COPPE/UFRJ

ARARA: UM SISTEMA MULTI-AGENTES PARA PROVISÃO DE PERCEPÇÃO EM DESENVOLVIMENTO DE SOFTWARE

Ester José Casado de Lima

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Geraldo Bonorino Xexéo

Rio de Janeiro

Setembro de 2010

ARARA: UM SISTEMA MULTI-AGENTES PARA PROVISÃO DE PERCEPÇÃO EM DESENVOLVIMENTO DE SOFTWARE

Ester José Casado de Lima

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____

Prof. Geraldo Bonorino Xexéo, D.Sc.

_____

Prof. Jano Moreira de Souza, Ph.D.

_____

Profª. Sergio Palma da Justa Medeiros, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

SETEMBRO DE 2010

Lima, Ester José Casado de Lima

ARARA: Um Sistema Multi-Agentes para Provisão de Percepção em Desenvolvimento de Software / Ester José Casado de Lima. – Rio de Janeiro: UFRJ/COPPE, 2010.

XIV, 103 p.: il.; 29,7 cm.

Orientador: Geraldo Bonorino Xexéo

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2010.

Referencias Bibliográficas: p. 71-75.

1. Awareness. 2. Ontologia. 3. Agentes. I. Xexéo, Geraldo Bonorino. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Titulo.

Àquele que me ama e me faz sentir amada,
Marcello Campos.
E ao Pedroca que também acompanhou, com
grande interesse, meus estudos.
Amo vocês!

Agradecimentos

Não tenho palavras para agradecer a tantas pessoas que contribuiram para o meu engrandecimento profissional, técnico e pessoal ao longo do meu curso de mestrado. Faço uso dessas linhas para expressar minha gratidão a todos que passaram e deixaram suas marcas em meu caminho.

Meu agradecimento em especial para:

Geraldo Xexéo, pela orientação, pela paciência e pela confiança dedicada a mim, e pelas grandiosas reuniões de alunos promovidas durante o período de orientação de tese. Sem dúvida foram muito motivadoras e enriquecedoras.

José Rodrigues Neto, amigo que me co-orientou, agradeço pelo apoio, incentivo e correções. Seus votos de confiança me deram forças em momentos de dúvida.

Marcello Campos, meu marido, agradeço por tudo (e você sabe o que quero dizer). Seu amor por mim é tudo!

Também deixo meus agradecimentos para:

Os professores *Marcello Lanza* e *Antônio Cláudio,* do Departamento de Eletrônica da UFRJ por terem confiado no meu potencial quando me recomendaram ao PESC.

Ao professor *Jano de Souza*, pela oportunidade concendida para a realização do meu mestrado na linha de banco de dados. Tenho a agradecer também pelo carinho, atenção e alegria sempre dispensados a mim ao me receber.

Aos *meus pais*, sempre devo agradecer-lhes, pois o que sou devo à educação que eles me deram.

Aos *amigos do mestrado* que me auxiliaram de diversas formas inimagináveis: *Fernando Morgado, Frederico Tosta, Michelle Machado, Rodrigo Aguas, Patrícia Fiuza, Vinícius Marques e Viviane Farias*.

Aos *membros da equipe COPPETEC* e à equipe com a qual trabalhei na PETROBRAS no projeto SISRES. Foi muito bom tê-los conhecido e trabalhado com vocês. Meus agradecimentos em particular para *Ana Paula* e *Márcia Helena*.

Às secretárias da linha de Banco de Dados: *Patrícia Leal* e *Ana Paula Rabello* e às secretárias do PESC: *Carol*, *Claudia Prata, Solânge* e *Sônia*. Todas sempre dispostas a me ajudar no que fosse.

À *Maria*, pelo carinho e preparo cuidadoso do meu cafezinho.

Aos *meus irmãos e amigos*. É sempre bom contar com vocês para tornar a vida mais agradável e para nos fornecer momentos de alegria e relaxamento.

Enfim, a todos que passaram pelo meu caminho durante esses anos de mestrado, agradeço pelas palavras de confiança, pela força e pelo carinho.

Obrigada!

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ARARA: UM SISTEMA MULTI-AGENTES PARA PROVISÃO DE PERCEPÇÃO EM DESENVOLVIMENTO DE SOFTWARE

Ester José Casado de Lima

Setembro/2010

Orientador: Geraldo Bonorino Xexéo

Programa: Engenharia de Sistemas e Computação

Esta dissertação apresenta um sistema capaz de auxiliar o processo de desenvolvimento de projetos, melhorando o aspecto de percepção dos membros da equipe de desenvolvimento quando mudanças nos artefatos do projeto forem detectadas, assim como o efeito da propagação dessas mudanças pelos demais artefatos relacionados. Esse sistema tem o suporte de agentes e do uso de ontologias aplicadas na construção dos artefatos do projeto, que serão utilizadas pelos agentes como base de conhecimento para relacionar os artefatos uns aos outros. O sistema proposto chama-se ARARA (*Artifacts and Requiremets Awareness Reinforcement Agents*).

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

ARARA: A MULTI-AGENTS SYSTEM TO PROVIDE AWARENESS TO SOFTWARE DEVELOPMENT PROCESSES

Ester José Casado de Lima

September/2010

Advisor: Geraldo Bonorino Xexéo

Department: Computer Science and Engineering

This dissertation presents a system capable of assisting development projects, improving perception of team members when changes to project artifacts are detected, as well as the effect of propagation of these changes for other related artifacts. This system is supported by agents that uses the ontology implemented in the construction of project artifacts as knowledge base to discover relationships among artifacts. The system proposed is called ARARA (Artifacts and Requirements Awareness Reinforcement Agents).

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

GLOSSÁRIO

ANSI *American National StandardsInstitute*

API *Application Programming Interface*

ARARA *Artifacts and Requiremets Awareness Reinforcement Agents*

BCD *Base de Conhecimento do Domínio*

BCP *Base de Conhecimento do Projeto*

CMMI *Capability Maturity Model Integration*

IEEE *Institute of Eletrical and Electronics Engineers*

MAS *Multi-Agents System*

OMG *Object Management Group*

OWL *Ontology Web Language*

P2P *Peer-to-Peer*

RA *Repositório de Artefatos*

RDF *Resource Description Framework*

REQM SP *REQuirements Management Specific Practice*

RSA *Rational Software Architech*

SVN *Subversion*

SWRL *Semantic Web Rule Language*

UML *Unified Modeling Language*

XMI *XML Model Interchange*

XML *eXtensible Markup Language*

W3C *World Wide Web Consortium*

Capítulo 1 - Introdução

Este capítulo apresenta o problema da provisão de percepção em processo de desenvolvimento de software e a motivação que geraram a proposta do sistema ARARA. Posteriormente são apresentados os objetivos da dissertação e por fim é apresentada a organização estrutural da dissertação.

1.1. Problema

O processo de desenvolvimento de software é um trabalho cooperativo e, portanto, a comunicação é reconhecida como sendo uma tarefa crítica (PRESSMAN, 2006). Manter conhecimento sobre as atividades dos outros para prover um contexto para a sua própria atividade é uma tarefa crítica a ser mantida ao longo do processo de desenvolvimento (DOURISH e BELLOTTI, 1992), principalmente nos tempos atuais quando a equipe de desenvolvimento pode estar distribuída geograficamente, como em projetos *open-source* ou desenvolvimentos de projetos *off-shore*.

O processo de desenvolvimento de software, baseado nos requisitos do negócio, gera um conjunto com grande número de artefatos (documentos e código) que são criados por membros diferentes da equipe. A interação entre os membros da equipe pode contribuir para a criação do conhecimento comum do projeto. Entretanto, nem sempre os membros da equipe sabem com detalhes o trabalho que está sendo realizado pelos demais membros e de que forma seus trabalhos se relacionam.

Os dois pontos principais que definem o grau de percepção dos membros da equipe em um projeto de desenvolvimento de softwares são:

1. percepção sobre o trabalho dos demais, ou seja, o que está sendo realizado, quando e por quem e
2. percepção sobre o relacionamento dos artefatos, ou seja, a capacidade de rastreabilidade entre os artefatos.

A falta de percepção sobre o trabalho dos demais (1) faz com que uma mudança realizada em determinados artefatos do projeto de responsabilidade de um membro específico da equipe não seja detectada e consequentemente não seja propagada para outros membros da equipe, cujas tarefas podem ser afetadas por tal mudança. A falta de percepção sobre o relacionamento dos artefatos (2) faz com que a mudança não seja propagada para todos os artefatos associados, mesmo que seja detectada uma mudança em um conjunto de artefatos.

A quebra no processo de propagação de mudanças, gerada pela falta de percepção do projeto como um todo ocasiona a inconsistência nos artefatos e a implementação de um sistema onde as regras de negócio podem estar inconsistentes com as regras mapeadas no processo de levantamento de requisitos. Como consequência, há perda de trabalho, por ser necessário refazer a implementação do que ficou inconsistente.

Para prover percepção quanto à mudança de requisitos e sua propagação pelos demais artefatos do projeto, faz-se necessário atender a dois requisitos:

1. que os artefatos, de alguma forma, sejam rastreáveis e
2. que a rastreabilidade entre os artefatos seja automática.

Neste contexto, é necessário um sistema capaz de prover aos membros da equipe a percepção necessária sobre o que os demais estão trabalhando, a fim de detectar mudanças, e as implicações das mesmas, de forma automática, para que seja propagada de forma adequada as mudanças pelos demais artefatos.

1.2. Motivação

O desenvolvimento do ARARA foi motivado pela necessidade de prover percepção *(awareness)* em ambientes de desenvolvimento de software, onde os requisitos mudam com frequência e existe falta de conhecimento dessas mudanças por parte de todos da equipe.

O nome ARARA foi a sigla obtida do título do primeiro artigo publicado sobre o assunto e que significa, em inglês, *"Artifacts and Requirement Awareness Reinforced Agents"* (LIMA *et al.*, 2008). A nome ARARA sugere uma associação entre a ave da fauna brasileira e o sistema proposto, onde ambas possuem uma certa característica de "fala". Das espécies das araras, a arara-canindé ou arara-de-barriga-amarela (WIKIPEDIA, 2010) é a que chega mais próximo às características do papagaio, ou seja, é a que possui alguma capacidade de fala. O sistema ARARA possui, de certa forma capacidade de "fala", ou seja, notifica aos membros de uma equipe de desenvolvimento de software o quanto uma mudança realizada em um artefato do projeto impacta nos demais artefatos e quais artefatos possivelmente serão impactados por essa mudança.

1.3. Objetivo

Esta dissertação apresenta um sistema capaz de prover percepção no processo de desenvolvimento de software, atualizando os membros da equipe de forma automática quanto a mudanças ocorridas nos artefatos e para quais outros artefatos propagá-las.

Para prover percepção de forma automática, o ARARA conta com um sistema multi-agentes capaz de perceber mudanças nos artefatos, recuperar os artefatos

relacionados e notificar os membros sobre as mudanças percebidas e os artefatos candidatos a serem modificados também.

Os conceitos mapeados na ontologia do domínio e utilizados na construção dos artefatos do sistema são usados pelo ARARA no processo de rastreabilidade dos artefatos, mostrando que é possível fazer com que os artefatos sejam rastreáveis entre si pela utilização da ontologia do domínio no processo de desenvolvimento de software.

Dispor de um sistema que possa auxiliar no processo cooperativo de desenvolvimento de software é uma proposta de prover percepção da evolução do sistema, garantir a consistência do projeto e otimizar o esforço da equipe, uma vez que é evitada perda de trabalho com implementações em desacordo com os requisitos.

1.4. Estrutura da Dissertação

Essa dissertação está organizada da seguinte forma: o primeiro capítulo descreve o problema, a motivação e os objetivos da dissertação. O capítulo 2 apresenta os princípios que levam à descrição do problema e alguns trabalhos relacionados. O capítulo 3 descreve as técnicas sugeridas nesta dissertação para solução do problema. O capítulo 4 explica a proposta conceitual da solução e o capítulo 5 apresenta em detalhe a implementação da solução. O capítulo 6 apresenta os experimentos realizados para avaliação do ARARA e capítulo 7 apresenta as conclusões e proposta de trabalhos futuros.

Capítulo 2 - Princípios do Problema

Este capítulo apresenta os conceitos associados ao problema, que foram tema de estudo para a dissertação. São eles: percepção e rastreabilidade de requisitos. Termina apresentando trabalhos relacionados à dissertação.

2.1. Percepção *(Awareness)*

Awareness ou "percepção" é definida como sendo o conhecimento criado pela interação entre um agente e seu meio, ou seja, é ter conhecimento (ciência) do que está ocorrendo (ENDSLEY, 1995). O conhecimento pode ser alcançado pela observação do que os outros membros da equipe estão fazendo, ou pela análise do resultado do que tais membros fizeram. Em processos de desenvolvimento de software esse conhecimento pode ser alcançando pela análise das mudanças feitas nos artefatos do projeto ao longo do desenvolvimento do sistema.

"Workspace awareness" ou "percepção do espaço de trabalho" é o conhecimento adquirido a cada instante quanto ao estado da interação das pessoas com o espaço compartilhado de trabalho. A percepção ajuda as pessoas a interagirem entre si, compartilhando suas atividades e provendo contexto para possibilitar antecipação das atividades dos outros e reduzir os esforços necessários para coordenar tarefas e conhecimento (GUTWIN *et al.*, 1996, GUTWIN e GREENBERG, 2001).

Existem várias ferramentas colaborativas que permitem aos membros da equipe trabalharem juntos, mesmo estando em lugares diferentes. Como exemplo, podemos citar: email e *bulleting boards,* que promovem comunicação remota assíncrona, *instant messaging*, vídeo conferência e ambiente virtuais colaborativos que promovem comunicação remota síncrona. No entanto o uso de tais ferramentas no processo de

desenvolvimento de software não é eficaz para prover percepção de mudanças e propagação das mesmas.

2.1.1. Percepção em Processo de Desenvolvimento de Software

Em processo de desenvolvimento de software cuja equipe é grande ou distribuída geograficamente, manter a percepção de tudo que está acontecendo se torna muito complexo. À medida que as tarefas vão sendo divididas entre os membros da equipe, nem sempre há interação entre os membors de forma que todos saibam o que cada um está executando. É comum membros da equipe estarem executando seu trabalho sem ter noção do todo, ou como a parte que ele está executando irá contribuir para o produto final. O uso, por exemplo, de email ou *instant messaging* não é suficiente para prover a percepção a qual estamos abordando.

Soma-se ao problema a questão de que o processo de desenvolvimento de software gera um grande número de artefatos[1] e a cada mudança realizada em um artefato é preciso que os demais artefatos que estejam relacionados pelo mesmo contexto de negócio sejam revisados. Para que a propagação das mudanças ocorra é preciso que todos da equipe tenham:

1. conhecimento de todo o projeto para saber dos relacionamentos entre os artefatos e
2. conhecimento da ocorrência de mudanças nos artefatos durante o processo de desenvolvimento para propagação das mesmas.

Em geral, a construção e manutenção desses conhecimentos são feitas individualmente e sem um método específico.

[1] Artefatos referem-se a tudo o que é gerado no processo de desenvolvimento de um sistema, por exemplo: diagramas de atividades, diagrama de classes conceitual e de análise, diagrama de casos de uso, descrição do caso de uso, diagramas de sequência, diagramas de comunicação, código fonte, etc.

A Figura 1 ilustra como os artefatos se relacionam e como estão divididos por alguns dos membros da equipe de desenvolvimento. No exemplo, o desenvolvedor D é responsável pelo artefato Art_9 que se relaciona com os artefatos Art_2 e Art_{10}, que por sua vez se relacionam ambos com o artefato Art_3, que é de responsabilidade do desenvolvedor B.

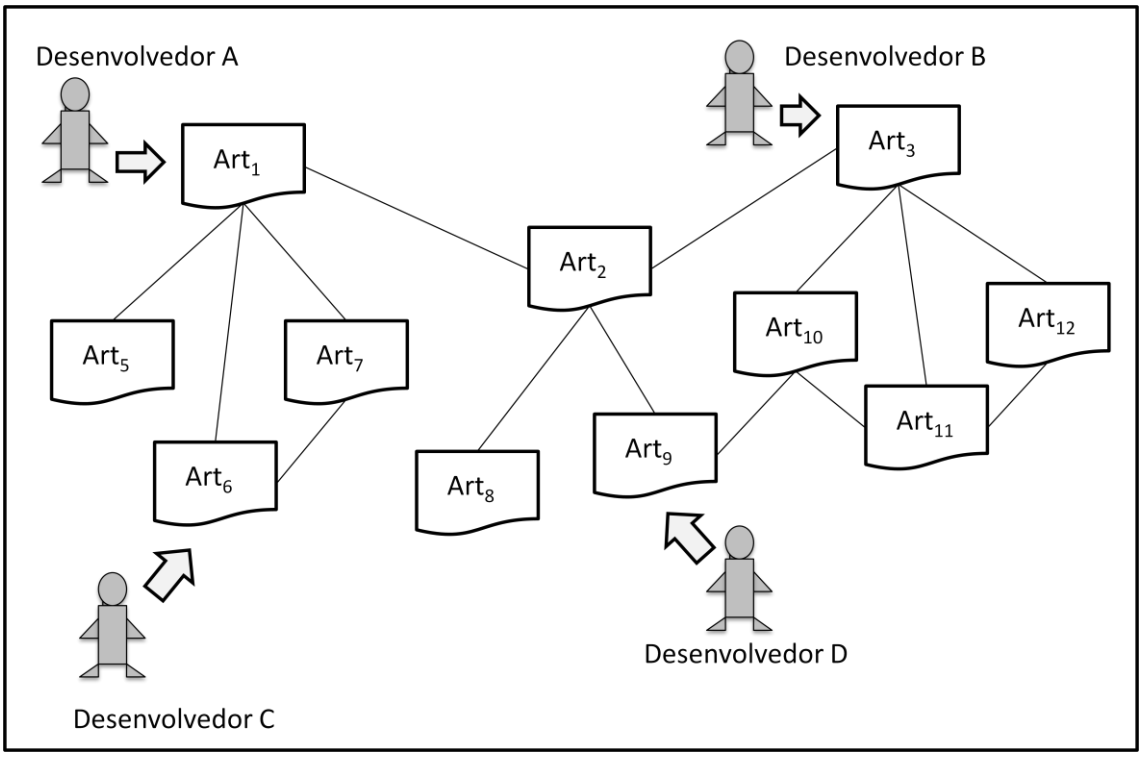

Figura 1. Relacionamento entre artefatos e distribuição entre os membros da equipe.

A Figura 2 simula a propagação de mudanças em um primeiro instante (T_1). Se o desenvolvedor B realizar uma modificação no artefato Art_3, a modificação pode impactar os demais artefatos que estão ligados ao artefato Art_3. Sendo assim os primeiros artefatos que podem receber a propagação da modificação são os artefatos que mais se relacionam com o Art_3, ou seja, são os que estão diretamente ligados ao Art_3 (Art_2, Art_{10}, Art_{11}, Art_{12}).

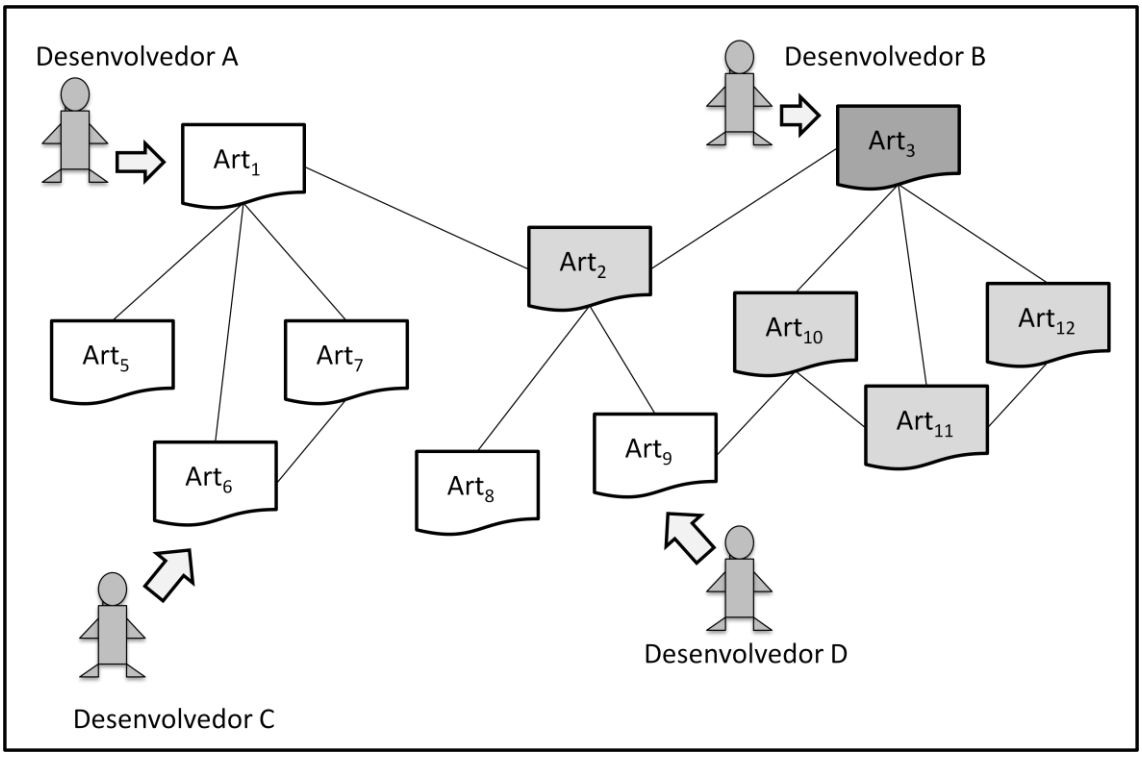

Figura 2. Propagação de mudanças no instante T_1.

A Figura 3 apresenta o caminho da propagação de uma mudança em um segundo momento (T_2).

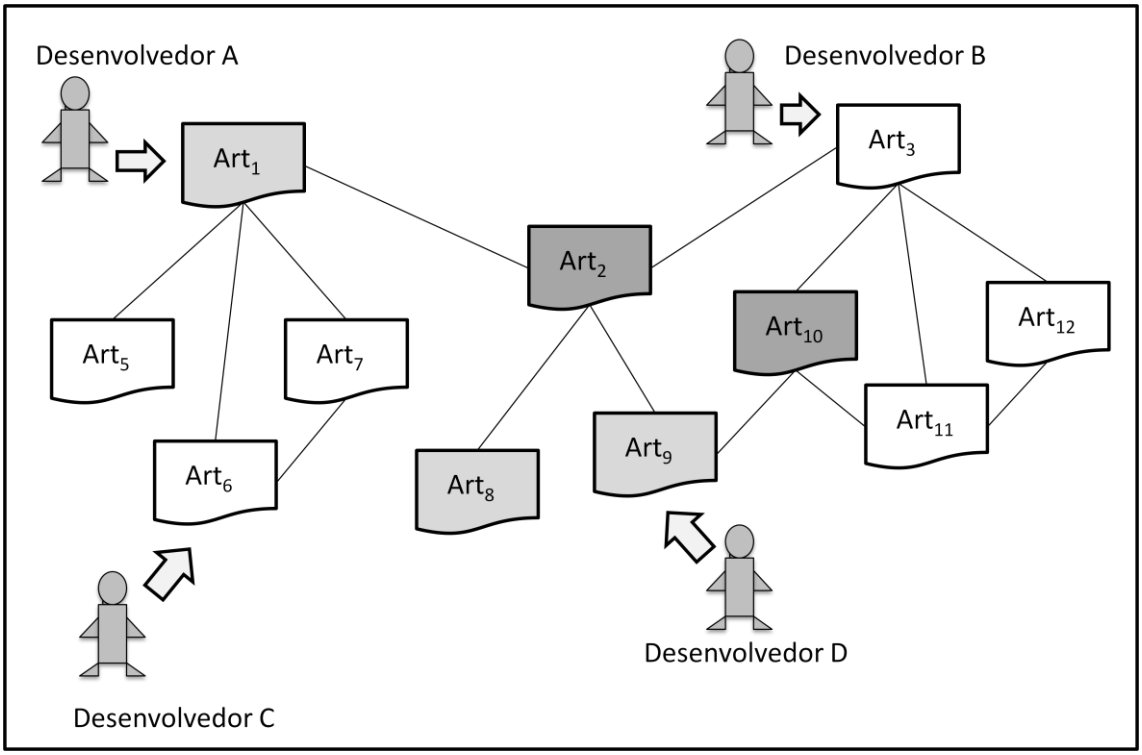

Figura 3. Propagação das mudanças pelos artefatos no intante T_2.

Sendo modificados os artefatos Art_2 e Art_{10}, após a propagação da modificação original realizada no artefato Art_3, os artefatos seguintes a serem propagadas tais mudanças são os artefatos Art_8, Art_9 e Art_1. Por sua vez Art_1 sendo modificado, poderá propagar a modificação para os artefatos Art_5, Art_6 e Art_7 em um terceiro instante (T_3).

Os exemplos apresentados na Figura 2 e Figura 3 mostram que para garantir a propagação de mudanças em artefatos e manter o projeto consistente com os requisitos mapeados é preciso manter conhecimento tanto dos relacionamentos entre artefatos quanto das mudanças que vão acontecendo ao longo do desenvolvimento do projeto.

