, 64% das tags foram associadas em dois ou mais ativos, o que sugere que a maioria dos ativos são inter-relacionados por ao menos uma característica representada pela tag.

A Figura 5.14 representa a nuvem de tags de todas as tags que classificam os ativos no repositório, junto da quantidade de ativos que classificam. A tag “acr” é a de maior valor, classificando cinquenta (50) ativos. Essa tag representa a sigla de um programa de desenvolvimento para reutilização denominado Administração de Componentes Reutilizáveis. As outras duas tags visivelmente relevantes são “java” e “.net”, representando as principais tecnologias envolvidas no desenvolvimento dos ativos reutilizáveis. Cada tag classificou em média 2,02 ativos, com desvio padrão de 4,11. As informações sobre essa extensão se encontram na subseção 3.1.8.

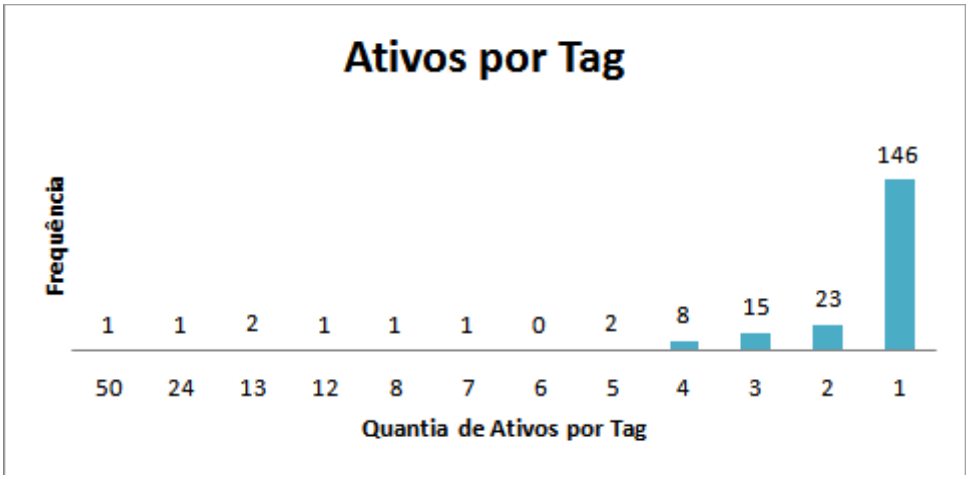


Figura 5.13: Histograma da quantia de ativos por tag.

Figura 5.14: Nuvem de tags dos ativos reutilizáveis com a quantidade ativos classificados pela tag entre parênteses.

5.2.3.3 Análise dos Dados da Solução

Os dados relacionados às extensões da solução no SW-RAS, descritas na seção 3.2, serão apresentados a seguir.

5.2.3.3.1 Dados de Requisitos Funcionais e Não-Funcionais

Não houve artefatos de requisitos funcionais ou de requisitos não-funcionais associados à solução dos ativos. As informações sobre essas extensões encontram-se na subseção 3.2.1.

5.2.3.3.2 Dados de Padrões de Projeto

São três os padrões de projeto relacionados à solução dos ativos. A Figura 5.15 exibe tais padrões de projeto: Façade (dois ativos), Builder (um ativo) e Specification (um ativo). As informações sobre essa extensão se encontram na subseção 3.2.2.

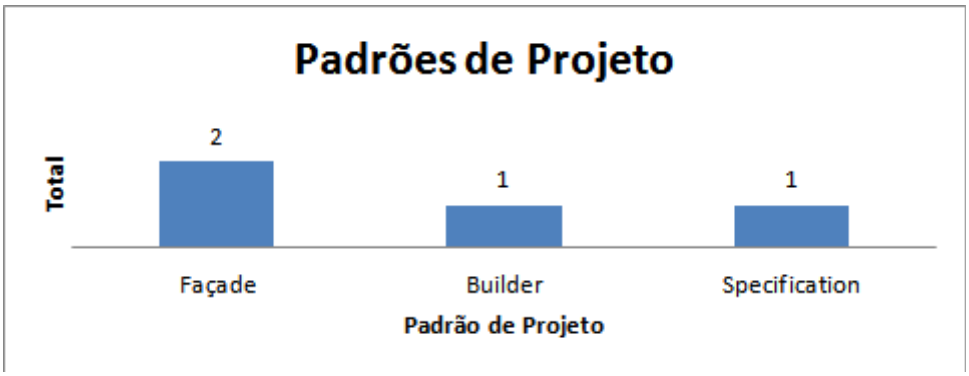


Figura 5.15: Padrões de projeto associados à solução dos ativos.

5.2.3.3.3 Dados de Interfaces de Usuários

Dez artefatos de interfaces de usuários estão contidos na solução dos ativos. Dentre esses artefatos, um é do nível de abstração de prototipação (Sketched UI), seis Final UI e três Executable (Figura 5.16). As informações sobre essa extensão se encontram na subseção 3.2.3.

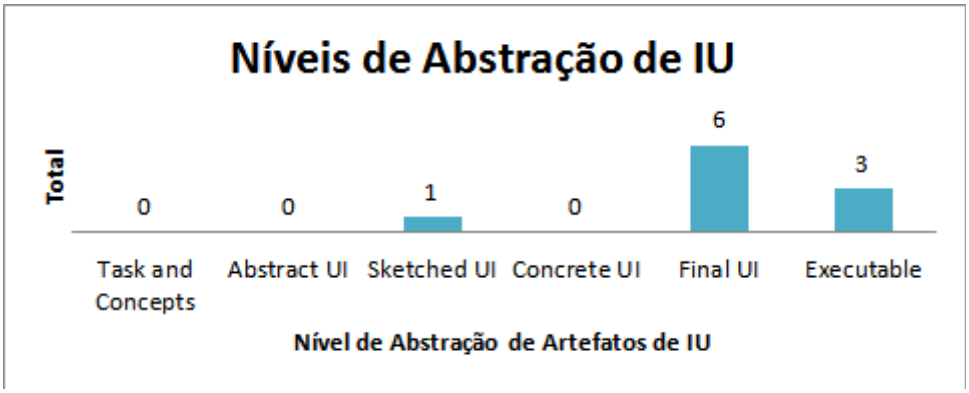


Figura 5.16: Níveis de abstração dos artefatos de interfaces de usuários.

5.2.3.3.4 Dados de Serviços

Foram atribuídos dezoito (18) artefatos de serviços (WSDL) na solução dos ativos, dentre os seis ativos do tipo de web services (Figura 5.2), com uma média de três

artefatos WSDL por ativo de web service. As informações sobre essa extensão se encontram na subseção 3.2.4.

5.2.3.3.5 Dados de Linguagens de Programação e Códigos-fontes

Ao todo são noventa e cinco (95) artefatos de código-fonte contidos na solução dos ativos. Os tipos desses artefatos de códigos-fontes são ilustrados na Figura 5.17: cinquenta e sete (57) artefatos do tipo biblioteca, dezenove pacotes, dez frameworks, nove objetos, quatro funções e um fragmento de código.

São noventa e três (93) as linguagens de programação referenciadas à implementação e ao teste da solução dos ativos publicados. As linguagens Java e .NET representam 54,84% do total. A Figura 5.18 exibe as linguagens de programação na implementação da solução dos ativos e a Figura 5.19 exibe as linguagens de programação no teste. As informações sobre essas extensões se encontram na subseção 3.2.5.

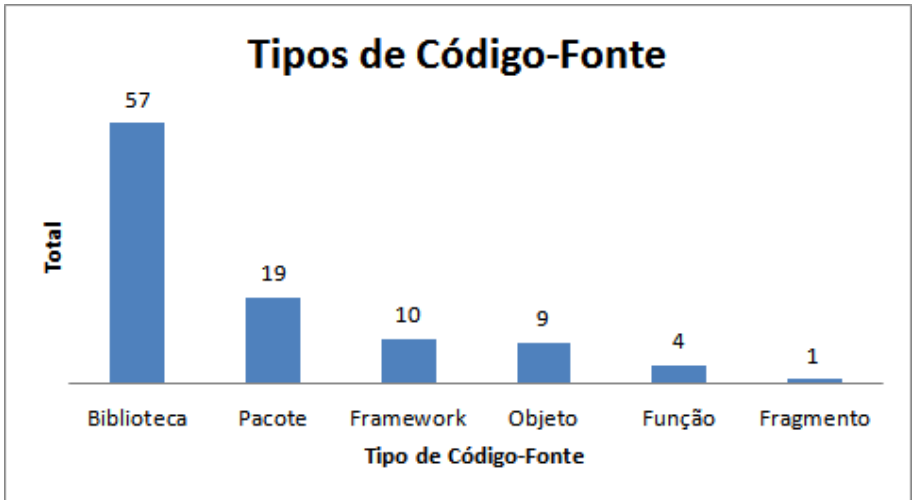


Figura 5.17: Tipos dos artefatos de código-fonte.

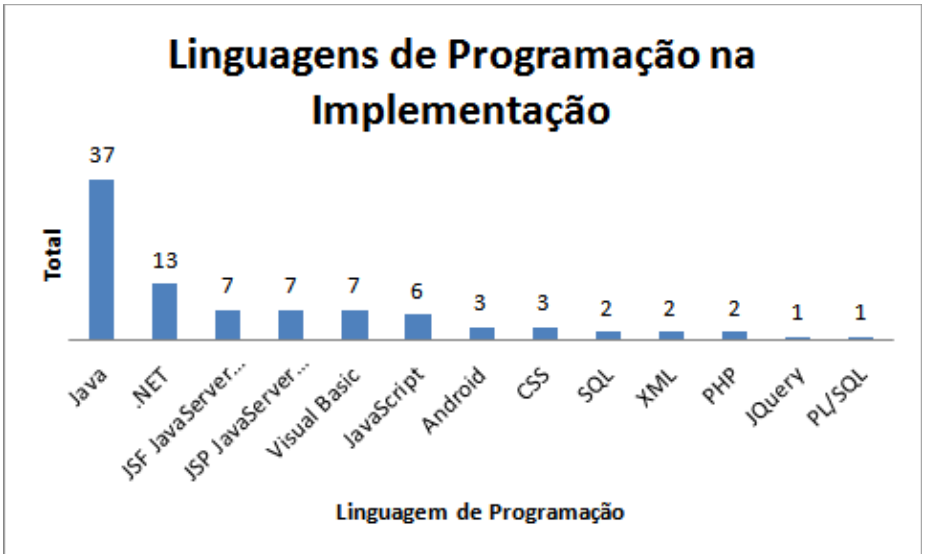


Figura 5.18: Linguagens de programação associadas à implementação da solução dos ativos.

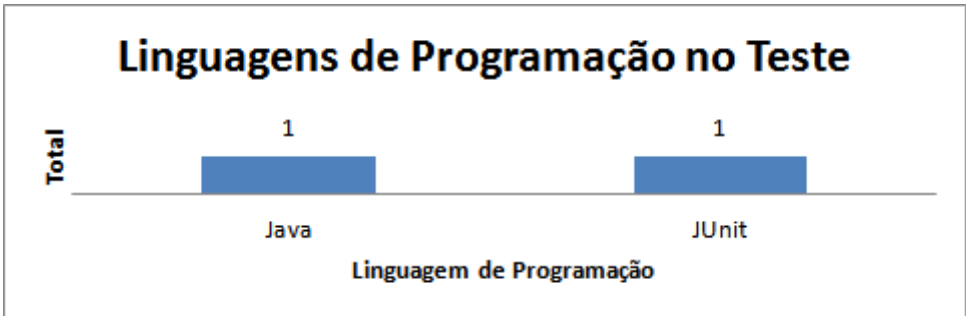


Figura 5.19: Linguagens de programação associadas ao teste da solução dos ativos.

5.2.3.3.6 Dados de Artefatos e Métodos de Teste

Apenas um artefato de teste, do tipo caso de teste, foi contemplado na solução dos ativos (Figura 5.20). Para os métodos de teste também apenas um tipo de teste foi associado, que é o teste de unidade (Figura 5.21). As informações sobre essa se extensão encontram na subseção 3.2.6.

Figura 5.20: Tipos dos artefatos de teste utilizados para o teste da solução dos ativos.

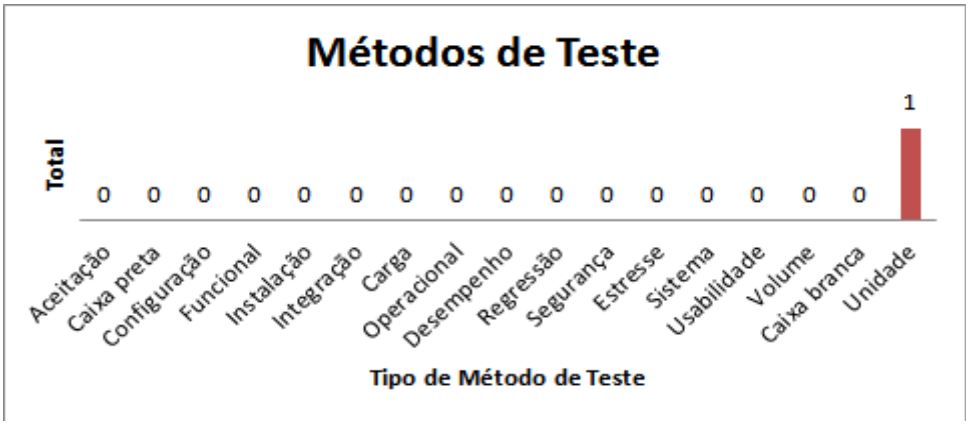


Figura 5.21: Métodos de teste aplicados no teste da solução dos ativos.

5.2.3.4 Análise dos Dados do Uso

Os dados relacionados às extensões do uso no SW-RAS, descritas na seção 3.3, são apresentados a seguir.

5.2.3.4.1 Dados de Perfis de Usuários

O total de usuários com o perfil publicador é dez (10), além de trinta e oito (38)

autores, apenas um certificador, e vinte e nove (29) consumidores, ilustrados na Figura 5.22. Cada usuário pode possuir um ou mais perfis (vide subseção 3.3.1). As informações sobre essas extensões se encontram na subseção 3.3.1.

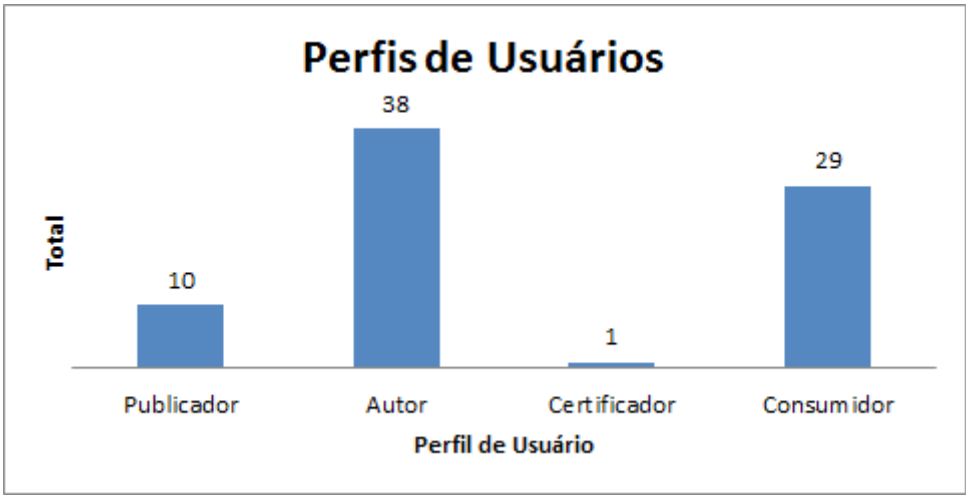


Figura 5.22: Totais de perfis de usuários publicadores, autores, certificadores e consumidores.

5.2.3.4.2 Dados de Publicações

Entre os dez publicadores, um deles cadastrou sessenta e dois (62) ativos reutilizáveis no repositório, por ter feita a migração dos ativos para o Lavoi. O restante desses usuários publicou apenas um ou dois ativos (Figura 5.23). As informações sobre essa extensão se encontram na subseção 3.3.1.

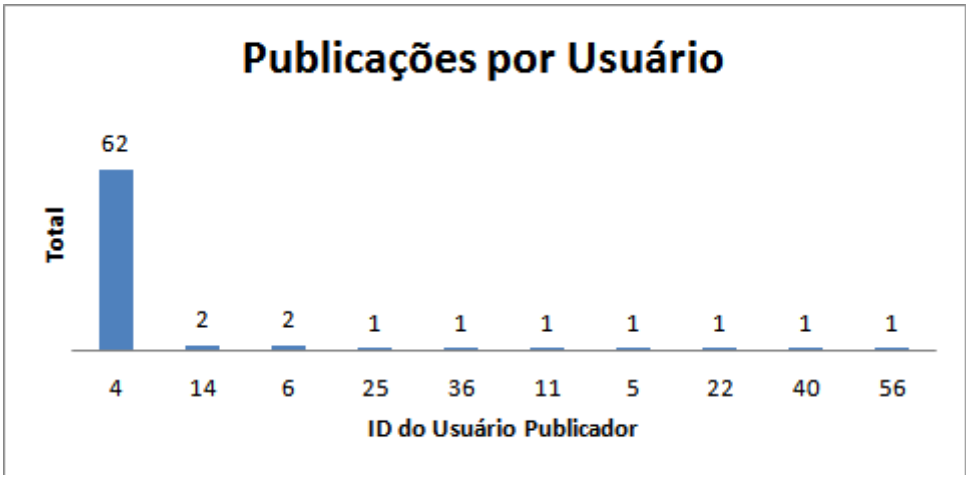


Figura 5.23: Total de publicações por usuário (publicador).

5.2.3.4.3 Dados de Consumo

O total de consumidores de ativos é vinte e nove (29) usuários, realizando quarenta e oito (48) consumos ao total. O histograma de consumo é ilustrado na Figura 5.24, indicando que dezessete (17) usuários consumiram apenas um ativo, e doze (12) usuários consumiram dois ou mais ativos. Esses consumos não detalham se o usuário realmente utilizou o ativo para construir um sistema, ou se apenas desejou entendê-lo ou experimentá-lo. As informações sobre essa extensão se encontram na subseção 3.3.1.