2.1.2. Efeito das Ferramentas de Controle de Versão na Percepção

O uso de uma ferramenta de controle de versão permite identificar as mudanças que acontecem ao longo do desenvolvimento de um software. Uma ferramenta de controle de versão armazena em seu repositório[2] os artefatos de um projeto com cópias evolutivas dos artefatos à medida que estes vão sendo alterados. Essas ferramentas foram desenvolvidas com o objetivo principal de rastrear mudanças nos artefatos, feitas ao longo do tempo (COLLINS-SUSSMAN *et al.*, 2008).

A ideia básica no uso das ferramentas de controle de versão é fazer com que a equipe não trabalhe diretamente nos artefatos do projeto e sim em um cópia desses artefatos. Os artefatos do projeto são armazenados no repositório da ferramenta de controle de versão, como sendo a cópia principal dos artefatos do projeto e cada membro da equipe de desenvolvimento possui em seu ambiente de desenvolvimento uma cópia de trabalho desses artefatos. Para isso os membros da equipe de desenvolvimento fazem uma cópia dos artefatos do repositório para sua área de trabalho, fazem as modificações necessárias e depois escrevem tais mudanças no

[2] Repositório se refere ao local onde são armazenados os artefatos criados no processo de desenvolvimento de software.

repositório. Toda vez que há uma escrita no repositório a ferramenta de controle de versão gera uma nova versão dos artefatos que estão sendo escritos, dessa forma o repositório armazena todas as versões de cada artefato e o desenvolvedor nunca trabalha no repositório (LOURIDAS, 2006).

A Figura 4 ilustra tal processo de trabalho.

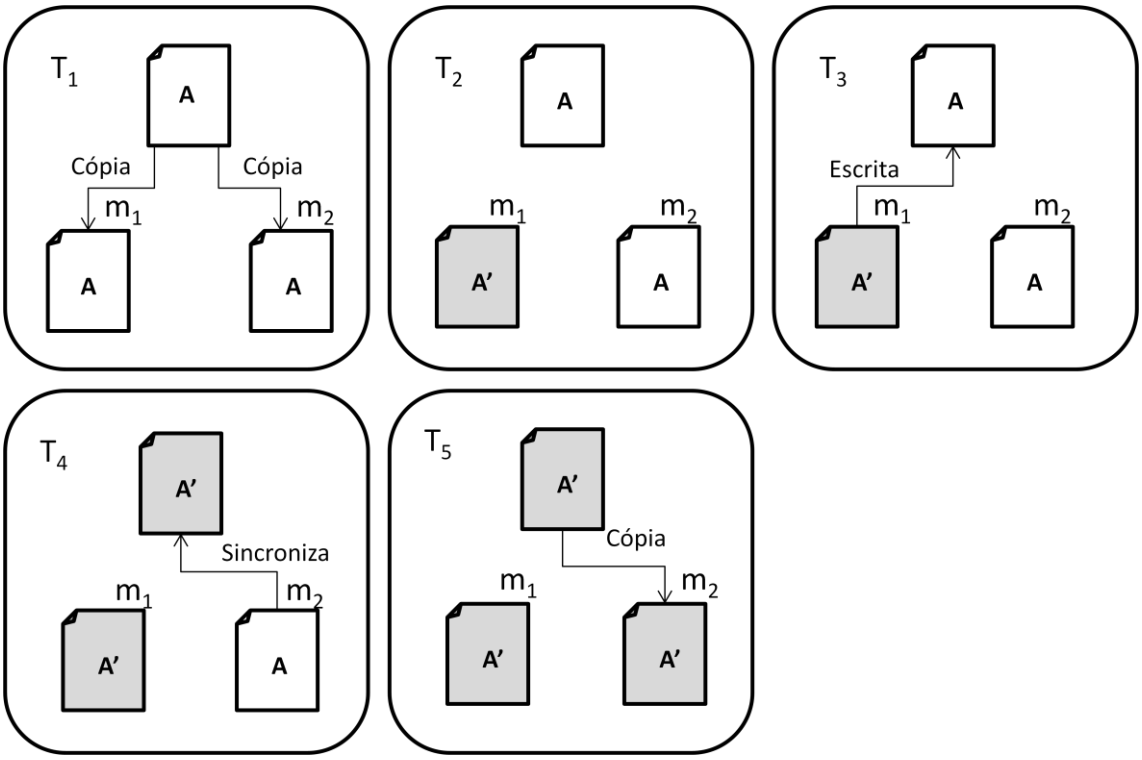

Figura 4. Processo de trabalho com uso de ferramenta de Controle de Versão (COLLINS-SUSSMAN *et al.*, 2008).

Em um primeiro momento (T_1) dois membros distintos (m_1 e m_2) da equipe de desenvolvimento fazem uma cópia do artefato A para seus repesctivos ambientes de trabalho. O membro m_1 faz alteração no artefato A em T_2 e em T_3 escreve no repositório o artefato modificado.

Uma escrita no repositório é definida como sendo um evento com mudanças nos artefatos que podem representar:

- inclusão de novos artefatos,
- exclusão de alguns artefatos e

- alteração nos dados dos artefatos que já fazem parte do repositório.

Quando m_2 sincroniza sua cópia de trabalho com o repositório descobre que o artefato A possui uma nova versão (A') e que a cópia que ele possui está desatualizada. Em T_5, m_2 faz uma cópia de trabalho da nova versão do artefato A, para seu ambiente de trabalho.

As ferramentas de controle de versão possibilitam examinar as diferenças entre a cópia de trabalho e a cópia principal informando linha a linha o que foi incluído, alterado ou excluído. Outras ferramentas, como por exemplo o plugin Subversive para o Eclipse, possibilitam examinar tais mudanças graficamente (Subversive 2010), quando o artefato é um código-fonte.

A análise dos artefatos modificados permite manter o conhecimento sobre o que está acontecendo no espaço de trabalho. No entanto, dependendo da escala do projeto de desenvolvimento de software, quando um membro da equipe sincronizar sua cópia de trabalho com o repositório haverá uma tal quantidade de artefatos desatualizados que será inviável a análise individual de cada artefato alterado para conhecimento das modificações que ocorreram no projeto.

O ideal é que o conhecimento seja passado entre os membros da equipe de forma automática por uma "entidade" que de certa forma possua o conhecimento geral do que está sendo produzido e dos relacionamentos entres os artefatos, podendo notificar os membros da equipe sobre o que está sendo produzido no projeto e sobre os impactos das novas produções nos artefatos já produzidos.

2.2. Rastreabilidade de Requisitos

Rastreamento de requisitos é considerado uma das práticas mais importantes no processo de desenvolvimento de software (CLELAND-HUANG *et al.*, 2007). É

também uma prática específica do CMMI (*Capability Maturity Model Integration*) especificada no REQM SP 1.4 (R*equirements Management Specific Practice 1.4*) (SEI, 2006), que define que é preciso manter uma rastreabilidade bidirecional entre os requisitos e o produto final.

As práticas recomendadas pelo ANSI/IEEE Standard 830-1998 para a especificação de requisitos de software quanto a rastreabilidade (IEEE, 1998) são de dois tipos:

1. **Rastreabilidade para trás** (*backward*) → rastreabilidade para versões anteriores do desenvolvimento,
2. **Rastreabilidade para frente** (*forward*) → rastreabilidade entre todos os documentos gerados pela especificação de requisitos de software.

O segundo tipo (rastreabilidade para frente) é especialmente importante quando o software entra em produção e subsequentemente em fase de manutenção. À medida que são realizadas modificações no produto é essencial que seja possível alcançar os requisitos que podem ser afetados por essas modificações.

Estes dois tipos de rastreabilidade foram definidos por GOTEL e FINKELSTEIN (1994) como sendo rastreabilidade Pré-Requisito de Software (Pré-RS), que concerne aos aspectos da produção dos requisitos e inclusão na especificação do projeto e Pós-Requisito de Software (Pós-RS), que concerne aos aspectos da propagação dos requisitos especificados nos demais artefatos do projeto. A Figura 5 ilustra essa definição.

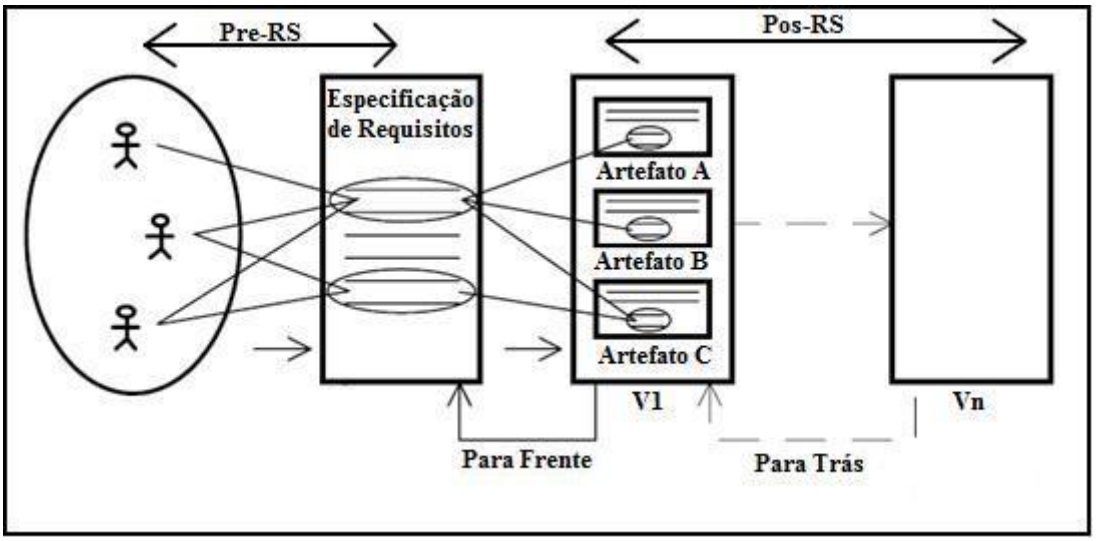

Figura 5. Dois tipos de rastreabilidade (Gotel & Finkelstein 1994).

Rastreabilidade também foi definida como uma técnica usada para prover relacionamento entre os requisitos, o projeto e a implementação do sistema (EDWARDS e HOWELL, 1991).

O processo de desenvolvimento de software cria grande quantidade e diversidade de artefatos. Alguns são derivados de outros dentro da modelagem do sistema e nem sempre esse relacionamento é feito de forma explícita dificultando a rastreabilidade entre os artefatos. É importante manter uma ligação entre os artefatos para facilitar o processo de rastreabilidade. Assim como também é necessário adotar técnicas automáticas, uma vez que a recuperação manual seria algo proibitivo considerando a grande quantidade de artefatos.

A falta de rastreamento automático entre esses diversos artefatos demanda da equipe conhecimento geral de todos os relacionamentos entre artefatos (explícitos ou implícitos) para que uma mudança inicial possa ser propagada pelos demais artefatos que estão relacionados, sem perda de integridade entre os relacionamentos.

CLELAND-HUANG *et al.* (2007) explicam que boas práticas para prover rastreabilidade automática recaem sobre três categorias. A primeira categoria descreve boas práticas para estabelecer um ambiente rastreável, a segunda descreve o conteúdo e

a estrutura dos artefatos para torná-los rastreáveis. A terceira descreve o processo para introduzir rastreabilidade automática em uma organização. As boas práticas para criar artefatos rastreáveis são**:**

1. usar um glossário do projeto, definido durante as etapas iniciais de levantamento de requisitos com os especialistas do domínio, durante todo o processo de desenvolvimento do produto,
2. escrever requisitos com qualidade, correção, não ambíguos, completos, consistentes, legíveis etc,
3. construir uma hierarquia da informação,
4. criar uma ponte de conhecimento intradomínios para criar ligação entre artefatos de diferentes domínios, que consequentemente podem ter termos diferentes para representar o mesmo conceito e
5. criar artefatos com conteúdos ricos em palavras e definições dos requisitos.

2.3. Trabalhos Relacionados

Essa seção apresenta, primeiramente, trabalhos relacionados ao tema percepção em trabalho cooperativo e em equipes de desenvolvimento distribuída geograficamente. Posteriormente apresenta os trabalhos relacionados ao tema recuperação de rastreabilidade.

2.3.1. Gossip

Um trabalho interessante foi apresentado por FARSHCHIAN (2000) como solução para o problema da falta de percepção em projetos de desenvolvimento

distribuído. Ele discute a necessidade de serviços genéricos para prover a percepção contínua do andamento do projeto. Introduz um modelo de percepcão com foco nas partes que compõem o produto em desenvolvimento e na propagação da percepção entre essas partes. Por fim apresenta o projeto e a implementação desse modelo em forma de um mecanismo de percepção que é chamado de Gossip.

A Figura 6 ilustra o modelo de percepção criado, que é chamado de Modelo de Percepção do Produto (*Product Awareness Model*). O modelo apresenta como é originada uma mudança e como deve ser feita a propagação da percepção através dos membros da equipe, ou seja, como um grupo que esteja trabalhando em uma determinada parte do desenvolvimento do sistema deve receber a propagação da informação de qualquer outra parte, especificamente no contexto que concerne ao seu trabalho.

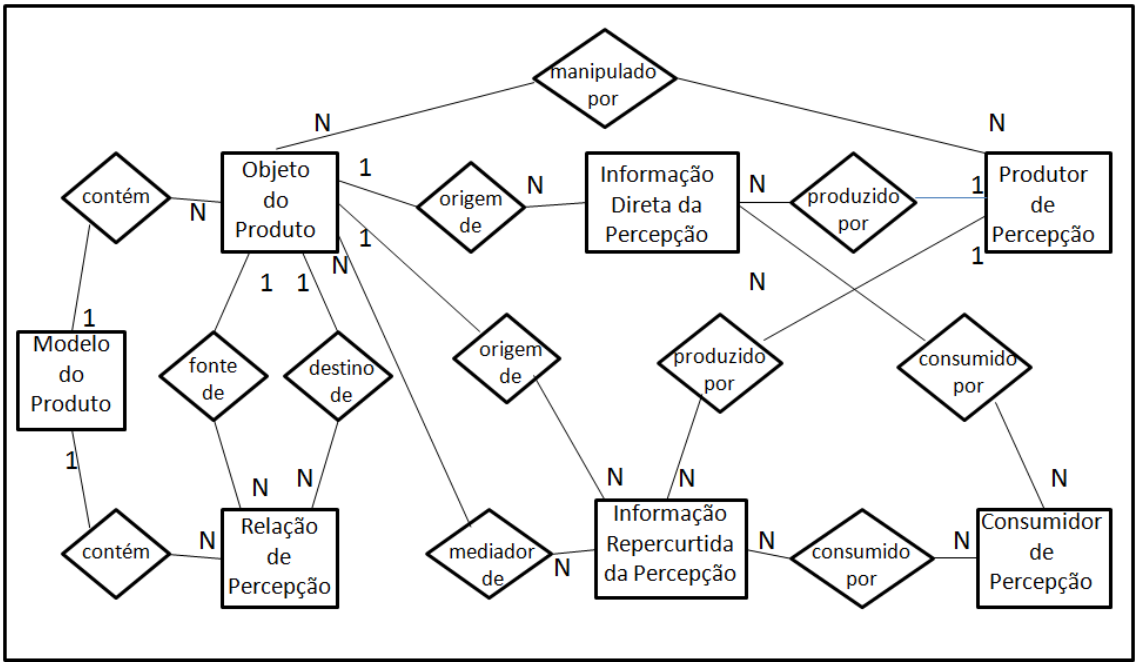

Figura 6. Modelo de percepção em Processo de Desenvolvimento de Software Distribuído (FARSHCHIAN, 2001).

Os modelos que compõem o Modelo de Percepção do Produto são:

- **Modelo do Produto** → é o produto em desenvolvimento compartilhado.

- **Objeto do Produto** → são as partes do produto, ou seja, os artefatos gerados no processo de desenvolvimento.
- **Relação de Percepção** → são os relacionamentos existente entre os artefatos.
- **Consumidor da Percepção** → são os membros da equipe em posição de receber informação (percepção).
- **Produtor da Percepção** → são os membros da equipe em posição de criar informação (gerar modificações, ou seja, gerar *awareness information*).
- **Informação Direta da Percepção** → é a mudança principal, que deve ser propagada para conhecimento dos consumidores de percepção.
- **Informação Repercurtida da Percepção** → é a repercursão da mudança nos demais artefatos do projeto, que também devem ser propagadas para conhecimento dos consumidores de percepção.

A implementação deste modelo de percepção é feita com um servidor de notificação especializado, com a responsabilidade de prover a propagação da informação para um diversificado conjunto de aplicações cliente. Estas aplicações são usadas para manipular estados compartilhados em um produto compartilhado.

2.3.2. Big Watch Framework

Existem sistemas colaborativos desenvolvidos com o objetivo de prover percepção em espaço de trabalho quanto a execução de atividades pelos membros de uma equipe de trabalho. Um exemplo desse esforço é o trabalho "*Big Watch Framework*" (BW Framework) que dá suporte à percepção de eventos passados. Foi desenvolvido de forma suficientemente flexível para acrescentar valor às ferramentas *groupware* já existentes e também para as futuras que venham a ser desenvolvidas (KIRSCH-PINHEIRO *et al.*, 2003).

BW Framework adota um mecanismo de percepção baseado em eventos e sua estrutura é composta por três fases:

1. registro,
2. monitoramento e
3. notificação.

Na primeira fase, os eventos de interesse para a percepção são registrados no framework.

Na segunda fase, as atividades acontecem dentro da ferramenta *groupware.* Uma vez que uma das atividades que foi registrada é executada a ferramenta passa para o BW Framework o evento relacionado à atividade executada.

A última fase consiste em informar aos usuários sobre o que aconteceu. Para isso o framework executa uma filtragem nas informações disponíveis, baseada em especificações sobre as preferências dos usuários ou papéis de usuários. As preferências são de quais atividades (dentre o grupo registrado no framework) devem ser notificadas aos usuários e qual o intervalo de tempo em que os usuários estão interessados em receber a notificação.

2.3.3. Rastreabilidade entre Código e Documentação

Existem também diversos trabalhos com foco em recuperação de rastreabilidade em artefatos de desenvolvimento de software. As metodologias são baseadas na aplicação de técnicas de recuperação de informação, como modelo de espaço vetorial, modelo probabilístico, indexação de semântica latente, etc.

Em MARCUS *et al.* (2005), a rastreabilidade é feita entre documentação e código fonte, e a metodologia semi-automática de recuperação de rastreabilidade é baseada na extração, análise e representação matemática dos comentários e identificadores do código fonte usando indexação de semântica latente.

Em ANTONIOL *et al.* (2002), o método implica em usar identificadores extraídos do código fonte do sistema, como *queries*, para recuperar os documentos relevantes de especificação do sistema. Para isso, assume-se que os programadores usam nomes (mnemônicos) significativos para os identificadores de classes, métodos, atributos e parâmetros e consequentemente esses mnemônicos são usados para servir como índice do processo de recuperação dos documentos associados. Os modelos de recuperação de informação usados foram os Modelo de Espaço Vetorial e o Modelo Probabilístico.

ZHANG *et al.*(2006) fez uso de ontologia em seu trabalho para promover integração entre a documentação do projeto e o código fonte do sistema. Através da representação ontológica para ambos os tipos de artefatos (documentos e código fonte), foi possível, em uma etapa seguinte, criar as ligações entre ambos, proporcionando também inferir ligações implícitas entre os artefatos.

Capítulo 3 - Técnicas Propostas para a Solução

Este capítulo apresenta as técnicas propostas nessa dissertação para prover percepção no processo de desenvolvimento de software. São eles: ontologia e sistema multi-agentes. A ontologia será aplicada como meio de relacionar os artefatos gerados no processo de desenvolvimento de software, e o sistema multi-agentes será responsável por fornecer a percepção para a equipe de forma automática, sempre que modificações ocorrerem no projeto.

3.1. Ontologia

Segundo a definição clássica, ontologia é a especificação explícita de uma conceituação (GRUBER, 1995).

Conceituação é uma visão abstrata do mundo que tentamos representar de alguma forma. Sendo assim, uma ontologia é uma especificação explícita da tentativa de representar a visão de algum domínio. É uma forma de tornar clara a estrutura do conhecimento e formalizá-la para representar adequadamente um domínio e permitir uma melhor comunicação entre humanos e software. Formalizar significa que a especificação do conhecimento é codificada em uma linguagem cujas propriedades formais são bem entendidas, sendo possível compartilhar e reutilizar esse conhecimento (USCHOLD e GRUNINGER, 2004).

USCHOLD e GRUNINGER (1996) subdividem os espaços de uso da ontologia em três categorias: comunicação entre pessoas, interoperabilidade entre sistemas e benefícios para a engenharia de sistemas. Os benefícios, em particular, são:

- **Reusabilidade** → o conhecimento compartilhado é a base para uma codificação formal de classes, atributos, processos e seus inter-

relacionamentos no domínio de interesse. Essa representação formal pode ser reutilizada ou compartilhada em sistemas de software.

- **Confiabilidade** → uma representação formal também torna possível a automação na verificação de consistências resultando em um software mais confiável.
- **Especificação** → o entendimento comum pode dar assistência ao processo de levantamento de requisitos e na definição da especificação do sistema. Isso é especialmente verdadeiro quando o levantamento de requisitos envolve grupos diferentes que usam terminologias diferentes no mesmo domínio ou em domínio múltiplo.

Um dos objetivos da ontologia é prover uma especificação fiel de uma unidade de conhecimento em que o modelo conceitual resultante representa de forma adequada o domínio ou o contexto de conhecimento de interesse (EDGINGTON *et al.*, 2005).

A ontologia pode ser vista como sendo o coração da representação do conhecimento de um dado domínio, e tudo em seguida pode ser derivado da ontologia. Com o apoio de ontologias, ou com a conceitualização que represente o conhecimento, é possível ter um vocabulário que represente o conhecimento de um domínio.

O primeiro passo para construir um sistema efetivo de representação de um domínio e vocabulário é fazer uma efetiva análise ontológica do domínio. Fracas análises levam a bases de conhecimento incoerentes. (CHANDRASEKARAN *et al.*, 1999).

3.1.1. Tipos de Ontologia

As ontologias foram classificadas nas seguintes categorias (GUARINO, 1997), conforme ilustrado na Figura 7.

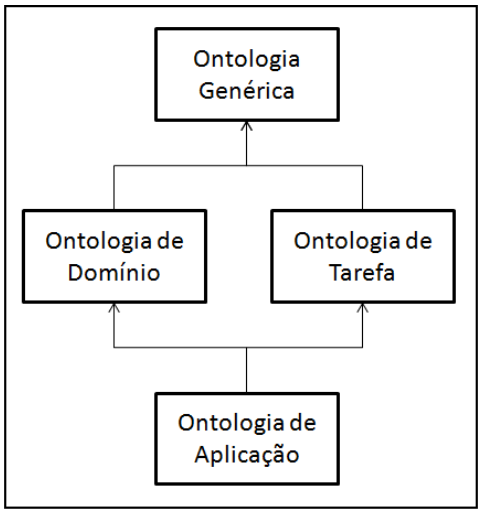

Figura 7. Tipos de Ontologias e suas especializações.

- **Ontologias Genéricas** → descrevem conceitos gerais, tais como espaço, tempo, matéria, objeto, evento, ação, etc., que são independentes de um problema ou domínio particular.
- **Ontologias de Domínio** → expressam os conceitos de domínios genéricos, descrevendo o vocabulário relacionado ao domínio.
- **Ontologias de Tarefas** → expressam os conceitos de tarefas ou atividades genéricas, descrevendo o vocabulário apropriado para representar tais conceitos.
- **Ontologias de Aplicação** → descrevem conceitos dependentes tanto do domínio quanto das tarefas. Estes conceitos frequentemente correspondem a papéis desempenhados por entidades do domínio quando da realização de uma certa atividade.

Uma categoria adicional é a **Ontologia de Representação,** que descreve a classificação dos formalismos usados pela linguagem de representação do conhecimento (VAN HEIJST *et al.*, 1997)

A **Ontologia de Domínio** é a categoria mais comumente desenvolvida enfocando áreas diversas (FALBO, 1998).

3.1.2. Metodologia

LÓPEZ (1999) faz uma revisão nas metodologias que surgiram desde 1995 até 1999. A primeira metodologia criada para desenvolver uma ontologia foi sugerida por USCHOLD e KING em 1995. Em 1996 AMAYA BERNERAS *et al.* apresentaram o método usado na construção da ontologia do domínio de redes elétricas. Na mesma época, GÓMEZ-PÉREZ *et al.* apresentaram o METHONTOLOGY, que foi extendido em publicações seguintes até 1997. Em 1997, uma metodologia foi proposta por SWARTOUT *et al* para a construção de ontologias baseadas na ontology SENSUS.

A metodologia sugerida por USCHOLD e KING (1995) está representada na Figura 8.

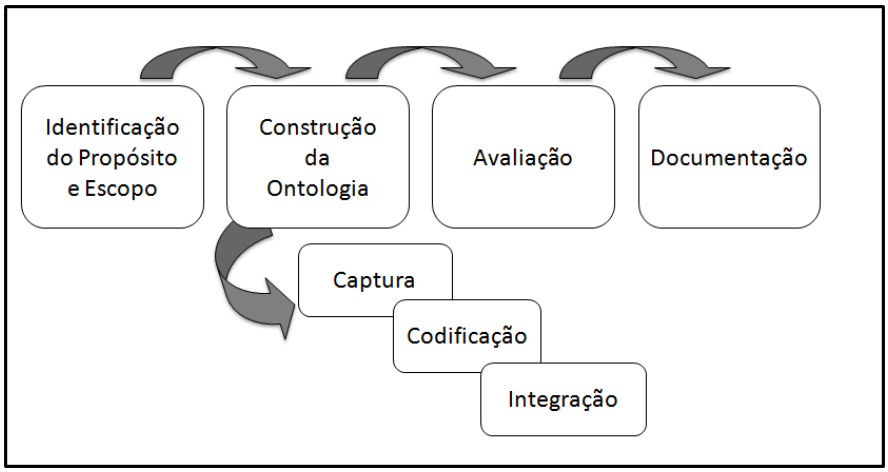

Figura 8. Representação da Metodologia sugerida por Uschold e Martin.

As etapas descritas nessa metodologia são:

- **Identificação do Propósito e Escopo** → consiste em definir por que a ontologia está sendo criada e onde ela será usada.
- **Captura** → identifica os conceitos e relacionamentos importantes no domínio de interesse.
- **Codificação** → envolve a representação explícita, em uma linguagem formal, do conhecimento adquirido na etapa anterior.

- **Integração de ontologias existentes** → durante os passos de captura e codificação é preciso avaliar como e onde usar uma ontologia já existente.
- **Avaliação** → uma boa definição para avaliação é fazer um julgamento técnico da ontologia, o ambiente o qual está associada e a documentação com relação a uma referência. A referência, por exemplo, pode ser uma especificação de requisitos.
- **Documentação** → é recomendado que sejam estabelecidas diretrizes para documentar a ontologia, podendo haver diferentes diretrizes de acordo com o tipo e o propósito da ontologia.

Uma ontologia define um vocabulário comum e uma estrutura de informação para pesquisadores e especialistas do domínio compartilharem conhecimento. Um especialista é caracterizado por possuir um desempenho superior em uma atividade de domínio específico (JOHNSON *et al.*, 1987).

A técnica predominante para construção da ontologia é a técnica que envolve o uso do especialista do domínio na construção e validação, usando lógica formal (López 1999).

3.1.3. Elementos da Ontologia

A ontologia é composta por instâncias, classes e propriedades que serão ilustradas nas figuras a seguir utilizando a mesma representação adotada por HORRIDGE *et al.* (2004).

As instâncias, ou indivíduos, representam objetos do domínio os quais estamos interessados em representá-los. A Figura 9 apresenta algumas instâncias de um domínio de negócio.

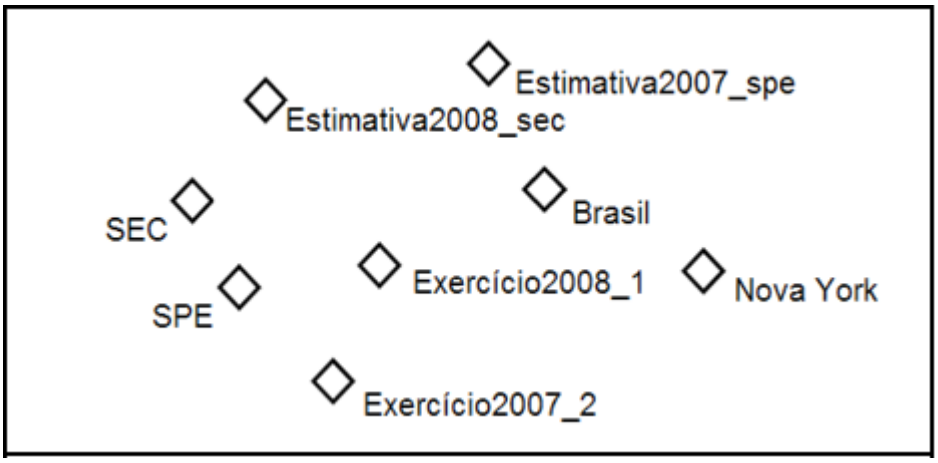

Figura 9. Exemplo de instâncias na ontologia.

O conjunto de instâncias de mesmas características compõem uma classe. Desta forma, as classes são interpretadas como um conjunto de instâncias. No exemplo apresentado na Figura 10, foram criadas as seguintes classes: Exercício, País, Critério de Estimativa e Estimativa de Reserva para agrupar os indivíduos apresentados na Figura 9.

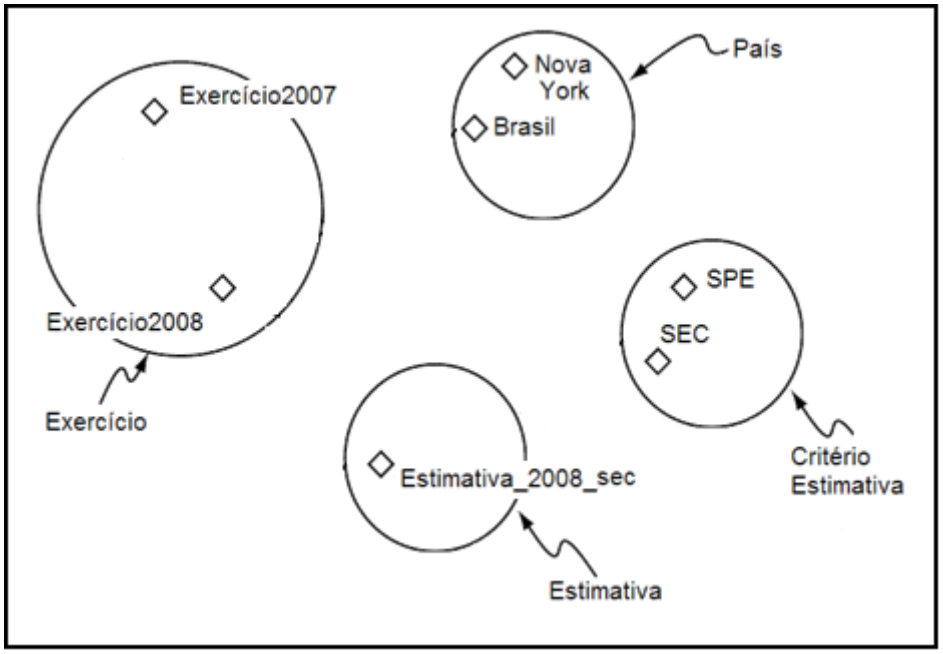

Figura 10. Exemplo de classes na ontologia.

É possível também criar subclasses nas classes. Neste caso, os indivíduos de uma subclasse também são indivíduos da classe e as subclasses herdam as expressões e restrições da classe pai.

As classes apresentam propriedades (atributos e relacionamentos) que podem ser:

- **Objeto** → representa o relacionamento entre duas instâncias, ou seja, é uma propriedade que liga instâncias de uma classe a instâncias de outra classe. Essas classes serão definidas como domínio e faixa. Um exemplo apresentado na Figura 11 como propriedade Objeto é: *PaísÉ*: liga as instâncias da classe domínio *Exercício* às instâncias da classe faixa *País*.

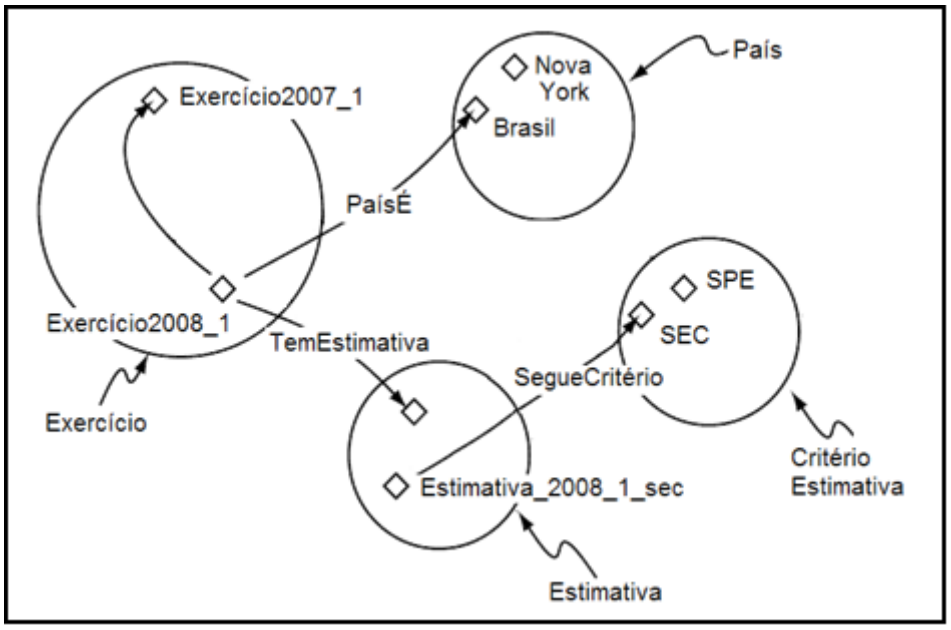

Figura 11. Exemplo de propriedades objeto na ontologia.

- **Tipo de dado** → representa o relacionamento de uma instância de um domínio e um *XML Schema Datatype value* (XML Schema 2004) ou um *rdf literal* (RDF Primer 2004). Ou seja, representa o atributo de uma classe. Um exemplo de propriedade tipo de dado é apresentado na Figura 12.

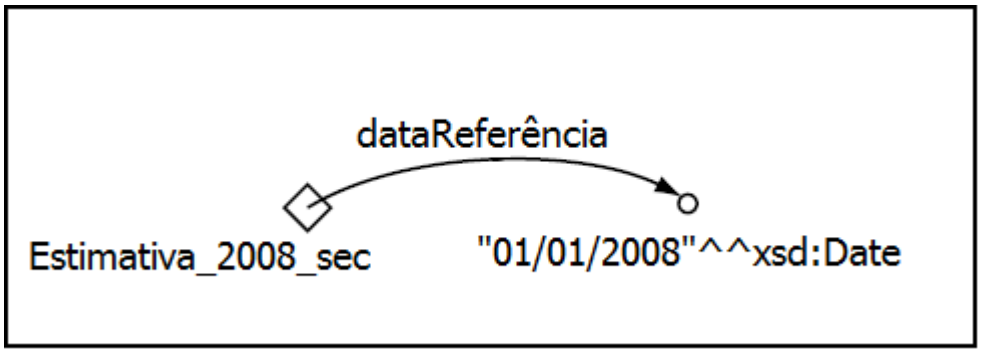

Figura 12. Exemplo de propriedade tipo de dado na ontologia.

3.1.4. Linguagem OWL

A linguagem mais recente de desenvolvimento de ontologias é a OWL (*Ontology Web Language*) do W3C (*World Wide Web Consortium*). Foi desenvolvida para uso em aplicações que necessitam processar ao invés de apenas apresentar o conteúdo da informação. Pode ser usada para representar explicitamente o significado dos termos em um vocabulário e o relacionamento entre esses termos (OWL, 2009).

OWL possui três sublinguagens, com diferentes níveis de expressão:

- **OWL Lite** → é uma sublinguagem menos complexa que a OWL DL. Dá suporte à classificação hierárquica e restrições simples. Por exemplo: na restrição de cardinalidade só são permitidos os valores 0 ou 1. Além disso prove uma rápida migração para tesauros e outras taxonomias.
- **OWL DL** → é chamada de DL (*Description Logic*) pela correspondência feita com a Lógica de Descrição[3] (BAADER *et al.*, 2003), um campo de pesquisa que estuda as lógicas que formam a base formal do OWL. Esta sublinguagem dá suporte à máxima expressividade, mantendo

[3] É um formalismo de representação do conhecimento que representa o conhecimento de um domínio de aplicação. Uma das características dessa linguagem é a semântica formal com base na lógica.

computabilidade[4] e decidibilidade[5]. Inclui todas as construções de linguagem do OWL, mas sendo usadas sob certas restrições.

- **OWL Full** → oferece máxima expressividade dentre todas as linguagens e liberdade sintática do RDF, mas sem garantias computacionais. Essa sublinguagem permite que uma ontologia aumente o vocabulário pré-definido de RDF ou OWL.

Uma lista dos construtores das sublinguagens OWL Lite é apresentado em anexo.

3.1.5. Ferramenta Protégé

Protégé é um editor de ontologias *opensource* e um framework de base de conhecimento. Com ele é possível modelar ontologias de duas formas: através do editor Protégé-Frame ou do editor Protégé-OWL. O editor Protégé-OWL permite:

- carregar e criar ontologias em RDF e OWL,
- editar e visualizar classes, propriedades e regras SWRL (*Semantic Web Rule Language*) (SWRL 2004),
- definir características lógicas das classes como expressões OWL,
- editar instâncias OWL para Web Semântica *markup e*
- executar *reasoners*[6].

O Protégé não favorece nenhuma linguagem específica. Ao contrário, possui facilidades para adaptar a ferramenta a diferentes linguagens, através do uso de plug-ins.

É possível exportar os modelos construídos usando os formatos RDF, RDF-S e XML (PROTÉGÉ, 2010).

[4] Todas as conclusões são garantidas de serem computadas.
[5] Todas as computações terminarão em tempo finito.
[6] Reasoner é um serviço que pode inferir informações que não estão explícitas na ontologia.

3.1.6. Análise de Domínio e Ontologia

O termo Análise de Domínio foi introduzido em 1980 por James Neighbors. A análise de domínio, diferente da análise de sistemas, não se preocupa com a ação específica em um sistema. Pelo contrário, ela se preocupa com quais ações e objetos ocorrem em todos os sistemas de uma área de aplicação, ou seja, de um domínio (NEIGHBORS, 1980).

A análise de domínio culmina na criação de um modelo geral dos objetos do domínio. O modelo de domínio pode ser gerado usando metologias, como, por exemplo, a linguagem específica de domínio. A linguagem pode ser criada para representar tais objetos, relações e operações que posteriormente podem ser usadas para descrever outros sistemas de um mesmo domínio. As principais fases da análise de um domínio são:

- **Planejamento** → esta fase é marcada por atividades como análise do negócio e análise de risco. Define-se, considerando custo e benefício se vale a pena ser feita a análise de domínio.
- **Aquisição e seleção dos dados** → Consiste em identificar as fontes de dados disponíveis.
- **Análise dos dados e modelagem do domínio** → consiste em avaliar o conhecimento capturado na etapa prévia para, em seguida, modelar o conhecimento , identificando-se entidades, relações, funções e axiomas.

As áreas de construção de ontologias e análise de domínio apresentam inúmeras similaridades, e faz sentido pensar em um modelo de processo para uma análise de domínio orientada a ontologias (GUIZZARDI, 2000).

A ontologia permite uma representação dos conceitos do domínio de negócio e das relações semânticas existentes entre os mesmos. Desta forma a ontologia pode atuar como complemento aos modelos de processos de negócio, facilitando o entendimento e explicitando os conceitos do domínio de negócio, com o propósito de identificar requisitos de sistema através dos requisitos do domínio de negócio (CAPELLI *et al.*, 2007).

Sistemas deveriam ser escritos com um comprometimento com o modelo de representação do domínio – entidades, atributos e relacionamentos do domínio - pois a estrutura de dados e os procedimentos explícita ou implicitamente fazem relação com a ontologia do domínio.

3.2. Sistema Multi-Agentes

Sistemas Multi-Agentes são compostos por agentes capazes de interagir entre si, trabalhando em conjunto para resolver um determinado problema. Com agentes, um sistema complexo pode ser quebrado em subsistemas descentralizados e cooperativos (WOOLDRIDGE, 1997). Existe, inclusive, uma arquitetura de multi-agentes para aplicações em Engenharia de Software Cooperativa por ser melhor em termos de simplicidade e flexibilidade (WANG *et al.*, 1999).

A Figura 13 apresenta a estrutura típica de um sistema multi-agentes, onde cada agente tem uma esfera de influência nesse ambiente compartilhado.

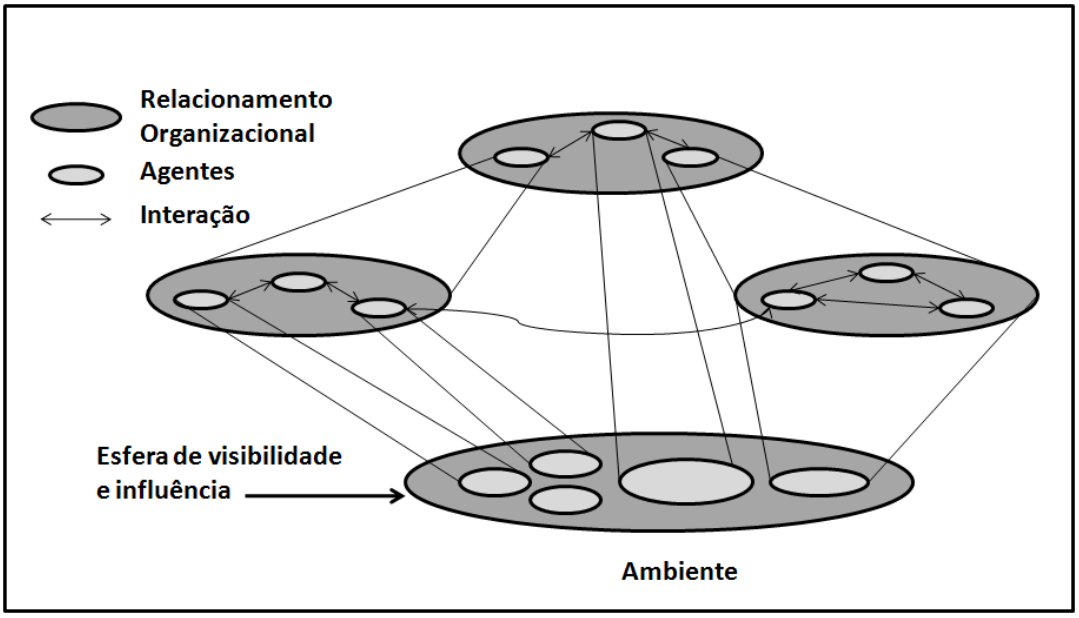

Figura 13. Estrutura típica de um sistema multi-agentes (JENNINGS, 2001).

Existem alguns contextos organizacionais que dão suporte à interação entre os agentes. Por exemplo, eles podem ser *peers* trabalhando juntos em equipe, ou pode haver um agente administrando os demais. A estrutura apresentada na Figura 13 mostra que adotar uma estratégia orientada a agentes significa decompor o problema em componentes múltiplos e autônomos, os quais podem agir e interagir de forma flexível para alcançar seus objetivos. Os modelos de abstração que definem o conjunto da orientação a agentes são: agentes, interações e organizações (JENNINGS, 2001).

3.2.1. Agentes

Os agentes são capazes de *sentir* o ambiente onde estão situados e agir segundo um repertório de possíveis ações a serem executadas com o objetivo de mudar o estado do ambiente (BORDINI *et al.*, 2007). Os agentes têm como características:

- autonomia,
- proatividade,
- reatividade e

- habilidade social.

Por serem **autônomos,** os agentes, sem a intervenção humana, tomam a iniciativa e decidem o que fazer, baseados no evento que os dispara. Por serem **proativos**, é esperado que os agentes façam o possível para alcançar seus objetivos. São **reativos** a mudanças, sendo capazes de responder em tempo as mudanças ocorridas no ambiente. A **habilidade social** é a capacidade que os agentes têm de cooperarem e coordenarem suas atividades com os demais agentes do ambiente (BORDINI *et al.*, 2007) (WANG *et al.*, 1999).

3.2.2. Framework COPPEER

COPPEER é um framework para sistema multi-agentes desenvolvido para dar suporte a aplicações cooperativas P2P (MIRANDA *et al.*, 2006).

Uma aplicação P2P desenvolvida no COPPEER será definida pelo seu **ambiente**, que é composto por células interconectadas que compartilham um espaço comum, conforme ilustrado na Figura 14. Cada computador que queira acessar a aplicação deve criar uma agência relacionada a essa aplicação executando seus agentes.

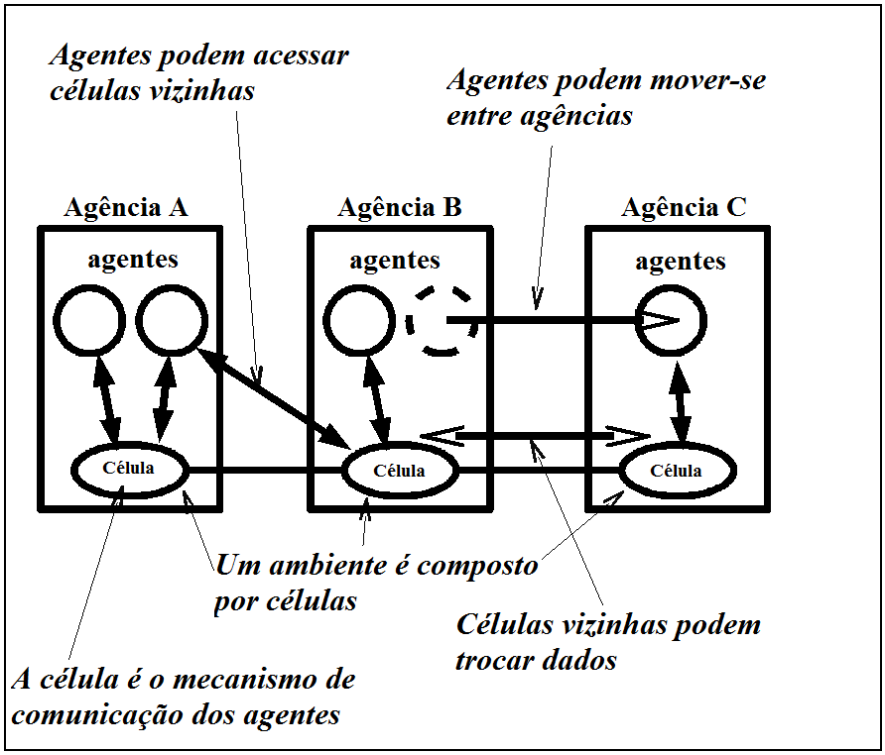

Figura 14. Framework COPPEER (MIRANDA *et al.*, 2007).

Agências gerenciam ambientes, células e agentes. Podem participar em vários ambientes simultaneamente, mas gerenciando apenas uma célula por ambiente.

Uma **célula** oferece aos agentes uma interface contendo operações para:

- escrita ou leitura de registros (ou entradas),
- inscrição para notificações sobre escrita nos registros e
- estabelecimento ou encerramento de conecções com outras células.

Uma **entrada** (ou registro) é um objeto de dados que pode ser armazenado em uma célula. É usada para carregar todos os dados trocados entre os agentes.

Um **agente** é um pedaço de software associado a um ambiente e que acessa a célula local ou vizinha e se move entre as agências para executar a computação distribuída.

A arquitetura do COPPEER é composta por quatro camadas, conforme ilustrado na Figura 15.

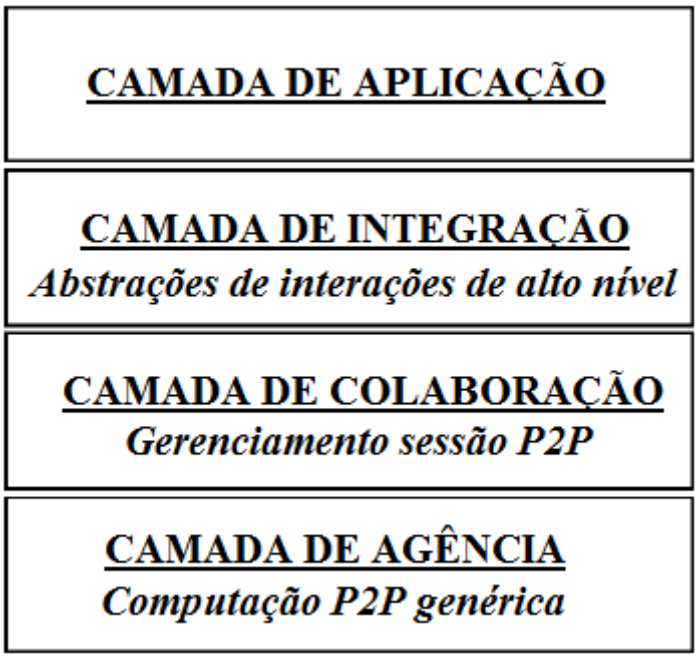

Figura 15. Arquitetura do Framework COPPEER (MIRANDA e XEXÉO, 2005).

Cada camada possui as seguintes responsabilidades:

- **Camada de agência** → é responsável por executar computações genéricas P2P para atender às camadas superiores.
- **Camada de colaboração** → é responsável pelo gerenciamento de sessão e locação de aplicação.
- **Camada de integração** → é responsável por oferecer distribuição de alto nível para as aplicações.
- **Camada de aplicação** → é responsável por serviços relacionados a colaboração.

Capítulo 4 - ARARA

Este capítulo apresenta o modelo conceitual do ARARA e as responsabilidade de cada grupo de agentes de forma abstrata. Apresenta também os requisitos para aplicação do ARARA em um processo de desenvolvimento de software.

4.1. Proposta do ARARA

O ARARA é um sistema sugerido para uso em projetos onde os requisitos mudam constantemente e a percepção dessas mudanças pode ocorrer por apenas uma partes dos membros da equipe, ou mesmo passar despercebida.

Uma vez que os artefatos mudam constantemente no processo de desenvolvimento de software, os membros da equipe de desenvolvimento devem tomar conhecimento das mudanças com o objetivo de finalizarem suas tarefas, mantendo o projeto consistente e com os artefatos produzidos de acordo com os requisitos definidos para o software em produção.