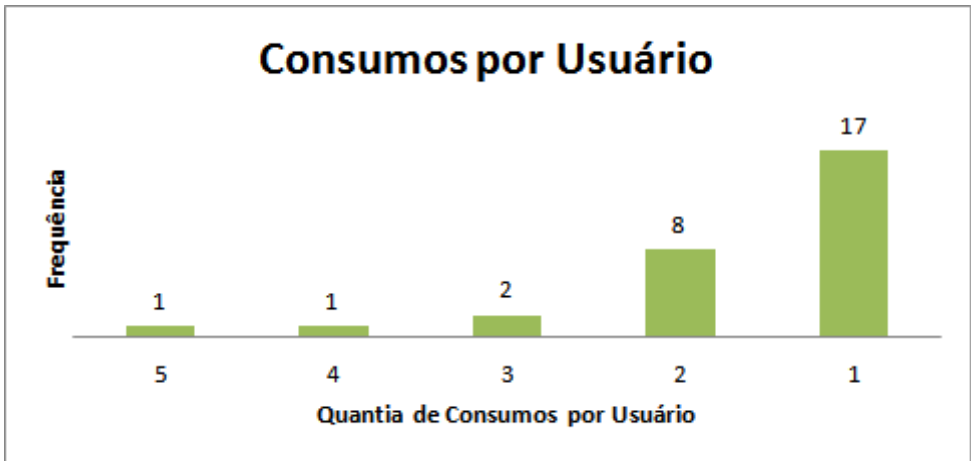


Figura 5.24: Histograma de consumos por usuário (consumidor).

5.2.3.4.4 Dados de Certificação

Apenas um usuário foi o certificador dos cinquenta e três (53) ativos certificados, ilustrados na Figura 5.1. As informações sobre essa extensão se encontram na subseção 3.3.1.

5.2.3.4.5 Dados de Ajustes

O total de ajustes realizados nos ativos é noventa (90). O histograma mostra que quatro ajustes foram realizados em cada um dos quatro ativos, três ajustes em dois ativos, dois ajustes em dezessete (17) ativos, e um ajuste em cada um dos trinta e quatro (34) ativos (Figura 5.25). As informações sobre essa extensão se encontram na subseção 3.3.1.

Figura 5.25: Histograma de ajustes por ativo.

5.2.3.4.6 Dados de Autoria

Foram trinta e oito (38) usuários atribuídos aos ativos como autores. Cada um pode ser autor de um ou mais ativos, sendo que dezessete (17) desses autores estão associados em apenas um ativo, e um dos usuários é autor de quinze (15) ativos reutilizáveis (Figura 5.26). As informações sobre essa extensão se encontram na subseção 3.3.1.

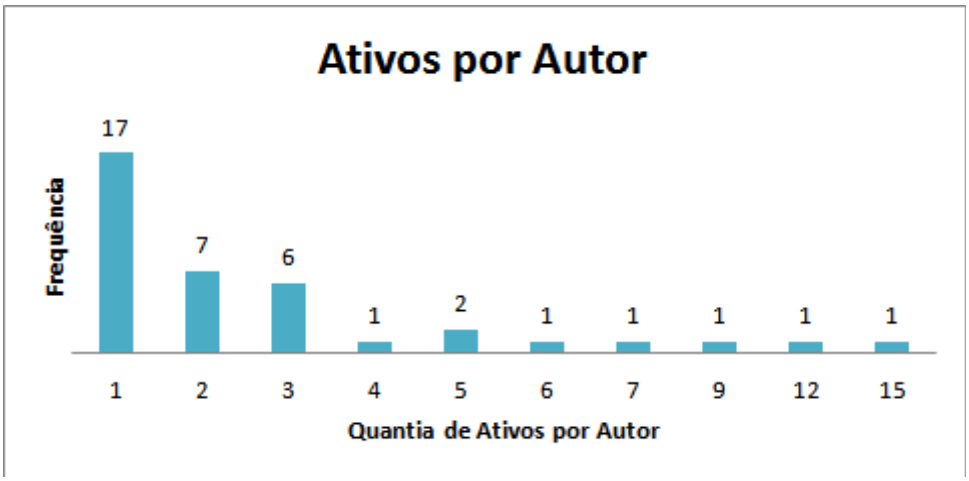


Figura 5.26: Histograma de ativos por autor.

5.2.3.4.7 Dados de Comentários de Usuários

Não houve comentários de usuários nos ativos. As informações sobre essa extensão se encontram na subseção 3.3.2.

5.2.3.4.8 Dados de Guias de Usuários

O total de guias de usuários contidos no uso dos ativos é noventa (90) artefatos. O histograma na Figura 5.27 exibe que quarenta e um (41) ativos contêm um guia de usuário, quatorze (14) ativos contêm dois guias, e cinco ativos possuem três ou mais guias de usuários. As informações sobre essa extensão se encontram na subseção 3.3.2.

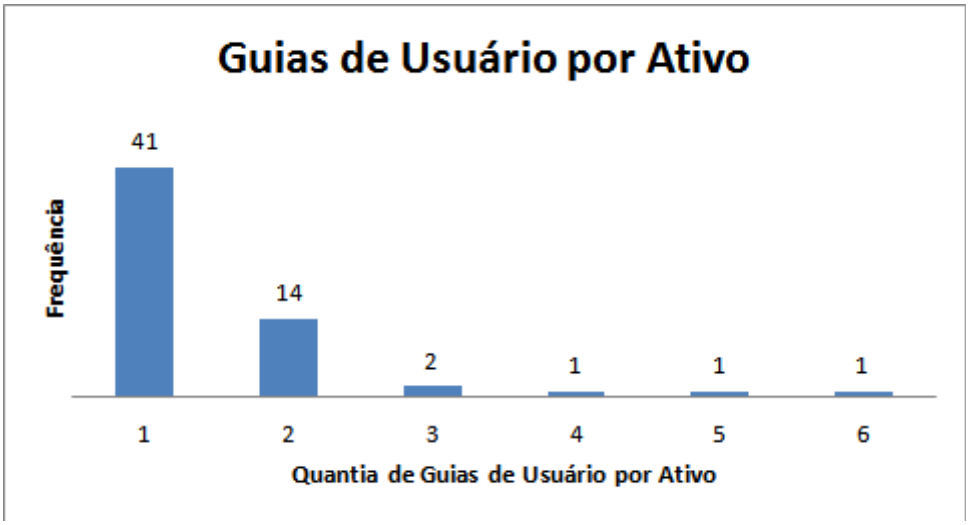


Figura 5.27: Histograma de artefatos de guias de usuários por ativo.

5.2.3.4.9 Dados de Descrições de Uso

As descrições de uso nos ativos foram de setenta e três (73) ao total, tendo o mesmo número de ativos publicados por serem um item obrigatório. As informações sobre essa extensão se encontram na subseção 3.3.2.

5.2.3.5 Análise dos Dados nos Tipos de Relacionamentos

Ao total foram dezesseis (16) relacionamentos estabelecidos entre ativos, dos quais

quinze (15) são do tipo de similaridade e um do tipo de dependência, ilustrados na Figura 5.28. As informações sobre essa extensão se encontram na subseção 3.4.1.

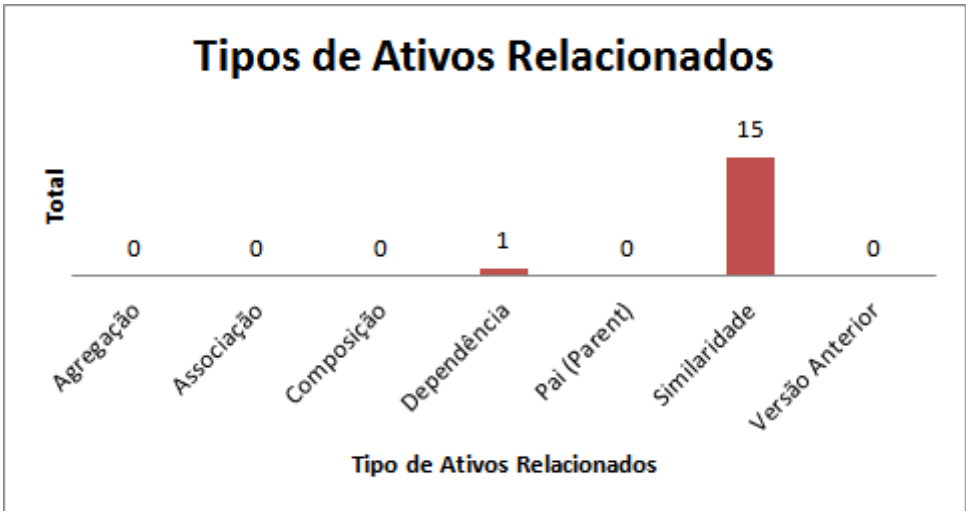


Figura 5.28: Tipos de ativos relacionados.

5.2.4 Discussão

Visto que o estudo de caso obteve dezenas de ativos reutilizáveis de software, diversas evidências de uso das extensões propostas no Software Profile RAS puderam ser investigadas (subseção 5.2.3). As considerações sobre essas evidências são discutidas a seguir, dentre as categorias de classificação (seção 3.1), de solução (seção 3.2), de uso (seção 3.3) e de ativos relacionados (seção 3.4).