Sempre que uma mudança acontece, o ARARA tem como objetivo notificar aos membros da equipe sobre os possíveis artefatos que podem ser impactados com tal mudança. As técnicas propostas para a implementação da solução são: uso de ontologias e agentes.

O princípio utilizado para relacionar os artefatos é o uso da ontologia do domínio de negócio aplicada nos artefatos do projeto. A ontologia não apenas proverá um vocabulário representativo dos conceitos do domíno de negócio, como proverá informações dos relacionamentos entre os conceitos da ontologia.

Fazendo da ontologia o centro de todos os artefatos gerados no processo de desenvolvimento de software, será possível rastrear as associações entre os artefatos.

Para isso, a boa prática a ser adotada para prover a rastreabilidade automática é o uso dos conceitos mapeados na ontologia na descrição dos artefatos do projeto. No processo de rastreabilidade tais conceitos servirão como identificadores que auxiliarão na busca pelos artefatos relacionados ao artefato modificado.

A base para automatizar tal processo é o uso de agentes, que proverão à equipe a percepção das mudanças ocorridas no projeto, notificando-a sobre quais mudanças ocorrerão no decorrer do desenvolvimento e alertando sobre os artefatos que podem ser afetados para que a equipe possa fazer a análise e correção do que for preciso.

Nesse contexto, ARARA é um sistema cooperativo que oferece percepção de mudanças em requisitos de forma automática, com apoio de sistema multi-agentes.

4.2. Requisitos para Aplicação do ARARA

Para uso do ARARA no desenvolvimento de um projeto alguns requisitos devem ser atendidos. São eles:

- dispor de uma Base de Conhecimento do Domínio (BCD),
- dispor de um Repositório de Artefatos (RA) e
- aplicar os conceitos da ontologia no preparo dos artefatos.

A Base de Conhecimento do Domínio é a base onde ficam armazenados os conceitos que estão representados na ontologia do domínio do negócio. A ontologia representativa do domínio de negócio deve ser preparada em uma etapa prévia do projeto onde os conceitos, seus significados e relacionamentos são levantados com os especialistas do domínio quando o modelo de processo do negócio é preparado.

O Repositório de Artefatos é usado para armazenar, sobre um controle de versão, todos os artefatos que serão construídos ao longo do desenvolvimento do projeto.

Os artefatos em construção devem ser criados com base na ontologia do domínio para que os conceitos da ontologia sejam aplicados na descrição dos mesmos.

4.3. Modelo Conceitual do ARARA

A Figura 16 apresenta as ações realizadas pelo ARARA.

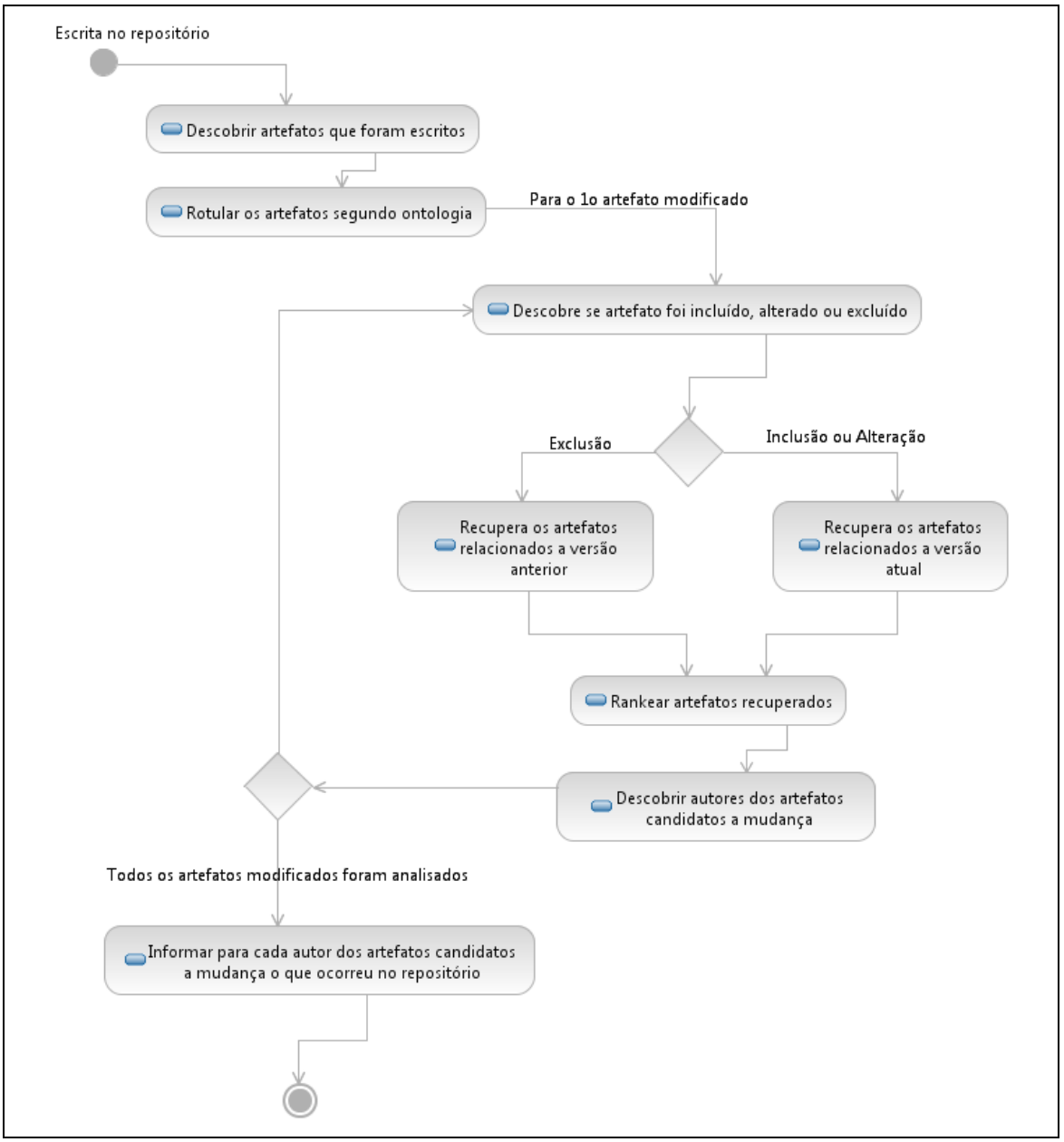

Figura 16. Fluxo conceitual das ações dos agentes no ARARA.

Os agentes trabalharão de forma a:

- perceber mudanças no Repositório de Artefatos,
- descobrir os artefatos modificados e o tipo de mudança (inclusão, alteração ou exclusão)
- rotular[7] os artefatos modificados com base na ontologia do domínio para que através desses rótulos seja possível relacionar os artefatos uns aos outros,
- identificar, com base na ontologia do negócio, os artefatos que podem ser impactados por estarem associados aos artefatos modificados e
- identificar e notificar aos responsáveis dos artefatos que podem ser impactados por propagação das mudanças ocorridas.

A Figura 17 apresenta de forma ilustrada a solução proposta para o problema.

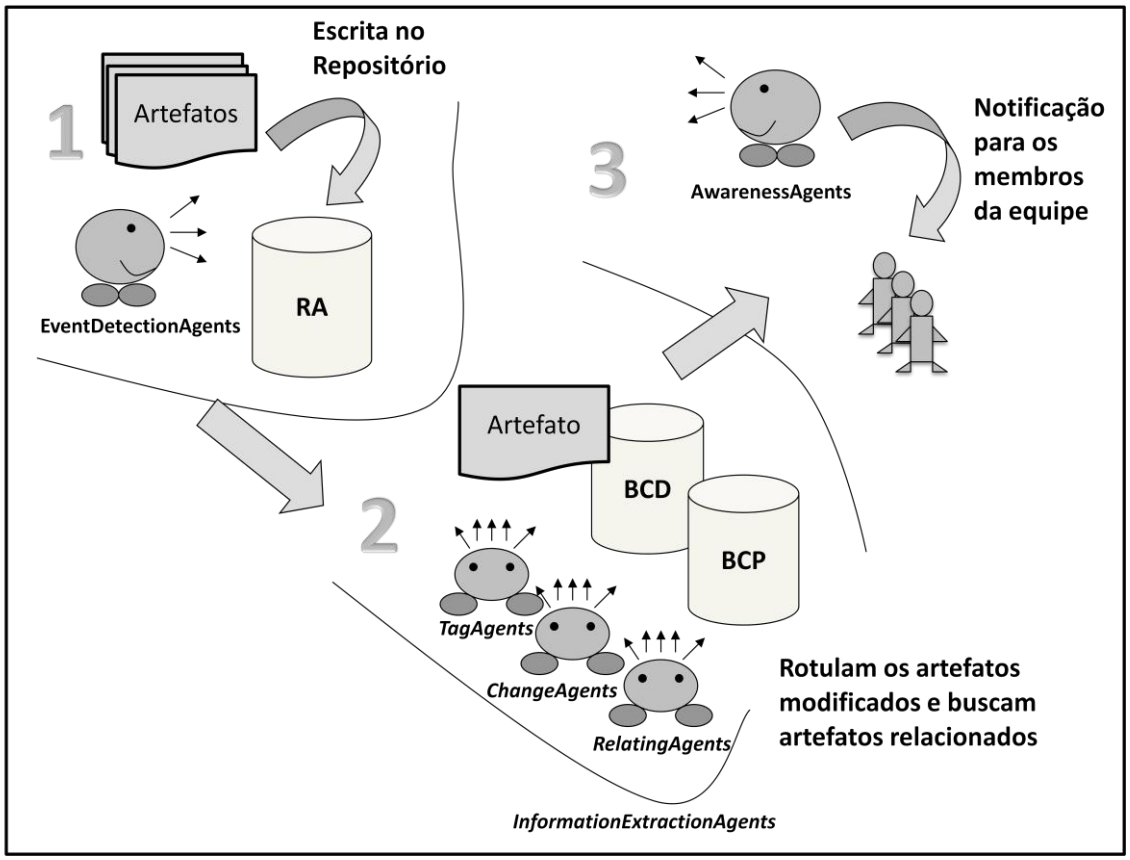

Figura 17. Diagrama de interação dos agentes de percepção de mudanças.

[7] Rotular se refere a definição de um conjunto de *tags* definidas para um artefato. Uma *tag* é definida como sendo um conceito do domínio de negócio representado na ontologia e utilizado no artefato.

O processo do funcionamento dos agentes no ARARA ocorre em três etapas.

A **etapa 1** é quando ocorre a percepção das mudanças em um projeto. O ponto inicial que dispara o funcionamento do ARARA é quando os agentes percebem a escrita no RA com mudanças na cópia principal dos artefatos do projeto.

A **etapa 2** é o momento quando é feito a análise da mudança para buscar os artefatos relacionados. Nessa etapa os agentes trabalham construíndo a Base de Conhecimento do Projeto (BCP). Além disso, eles também trabalham na identificação das mudanças que ocorreram entre a nova versão e a anterior do artefato modificado e na busca dos artefatos que podem ser afetados. A BCP é a base de dados onde fica armazenada a representação ontológica da última versão de cada artefato do projeto. A representação dos artefatos é feita com base na extração de informação de cada artefato, com base nas informações da BCD, onde são definidos rótulos para o artefato. Os rótulos representam um conjunto de termos da ontologia que foram recuperados do artefato. A BCP que auxiliará na descoberta dos artefatos relacionados.

A **etapa 3** é quando ocorre a propagação da percepção das mudanças aos membros da equipe. Os agentes irão finalizar o processo notificando os membros da equipe, ou seja, provendo a percepção desejada após mudanças percebidas em parte do projeto.

A seguir serão descritos em detalhes as responsabilidades de cada grupo de agentes do ARARA.

4.4. ARARA e seus Agentes

O ARARA é um sistema multi-agentes. Seus agentes atuam no Repositório de Artefatos e na Base de Conhecimento do Domínio com o objetivo de preparar a Base de Conhecimento do Projeto. A BCP auxilia no processo de relacionamento dos artefatos e

na recuperação dos mesmos para apresentação aos membros da equipe do projeto, como resultado da propagação da percepção de uma mudança no projeto.

4.4.1. *EventDetectionAgents*

Sempre que ocorre uma escrita no repositório de artefatos do projeto, os agentes que compõem o conjunto de *EventDetectionAgents* são os responsáveis por detectarem tal evento. Eles descobrem quais foram os artefatos envolvidos na escrita no repositório e para cada artefato qual foi o tipo de escrita no repositório (inclusão, alteração ou exclusão), conforme ilustrado na Figura 18.

Em seguida, esses agentes se comunicam com um outro conjunto de agentes chamados *InformationExtractionAgents*, informando quais artefatos foram modificados para serem tratados por eles, um a um.

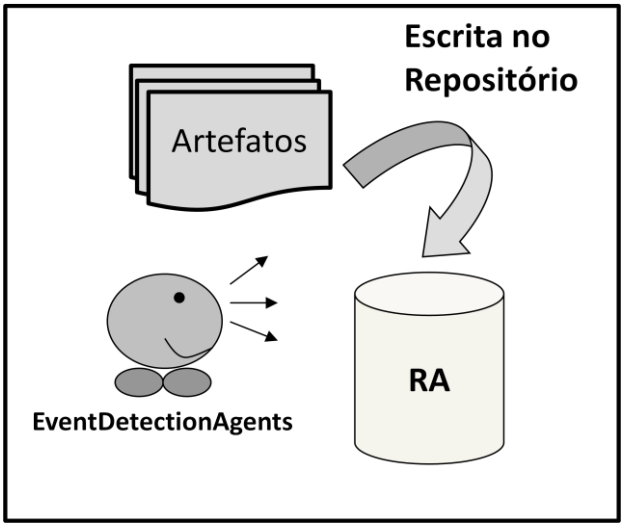

Figura 18. Agentes que compõem o ARARA.

4.4.2. TagAgents

Estes agentes compõem o conjunto de agentes chamados de *InformationExtractionAgents*. Eles são responsáveis por criar uma representação de

cada artefato que é escrito no RA, conforme apresentado na Figura 19. Essa representação ficará armazenada na BCP.

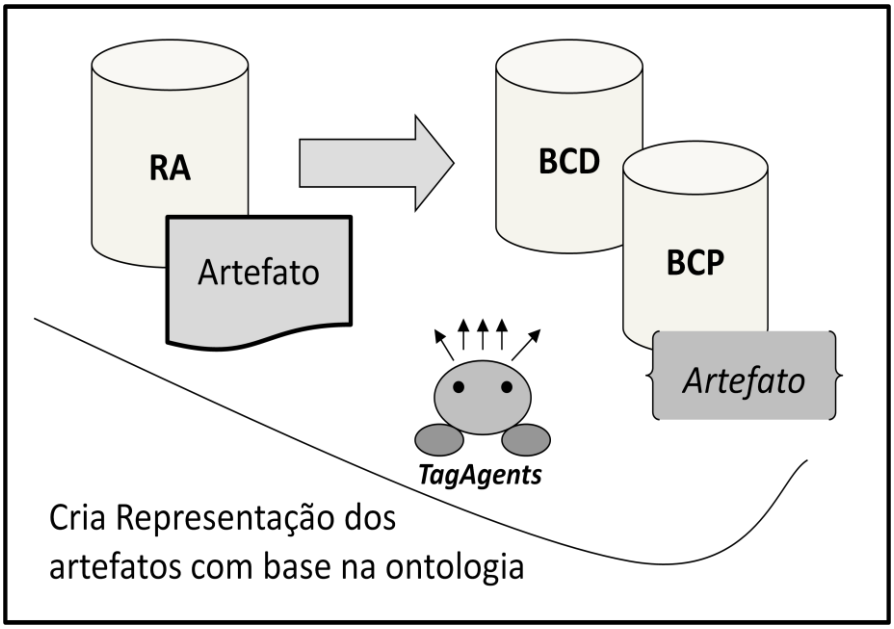

Figura 19. Recursos utilizados pelos *TagAgents*.

A representação do artefato na BCP é feita com base nas informações da BCD e das informações extraídas do artefato que está sendo analisado. O processo que os *TagAgents* seguem consiste das seguintes etapas:

- buscar no RA a última versão do artefato que foi escrito,
- descobrir o tipo do artefato: diagrama de atividades, diagrama de classes, diagrama de componentes, diagrama caso de uso etc,
- descobrir o autor responsável pela última escrita do artefato no RA e
- criar a representação do artefato na BCP, com um conjunto de termos que o representem na ontologia, além de informar o tipo do artefato e o autor da escrita do artefato no repositório.

Criada a representação do artefato, esta é escrita na BCP como uma nova versão do artefato, mantendo-se também na BCP a representação anterior.

O conjunto de termos que definem os conceitos da ontologia que o artefato aborda é composto pelas classes da ontologia.

4.4.3. ChangeAgents

Os *changeAgents* também compõem o conjunto de agentes definidos como *InformationExtractionAgents*. Eles agem com base na BCP recuperando a representação do artefato em análise para comunicar aos agentes seguintes. O tipo de escrita do artefato no RA (inclusão, alteração, ou exclusão) definirá qual versão da representação do artefato esses agentes recuperarão. Para cada tipo de escrita os agentes agem de forma distinta, como descrito abaixo:

- **Inclusão de um artefato** → uma vez que o artefato é novo no repositório não há representação anterior à atual gerada na BCP. Dessa forma os *changeAgents* recuperam versão atual e única da representação do artefato.
- **Exclusão de um artefato** → uma vez que o artefato foi excluído do repositório do projeto em desenvolvimento, não há como os *TagAgents* criarem uma nova representação para tal artefato. Então é preciso que os *ChangeAgents* recuperem a última versão da representação do artefato, que se encontra na BCP.
- **Alteração de um artefato** → nesse caso os *ChangeAgents* comparam as duas versão para verificar se houve modificação na representação do artefato para saber se é preciso recuperar ambas representações. A Figura 20 ilustra esse processo.

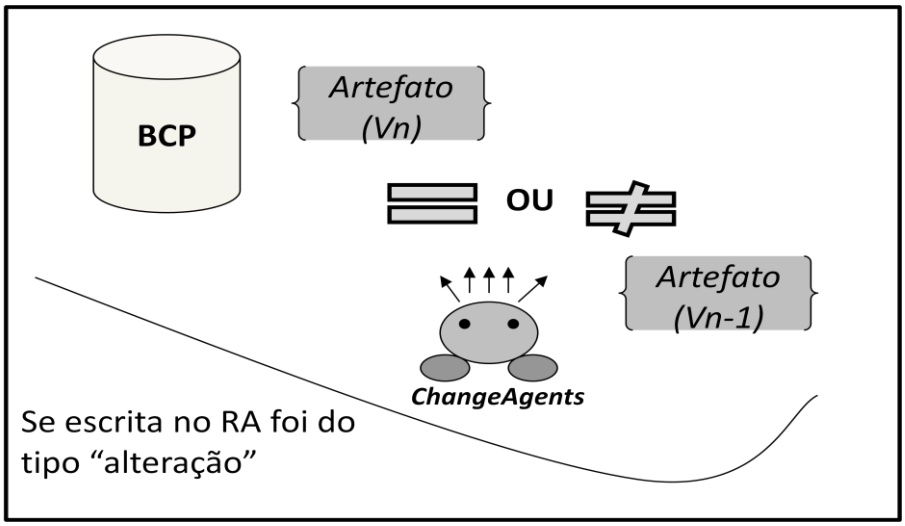

Figura 20. Análise das representações dos artefatos se escrita for alteração pelos *ChangeAgents*.

Os *TagAgents* se comunicam com os *RelatingAgents* passando a versão (ou versões) da representação que deve ser analisada para que sejam recuperados os artefatos relacionados.

4.4.4. RelatingAgents

Para cada artefato escrito no RA, os *RelatingAgents* são responsáveis por rastrear a BCP em busca do conjunto de artefatos que se relacionam pelos mesmos conceitos de negócio e que são fortes candidatos a serem analisados, pois são artefatos passíveis de serem impactados pela mudança detectada no RA. O relacionamento de um artefato com os demais é definido de duas formas:

1. semelhança entre os artefatos baseado nos conceitos da ontologia comuns entre eles e
2. navegação pelos conceitos da ontologia abordados no artefato em análise.

O resultado é listado em ordem dos mais semelhantes aos menos semelhantes. Em seguida, o corte é aplicado a partir de um determinado ponto para que apenas os artefatos mais semelhantes sejam apresentados. Esse procedimento evitará sobrecarga de informação. A Figura 21 ilustra o trabalho dos *RelatingAgents*.

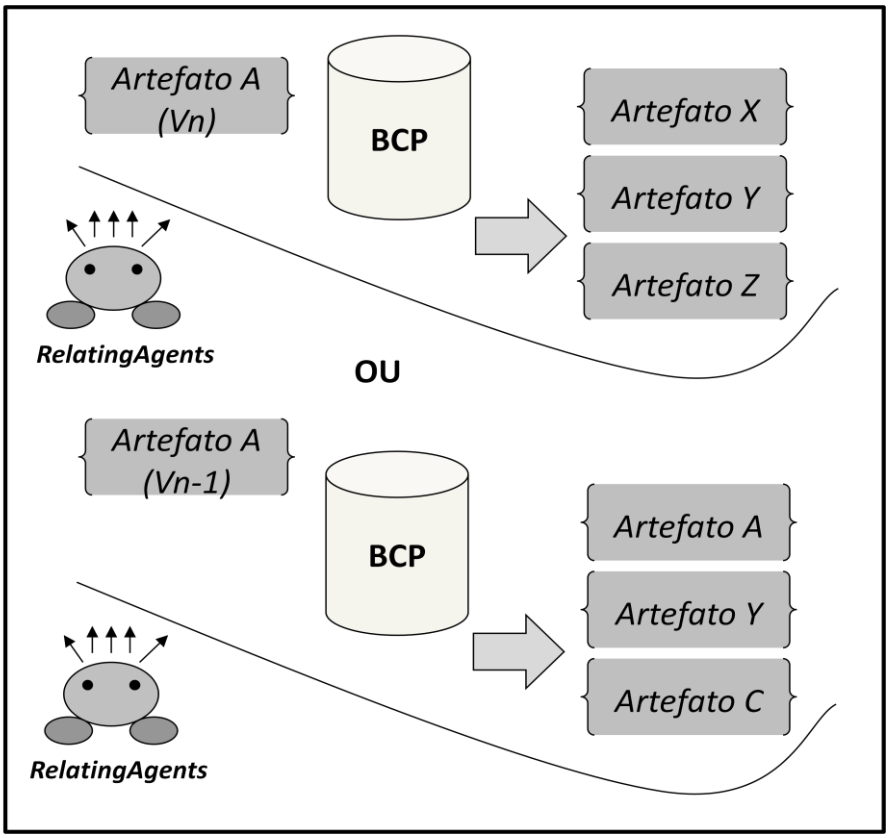

Figura 21. Atividades dos agentes *RelatingAgents*.

O tipo de escrita do artefato no RA (inclusão, exclusão, ou alteração) define qual versão da representação do artefato será usada para recuperação dos artefatos relacionados. Na etapa anterior os *ChangeAgents* já se responsabilizaram por selecionar a versão correta para uso na recuperação dos relacionamentos. Será usado o {ArtefatoA (V_n)} no caso de inclusão; o {ArtefatoA (V_{n-1})} no caso de exclusão e no caso de alteração pode ser usado somente o {ArtefatoA (V_n)} ou {ArtefatoA (V_n)} e {ArtefatoA (V_{n-1})}.

4.4.5. AwarenessAgents

Após os *InformationExtractionAgents* terem feito seu trabalho, eles então se comunicam com os *AwarenessAgents* que irão preparar a notificação das mudanças detectadas com a coleção recuperada de artefatos relacionados. Eles são responsáveis

por encontrar os membros da equipe que devem receber a notificação sobre as mudanças ocorridas no RA e os artefatos que foram recuperados como candidatos a serem revistos para propagação das mudanças

Cada membro será notificado apenas sobre os artefatos recuperados que sejam de sua responsabilidade para análise de impacto. Na representação dos artefatos na BCP há informações dos autores de cada artefato, ou seja, quem o manipulou por último no RA.

Capítulo 5 - Recursos e Implementação do ARARA

Este capítulo apresenta como foram criadas a Base de Conhecimento do Domínio e a Base de Conhecimento do Projeto. Apresenta a forma como é acessado o Repositório de Artefatos pelo ARARA e as técnicas utilizadas para relacionar os artefatos para recuperação no processo de percepção de mudanças.

5.1. O Projeto de Desenvolvimento de Software

O projeto que motivou a criação do ARARA foi o projeto COPPETEC de desenvolvimento de software para a PETROBRAS.

Era de conhecimento do projeto a possibilidade da saída de alguns membros e da chegada de outros para compor a equipe, ao longo do desenvolvimento do projeto. Além disso, alguns membros da equipe trabalhariam com carga horária reduzida e horários não fixos, além de estarem distribuídos em diferentes setores dentro da companhia.

Era preciso estabelecer uma forma de representar e armazenar o conhecimento do domínio do negócio para que toda a equipe pudesse acessá-lo a qualquer instante no projeto. O conhecimento seria mapeado durante a etapa de preparo do modelo de processo juntamente com os especialistas do domínio do negócio.