5.2.4.1 Extensões na Classificação

Diversos tipos foram classificados nos ativos, identificando cada ativo de um modo simplificado. Além do tipo, a reusabilidade classificou diversos ativos pela relevância através da quantidade de consumos. Porém, poucas avaliações de qualidade foram realizadas, classificando poucos ativos no repositório quanto à relevância de qualidade tanto no uso e quanto do produto.

Alguns domínios de aplicação foram associados aos ativos desenvolvidos com foco em famílias específicas, ativos na forma de web services. A maior parte dos ativos publicados são de níveis horizontais, de infraestrutura ou de funcionalidades amplas, não sendo desenvolvidos para certos domínios e famílias de aplicações. Baixa foi a classificação por custos e esforços de desenvolvimento dos ativos, ocorrendo principalmente porque essas informações foram perdidas na baixa documentação dos ativos já existentes.

Pelas características intrínsecas da respectiva companhia de TI, as licenças de software são proprietárias e não foram associadas aos ativos. Além disso, a classificação por organizações e projetos também não foi utilizada pelo fato de que o desenvolvimento de ativos não é distribuído, e sim centralizado.

A classificação por social tagging se mostrou potencial pelo fato de que diversas tags foram associadas a cada ativo. Isso fez com que a nuvem de tags exibisse visualmente uma base de entendimento da natureza dos ativos no repositório (Figura 5.14). Uma das tags foi amplamente utilizada (“acr”), especificando os ativos reutilizáveis

desenvolvidos e mantidos pela divisão de tecnologia da organização.

5.2.4.2 Extensões na Solução

Artefatos de requisitos funcionais e não-funcionais não foram utilizados. Uma das razões é de que os ativos não são desenvolvidos com documentos de requisitos, e que parte desses requisitos estejam incorporados nos guias de usuários, utilizados extensivamente nos ativos publicados. Houve também um baixo uso de padrões de projeto nos ativos.

Um número moderado de artefatos de interfaces de usuários dentre seus níveis de abstração foram dispostos nos ativos, sendo sua maioria artefatos executáveis, prontos para serem integrados nos sistemas. Além disso, alguns ativos de serviços (web services) foram disponibilizados com artefatos WSDL.

Uma grande quantidade de linguagens de programação e códigos-fontes foram dispostos na solução dos ativos, apontando que os ativos são desenvolvidos com foco em códigos e tecnologias de desenvolvimento. Houve uma baixa quantidade de artefatos e métodos de teste dispostos nos ativos, o que indica que a verificação e a validação não são automatizadas, podendo trazer alguns problemas na integração dos ativos em novas aplicações de software.

5.2.4.3 Extensões no Uso

O uso dos perfis de usuários foi significativo. Um dos usuários publicadores foi o responsável por migrar os ativos já existentes na organização para o Lavoi, por essa razão teve dezenas de publicações. O restante dos usuários publicaram ativos dos quais eles próprios eram autores, desenvolvidos em colaboração para o desenvolvimento de software da organização. Em geral, houve mais de um autor por ativo, havendo um caso em que um único usuário era autor de quinze ativos.

Diversos ativos já reutilizados e homologados ao longo do tempo na organização foram certificados, havendo um usuário certificador. Houve uma quantidade considerável de usuários que consumiram ativos. A quantia de ajustes foi grande pela razão de que os ativos foram migrados para o repositório, e por algumas vezes foi necessário alterar as informações ou os artefatos dos ativos.

A quantidade de descrições de uso dos ativos foi igual à quantidade de ativos publicados, pelo fato de serem obrigatórias para cada ativo. Algumas dessas descrições referenciaram a utilização dos guias de usuários associados ao uso do ativo. A quantidade de artefatos de guias de usuários foi alta, com um pouco mais de um guia de usuário por ativo em média. Nenhum comentário de usuário foi realizado para a comunicação de dúvidas, sugestões ou relatos de defeitos de ativos.

5.2.4.4 Extensão nos Ativos Relacionados

A maior parte dos ativos relacionados teve o relacionamento de similaridade, não havendo relacionamentos de versão anterior. Isso acontece pelo fato de que as versões anteriores dos ativos não foram migradas, além do tempo que ainda não foi suficiente para existir uma nova versão a ser publicada e relacionada à versão anterior de um ativo.

5.2.4.5 Considerações Finais

Os resultados do estudo de caso refletem intrinsecamente o ambiente da organização e a maturidade de seu processo de desenvolvimento de software. O processo de Desenvolvimento para Reutilização dos ativos na organização deverá ser revisado para desenvolver ativos também para famílias de produtos e domínios de aplicação. Além disso, trabalhar em parceria com outras entidades do governo com base no desenvolvimento distribuído poderá beneficiar a reutilização de software em todas entidades envolvidas. A utilização dos custos e esforços dos ativos poderá auxiliar a gerir o desenvolvimento dos seus ativos, mas para isso se tornar real serão necessárias a implantação de um programa reúso a nível organizacional.

A padronização de critérios mínimos para a publicação e para a certificação dos ativos poderá beneficiar a qualidade, a confiança e a reusabilidade dos ativos. Esses critérios poderão englobar itens que faltaram na composição dos ativos, principalmente na solução, tais como requisitos bem definidos, padrões de projeto e artefatos de testes automatizados. O programa de reúso também poderá definir esses critérios, além do treinamento com o foco na produção de ativos reutilizáveis de software e no Lavoi. Além disso, esse programa poderá abordar incentivos para os publicadores que colaboram com os ativos reutilizáveis, através de alguma forma de reconhecimento.

Acredita-se que o treinamento em reúso de software e na utilização do repositório do Lavoi potencializará a importância do reúso de software na organização. Essa importância não é alta ainda por não se reconhecer ou não se fazer uso dos benefícios proporcionados, pela falta de cultura de reúso e pela falta de incentivos. A infraestrutura de SOA poderá incentivar mais ainda o reúso de software na organização.

Como trabalho futuro desse estudo de caso, será válido reanalisar tais dados após o programa de reutilização ser definido e implantado, além dos treinamentos serem realizados e da implantação dos serviços através de SOA. O estudo de caso também poderá ser aplicado em outros ambientes, tais como os que gerenciam os custos e esforços de desenvolvimento de ativos, que utilizem licenças de software ou que tenham seu desenvolvimento distribuído entre organizações e equipes.

5.3 Pesquisa com os Usuários do Lavoi

Uma pesquisa de campo foi realizada ao final do estudo de caso (seção 5.2) com os usuários do Lavoi visando a obter suas opiniões sobre o Lavoi, junto de seus perfis e opiniões quanto ao contexto do reúso de software na organização. O total de usuários respondentes foi de vinte e seis (26) pessoas. O uso foi feito de uma forma livre, visto que o Lavoi ainda não foi incorporado ao processo de desenvolvimento de software da organização. Essa pesquisa explora questões sobre o Lavoi, não diretamente sobre o SW-RAS porque é o que os usuários conhecem e utilizam.

As questões para cada usuário foram sobre:

- Nível de escolaridade (Figura 5.29).
- Idade (Figura 5.30).
- Experiência em desenvolvimento de software (Figura 5.31).
- Experiência efetiva em reúso de software (Figura 5.32).

- Quantidade de ativos reutilizáveis da empresa utilizados por ele (ao longo dos anos) (Figura 5.33).
- Quantidade de ativos reutilizáveis da empresa conhecidos por ele (Figura 5.34).
- Se seus projetos podem contribuir com ativos reutilizáveis (Figura 5.35).
- Se considera importante o treinamento em reúso de software proporcionado pela empresa (Figura 5.36).
- Se um repositório de ativos reutilizáveis contribui para o reúso de software (Figura 5.37).
- Se o repositório Lavoi apoia a prática de reúso de software (Figura 5.38).
- Frequência de uso do Lavoi pelo usuário (Figura 5.39).
- A opinião de quais funcionalidades podem ser adicionadas, modificadas ou removidas do Lavoi.

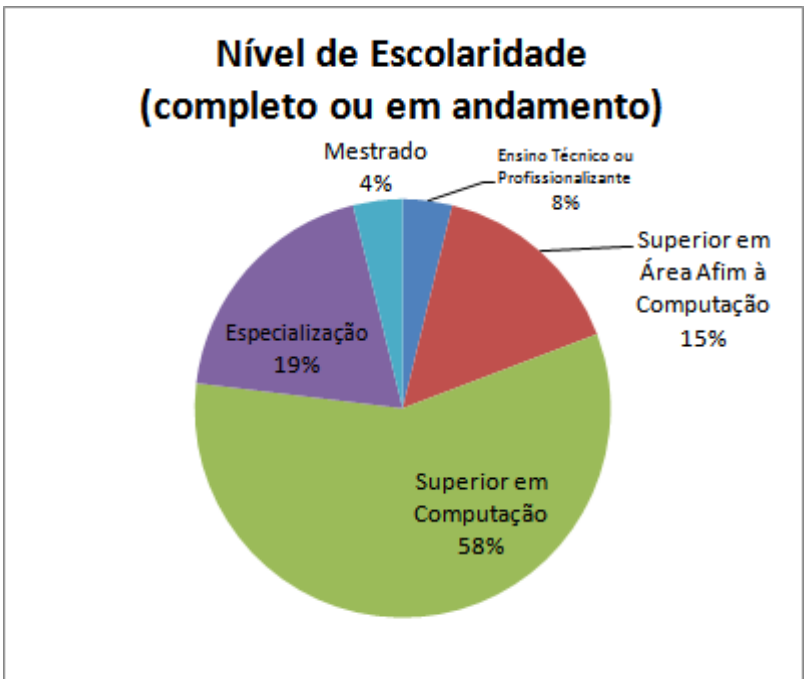


Figura 5.29: Nível de escolaridade dos usuários do Lavoi.