O projeto teve como principais objetivos:

1. representar formalmente o conhecimento do domínio de negócio em análise para reuso em diferentes sistemas que venham a ser desenvolvidos futuramente e
2. modelar e desenvolver um sistema de controle de reservas para reportar à companhia, às agências reguladoras e a bolsas de valores.

5.2. A Ontologia utilizada pelo ARARA

O domínio de negócio mapeado na ontologia é o domínio das reservas de petróleo. A ontologia do domínio foi construída a partir do conhecimento obtido nas reuniões realizadas com os especialistas do domínio, durante a etapa de mapeamento do processo do negócio. A metodologia utilizada foi a sugerida por USHCOLD e KING (USCHOLD e KING, 1995).

Dadas as características do projeto, já havia sido identificado o propósito da criação da ontologia, que consistiu em definir o motivo pelo qual a ontologia estava sendo criada e onde seria utilizada. Nesse caso a ontologia foi criada para:

1. representar formalmente o conhecimento do domínio do negócio,
2. representar os conceitos do domínio com um termo único,
3. representar os conceitos do domínio com definições precisas,
4. ter um vocabulário adequado e comum como base para uso ao longo do desenvolvimento do projeto,
5. servir como base de conhecimento disponível para acesso a todos da equipe e
6. ser usada como complemento ao modelo de processo do sistema.

A captura se deu através de reuniões realizadas com os especialistas do domínio a fim de serem explicados os conceitos do domínio e os relacionamentos entre eles. As reuniões foram direcionadas seguindo o modelo de processo *AS-IS* da Reserva. Os especialistas explicavam sobre as partes do processo e como cada uma dessas partes se encaixava no todo. Após cada reunião, eram extraídos os termos que foram mais citados ao longo da reunião, para associá-los aos conceitos mapeados, seus significados, seus

relacionamentos e seus atributos. Essas informações eram apresentadas em resumo para os especialistas, em reuniões seguintes, que avaliavam se houve entendimento correto (ou não) do que foi explicado. Outra forma de avaliar a ontologia foi através do julgamento da ontologia, após codificação, com relação a outras referências. As referências usadas para esse fim foram: o modelo *TO-BE* de processo da Reserva e os artefatos do projeto.

A codificação que envolve a representação explícita do conhecimento adquirido na etapa anterior foi feita com o uso da ferramente Protégé, versão 3.4 (PROTÉGÉ, 2010).

5.3. Representação da Ontologia na BCD

Para atuação do ARARA no processo de percepção de mudanças é preciso ter configurada a Base de Conhecimento do Domínio.

O ARARA funciona independentemente do domínio de negócio. É preciso apenas que a ontologia tenha sido construída em OWL.

Para criação da BCD foi utilizado o Protégé-OWL API (PROTÉGÉ-OWL API, 2010), uma biblioteca java e open-source disponibilizada pelo Protégé. Essa API provê classes e métodos para leitura e escrita de ontologias OWL.

Na criação da BCD, foram extraídos os seguintes elementos da ontologia:

- classes e subclasses (ambas representadas como classes),
- hierarquia entre classes e subclasses,
- propriedades objeto,
- domínio e faixa das propriedades objeto,
- propriedades tipo de dado e
- domínio das propriedades tipo de dado.

5.4. Artefatos usados pelo ARARA

Com o modelo de processo do negócio finalizado e com os conceitos do negócio representados pela ontologia, a etapa seguinte do projeto foi a criação dos modelos de análise e *design*[8].

Esta etapa do processo de modelagem do sistema contou com a criação de modelos UML (*Unified Modeling Language*) para representação da especificação do sistema. A UML é uma linguagem de modelagem que combina métodos de análise e *design* orientados a objetos em um método unificado (PRESSMAN, 2006). Hoje a UML é um padrão OMG (*Object Management Group*), dos mais usados para modelar estruturas, comportamentos e arquitetura de aplicações, processo de negócios e estruturas de dados (OMG/UML 2007).

A ferramenta RSA (*Rational Software Architect*) (RSA 2010) foi utilizada para criar os modelos UML que formam os artefatos do sistema. Os modelos UML são representados em arquivos XMI (XML Model Interchange) (XMI 2003), (MOF/XMI 2007).

5.5. Aplicação da Ontologia nos Artefatos do Projeto

O preparo dos artefatos seguiu regras básicas de boas práticas definidas pela equipe.São exemplos de algumas regras:

1. Os termos que foram definidos na ontologia do negócio devem ser empregados nos artefatos evitando criação de novos termos, ou termos similares, para representar os conceitos já mapeados na ontologia.

[8] Modelo de projeto.

2. Utilizar os termos da ontologia nos nomes dos artefatos e nos elementos que compõem cada artefato.
3. Abreviações e omissões de termos devem ser evitadas.
4. Nomes dos artefatos devem ter as palavras separadas e devem ser iniciadas em maiúsculas.
5. Nomes dos elementos que compõem os artefatos e que são formados por mais de uma palavra, devem obrigatoriamente ter da segunda palavra em diante as iniciais em maiúscula. Por exemplo: **registrarArvoreHierarquicaSisres()**, **recuperarExercícioAberto()**.
6. Preposições, artigos e conjunções ficam em minúscula nos elementos de diagramas de atividades.

A Figura 22 ilustra a aplicação dessas regras no preparo dos artefatos. À direita da figura estão listadas as classes da ontologia. O conceito *Exercício* foi aplicado nos nomes dos artefatos e também nos elementos que compõem cada artefato.

O uso dos conceitos da ontologia na criação dos artefatos viabilizou o processo do ARARA para relacionar os artefatos e prover percepção quanto às mudanças nos artefatos do projeto ao longo do desenvolvimento do sistema.

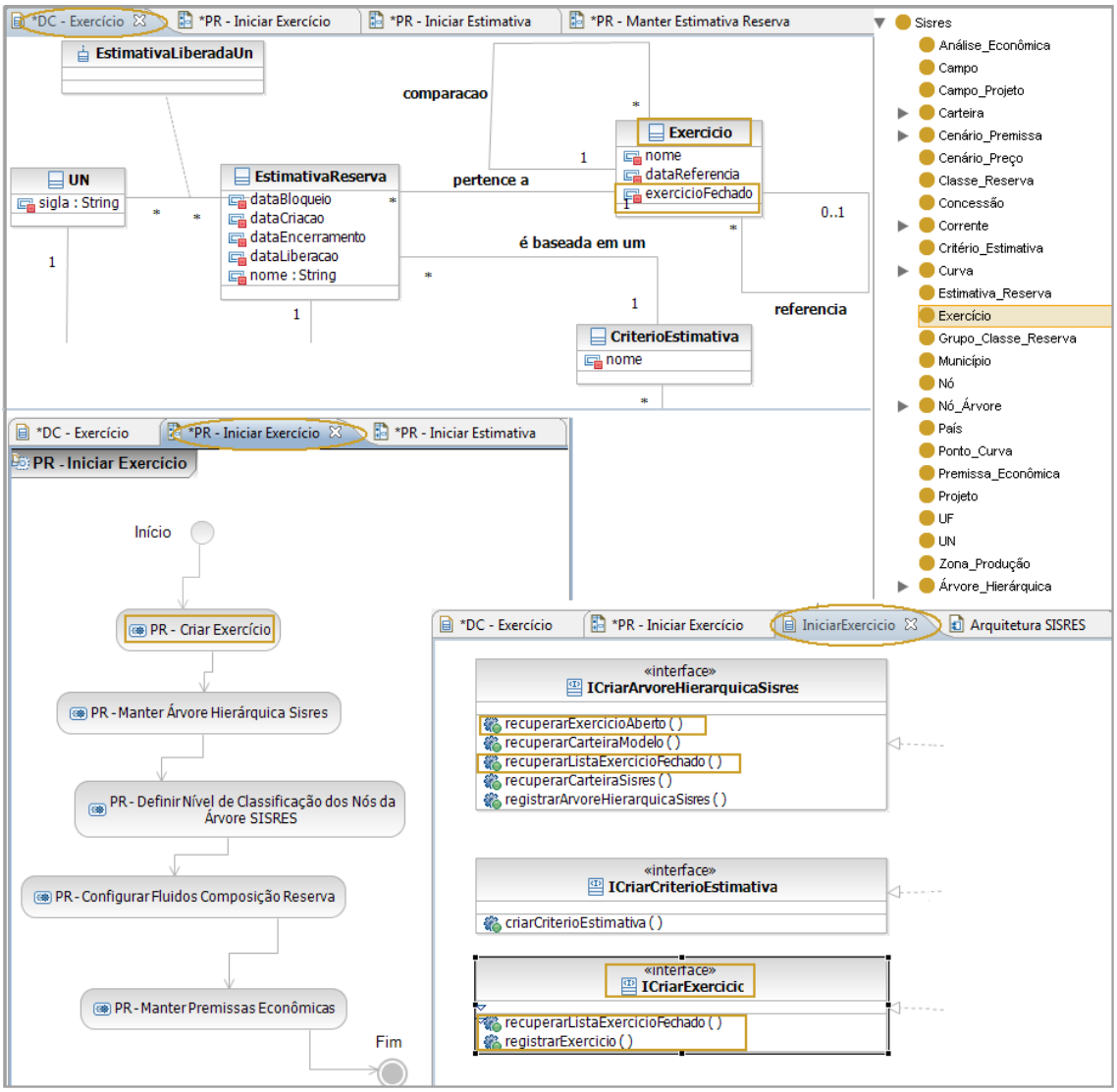

Figura 22. Exemplo da aplicação da ontologia nos artefatos do projeto.

5.6. Acesso ao Repositório de Artefatos

A ferramenta de controle de versão SubVersion (SVN, 2010) foi utilizada no projeto como Repositório dos Artefatos (RA), armazenando-os sob um controle de versões.

A comunicação dos agentes com o repositório de artefatos se deu através do uso do SVNKit (SVNKit, 2010), que provê uma API java para trabalhar acessando e manipulando os arquivos no SVN através de aplicações Java.

A comunicação dos agentes com o RA (detecção de evento) se dá através de um *repository hook*, que é um *trigger* disparado na ocorrência de um evento. O evento detectado pelos agentes é o evento de *commit*. Este evento representa a escrita por um membro da equipe de desenvolvimento que pode ser inclusão de novos artefatos, exclusão de alguns outros, ou alteração em artefatos que já estejam no RA.

O *repository hook* usado para iniciar o processo do ARARA é chamado de *post-commit hook*, pois é executado logo após a finalização do evento de *commit* que contém informações suficientes a respeito do que foi modificado no RA.

5.7. Representação dos Artefatos na BCP

A Base de Conhecimento do Projeto é onde são armazenados os artefatos segundo uma representação específica criada pelo ARARA. Nesta base são armazenados os rótulos definidos para os artefatos em cada versão do mesmo.

Para criação da BCP foi utilizada a API JDOM (JDOM, 2010), uma biblioteca java disponibilizada para manipulação de arquivos XML.

Os modelos UML que compõem os artefatos de aplicação do ARARA são:

- diagramas de atividades e
- diagramas de classes (modelo de análise e interfaces dos componentes).

A Figura 23 apresenta o metamodelo para o diagrama de atividades(OMG/UML, 2009). As metaclasses que estão apresentadas na cor cinza são os elementos dos modelos UML lidos em busca de termos relevantes para a representação do artefato na BCP. A Figura 24 apresenta o metamodelo para o diagrama de classes com os elementos lidos tanto nos diagramas representativos do modelo de análise do projeto, quanto das interfaces dos componentes do projeto.

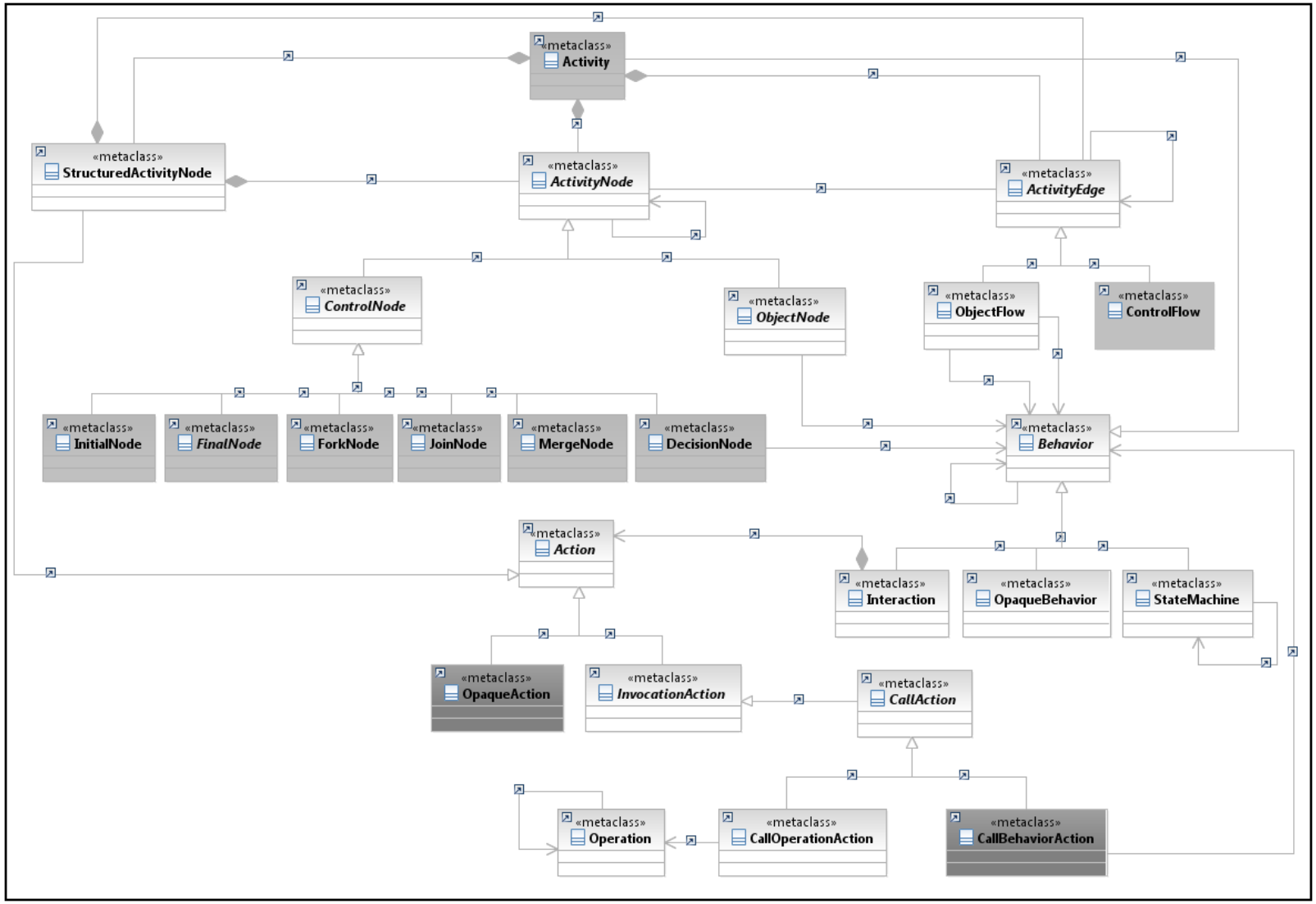

Figura 23. Metamodelo do diagrama UML de atividades.

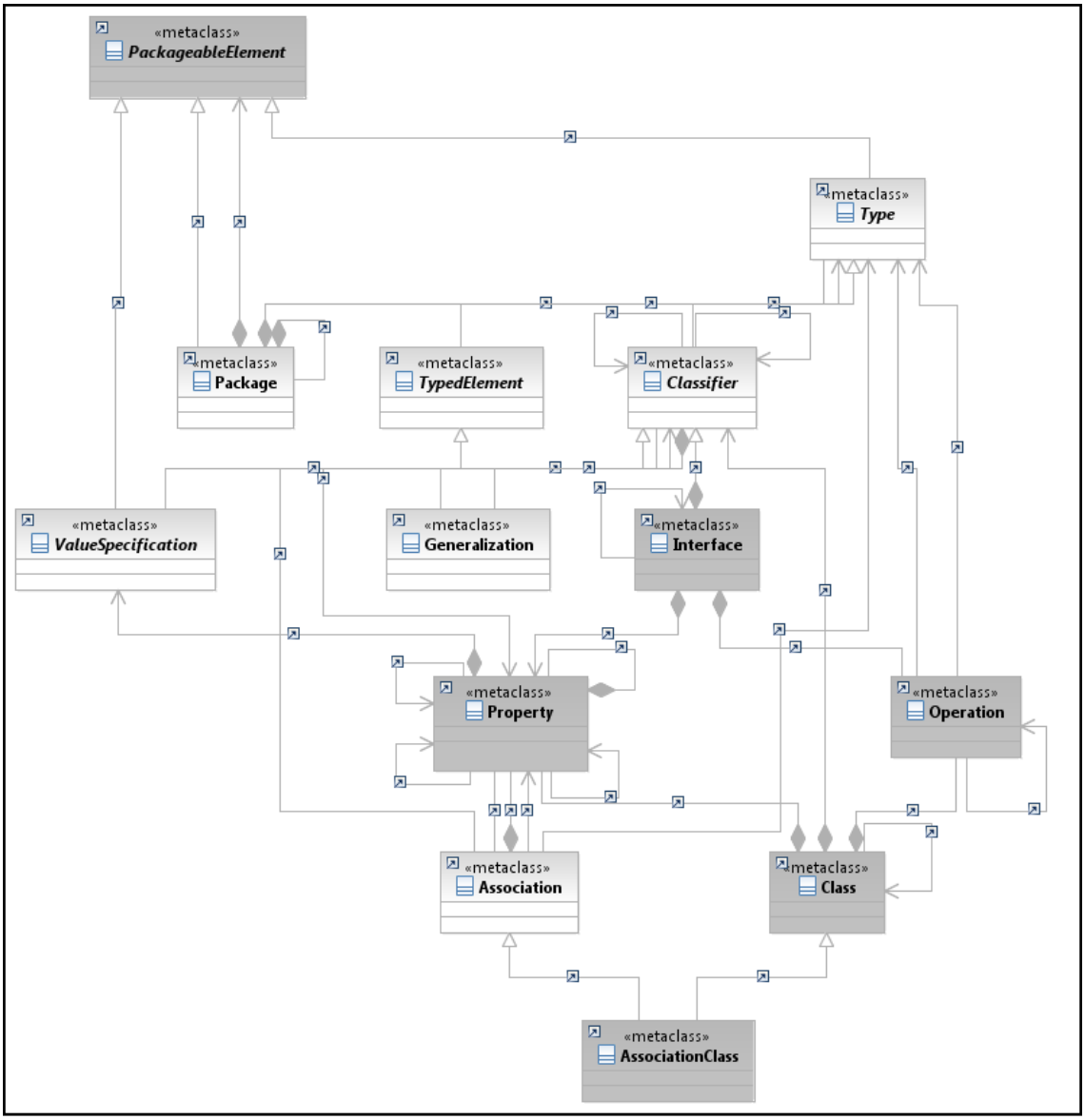

Figura 24. Metamodelo do diagrama UML de classes.

5.7.1. Rotular os Artefatos

O processo de rotular consiste em buscar termos da ontologia em cada elemento lido em um artefato. Cada elemento é submetido a um pré-processamento, cujas etapas são:

1. realizar a análise léxica das sequências de caracteres transformando-as em palavras (*tokens*),

2. remover *stopwords* das sentenças e
3. extrair a raíz das palavras (*stemmeing*) seguindo o algoritmo de Porter (PORTER, 1997).

O resultado final do pré-processamento é submetido à comparação com os termos da ontologia, que também passam pelo mesmo processo de pré-processamento. São eles:

- As **classes** da ontologia são usadas para gerar os rótulos dos artefatos.
- As **propriedade tipo de dado** são usadas para auxiliar na definição precisa do rótulo. As propriedades tipo de dado são aplicadas nos artefatos de forma explícita como atributos de classes nos diagramas UML. Por exemplo: {**parceiro**, **descolado**} são atributos da classe **Campo** no diagrama de classes e são propriedades tipo de dado da classe **Campo** na ontologia, pois **Campo** possui apenas 1 **parceiro** e pode ser ou não **Campo descolado**. Para os casos onde a classe **Campo**, por algum motivo, teve seu nome abreviado em um diagrama UML, é possível inferi-la analisando as propriedades {parceiro, descolado} cujo domínio é a classe **Campo**.
- As **propriedade objeto** são usadas também com o objetivo de auxiliar na definição precisa do rótulo, uma vez que as propriedades objeto não aparecem representadas de forma explícita nos artefatos. Com a análise da faixa e domínio da propriedade objeto é possível confirmar se um rótulo se relaciona com os demais. Por exemplo: para um artefato que aborde o conceito **Árvore Hierárquica,** serão recuperados como rótulo as classes **Nó** e **Nó_Árvore**. No entanto, analisando as faixas das propriedades objeto cujo domínio é a classe **Nó** conclui-se que a classe **Nó** não possui

relacionamento com o conceito **Árvore Hierárquica**. Por exemplo: **Nó_CorrenteÓleo e Nó_CurvaVolumétrica** são propriedades objeto cujo domínio é **Nó**. Se no artefato não houver os conceitos **CorrenteÓleo** ou **CurvaVolumétrica** é inferido que o rótulo **Nó** não é representativo no artefato.

5.8. Recuperação dos Artefatos

A recuperação de artefatos para percepção de mudanças é encontrar os que se relacionam com o artefato motivo da mudança.

A relacionamento é dado pelo Coeficiente de Similaridade Jaccard (JAIN e DUBES, 1988) que mede a similaridade entre o artefato modificado e os demais artefatos do RA. A definição do Coeficiente de Similaridade de Jaccard é:

$$\mathrm{JS}(\mathrm{A},\mathrm{B}) = \left[\frac{\mathrm{A} \cap \mathrm{B}}{\mathrm{A} \cup \mathrm{B}}\right]$$

Reescrevendo esse coeficiente para aplicação no ARARA, temos:

$$JS_{(ArtfTag_1, ArtfTag_2)} = \left[\frac{ArtfTag_1 \cap ArtfTag_2}{Artf_{Tag1} \cup Artf_{Tag2}}\right], \text{ onde:}$$

- *JS* → representa o coeficiente de similaridade entre dois artefatos,
- $ArtfTag_T$ → é a representação do artefato *T* ($Artf_T$) na BCP, ou seja, é o conjunto de termos da ontologia que o artefato *T* ($Artf_T$) possui:

$$ArtfTag_T = \{BT_1, BT_2, \ldots, BT_n\} \text{ e}$$

- BT_i *(Business Tag)* → representa o *iéssimo* termo do conjunto de termos que representa um artefato na BCP.

Assim, para cada artefato que é escrito no RA são feitos os cálculos de similaridade deste com os demais já representados na BCP.

A Tabela 1 apresenta o grau de similaridade de cada artefato do projeto com o artefato $\boldsymbol{Artf_1}$. Os artefatos com coeficiente de similaridade maior que zero ($\mathbf{JS}(\mathbf{ArtfTag1}, \mathbf{ArtfTagx}) > 0$) são listados como sendo os que se relacionam com $\boldsymbol{Artf_1}$, que são os canditatos a mudança no processo de propagação da percepção.

Tabela 1. Similaridade entre artefatos com $Artf_1$.

$JS(ArtfTag_1, ArtfTag_x)$	Artefatos Relacionados
0,75	$Artf_2$
0,66	$Artf_{10}$
0,6	$Artf_{13}$
0,5	$Artf_5$
0,5	$Artf_6$
0,5	$Artf_7$
0,5	$Artf_8$
0,5	$Artf_{14}$
0,5	$Artf_{16}$

A modificação do artefato $\boldsymbol{Artf_2}$, após propagação da percepção, gera uma nova relação de artefatos candidatos a mudança que é apresentada na Tabela 2.

Tabela 2. Similaridade entre artefatos com $Artf_2$.

$JS(Artftag_2, ArtfTag_x)$	Artefatos Relacionados
0,75	$Artf_1$
0,66	$Artf_5$
0,66	$Artf_6$
0,66	$Artf_7$
0,66	$Artf_8$
0,5	$Artf_{11}$
0,5	$Artf_{12}$
0,5	$Artf_{17}$
0,5	$Artf_{20}$

Pode-se observar na Tabela 1 que a partir do **JS(ArtfTag$_1$, ArtfTag$_x$) = 0,5** os artefatos recuperados {***Artf$_5$***, ***Artf$_6$***, ***Artf$_7$*** e ***Artf$_8$***} são uns dos primeiros artefatos recuperados com a mudança feita no ***Artf$_2$***, conforme apresentado na Tabela 2. Esses artefatos {***Artf$_5$***, ***Artf$_6$***, ***Artf$_7$*** e ***Artf$_8$***} possuem um grau maior de semelhança com ***Artf$_2$*** do que com ***Artf$_1$***.

Analisando esse caso, poderia-se sugerir o procedimento de definir uma linha de corte a partir do grau de similaridade < 0,5 já que após modificação no ***Artf$_2$***, {***Artf$_5$***, ***Artf$_6$***, ***Artf$_7$*** e ***Artf$_8$***} serão recuperados. No entanto {***Artf$_{14}$*** e ***Artf$_{16}$***} não foram recuperados e podem ser perdidos adotando tal procedimento.

Essa linha de corte reduziria o número de artefatos recuperados na tentativa de evitar sobrecarga de informação, no entanto o processo de propagação de percepção pode ser afetado, pela possibilidade de deixar de fora alguns artefatos.