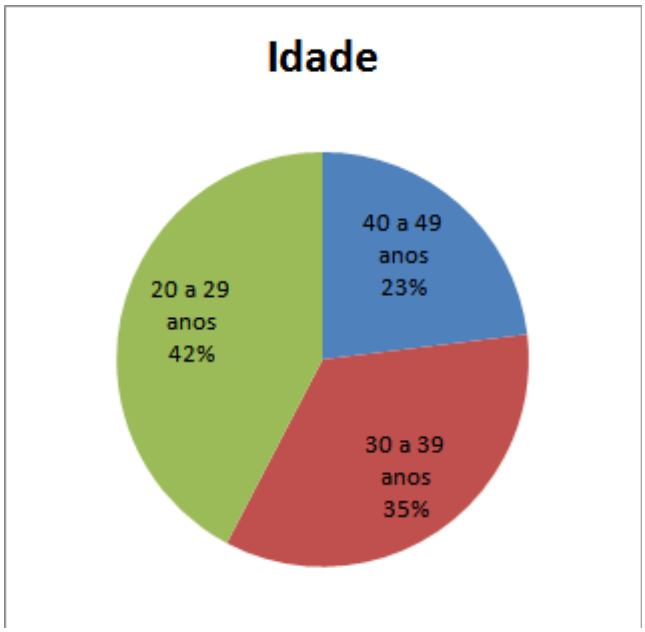


Figura 5.30: Idade dos usuários do Lavoi.

Figura 5.31: Experiência em desenvolvimento de software dos usuários do Lavoi.

Figura 5.32: Experiência efetiva em reúso de software dos usuários do Lavoi.

Figura 5.33: Quantidade de ativos reutilizáveis da empresa utilizados pelos usuários do Lavoi.

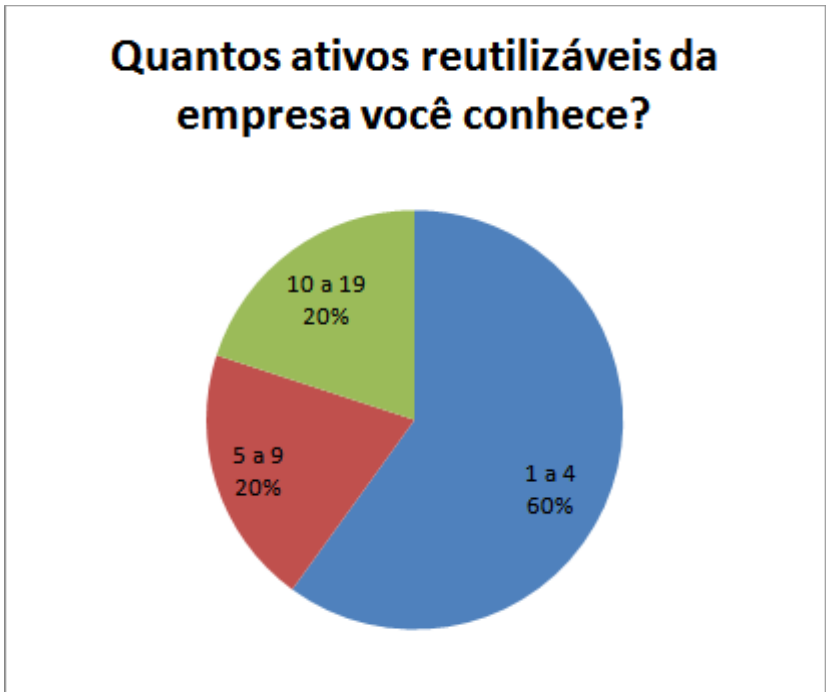


Figura 5.34: Quantidade de ativos reutilizáveis da empresa conhecidos pelos usuários do Lavoi.

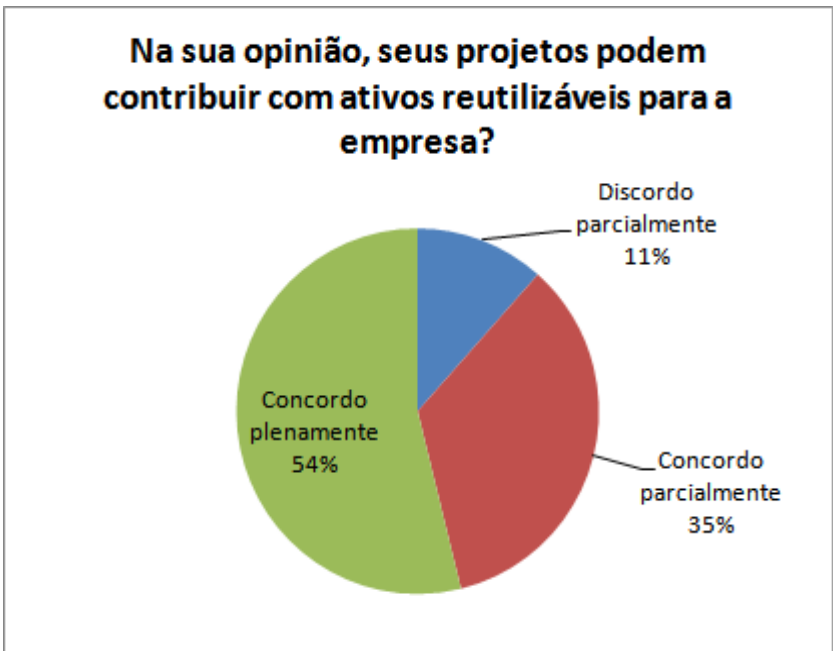


Figura 5.35: Opinião dos usuários do Lavoi de se seus projetos podem contribuir com ativos reutilizáveis para a empresa.

Figura 5.36: Opinião dos usuários do Lavoi de se acham importante obter o treinamento em reúso de software proporcionado pela empresa.

Figura 5.37: Opinião dos usuários do Lavoi de se um repositório de ativos reutilizáveis pode contribuir para o reúso de software.

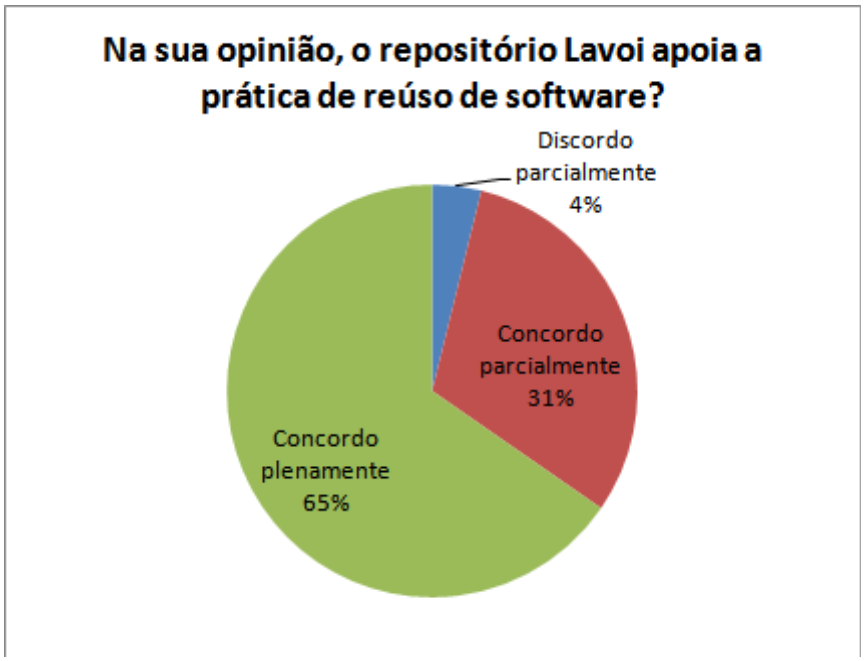


Figura 5.38: Opinião dos usuários de se o repositório Lavoi apoia a prática de reúso de software.

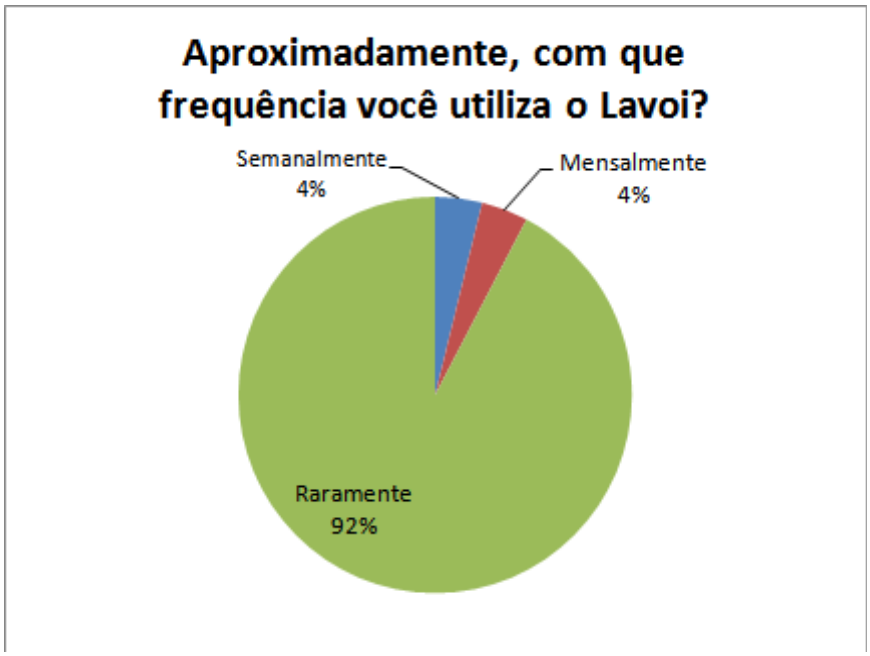


Figura 5.39: Frequência de utilização do Lavoi por seus usuários.

5.3.1 Discussão

A maior parte (58%) dos usuários respondentes da pesquisa possuem ao menos o ensino superior em Computação (completo ou em andamento) e vinte e três por cento (23%) possuem pós-graduação (completa ou em andamento). Tais usuários possuem uma faixa de idade bem distribuída entre 20 e 49 anos. Metade deles têm experiência de dez anos ao menos em desenvolvimento de software, e sessenta e cinto por cento (65%) têm três anos ao menos de experiência efetiva de reúso de software.

Sessenta e dois por cento (62%) utilizou ou utiliza apenas de 1 a 4 ativos

disponibilizados pela empresa (mesmo sem o Lavoi implantado), e apenas vinte por cento (20%) conhece de 10 a 19 desses ativos. Grande parte (89%) desses usuários concordam (plena ou parcialmente) que seus projetos podem contribuir com ativos reutilizáveis para o reúso na empresa. Além disso, todos os usuários concordam (plena ou parcialmente) que é importante obter o treinamento em reúso de software proporcionado pela empresa.

Todos esses usuários do Lavoi concordam (plena ou parcialmente) que um repositório de ativos reutilizáveis pode contribuir para o reúso de software. Noventa e seis por cento (96%) deles concordam (plena ou parcialmente) que o Lavoi apoia a prática de reúso de software. Apenas oito por cento (8%) deles acessam o repositório Lavoi a cada semana ou a cada mês.

As sugestões dos usuários para adição de funcionalidades ao Lavoi foram sobre:

- A tradução da ferramenta para o português.
- A notificação de novos ativos com resumos por e-mail aos usuários.
- O versionamento da documentação
- A integração com Maven[12], um gerenciador de build para projetos Java.

A única funcionalidade sugerida a ser modificada (melhorada) foi a de deixar mais explícito que o download do pacote do ativo trará apenas a URI dos artefatos que são referenciados, não os empacotando. Já para a remoção de funcionalidade do Lavoi, não houve sugestões.

Os usuários concordam que repositórios, assim como o Lavoi, apoiam a prática de reúso de software. Mesmo sem treinamento e com o pouco uso que ainda fazem do repositório, acreditam que podem se beneficiar com o reúso de software através do treinamento, além de concordarem que seus projetos poderem colaborar com ativos reutilizáveis. As notificações de novos ativos poderão fazer com que os usuários acessem mais o repositório, além da tradução da ferramenta para o português que poderá facilitar o entendimento de suas funcionalidades. O treinamento e o uso do Lavoi no processo de desenvolvimento de software poderão ser incorporados no programa de reúso da organização para tornar seu uso mais efetivo, e potencializar a produção e reutilização de tais ativos.

12 http://maven.apache.org/

6 CONCLUSÃO

O Software Profile RAS é proposto para solucionar lacunas relevantes do Reusable Asset Specification, um padrão de modelo de ativos reutilizáveis *de facto* criado pela OMG. Os resultados obtidos por esse trabalho foram:

1. Uma extensão compatível com o padrão RAS (seção 2.9), estendendo o Default Component Profile RAS em todas as categorias dos ativos: classificação, solução, uso e ativos relacionados. Suas diversas extensões visam ao empacotamento dos ativos com informações e artefatos úteis para o processo de reutilização de software, evitando a inexistência ou a baixa descrição dos ativos reutilizáveis em seus repositórios. Entre os pontos fortes da extensão proposta temos:

    - Um Profile RAS que permite uma maior diversidade de ativos através dos níveis horizontais e verticais dos ativos, além das dimensões de granularidade, de variabilidade e de articulação, por suas extensões com informações e artefatos, inclusive novas ao contexto do RAS (vide seção 5.1);

    - A classificação dos ativos através de seus tipos (subseção 3.1.1), além da relevância pela qualidade avaliada por usuários (subseção 3.1.2), da relevância pela reusabilidade (subseção 3.1.3), da classificação por domínios de aplicação (subseção 3.1.4), por organizações e projetos (subseção 3.1.5), por licenças de software (subseção 3.1.6), por custos e esforços de desenvolvimento (subseção 3.1.7) e por tags descritas livremente por usuários através de social tagging (subseção 3.1.8);

    - Um modelo com extensões na solução dos ativos, empacotando uma gama de artefatos reutilizáveis e informações relacionadas ao ciclo de vida de software, tais como requisitos funcionais e requisitos não funcionais (subseção 3.2.1), padrões de projeto (subseção 3.2.2), interfaces de usuário (subseção 3.2.3), serviços (web services) (subseção 3.2.4), códigos-fontes (subseção 3.2.5), linguagens de programação (subseção 3.2.5), artefatos e métodos de teste (subseção 3.2.6);

    - Extensões na categoria de uso com informações úteis para o processo de reúso de software, com perfis de usuários definidos para a produção, consumo e certificação de ativos (subseção 3.3.1). Por abranger as ações realizadas pelos usuários no ciclo de vida do ativo, a comunicação é viabilizada entre os usuários produtores, consumidores e certificadores. Os guias de usuários, as descrições de uso e os comentários de usuários são

definidos para a troca de conhecimento sobre os ativos entre os usuários produtores e consumidores (subseção 3.3.2).

2. O desenvolvimento de um sistema de repositório de ativos reutilizáveis, o Lavoi, estruturado pelo Software Profile RAS e construído para dar apoio a ele (Capítulo 4). O Lavoi foi refinado e consolidado por uso em contexto real em uma grande empresa pública de TI (seção 5.2), com a maioria de seus usuários na experimentação concordando que o Lavoi apoia a prática de reúso de software (seção 5.3). A extensão proposta foi fundamental por guiar e facilitar os mecanismos de armazenamento e de busca com relevância, além da nuvem de tags, da navegação por tags e de viabilizar notificações automáticas do repositório para os usuários. Pontos positivos do Lavoi inclui disponibilidade do Lavoi como software livre para sua utilização pela comunidade internacional de software, podendo ser adaptado e evoluído livremente. Além disso, pode ser utilizado como suporte de experimentações em outros estudos relacionados à reutilização de software.
3. Além disto, uma consequência indireta do trabalho foi a padronização de dezenas de ativos reutilizáveis na companhia de TI do estudo de caso (seção 5.2).

Esse trabalho propôs unificadamente diversas extensões com o intuito de resolver satisfatoriamente as lacunas do RAS, através do SW-RAS. No entanto, devido às particularidades do ambiente da companhia onde foi feito sua avaliação, algumas extensões não foram efetivamente avaliadas por não terem sido utilizadas durante o estudo de caso (seção 5.2).

As maiores limitações da experimentação foram:

(a) não possuir um ambiente de desenvolvimento distribuído entre organizações e projetos, por possuir o desenvolvimento de ativos centralizado;

(b) não utilizar as licenças de software por possuir desenvolvimento para reúso próprio;

(c) não possuir as informações dos custos e esforços de desenvolvimento de grande parte dos ativos.

Além disto, na solução dos ativos, não foram utilizados requisitos funcionais e requisitos não-funcionais, além de uma baixa quantidade de artefatos e métodos de teste para os ativos. Não houve usuários que comentaram sobre os ativos, e poucos avaliaram a qualidade dos ativos. Para mitigar alguns destes pontos, deverá ser elaborado um programa de reutilização, abrangendo treinamento para incentivar a cultura de reutilização de software da organização com mais qualidade, efetividade e gerenciamento dos ativos reutilizáveis de software na organização. Assim, constata-se mais uma vez que a maior barreira ao reúso de software continua sendo a cultural, pois a mudança de atitude dos desenvolvedores é ainda muito significativa para o sucesso de uma estratégia de reutilização.