Capítulo 6 - Análise dos Resultados

Este capítulo apresenta os resultados de como o ARARA se comporta provendo percepção à equipe quanto a mudanças nos artefatos de um projeto de desenvolvimento de software. A primeira avaliação feita apresenta os resultados dos rótulos gerados pelo ARARA para os artefatos do projeto. A segunda avaliação feita apresenta os resultados do artefatos recuperados para propagação da percepção de mudanças.

6.1. Avaliação da Representação dos Artefatos pelo ARARA

A primeira etapa no processo do ARARA de prover percepção de mudanças é gerar uma representação do artefato modificado na BCP para em seguida recuperar os relacionamentos deste com os demais. Essa representação é feita com base na ontologia do domínio, rastreando o artefato e recuperando informações que auxiliem na geração de rótulos que são compostos pelos termos da ontologia.

A avaliação teve como objetivo validar a representação criada pelo ARARA para cada artefato, validando a precisão na geração de rótulos para os artefatos do projeto.

Para validar recuperação de dados não ordenados (*rank*) a precisão e revocação (*recall*) são os dois métodos básicos e mais utilizados para avaliar a eficiência em recuperação de informação (MANNING *et al.*, 2009).

Precisão *(P)* é a fração dos documentos recuperados que são relevantes:

$$Precisão = \frac{\sum(itens\ relevantes\ recuperados)}{\sum(itens\ recuperados)}$$

$$Precisão = P(relevantes/recuperados)$$

Revocação *(R)* é a fração dos documentos relevantes que são recuperados:

$$Revocação = \frac{\sum(itens\ relevantes\ recuperados)}{\sum(itens\ relevantes)}$$

$$Revocação = R(recuperados/relevantes)$$

A Tabela 3 apresenta a definição dos rótulos relevantes e não relevantes, recuperados ou não recuperados, para clarear a ideia de precisão e revocação.

Tabela 3. Tabela de contingência

	Relevantes	**Não Relevantes**
Recuperados	Verdadeiro positivo *(vp)*	Falso positivo *(fp)*
Não recuperados	Falso negativo *(fn)*	Verdadeiro negativo *(vn)*

Onde:

$$Precisão = \frac{vp}{(vp + fp)} \quad \text{e} \quad Revocação = \frac{vp}{(vp + fn)}$$

6.1.1. Experimento 1

O primeiro experimento foi para validar a precisão do ARARA em gerar rótulos, comparando-os com os definidos pela autora, que seguiu as mesmas regras utilizadas na configuração do ARARA.

A autora definiu rótulos para todos os artefatos do projeto, que conta com 124 artefatos.

A precisão nos experimentos foi calculada da seguinte forma:

$$P_{Artf_T} = \frac{R_c}{R_t}, \text{ onde:}$$

- $\boldsymbol{P_{Artf_T}}$ *(Precisão para um artefato* $Artf_T$*)* representa a precisão no acerto do ARARA no processo de geração de rótulos.
- $\boldsymbol{R_c}$ (*Rótulos Coincidente)* é o total de rótulos gerados pelo ARARA para um artefato coincidentes com os rótulos definidos por um outro processo.

- $\boldsymbol{R_t}$ (*Rótulos Total) é* o total de rótulos gerados pelo ARARA para representar um artefato.

O resultado obtido está apresentado na Tabela 4.

Tabela 4. Precisão do ARARA comparação rótulos do autora.

ARARA X Autora	$P_{M_{Artf}}$(*Precisão média*)	$R_{M_{Artf}}$(*Revocação média*)
Diagramas de Atividades	**0,99**	**0,92**
Diagramas de Classe (análise)	**0,86**	**1**
Diagrama de Classes (interface)	**0,93**	**0,96**
Total de Artefatos	**0,97**	**0,93**

A técnica de utilizar as propriedades tipo de dado para inferir termos da ontologia que foram abreviados ou tiveram parte do nome suprimido aperfeiçoou a geração de rótulos. Para os diagramas de classe o índice de rótulos falso negativos *(fn)* diminuiu, melhorando a revocação uma vez que tal propriedade aparece como atributo das classes. Para os diagramas de atividades e de classe (interface) essa técnica não alterou os resultados, pois não é apresentado explicitamente propriedades tipo de dado.

A técnica de utilizar as propriedades objeto melhorou o resultado de precisão para todos os artefatos. Anteriormente ao uso dessa técnica de navegação na ontologia, para confirmar se um rótulo definia mesmo o artefato, o índice de falso positivo *(fp)* era maior. Os casos comuns apresentados foi a recuperação do termo **Nó**, quando o artefato trata de conceitos associados apenas a **Nó_Árvore** e a recuperação do termo **Curva**, quando o artefato trata de conceitos mais específicos como **Curva_Econômica**, **Curva_Volumétrica**, ou **Curva_Preços**.

Esse experimento apresentou uma precisão $P = 1$ em 96% dos artefatos avaliados e revocação $R = 1$ em 72,6% dos mesmos artefatos avaliados. Os demais artefatos que

tiveram revocação $R_M < 1$ a autora atribui ao fato de alguns artefatos não terem sido criados em conformidade com as regras de boas práticas para aplicação da ontologia na criação de artefatos, principalmente nos diagramas de atividades, onde os conceitos foram apresentados abreviados e com omissão de palavras que compõem o termo.

6.1.2. Experimento 2

O segundo experimento foi validar a precisão do ARARA em gerar rótulos comparando-os com os definidos pelos membros da equipe do projeto. Sete membros da equipe participaram do experimento, sendo que todos possuem tempos diferentes de atuação no projeto.

Foi disponibilizado para os participantes: as imagens dos diagramas selecionados para avaliação e as classes da ontologia. O objetivo era definir os rótulos (classes e subclasses) para os conceitos que eles julgassem que eram abordados nos artefatos.

A análise dos rótulos definidos pelos participantes mostrou que cada participante possui um parâmetro próprio para definir rótulos para representar os conceitos mapeados pelo artefatos. Alguns participantes, além de definir rótulos com base no que estava descrito nos artefatos, definiram também rótulos de conceitos associados, segundo entendimento próprio, mas não descritos nos artefatos. Outros especializaram a representação se utilizando das subclasses, enquanto outros utilizaram apenas a classe pai.

Uma resposta padrão (gabarito) para os rótulos definidos pelos participantes foi obtida utilizando-se somente os termos que tiveram ocorrência maior que 50% dentro das respostas dos participantes.

A Tabela 5 apresenta os resultados de precisão e cobertura alcançados comparando o ARARA com este rótulo padrão da equipe.

Tabela 5. Precisão média entre os rótulos do ARARA com rótulo padrão da equipe.

ARARA X Padrão Equipe	$P_{M_{Artf}}$(*Precisão média*)	$R_{M_{Artf}}$(*Revocação média*)
Diagramas de Atividades	**0,64**	**0,86**
Diagramas de Classes (análise)	**0,27**	**0,95**
Diagramas de Classes (interface)	**0,59**	**0,92**
Total de Artefatos	**0,76**	**0,70**

Os resultados mostram que o ARARA tem um melhor desempenho na revocação do que na precisão. Nesse experimento, em particular, esse resultado já era previsto, uma vez que ao padronizar as respostas dos participantes o conjunto de rótulos foi reduzido favorecendo um número maior de coincidências no conjunto de rótulos relevantes. Esse aspecto da padronização foi mais acentuado nos diagramas de classes onde o resultado de precisão foi baixo em comparação com os demais artefatos. Era esperado que a representação dos diagramas de classe pelo ARARA apresentasse um grande conjunto de rótulos recuperados, cujo motivo está diretamente relacionado ao fato dos nomes das classes serem também os nomes das classes da ontologia.

A precisão nos diagramas de atividades e de classes (interface) só não foi maior, pois houveram casos onde foram omitidas palavras que compõem um termo. Por exemplo, **Estimativa_Reserva** sendo apresentado somente **Estimativa** ou **Nó_Árvore** sendo apresentado somente **Nó**.

6.2. Avaliação da Recuperação dos Artefatos Relacionados

Para recuperar artefatos semelhantes a um artefato modificado o ARARA utiliza a representação ontológica dos artefatos na BCP para relacioná-los por semelhança de conceitos.

Esta avaliação teve como objetivo validar a recuperação de artefatos após uma modificação no RA. A validação foi realizada com base nos resultados de revocação avaliando a capacidade do ARARA em prover percepção de mudanças.

6.2.1. Experimento 3

O terceiro experimento validou a revocação do ARARA na recuperação de artefatos. O parâmetro utilizado para avaliação foi o conjunto de artefatos que a autora esperava ser notificada dada uma mudança.

A preocupação nesse experimento era avaliar a capacidade do ARARA em, dada uma modificação em um artefato, recuperar o maior número de artefatos relacionados relevantes. Artefatos não relevantes recuperados (*falsos positivos*) afetam a precisão, geram sobrecarga de informação, mas não afetam a propagação da percepção da mudança. Sendo assim, o que deve ser evitado nesse processo são os falsos negativos *(relevantes não recuperados).*

Para evitar sobrecarga de informação uma solução possível de ser adotada é trabalhar com uma sequência de preparo dos artefatos. Nesta hierarquia é especificada uma ordem de preparo dos diagramas e a dependência entre eles. A Figura 25 apresenta a ordem de preparo dos artefatos. É possível observar que os diagramas de componentes do sistema serão preparados somente após o preparo dos diagramas de classe referente ao modelo de análise do projeto.

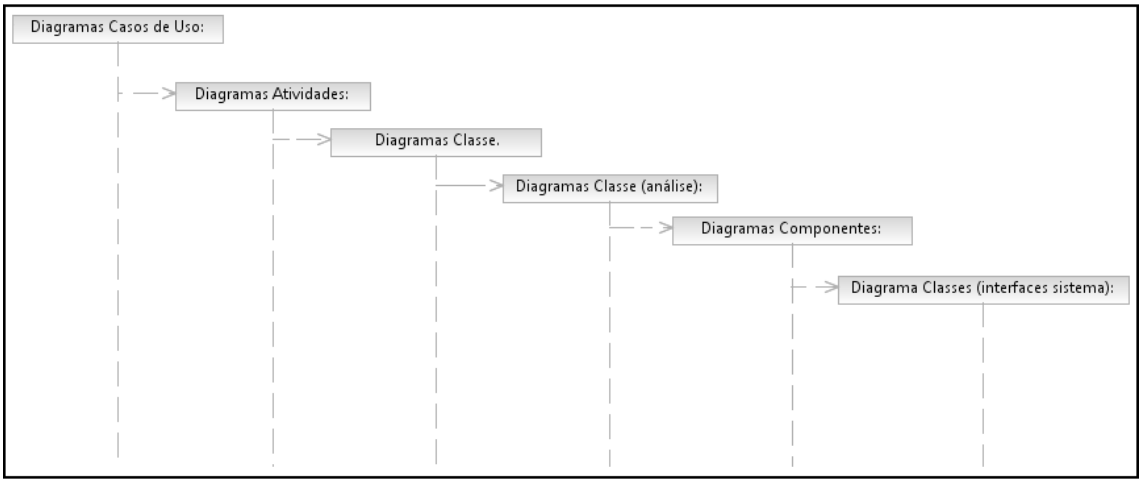

Figura 25. Sequência de preparo dos artefatos no desenvolvimento do projeto.

Quando houver uma modificação, por exemplo, em um diagrama de atividades, somente os artefatos que se relacionem com este e que tenham sido preparados após, dentro dessa hieraquia, que serão incluídos na lista dos artefatos sujeitos a mudança.

Para 20% do total de artefatos do projeto, a autora definiu os conjuntos de artefatos a serem recuperados para cada um que fosse modificado. Essa definição foi feita com base no conhecimento da autora dos modelos UML do projeto, uma vez que a autora era uma dos responsáveis pela criação e atualização dos mesmos.

Para esse experimento foram utilizados somente o diagrama de atividades como sendo o artefato modificado para recuperação dos relacionados a ele. Os diagramas de atividades foram vistos como sendo os artefatos que continham os requisitos e regras de negócio. Os diagramas de classe (análise e interface), como sendo os de implementação do sistema. Uma mudança nesses diagramas dificilmente implicará em mudança no diagrama de atividades. Em geral, modificações nesses diagramas se dão em virtude das modificações realizadas em diagramas que estão acima na sequência de criação de dos artefatos.

A Tabela 6 apresenta o resultado de revocação obtida no experimento.

Tabela 6. Revocação média artefatos recuperados

$R_{M_{Artf}}$(Revocação média)
0,83

Embora a meta fosse revocação de 100%, o resultado encontrado foi bem satisfatório considerando os resultados encontrados no experimento 1. Essa divergência entre o esperado e o encontrado ocorreu devido ao fato de alguns artefatos não terem sido rotulados precisamente. Foram os casos onde as classes da ontologia foram omitidas ou abreviadas quando do preparo do artefato, por exemplo: Estimativa_Reserva sendo utilizado apenas Estimativa, Critério_Estimativa sendo utilizado apenas Critério. A falta desses rótulos fez com que tais artefatos não fossem recuperados como relacionados a um artefato cuja representação possuía o rótulo Estimativa_Reserva ou Critério_Estimativa.

Capítulo 7 - Conclusão

A deficiência ou falta de percepção de mudanças no processo de desenvolvimento de software gera inconsistências no projeto, e a tentativa de acerto dessas inconsistências leva à perda de trabalho já desenvolvido. Manter percepção quanto a mudanças no projeto minimiza o disperdício de tempo e garante consistência nos artefatos. No entanto, manter percepção a todo instante do que está acontecendo ao longo do desenvolvimento, ou manter percepção dos relacionamentos de todos os artefatos, é inviável sem o auxílio de uma ferramente capaz de prover essa necessidade.

Análisando o projeto de desenvolvimento de software utilizado para avaliação do ARARA podemos ressaltar as seguintes contribuições que a ontologia proporcionou:

- Sua criação apoiou o processo de modelagem de negócio e auxiliou no levantamento de requisitos do sistema.
- Serviu como base de conhecimento do domínio auxiliando na distribuição do conhecimento para todos da equipe.
- Formalizou a definição dos conceitos, evitando entendimentos ambíguos dentro do projeto.
- Forneceu um vocabulário preciso do domínio para uso nos artefatos do projeto.
- Forneceu meios de identifcar os conceitos abordados nos artefatos, uma vez que além das classes da ontologia também foram utilizadas as propriedades objeto e propriedades tipo de dado, refinando o processo de representação dos artefatos.

- Por ter sido possível representar os artefatos segundo a ontologia, foi possível relacioná-los por semelhança dos conceitos do domínio para recuperação para propagação de percepção de mudanças.

Em resumo, a aplicação da ontologia do domínio nos artefatos do projeto mostrou-se, pelos experimentos, um meio de grande valia para manter a rastreabilidade entre os artefatos e recuperar os relacionamentos entre eles. A ontologia provê mais do que um vocabulário para uso no processo de rastreabilidade. Dispondo dos demais elementos da ontologia (propriedades objeto e tipo de dado) a precisão no processo de rotular os artefatos aumenta, aumentando também a revocação na recuperação dos relacionamentos para propagação da percepção. É preciso que a equipe se comprometa em adotar as boas práticas definidas para criação dos artefatos, com base no uso da ontologia.

Com o apoio da ontologia, o ARARA se mostrou capaz de reconhecer os conceitos do domínio representados em cada artefato, uma vez que apresentou um bom desempenho em termos de precisão e revocação no processo de rotular os artefatos. Se mostrou capaz também de relacionar os artefatos que tratam dos mesmos conceitos. Isso é especialmente positivo por satisfazer à prática recomendada pelo ANSI/IEEE Standard 830-1998 para a especificação de requisitos de software quanto a rastreabilidade para frente (*forward*) (IEEE, 1998). Satisfaz também a prática específica do CMMI especificada no REQM SP 1.4 (SEI, 2006), que define que é preciso manter rastreabilidade bidirecional entre os requisitos e o produto final.

Os estudos realizados nesta dissertação mostram que o ARARA é um sistema colaborativo capaz de prover percepção de mudanças em desenvolvimento de software. Proporciona à equipe redução nos esforços necessários para coordenar tarefas e manter conhecimento do que está ocorrendo na área de trabalho. Esse aspecto é ainda mais

relevante para equipes de desenvolvimento cujos membros trabalham dispersos geograficamente, pois o sistema permite que, mesmo com a deficiência na comunicação e na percepção do ambiente de trabalho, a equipe seja informada das mudanças que vão ocorrendo ao longo do desenvolvimento. Isso também traz como benefício a eficiência nos processos de desenvolvimentos de softwares. Ao prover percepção de mudanças, o tempo gasto com implementações em desacordo com os requisitos, e que futuramente sendo percebidas terão de ser refeitas, é minimizado. Além disso, o ARARA possibilita que a implementação do software seja consistente com o projeto, uma vez que mudanças nos requisitos são percebidas e propagadas para que a equipe implemente as modificações necessárias.

7.1. Trabalhos Futuros

Como trabalhos futuros a autora sugere a avaliação do ARARA em projetos de desenvolvimento de software de outros domínios de negócio. Embora a proposta do ARARA tenha sido criar uma ferramenta colaborativa para prover percepção de mudanças em qualquer que seja o projeto[9] e domínio, o trabalho apresentado nesta dissertação só pôde ser aplicado em um projeto, cuja autora fez parte da equipe.

Como o escopo deste trabalho ficou restrito ao uso do ARARA, representando e relacionando somente os artefatos do tipo diagrama de atividades e de classes (análise e interface), a autora sugere a expansão do ARARA para representação e recuperação também dos demais tipos de artefatos. São eles:

- Diagrama de caso de uso,
- Diagrama de sequência,

[9] Desde que o projeto tenha uma ontologia de domínio, cujos conceitos sejam aplicados na criação dos artefatos do projeto.

- Diagrama de componentes,

Além disso fica a sugestão de melhorar o processo de representação dos artefatos através da expansão da leitura dos demais elementos que compõem um diagrama UML. Outros elementos, que não foram lidos nesse trabalho, podem também fornecer informações para refinar o processo de representação dos artefatos, por exemplo, a descrição textual que cada elemento possui.

Fazer análises estatísticas com bases maiores de artefatos também é uma sugestão para trabalhos futuros.

Implementar no ARARA um mecanismo de aprendizado para aperfeiçoar a recuperação de artefatos relacionados. Isso poderia ser alcançado com base na verificação dos artefatos que foram modificados após propagação da percepção de uma mudança. Ou seja, supondo que com a modificação do artefato $\boldsymbol{Artf_1}$, foram recuperados como artefatos relacionados o conjunto {$\boldsymbol{Artf_3}$, $\boldsymbol{Artf_6}$, $\boldsymbol{Artf_5}$, $\boldsymbol{Artf_{11}}$ e $\boldsymbol{Artf_9}$}. Desse conjunto foram modificados num segundo momento apenas os artefatos {$\boldsymbol{Artf_3}$, $\boldsymbol{Artf_6}$ e $\boldsymbol{Artf_5}$}. Após um tempo sempre que houver mudanças no artefato $\boldsymbol{Artf_1}$ o ARARA poderia apresentar apenas os artefatos {$\boldsymbol{Artf_3}$, $\boldsymbol{Artf_6}$ e $\boldsymbol{Artf_5}$} como candidatos a mudanças.

Esse aprendizado pode ser feito, por exemplo, através da interação da equipe com o ARARA. Os membros, após notificação de uma mudança, informam quais artefatos, dentre os recuperados, de fato foram impactados e necessitam ser revistos. Após um tempo o ARARA se configuraria de forma a já saber quais os artefatos são mais relevantes a serem apresentados dentre os relacionados.

Este procedimento também trataria o problema de sobrecarga de informação. Ocorrências de grande quantidade de falsos positivos *(fp)* podem impactar no

andamento do projeto, por requisitar dos membros notificados um tempo a ser gasto na análise de todos os artefatos recuperados para verificar a necessidade de mudanças.

Também com o objetivo de minimizar a sobrecarga de informação e aumentar a precisão nos artefatos recuperados como relacionados a um modificado, a autora sugere a implementação de análise semântica das mudanças realizadas de forma a verificar a real necessidade de propagar a percepção de mudança.

Por último a autora sugere a implementação do ARARA como um plugin para o Eclipse.

Referências Bibliográficas

Alani, H., 2003. TGVizTab: An Ontology Visualisation Extension for Protégé. In *Proceedings of Knowledge Capture (K-Cap'03), Workshop on Visualization Information in Knowledge Engineering*. Workshop on Visualization Information in Knowledge Engineering. Saniebel Isaland, Florida, USA.

Antoniol, G. et al., 2002. Recovering Traceability Links between Code and Documentation. *IEEE Transactions on Software Engineering*, 28(10), 970-983.

Baader, F. et al., 2003. *The Description Logic Handbook: Theory, Implementation, and Applications* 1º ed., United Kingdom: Cambridge University Press.

Bordini, R.H., Hübner, J.F. & Wooldridge, M.J., 2007. *Programming Multi-Agent Systems in AgentSpeak Using Jason* 1º ed., United Kingdom: Wiley-Interscience.

Capelli, C. et al., 2007. Uma Abordagem de Construção de Ontologia de Domínio a partir do Modelo de Processo de Negócio. In *Proceedings of the 2nd Workshop on Ontologies and Metamodels in Software and Data Engineering*. 2nd Workshop on Ontologies and Metamodels in Software and Data Engineering. João Pessoa/PB, Brazil, pp. 85-96.

Chandrasekaran, B., Josephson, J.R. & Benjamins, V.R., 1999. What Are Ontologies, and Why Do We Need Them? *IEEE Intelligent Systems*, 14(1), 20-26.

Cleland-Huang, J., Settimi, R. & Romanova, E., 2007. Best Practice for Automated Traceability. *IEEE Computer*, 40(6), 27-35.

Collins-Sussman, B., Fitzpatrick, B.W. & Pilato, C.M., 2008. Version Control with SubVersion.

Dourish, P. & Bellotti, V., 1992. Awareness and Coordination in Shared Workspaces. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*. Toronto, Ontario, Canada: ACM, pp. 107-114.

Edgington, T., Raghu, T.S. & Vinze, A., 2005. Knowledge Ontology: A Method for Empirical Identification of 'As-Is' Contextual Knowledge. In *Proceedings of the 38th Hawaii International Conference on System Sciences*. 38th Hawaii International Conference on System Sciences.

Edwards, M. & Howell, S.L., 1991. *A Methodology for Systems Requirements Specification and Traceability for Large Real Time Complex Systems*.

Endsley, M.R., 1995. Toward a Theory of Situation Awareness in Dynamic Systems. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 37, 32-64.

Falbo, R.D.A., 1998. *Integração de Conhecimento em um Ambiente de Desenvolvimento de Software*. DSc. Rio de Janeiro, RJ.: Universidade Federal do Rio de Janeiro.

Farshchian, B.A., 2000. Gossip: An Awareness Engine for Increasing Product Awareness in Distributed Development Projects. In *Proceedings of the 12th International Conference on Advanced Information Systems Engineering*. Springer-Verlag, pp. 264-278.

Farshchian, B.A., 2001. Integrating Geographically Distributed Development Teams Through Increased Product Awareness. *Information Systems*, 26(3), 123-141.

Gotel, O. & Finkelstein, C., 1994. An Analysis of the Requirements Traceability Problem. In *Proceedings of the 1st International Conference on Requirements Engineering*. First International Conference on Requirements Engineering. Colorado Springs, CO, pp. 94-101.

Gruber, T.R., 1995. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal of Human-Computer Studies*, 43(5-6), 907-928.

Guarino, N., 1997. Semantic Matching: Formal Ontological Distinctions for Information Organization, Extraction, and Integration. In *International Summer School on Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*. Springer-Verlag, pp. 139-170.

Guizzardi, G., 2000. *Uma Abordagem Metodológica de Desenvolvimento Para e Com Reuso Baseada em Ontologias Formais de Domínio*. MSc. Espírito Santo, Vitória: Universidade Federal do Espírito Santo.

Gutwin, C. & Greenberg, S., 2001. The Importance of Awareness for Team Cognition in Distributed Collaboration. In *In E. Salas and S. M. Fiore (Editors), Team Cognition: Understanding the Factors that Drive Process and Performance, Cap 9*. pp. 177-201.

Gutwin, C., Greenberg, S. & Roseman, M., 1996. Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation. In *Proceedings of HCI on People and Computers XI*. Springer-Verlag, pp. 281-298.

Horridge, M. et al., 2004. *A practical Guide to Building OWL Ontologies Using the Protégé-OWL Plugin and CO-ODE Tools* 1º ed., United Kingdom: The University Of Manchester.

IEEE, 1998. IEEE recommended practice for software requirements specification - IEEE std. 830-1998.

Jain, A.K. & Dubes, R.C., 1988. *Algorithms for Clustering Data*, Prentice-Hall, Inc.

Jambalaya, 2010. Jambalaya | The CHISEL Group, University of Victoria. Available at: http://www.thechiselgroup.org/jambalaya.

JDOM, 2010. JDOM. Available at: http://www.jdom.org/.

Jennings, N.R., 2001. An Agent-Based Approach for Building Complex Software Systems. *Communications of the ACM*, 44(4), 35-41.

Johnson, P.E., Zaulkernan, I. & Garber, S., 1987. Specification of Expertise. *International Journal of Man-Machine Studies*, 26(2), 161-181.