Como perspectiva desse trabalho, esperamos que o Software Profile RAS possa beneficiar o reúso de software na comunidade de desenvolvimento de software através de seu modelo de ativos reutilizáveis. Por ser um Profile compatível com o RAS, também esperamos que seja continuado com extensões e refinamentos para melhor

atender o processo de reutilização de software, com o empacotamento dos ativos e a estrutura dos repositórios. Como trabalho futuro, pode-se experimentar soluções para as lacunas do RAS que já são conhecidas, tais como a de arquitetura de domínio, a de experiência dos usuários com o nível de acessibilidade do ativo, a de automatização da execução do ativo e do deploy de seus artefatos, e a de descrição do repositório onde se encontra cada ativo (seção 5.1). A experimentação de outros tipos de padrões e de serviços, além de padrões de projeto e de WSDL respectivamente, poderá ser um trabalho futuro. Análises aprofundadas em contextos de reutilização de software poderão encontrar novas lacunas e oportunidades de extensão no RAS. Temos a expectativa de que o SW-RAS e o sistema de repositório Lavoi auxiliem o empacotamento, o armazenamento, a busca e o gerenciamento do uso de ativos reutilizáveis de software em diversos ambientes de reutilização de software.

REFERÊNCIAS

ACM – Association for Computing Machinery. **The ACM Computing Classification System**. New York: The Association for Computing Machinery, Inc.. Disponível em: <http://www.acm.org/about/class/>. Acesso em: dez. 2012.

AGRESTI, W. W. Software Reuse: Developer's Experiences and Perceptions. **Journal of Software Engineering and Applications,** [S.l.], v.4, n.1, p. 48-58, Jan. 2011.

ALMEIDA, E. S. DE; ALVARO, A.; GARCIA, V. C.; MASCENA, J. C. C. P.; BURÉGIO, V. A. DE A. et al. **C.R.U.I.S.E: Component reuse in software engineering**. Brasil: C.E.S.A.R e-book, 2007. Disponível em: <http://cruise.cesar.org.br>. Acesso em: abr. 2012.

ALMEIDA, T. O. DE. **Ras4Nexus – Promovendo reuso utilizando o gerenciador de repositórios Nexus com o padrão RAS**. 2009. 46 f. Projeto de Diplomação (Bacharelado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

AUSSTATS. **1297.0 – Australian and New Zealand Standard Research Classification (ANZSRC)**. AUSSTATS Australian Bureau of Statistics, Austrália, dez. 2008. Disponível em: <http://www.abs.gov.au/AUSSTATS/abs@.nsf/DetailsPage/1297.02008?OpenDocument>. Acesso em: jun. 2013.

BASSO, F. P.; WERNER, C. M. L.; PILLAT, R. M.; OLIVEIRA, T. C.. A Common Representation for Reuse Assistants. In: FAVARO, J.; MORISIO, Maurizio. Safe and Secure Software Reuse. **13th International Conference on Software Reuse, ICSR 2013 Proceedings**. [S.l.]: Springer Berlin Heidelberg, 2013.

BRAGA, R. M. M.; WERNER, C. M. L.; MATTOSO, M. Odyssey: A reuse environment based on domain models. **IEEE Symposium on Application-Specific Systems and Software Engineering and Technology**. Richardson, Texas, p. 50-57, mar. 1999.

BURÉGIO, V. A. de A. **Specification, design and implementation of a reuse repository**. 2006. 131 f. Dissertação (Mestrado em Ciência da Computação) – Pós-Graduação em Ciência da Computação, UFPE, Recife.

BURÉGIO, V. A.; ALMEIDA, E. S.; LUDRÉDIO, D.; MEIRA, S. L.. A Reuse Repository System: From Specification to Deployment. **ICSR '08 Proceedings of the 10th international conference on Software Reuse: High Confidence Software Reuse in Large Systems.** Beijing, China, v.5030, p. 88-99, mai. 2008.

CHAVEZ, A.; TORNABENE, C.; WIEDERHOLD, G. Software component licensing: a primer. **IEEE Software.** [S.l.], v.15, n.5, p. 47-53, set./out. 1998.

DAVIES, J. Measuring Subversions: Security and Legal Risk in Reused Software Artifacts. **ICSE '11 Proceedings of the 33rd International Conference on Software Engineering**, Honolulu, HI, p. 1149-1151, mai. 2011.

ELSEVIER. **Subjects**. Elsevier Inc., Philadelphia, PA. Disponível em: <http://www.elsevier.com/books/subjects>. Acesso em: jun. 2013.

EZRAN, M.; MORISIO, M.; TULLY, C. **Practical software reuse**. 1st ed. [S.l.]: Springer London, 2002.

FOWLER, M. Patterns. **IEEE Software**. [S.l.], v.20, n.2, p. 56-57, mar./abr. 2003.

FRAKES, W. B.; NEJMEH, B. A. Software reuse through information retrieval. **Newsletter, ACM SIGIR Forum Homepage archive**, New York, NY, USA, v.21, n.1-2, p. 30-36, set. 1986/mar. 1987.

FRAKES, W. B.; Gandel, P. B. Representing reusable software. **Information and Software Technology**. Newton, MA, USA, v.32, n.10, p. 653-664, dez. 1990.

FRAKES, W. B.; ISODA, S. Success factors of systematic reuse. **IEEE Software**. [S.l.], v.11, n.5, p. 14-19, set. 1994.

FRAKES, W.. TERRY, C. Software reuse: metrics and models. **ACM Computing Surveys (CSUR)**. New York, NY, USA, v.28, n.2, p. 415-435, jun. 1996.

FRAKES, W. B.; KANG, K. Software reuse research: Status and future. **IEEE Transactions on Software Engineering**. Piscataway, NJ, USA, v.31, n.7, p. 529-536, jul. 2005.

FRANCESCHINI, P. M. **Estendendo ReUse: Rumo a uma ferramenta mais efetiva de suporte a reuso**. 2008. 60 f. Projeto de Diplomação (Bacharelado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

FSF – Free Software Foundation, Inc. **Various Licenses and Comments about Them**. Free Software Foundation, Inc., [S.l.]. Disponível em: <http://www.gnu.org/licenses/license-list.html>. Acesso em: jul. 2013.

GAMMA, E; HELM, R.; JOHNSON, R.; VLISSIDES, J.. **Design patterns elements of reusable object-oriented software**. 1st edition. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

GAO, J. Z.; TSAO, J.; WU, Y.; JACOB, T. H.-S.. **Testing and Quality Assurance for Component-Based Software**. Testing and Quality Assurance for Component-Based Software. Norwood, MA, USA: Artech House, Inc., 2003.

GARTNER, Inc. **Building a Reuse Culture in the Development Organization**. [S.l.]: GARTNER, Inc, jun 2010.

IBM. **IBM Rational Asset Manager**. 2007. Disponível em: <http://www.ibm.com/software/rational/products/ram/>. Acesso em: jun. 2013.

IBM. **Strategic Reuse with Asset-Based Development**. First Edition. [S.l.]: IBM

Redbooks, 2008.

HADJI, H. B.; KIM, S.-K.; CHOI, H.-J. A representation model for reusable assets to support user context. **SOSE '08 Proceedings of the 2008 IEEE International Symposium on Service-Oriented System Engineering**, Jhongli, Taiwan, p. 91-96, 2008.

HESA. **Joint Academic Coding System (JACS).** HESA – Higher Education Statistics Agency, Cheltenham, United Kingdom, 2008. Disponível em: <http://www.hesa.ac.uk/jacs/>. Acesso em: jun. 2013.

HOBBS, E. T. A Uniform Data Model for Reuse Library Interoperability. **WISR 6 - Workshop In Software Reuse**. Owego, NY, nov. 1993.

HONG-MIN, R.; ZHI-YING, Y.; JING-ZHOU, Z. Design and implementation of RAS-based open source software repository. **FSKD '09 Proceedings of the 2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery**. Washington, DC, USA, v.2, p. 219-223, ago. 2009.

HULL, E.; JACKSON, K.; DICK, J. **Requirements Engineering.** [S.l.]: Springer, 2nd edition, 2005.

ISO/IEC. **ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models**. ISO/IEC, Geneva, Switzerland, 2010.

ISO/IEC. **ISO/IEC 9126-1:2001. Software engineering – Product quality – Part 1: Quality model**. ISO/IEC, Geneva, Switzerland, 2001.

JARZABEK, S. **Effective Software Maintenance and Evolution**: A Reuse-Based Approach. [S.l.]: Auerbach Publications, 2007.

KEQIN, L.; LIFENG, G.; HONG, M.; FUQING, Y.. An overview of JB (Jade Bird) Component Library System JBCL. **TOOLS '97 Proceedings of the Technology of Object-Oriented Languages and Systems-Tools – 24**, Washington, DC, USA, p. 206-213, set. 1997.

KRUEGER, C. W. Software product line reuse in practice. **ASSET '00 Proceedings of the 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET'00)**. Washington, DC, USA, p. 117-118, mar. 2000.

LACERDA, G. S. **FrameworkDoc: Ferramenta de documentação e geração de artefatos de software**. 2005. 77 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

LIONS, J. L. **ARIANE 5. Flight 501 Failure. Report by the Inquiry Board.** Paris, [s.n], jul. 1996. Disponível em: <http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html>. Acesso em: ago. 2013.

LLORENS, J.; AMESCUA, A.; VELASCO, M. Software Thesaurus: a tool for reusing software objects. **SAST '96 Proceedings of the Proceedings of the Fourth International Symposium on Assessment of Software Tools (SAST '96)**, Washington, DC, USA, p. 99, 1996.

MACCHINI, B. Reusing software with ESTRO (Evolving Software Repository).

Fourth International Conference on Software Engineering and Knowledge Engineering, Capri, Italy, p. 150-157, jun. 1992.

MACHADO, F; ALMEIDA, E. S.; MEIRA, S. R. L. Designing a set of Service-Oriented Systems as a Software Product Line. **SBCARS '10 Proceedings of the 2010 Fourth Brazilian Symposium on Software Components, Architectures and Reuse**, Salvador, Brazil, p. 70-79, 2010.

MARSHALL, J. J.; DOWNS, R. R.; MATTMAN, C. A. Progress towards a NASA Earth Science Reuse Enablement System (RES). **Information Reuse and Integration (IRI), 2010 IEEE International Conference on**, Las Vegas, Nevada, USA, p. 340-343, ago. 2010.

MARTINS, J. de S. ReUse: **Uma ferramenta de suporte a reuso**. 2008. 45 f. Projeto de Diplomação (Bacharelado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

MCILROY, M. D. Mass produced software components. **Software Engineering, Report on a conference sponsored by the NATO Science Committee**, Garmisch, Germany, p. 138-155, out. 1968.

MILI, A; MILI, R.; MITTERMEIR, R. T. A survey of software reuse libraries. **Annals of Software Engineering 5,** Red Bank, NJ, USA, v.5, n.1, p. 349-414, 1998.

MILLS, E. E. **Software Metrics**. Seattle, Washington: Carnegie Mellon University, Software Engineering Institute, 1988.

MOON, M.; CHAE, H. S.; YEOM, K. A metamodel approach to architecture variability in a product line. **ICSR'06 Proceedings of the 9th international conference on Reuse of Off-the-Shelf Components,** Turin, Italy, Springer-Verlag, v.4039, p. 115-126, 2006.

MOREIRA, A. A. **Reuso de IHC orientado a padrões concretos de interação e dirigido por casos de uso**. 2007. 181 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

NASA. **Software Packaging for Reuse**. Version 1.0, [S.l.]: NASA Earth Science Data Systems – Software Reuse Working Group, fev. 2011.

OMG. **Reusable Asset Specification. OMG Available Specification**. Version 2.2. [S.l]: Object Management Group, Inc, nov. 2005. Disponível em: <http://www.omg.org/spec/RAS/>. Acesso em: jun. 2013.

O'REILLY, T. **The architecture of participation**. O'Reilly Media, Inc, [S.l.], jun. 2004. Disponível em: <http://oreilly.com/pub/a/oreilly/tim/articles/architecture_of_participation.html>. Acesso em: jun. 2013.

PALUDO, M.; REINEHR, S.; MALUCELLI, A.; BRUZON, L.; PINHO, P.. Applying Pattern Structures to Document and Reuse Components in Component-Based Software Engineering Environments. **Information Reuse and Integration (IRI), 2011 IEEE International Conference on**, Las Vegas, Nevada, USA, p. 378-383, ago. 2011.

PARK, S.; PARK, S.; SUGUMARAN, V. Extending reusable asset specification to improve software reuse. **SAC '07 Proceedings of the 2007 ACM symposium on**

Applied computing, Seoul, Korea, p. 1473-1478, 2007.

PRESSMAN, R. S. **Engenharia de Software** – Uma abordagem profissional. 7ª edição. Porto Alegre: Bookman, 2011.

RAMACHANDRAN, M. Software reuse guidelines. **ACM SIGSOFT Software Engineering Notes,** New York, NY, USA, v.30, n.3, p. 1-8, mai 2005.

RIVADENEIRA, A. W.; GRUEN, D. M.; MULLER, M. J.; MILLEN, D. R.. Getting our head in the clouds: toward evaluation studies of tagclouds. **CHI '07 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems**, New York, NY, USA, p. 995-998, 2007.

ROSA, F. R. DA. **RASPUTIN Uma infra-estrutura de suporte para promover o reuso de software através do padrão RAS**. 2009. 55 f. Projeto de Diplomação (Bacharelado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

SAMETINGER, J. **Software Engineering with Reusable Components**. New York, NY, USA: Springer-Verlag New York, Inc., 1997.

SANDHU, P. S.; AASHIMA; KAKKAR, P.; SHARMA, S.. A Survey on Software Reusability. **2010 International Conference on Mechanical and Electrical Technolog (ICMET 2010)**, Singapore, p. 769-773, set. 2010.

SCHIROKY, A. M. **Modelagem e prototipação de um repositório extensível para componentes de software**. 2002. 114 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

SCHUENCK, M.; ELIAS, G. Suporte à Certificação de Componentes no Modelo de Representação X-ARM. **Anais do Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software**, Campinas, p. 135-148, 2007.

SHERIF, K.; VINZE, A. Barriers to adoption of software reuse: A qualitative study. **Information and Management**, Amsterdam, The Netherlands, v.41, n.2, p. 159-175, dez. 2003.

SHIVA, S. G.; SHALA, L. A.. Software Reuse: Research and practice. **Fourth International Conference on Information Technology**, Las Vegas, NV, USA, p. 603-609, abr. 2007.

SILVA, R. P. e. **Suporte ao desenvolvimento e uso de frameworks e componentes**. 2000. 262 f. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

SINDRE, G.; CONRADI, R.; KARLSSON, E. A. The REBOOT approach to software reuse. **Journal of Systems and Software - Special issue on software reuse**, v.30, n.3, p. 201-212, set. 1995.

SOFTEX – Associação para a Promoção da Excelência do Software Brasileiro. **MPS.BR – Melhoria de Processo do Software Brasileiro – Guia de Implementação – Parte 3: Nível E:2011**. Disponível em: <http://www.softex.br/mpsbr/guias/>. Acesso em: set. 2013.

SOFTEX – Associação para a Promoção da Excelência do Software Brasileiro.

MPS.BR – Melhoria de Processo do Software Brasileiro – Guia de Implementação – Parte 5: Nível C:2011. Disponível em: <http://www.softex.br/mpsbr/guias/>. Acesso em: set. 2013.

SOJER, Manuel. **Reusing Open Source Code.** Value Creation and Value Appropriation Perspectives on Knowledge Reuse. 1st Edition. [S.l.]: Gabler Verlag, 2011.

SOLDERITSCH, J. J.; CREPS, D. **Asset Library Open Architecture Framework** – Sharing reusable assets. [S.l.:s.n], ago. 1992.

SOMMERVILLE, I. **Software Engineering**. 9th Edition. [S.l.]: Addison-Wesley, 2011.

SUTCLIFFE, A. G.; CARROLL, J. M. Designing Claims for Reuse in Interactive Systems Design. **International Journal of Human-Computer Studies**. v.50, n.3, p. 213-241, mar. 1999.

THOM, L. H.; CHIAO, C.; IOCHPE, C. Padrões de Workflow para Reuso em Modelagem de Processos de Negócio. **Conferência Latino Americana em Linguagens de Programação, SugarloafPlop**, Porto de Galinhas, v.6, 2007.

VOTH, D. Packaging reusable software assets. **IEEE Software**. v.21, n.3, p. 107-108, 110, ago. 2004.

WAL, T. V. **Folksonomy**. [S.l.:s.n], fev. 2007. Disponível em: <http://vanderwal.net/folksonomy.html>. Acesso em: jun. 2013.

WANG, Z.-S.; XU, R.-Z.; ZHANG, K.-K.; FENG, D.-L.. Design and implementation of RAS-Based reusable asset management tool. **Internet Computing in Science and Engineering, 2008. ICICSE '08. International Conference on**, Harbin, p. 363-366, jan. 2008.

WASHIZAKI, H.; YAMAMOTO, H.; FUKAZAWA, Y. A Metrics Suite for Measuring Reusability of Software Components. **METRICS '03 Proceedings of the 9th International Symposium on Software Metrics**, Washington, DC, USA, p.211-223, set. 2003.

WERNER, C. M. L.; TRAVASSOS, G. H.; ROCHA, A. R. DA; CIMA, A. M. DE; SILVA, M. F. DA et al. Memphis: A reuse based O.O. software development environment. **Technology of Object-Oriented Languages, 1997. TOOLS 24. Proceedings**, Beijing, p. 182-191, set. 1997.

WERNER, C.; MURTA, L.; LOPES, M.; DANTAS, A.; LOPES, L. G. et al. Brechó: Catálogo de componentes e serviços de software. **In: Anais da XIV Sessão de Ferramentas, XXI Simpósio Brasileiro de Engenharia de Software**, João Pessoa, p. 24-30**,** out. 2007.

WOHLIN, C.; RUNESON, P.; HOST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A.. **Experimentation in Software Engineering**. [S.l.]: Springer Berlin Heidelberg, 2012.

XEXEO, G.; MORGADO, F.; FIUZA, P. Differential Tag Clouds: Highlighting Particular Features in Documents. **Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT '09. IEEE/WIC/ACM International Joint Conferences on**, Milan, Italy, v.3, p. 129-132, set. 2009.

YE, Y. An active and adaptive reuse repository system. **System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on**, Washington, DC, USA, v.9, p. 9065, jan. 2001.

YU, W. D.; ONG, C. H. A SOA Based Software Engineering Design Approach in Service Engineering. **E-Business Engineering, 2009. ICEBE '09. IEEE International Conference on**, Macau, p. 409-416, out. 2009.