Kirsch-Pinheiro, M., Lima, J.V.D. & Borges, M.R.S., 2003. A Framework for Awareness Support in Groupware Systems. *Computers in Industry*, 52(1), 47-57.

Lima, E.J.C. et al., 2008. ARARA – Artifacts and Requirements Awareness Reiforcement Agents. In *Proceedings of the IADIS International Conference ISA (part of MCCSIS 2008)*. Conference ISA (part of MCCSIS 2008). Amsterdan, pp. 92-99.

López, M.F., 1999. Overview Of Methodologies For Building Ontologies. In *Proceedings of the IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5)*. Stockholm, Sweden.

Louridas, P., 2006. Version Control. *IEEE Software*, 23, 104-107.

Manning, C.D., Raghvan, P. & Schütze, H., 2009. *Introduction to Information Retrieval* Online ed., United Kingdom: Cambridge University Press.

Marcus, A., Maletic, J.I. & Sergeyev, A., 2005. Recovery of Traceability Links between Software Documentation and Source Code. *International Journal of Software Engineering and Knowledge Engineering*, 15(5), 811-836.

Miranda, M., Xexéo, G.B. & Souza, J.M., 2006. Building Tools for Emergent Design with COPPEER. In *Proceedings of the 10th International Conference on Computer Supported Cooperative Work in Design*. pp. 550-555.

Miranda, M.D.G. & Xexéo, G.B., 2005. A Complex Adaptive System Approach for Agent-Based Peer-To-Peer Collaborative Applications. In *Workshop de Teses e Dissertações em Banco de Dados (WTDBD)*.

Miranda, M.G., Xexéo, G.B. & Souza, J.M., 2007. Coppeer Documents: An Agent Based Approach to Collaborative and Incremental Development of Document Oriented Peer-to-peer Systems. In *Proceedings of the 11th International Conference on Computer Supported Cooperative Work in Design*. Melbourne, Vic., pp. 383-388.

MOF/XMI, 2007. MOF 2.0/XMI Mapping Version 2.1.1.

Neighbors, J.M., 1980. *Software Construction Using Components*. Ph.D. California, Irvine: Department of Information and Computer Science University of California.

OMG/UML, 2007. OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2.

OMG/UML, 2009. OMG Unified Modeling Language (OMG UML), Superstructure, V2.2.

OWL, 2009. OWL Web Ontology Language Guide. Available at: http://www.w3.org/TR/owl-guide/.

OWLViz, 2010. OWLViz - Protege Wiki. Available at: http://protegewiki.stanford.edu/wiki/OWLViz.

Porter, M.F., 1997. An Algorithm for Suffix Stripping. In *In Readings in Information Retrieval*. United Kingdom: Morgan Kaufmann Publishers Inc., pp. 313-316.

Pressman, R.S., 2006. *Engenharia de Software* 6 ed., McGraw-Hill.

Protégé, 2010. The Protégé Ontology Editor and Knowledge Acquisition System. Available at: http://protege.stanford.edu/.

Protégé-OWL API, 2010. Protégé-OWL API. Available at: http://protege.stanford.edu/plugins/owl/api/.

RDF Primer, 2004. RDF Primer. Available at: http://www.w3.org/TR/rdf-primer/.

RSA, 2010. IBM - Rational Software Architect for WebSphere Software - Software. Available at: http://www-01.ibm.com/software/awdtools/swarchitect/websphere/.

SEI, 2006. Software Engineering Institute. Available at: http://www.sei.cmu.edu/.

Storey, M. et al., 1997. On Integrating Visualization Techniques for Effective Software Exploration. In *Proceedings of IEEE Symposium on Information Visualization, 1997*. Information Visualization, 1997. Proceedings., IEEE Symposium on. pp. 38-45.

Storey, M. et al., 2004. Visualization and Protégé. In *Proceedings of 7th Internationa Protégé Conference*. 7th International Protégé Conference. Bethesda, Maryland, pp. 1-4.

Subversive, 2010. Subversive. Available at: http://www.eclipse.org/subversive/documentation/preferences/pref_diff_viewer.php.

SVN, 2010. subversion.tigris.org. Available at: http://subversion.tigris.org/.

SVNKit, 2010. SVNKit :: Subversion for Java. Available at: http://svnkit.com/.

SWRL, 2004. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Available at: http://www.w3.org/Submission/SWRL/.

TGViz, 2009. TGViz - Protege Wiki. Available at: http://protegewiki.stanford.edu/wiki/TGViz.

TouchGraph, 2010. TouchGraph | Products: Navigator. Available at: http://www.touchgraph.com/navigator.html.

Uschold, M. & Gruninger, M., 2004. Ontologies and Semantics for Seamless Connectivity. *ACM SIGMOD Record*, 33(4), 58-64.

Uschold, M. & Gruninger, M., 1996. Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review*, 11(02), 93-136.

Uschold, M. & King, M., 1995. Towards a Methodology for Building Ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing: International Joint Conference on Artificial Intelligence*.

Van Heijst, G., Schreiber, A.T. & Wielinga, B.J., 1997. Using Explicit Ontologies in KBS Development. *International Journal of Human-Computer Studies*, 46(2-3), 183-292.

Wang, A.I., Conradi, R. & Liu, C., 1999. A Multi-Agent Architecture for Cooperative Software Engineering. In *Proceedings of Eleventh International Conference on software Engineering and Knowledge Engineering*.

Wikipedia, 2010. Arara-de-barriga-amarela - Wikipédia, a enciclopédia livre. Available at: http://pt.wikipedia.org/wiki/Arara-de-barriga-amarela.

Wooldridge, M., 1997. Agent-Based Software Engineering. *Software Engineering IEEE Proceedings*, 144, 26--37.

XMI, 2003. XML Metadata Interchange (XMI) Specification Version 2.0.

XML Schema, 2004. XML Schema Part 2: Datatypes Second Edition. Available at: http://www.w3.org/TR/xmlschema-2/.

Zhang, Y. et al., 2006. An Ontology-Based Approach for Traceability Recovery. In *Proceedings of 3rd International Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering*. pp. 36-43.

Anexos

A Ontologia

Criando Elementos da Ontologia no Protégé

A ferramenta disponibiliza um local específico para criação e visualização de cada um dos elementos da ontologia (*Subclass Explorer*, *Property Browser*, *Instance Browser* e *Form Browser*) e também um local para edição dos elementos criados (*Class Editor*, *Property Editor*, *Individual Editor* e *Form Editor*).

As classes e suas subclasses são criadas no *Subclass Explorer* e a descrição e restrições de cada classe são criadas no *Class Editor*. A Figura 26 apresenta a esquerda o *Subclass Explorer* e a direita o *Class Editor*. As restrições são criadas através da tela apresentada na Figura 27.

As propriedades são criadas no *Property Browser*. Se desejável as propriedades podem ser criadas separadamente por tipo: Objeto, Tipo de Dado e Anotação. A Figura 28 apresenta o *Property Browser* para as propriedades do tipo Objeto e a Figura 29 apresenta o *Property Browser* para as propriedades do Tipo de Dado. À direita em ambas as figuras é apresentado o *Property Editor*. É no editor das propriedades que são definidos os domínios e contra-domínios de cada propriedade (relacionamento entre os conceitos) e também a descrição de cada propriedade.

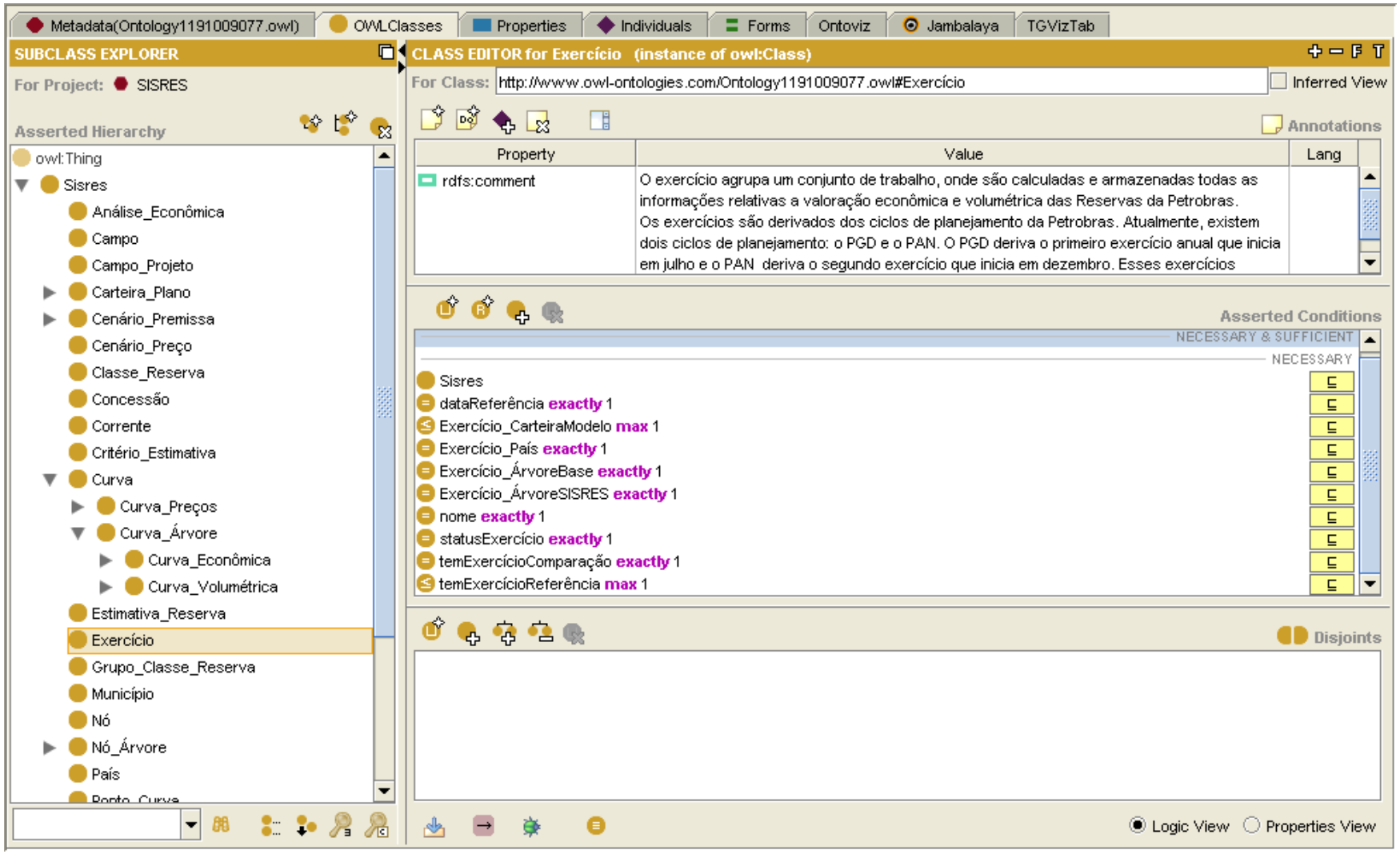

Figura 26. Editor e browser de classes na ferramenta Protégé.

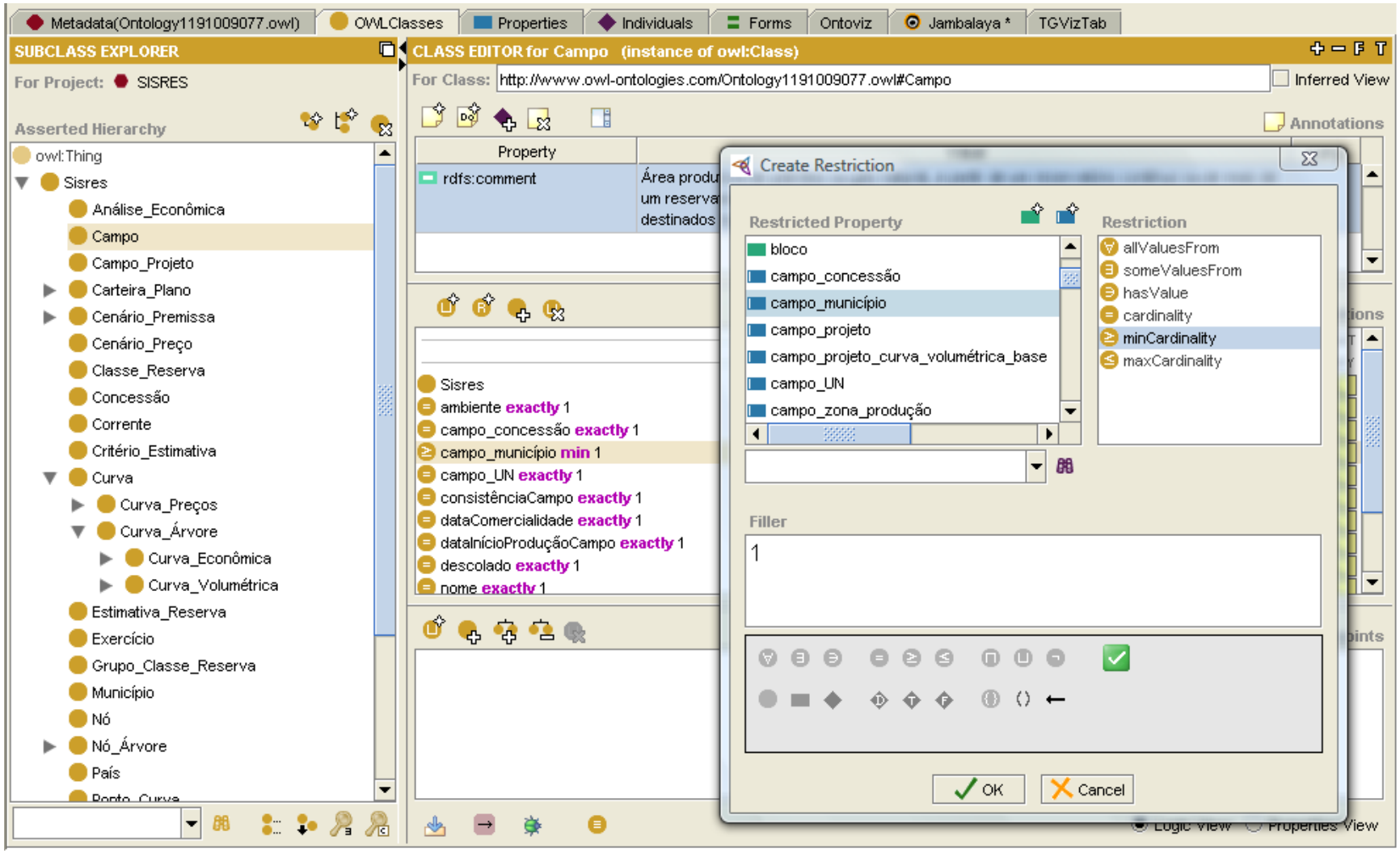

Figura 27. Criando restrições em um classe da ontologia.

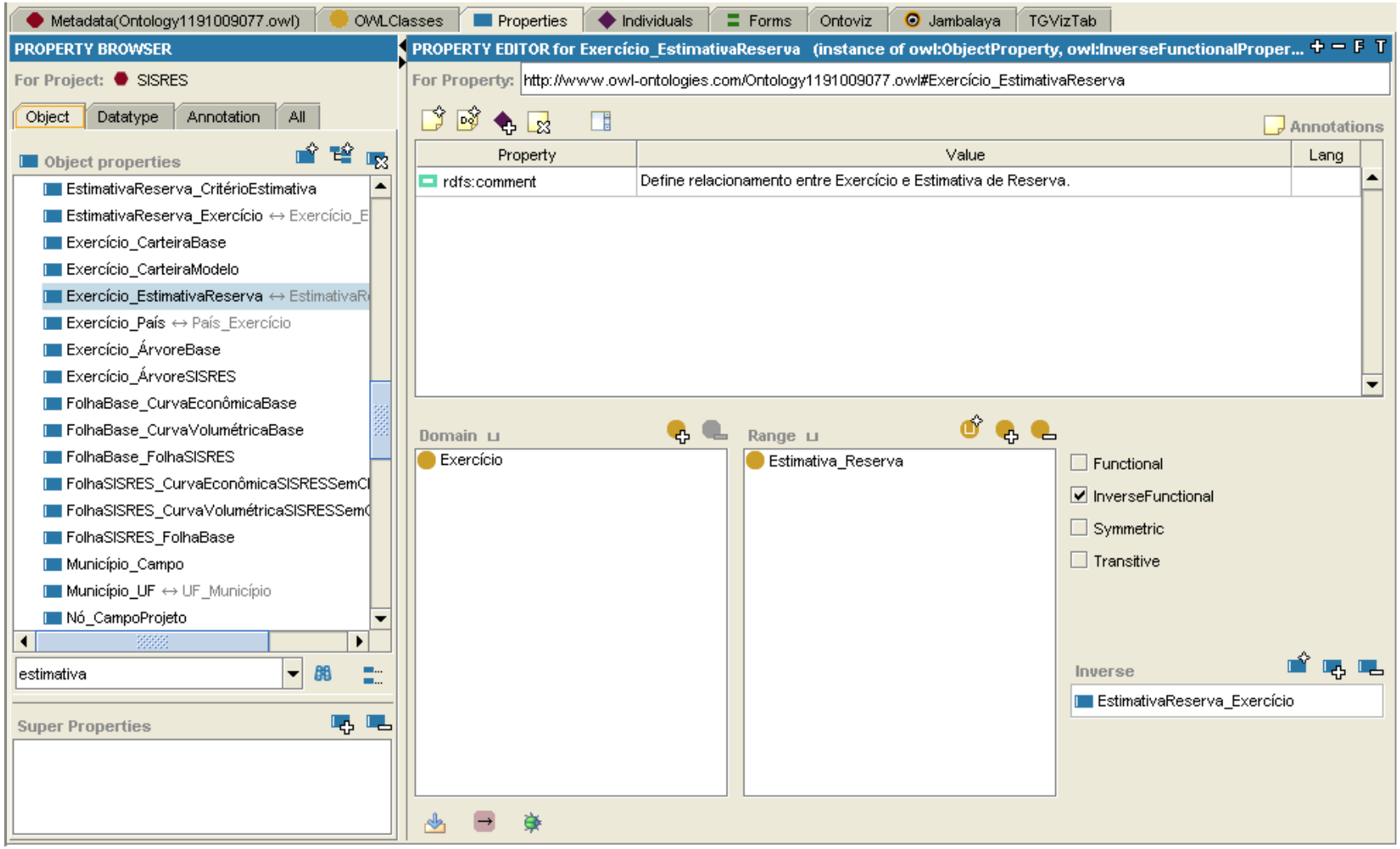

Figura 28. Editor e browser de Propriedade do tipo Objeto na ferramenta Protégé.

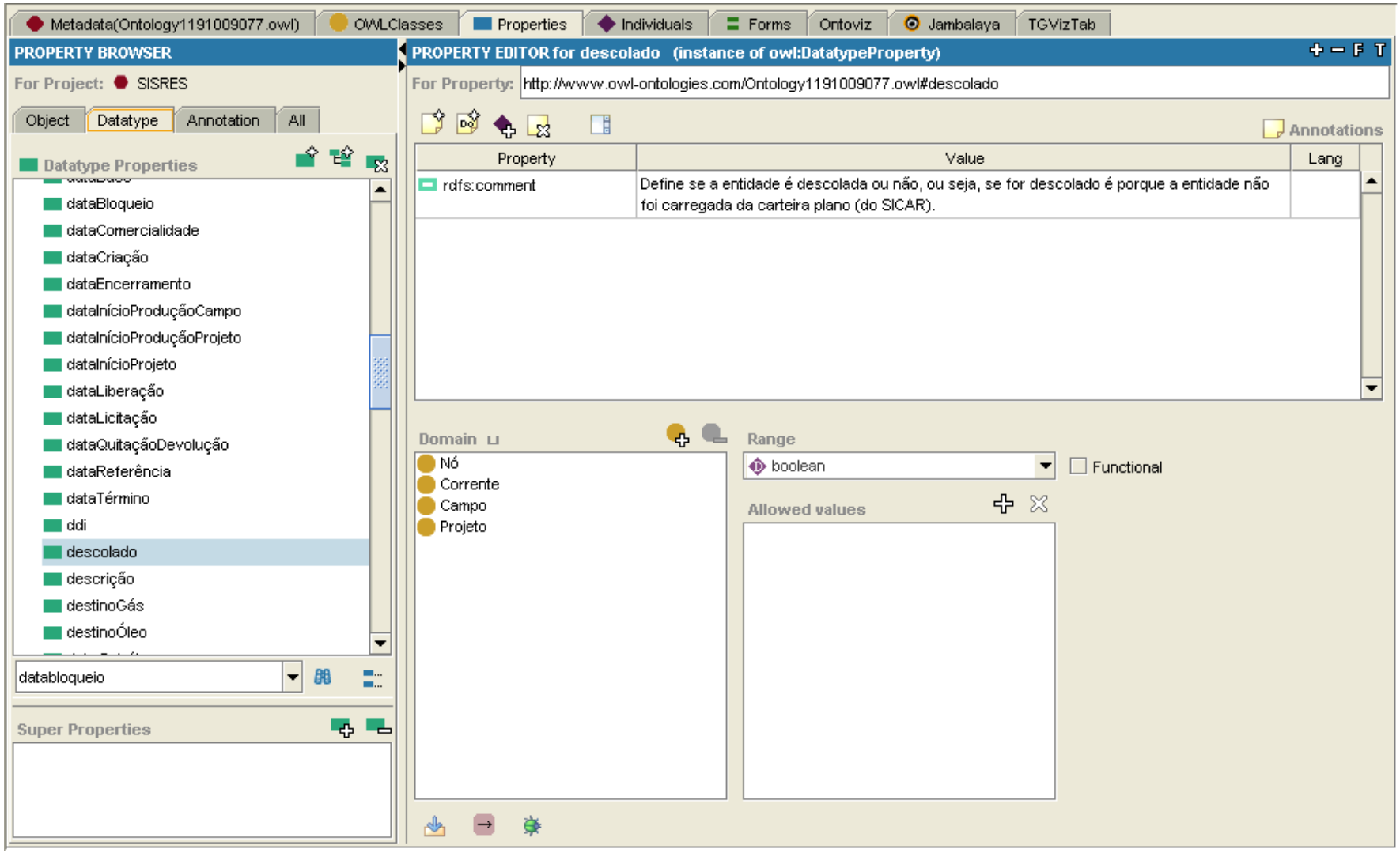

Figura 29. Editor e browser de propriedade do tipo de dado na ferramenta Protégé.

Visualização da Ontologia

É importante que de alguma forma a ontologia possa ser apresentada para visualização sem ser através da interface de criação, onde os elementos da ontologia são apresentados de forma estruturada, como pode ser observado nas figuras apresentadas anteriormente. Uma apresentação gráfica da ontologia ajuda a visualizar a ontologia e navegar de forma interativa entre os conceitos da ontologia.

O Protégé dispõem de uma série de plug-ins para apresentar graficamente uma ontologia. No projeto foram usados alguns plug-ins para facilitar determinadas visualizações da ontologia que estava sendo criada. Foram eles:

- OWLViz (OWLViz 2010) → permite a visualização das classes e a navegação por sua hierarquia.
- Jambalaya (Jambalaya 2010) → baseado no ShriMP (M.-A.D. STOREY *et al.*, 1997) apresenta a ontologia (Margaret-Anne Storey et al. 2004).
- TGViz (TGViz, 2009) → baseado no TouchGraph (TOUCHGRAPH, 2010), que é um ambiente Java open source para criação e navegação de redes gráficas iterativas, o TGViz permite a visualização das classes, propriedades e instâncias em árvore hiperbólica (ALANI, 2003).

A Figura 30 apresenta uma imagem, capturada através do plug-in Jambalaya, da visualização das classes da ontologia.

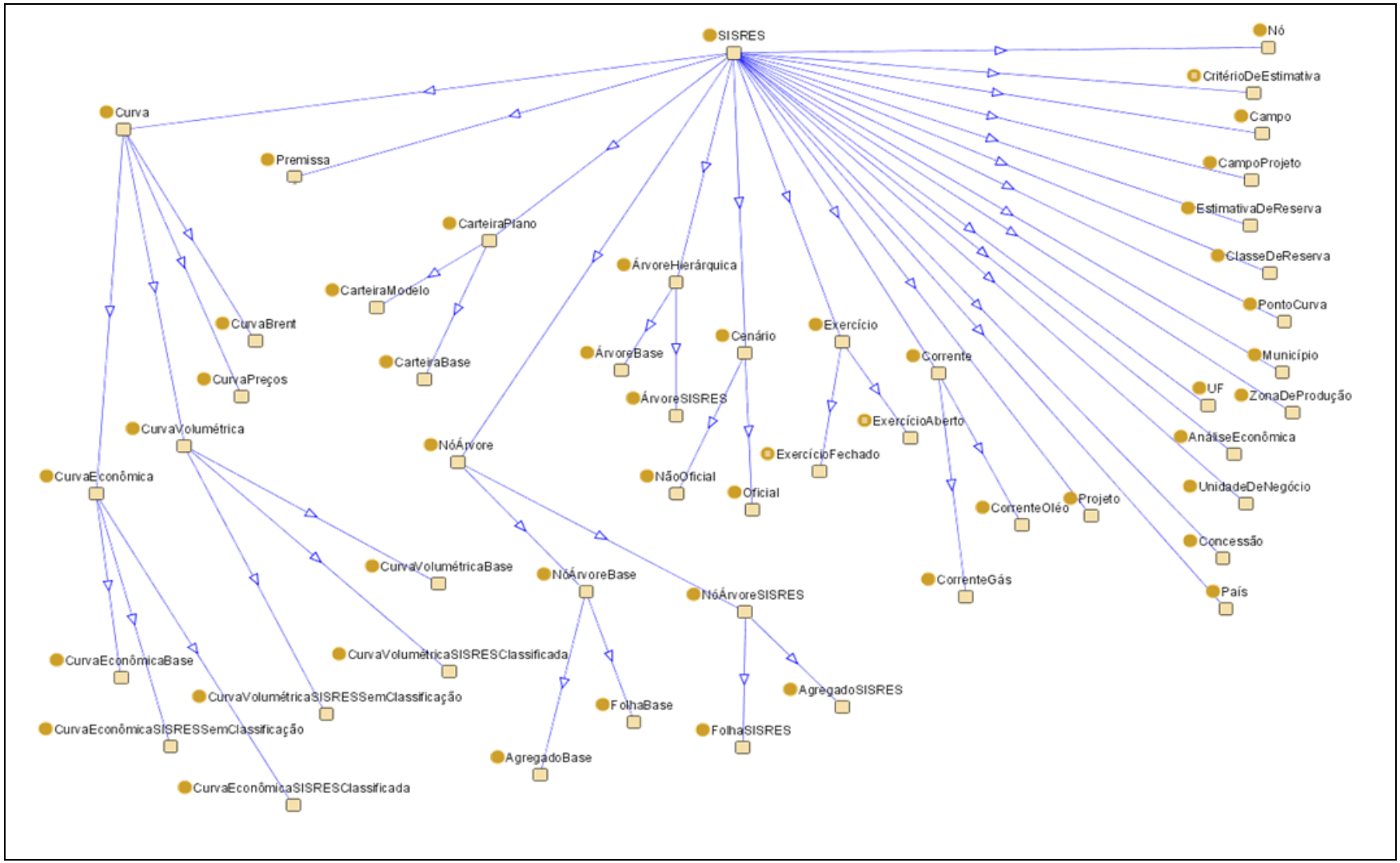

Figura 30. Visualização das classes da ontologia através do Plug-in Jambalaya.

Os relacionamentos entre as classes (domínios e contra-domínios das propriedades do tipo objeto) também podem ser visualizados através do Jambalaya conforme apresentado na Figura 31. Essa figura ilustra como os conceitos mapeados na ontologia estão ligados entre si e como navegar entre eles. Essa navegação entre os conceitos que forma a base proposta para ser alcançada a rastreabilidade nos artefatos do projeto para percepção de mudanças e propagação das mesmas.

A árvore hiperbólica da ontologia criada é apresentada através do TGViz conforme ilustrado na Figura 32. São apresentados nessa figura o relacionamento entre as classes até o terceiro nível.

A Figura 33 apresenta também uma visualização em árvore hiberbólica, mas de uma determinada classe da ontologia (exemplo: Exercício) e até três níveis de relacionamentos. É possível observar pela figura os conceitos que estão ligados direta e indiretamente ao conceito apresentado.

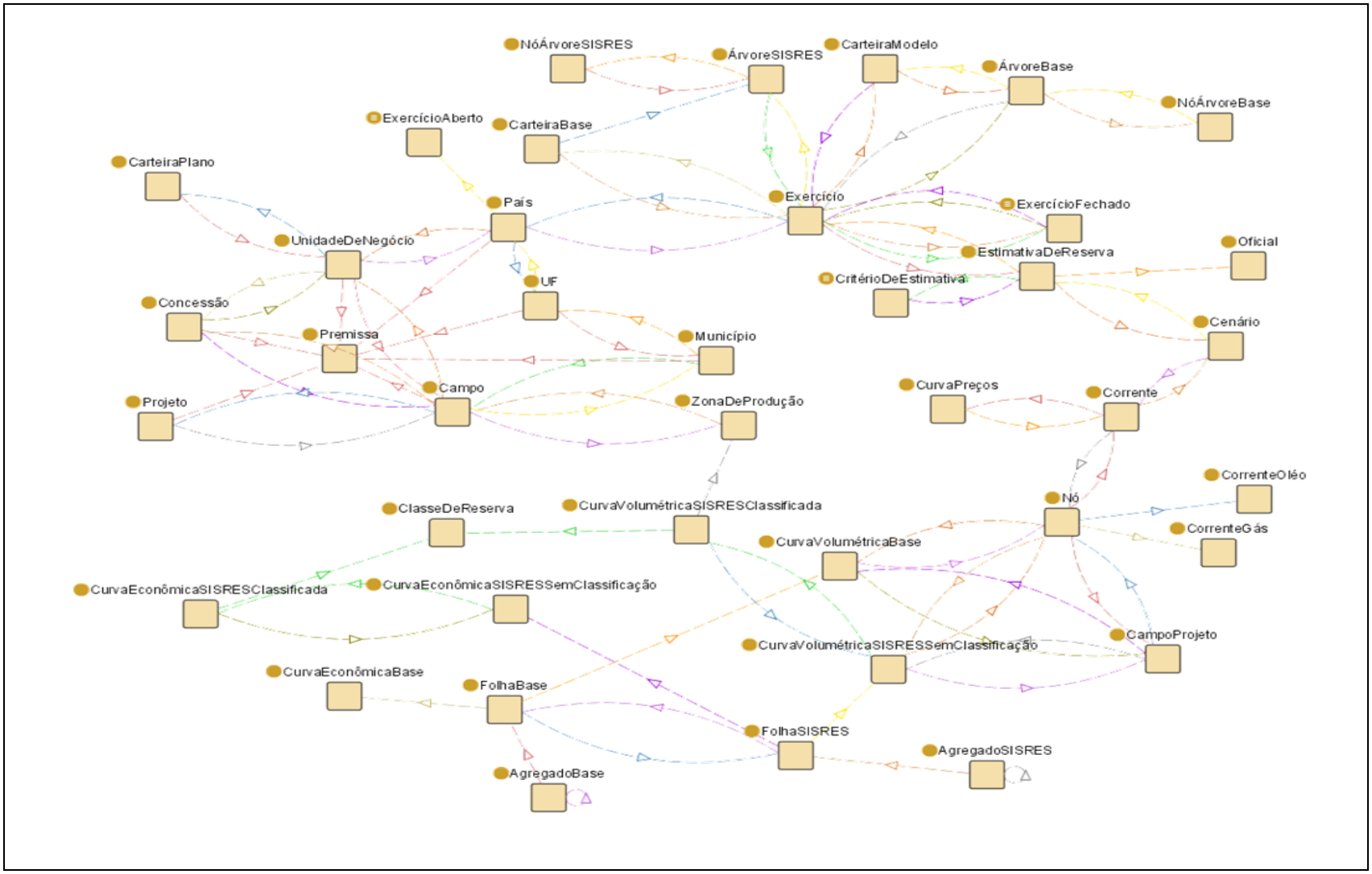

Figura 31. Visualização domínio e contra-domínio das propriedades do tipo objeto (relacionamento entre as classes) através do plug-in Jambalaya.

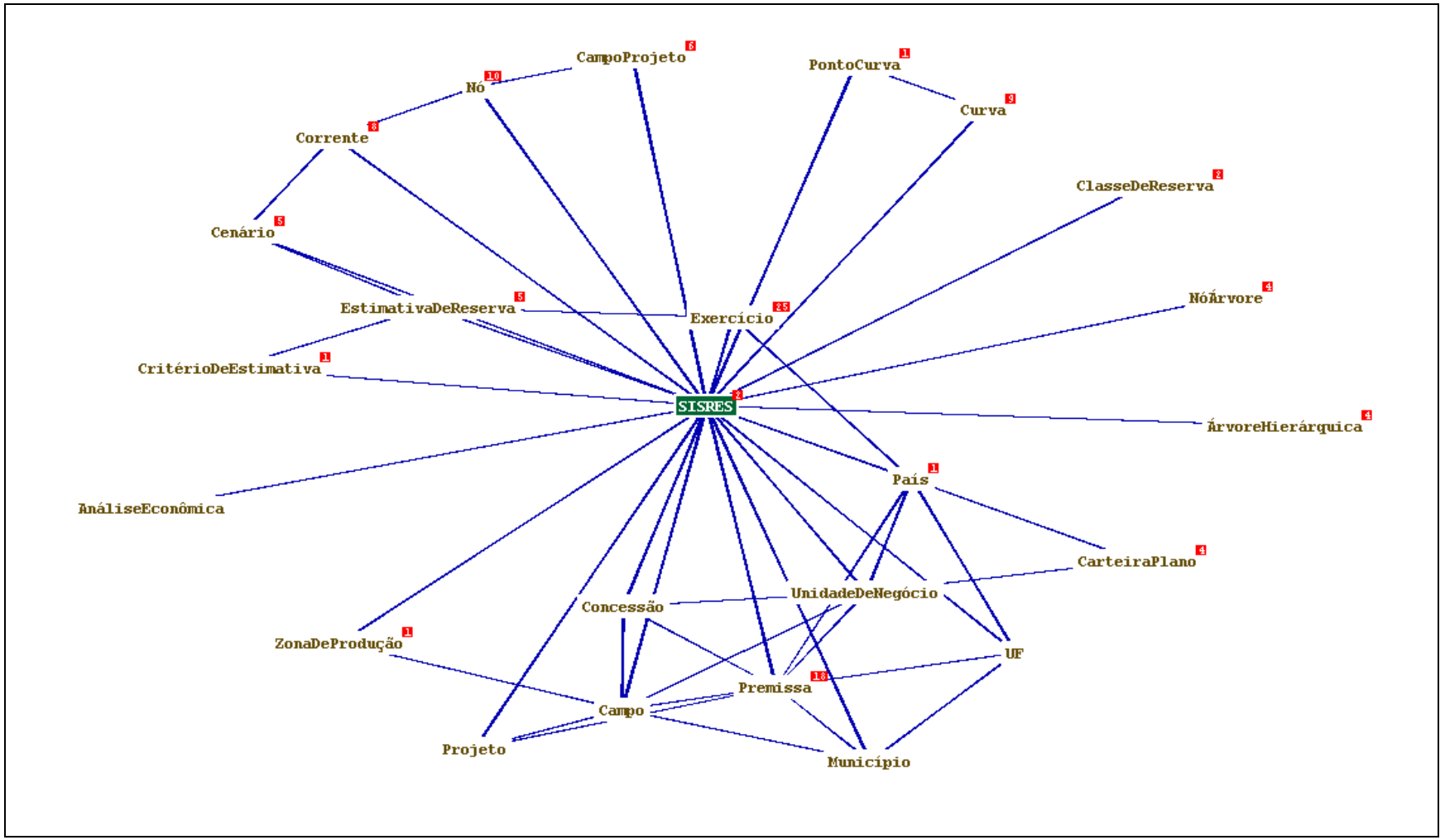

Figura 32. Visualização das classes da ontologia em árvore Hiperbólica através do plug-in TGViz.

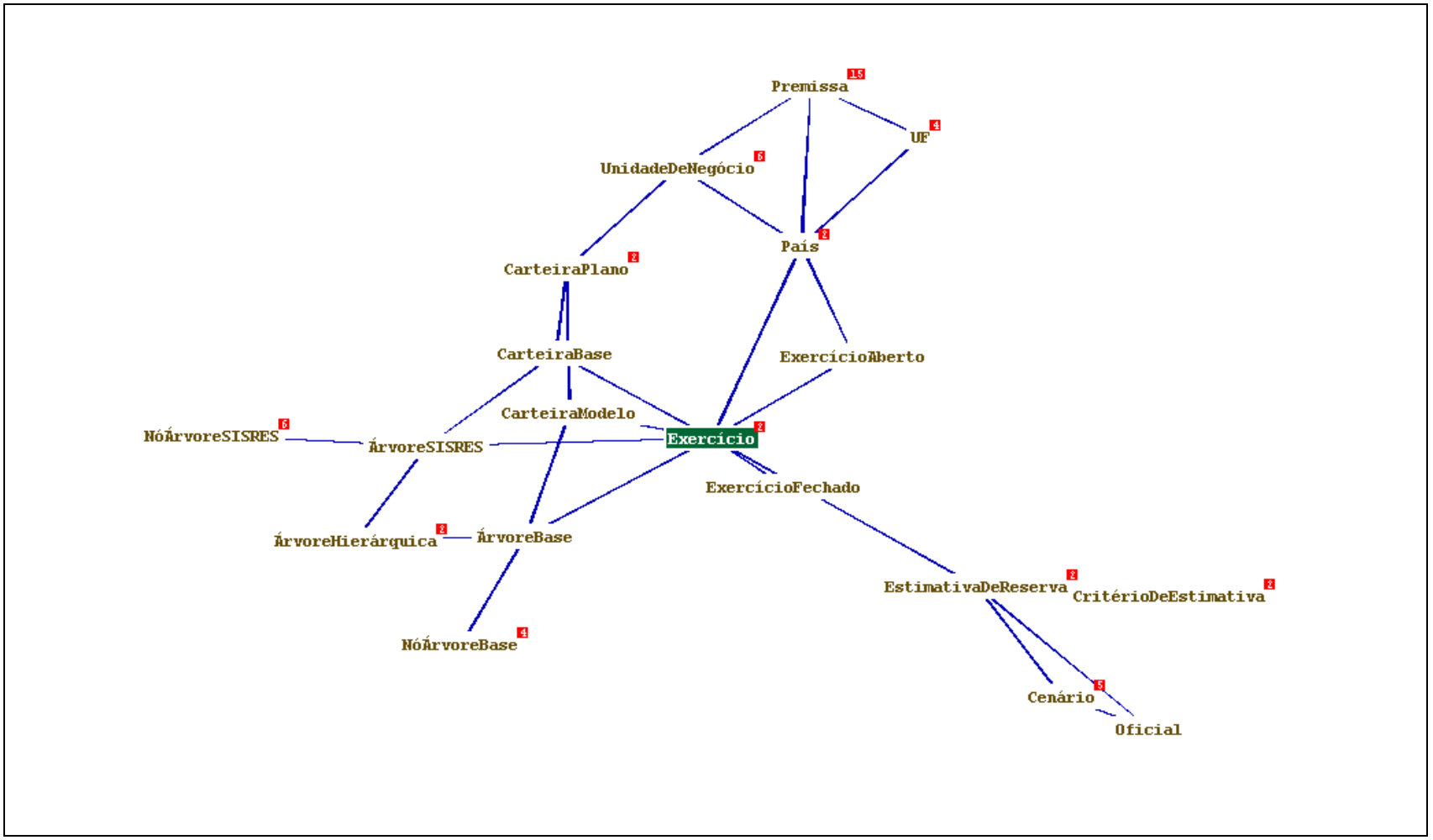

Figura 33. Visualização em árvore hiperbólica até o terceiro nível de relacionamento do conceito Exercício mapeado na ontologia da reserva.

Construtores OWL

Tabela 7. Construtores da sublinguagem OWL Lite.

RDF Schema Features:	(In)Equality:	Property Characteristics:
Class (Thing, Nothing) rdfs:subClassOf rdf:Property rdfs:subPropertyOf rdfs:domain rdfs:range Individual	equivalentClass equivalentProperty sameAs differentFrom AllDifferent distinctMembers	ObjectProperty DatatypeProperty inverseOf TransitiveProperty SymmetricProperty FunctionalProperty InverseFunctionalProperty
Property Restrictions:	**Restricted Cardinality:**	**Header Information:**
Restriction onProperty allValuesFrom someValuesFrom	minCardinality (only 0 or 1) maxCardinality (only 0 or 1) cardinality (only 0 or 1)	Ontology imports
Class Intersection:	**Versioning:**	**Annotation Properties:**
intersectionOf	versionInfo priorVersion backwardCompatibleWith incompatibleWith DeprecatedClass DeprecatedProperty	rdfs:label rdfs:comment rdfs:seeAlso rdfs:isDefinedBy AnnotationProperty OntologyProperty
Datatypes		
xsd datatypes		

A lista dos construtores para as sublinguagens OWL DL e OWL Full é a lista apresentada na Tabela 7 em adição a lista apresentada na Tabela 8.

Tabela 8. Construtores adicionais aos construtores da sublinguagem OWL Lite

Class Axioms:	Boolean Combinations of Class Expressions:
oneOf, dataRange disjointWith equivalentClass (applied to class expressions) rdfs:subClassOf (applied to class expressions)	unionOf complementOf intersectionOf
Arbitrary Cardinality:	**Filler Information:**
minCardinality maxCardinality cardinality	hasValue

B Publicações da Autora Relacionadas ao ARARA

A seguir encontram-se reproduzidos os seguintes artigos relacionados ao ARARA:

1. Lima, E.J.C. et al., 2008. ARARA – Artifacts and Requirements Awareness Reiforcement Agents. In *Proceedings of the IADIS International Conference ISA (part of MCCSIS 2008)*. Conference ISA (part of MCCSIS 2008). Amsterdan, pp. 92-99.

2. Lima, E.J.C. et al., 2008. ARARA – A Collaborative Tool to Requirement Change Awareness. In *Proceedings of the 14th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2010),* Shanghai, 2010.

ARARA: Artifacts and Requirements Awareness Reinforcement Agents

Ester J. C. de Lima[1], José A. Rodrigues Nt.[1], Geraldo B. Xexéo[1,2], Jano M. de Souza[1,2]
1- Graduate School and Research in Engineering/Federal University of Rio de Janeiro (COPPE/UFRJ)
2- Computer Science Department - Institute of Mathematics/Federal University of Rio de Janeiro (IM/UFRJ)
PO Box 68.511 – ZIP 21.941-972 – Rio de Janeiro – Brazil
{esterlima, rneto, xexeo, jano}@cos.ufrj.br

ABSTRACT

ARARA, an agent-based system, is an application where a myriad of simple agents track, identify, and notify changes on software-process artifacts. ARARA provides workspace awareness to team-development members, which are notified when changes occur on artifacts. Ontologies support this application for they are used by agents as knowledge base to retrieve information, tag the artifacts, detect changes, and search for responsibilities of project members.

KEYWORDS

Agents, Awareness, Ontology, Information Retrieval and Traceability.

1. INTRODUCTION

In this article we propose ARARA,[10] **A**rtifacts and **R**equirements **A**wareness **R**einforcement **A**gents, an agent-based system (ABS) where a myriad of simple agents track, identify, and notify changes on software-process artifacts to aware project members whose work is affected by them. In the scope of this work, software-process artifacts are meant to be conceptual and design models, like activity diagrams, use-case diagrams and class diagrams, and system code, all of them based on user requirements.

Considering that software development process (SDP) is a cooperative work, and communication is recognized as a critical task (Pressman, 2006), ARARA's goal is to inform members of a project of what is going on, and specifically, what changes can affect their work. This problem is even more relevant in present-day projects where the team can be distributed over a large geographical area, such as in open-source and offshore development projects (Gutwin, C. et al, 1995).

Awareness is the understanding of the activities of others and it provides a context for your own activity (Dourish, P. and Bellotti, V., 1992). This understanding can be achieved by observing what other members of the project are doing or the result of their actions, which can be identified by the changes they do to the software-process artifacts. According to Gutwin et al. they call "*workspace awareness as the collection of up-to-the minute knowledge a person holds about the state of another's interaction with the workspace. Workspace awareness helps people move between individual and shared activities, provides a context in which to interpret other's utterances, allows anticipation of other's actions, and reduces the effort needed to coordinate tasks and resources" (*Gutwin, C. et al, 1996) (Gutwin, C. and Greenberg, S., 2004).

Once SDP produces a large set of documents and code, which are usually shared by subsets of the project members, a change in a document can propagate through several artifacts. Therefore, it is important that all project's members are always aware of the changes made.

The first task of ARARA is to identify artifact changes and effectively notify project members. Members must be traceable to guarantee that changes will be reported to them. However, if all project members receive a notification of every change, they will spend too much time filtering out information that does not concern to them. To solve this problem, ARARA provides a way to identify to whom a change should be reported, i.e., who should benefit from this advice. This is the second task of ARARA.

In addition to identifying changes, ARARA also takes care of relating them to the interests or responsibilities of project members. It establishes such a relation between artifact and member tracking which artifacts a project member works with. It uses document's tags and ontologies to identify the concepts related to other artifacts affected by the change, which may also be subject to revision (Figure

[10] ARARA is a Brazilian bird known by its eloquence.

1). When an artifact is changed, ARARA navigates through its ontologies, finds related concepts and its associated artifacts, and notifies the proper members. Artifacts are linked to concepts using information retrieval techniques.

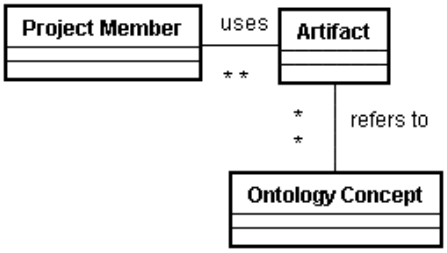

Figure 1. Conceptual view of the relation between Project Members, Artifacts and Ontology Concepts

The third task of ARARA is to identify the exact portion of the artifact that was changed as a way to track the artifacts that shall be affected in consequence. Since artifacts of a SDP can be large and complex, ARARA tries to discover the concepts on the artifact that was affected. Then it uses a project knowledge base, which is used to maintain the project artifacts tagged with a set of concepts that represents them on the business ontology, to find the other artifacts that are associated with the one changed.

Finally, ARARA must verify if there are new concepts or if concepts already mapped have changed in order to implement or suggest changes to the ontologies. This is the most difficult task, because it implies reorganization of ontologies, which are themselves the knowledge base used by ARARA in its other tasks.

The paper is organized as follows. In the next section we present the problem and provide the background necessary to build the proposed solution. In section 3 we present ARARA and Section 4 presents conclusions and some considerations pointing out some directions for future works.

2. THE PROBLEM

SDP generates a large amount of artifacts, e.g., activity diagrams, use-case diagrams, class diagrams, conceptual and design models, and system code, all of them based on user requirements.

Tracking requirements is considered one of the important practices of SDP (Cleland-Huang, J. et al, 2007), being a specific practice of requirements management in CMMI (REQM SP 1.4) (SEI, 2006). Although SDP artifacts are all related, at least referring to common purposes, and current standards require traceability (IEEE, 1998), relations among the artifacts are not always explicit. Moreover, as pointed out by other authors, traceability of requirements and other SDP documents is still a problem (Cleland-Huang, J. et al, 2007). Considering changes are common place in development projects, this lack of traceability demands that project members have full knowledge about artifacts' relations, be them explicit or implicit, and other members' work. This knowledge is necessary for correctly introducing changes to the project, without compromising its integrity.

Required knowledge is usually obtained from understanding artifacts' relations in the software development project, from the intrinsic relations implied by the concepts treated by each of them, and from the activities of other project members. The construction and maintenance of such knowledge is done individually and with no specific method. It is usually hard to obtain and extremely fragile. The issues associated to knowledge maintenance and consequent project integrity implications are amplified in development projects that are long-termed or have large teams. This gets even worse in the case of distributed development projects, nowadays common due to open-source software development and outsourcing practices.

In this context, if members are not aware of what others are doing, it is difficult to detect changes and resulting implications, e.g., the propagation of changes among artifacts. This is even more pronounced in the absence of a common physical workplace, where members' interactions can contribute to the leverage of project knowledge.

The use of an application that can improve team members awareness in an automatic way, notifying them of changes and of which artifacts may have been affected, can help guarantee that artifacts will always be consistent and traceable. Automatic awareness optimizes work without excessive cost to the project. ARARA is an agent-based system that provides automatic awareness.

2.1 Agent-based system

Nowadays, it is believed that Multi-Agent Systems (MAS) are a better way to model and support distributed, open-ended systems and environments. A MAS is a loosely coupled network of problem solvers (agents) that work together to solve a given problem (Wang, A. et al, 1999).

With agents a complex system can be break down into a set of decentralized, cooperative subsystems being able to cope with the characteristics features of a distributed environment. An intelligent agent is generally regarded as an autonomous decision making system, which senses and acts in some environment. In addition, an agent is pro-active as it is able to exhibit goal-directed behavior by *taking the initiative.* (Wooldridge, M., 1997).

Agents interact with other agents, via some agent-communication language, in order to achieve their goals. As Jennings says (Jennings, N. R., 2001), there is typically some underpinning organizational context to agents' interaction. For example, they may be peers working together in a team or one may be the manager of the others. Adopting an agent-oriented approach to software engineering means decomposing the problem into multiple, autonomous components that can act and interact in flexible ways to achieve their set objectives (Jennings, N. R., 2001).

In this context, ARARA is a cooperative system reinforced by agents, as MAS is the best option to model and support this kind of system, to provide artifacts and requirement awareness.

2.2 Ontology

Gruber (Gruber, T.R., 1993) defines Software Engineering Ontology as "a formal, explicit specification of a shared conceptualization in the domain of software engineering". It is a way to clarify knowledge structure and formalize it to represent adequately the domain and to enable a better communication between humans and machines.

With ontology one can have a vocabulary to represent the domain that can serve as pattern to be used in all other artifacts, which are the result of system analysis. Object-Oriented design of software systems is similar to domain ontology, due to existence of objects, their attributes, and relation between them. It more or less mirrors aspects of the domain that are relevant to the application (Chandrasekaran, B. et al, 1999).

Ontology can be seen as the heart of domain knowledge representation. Everything next is derived from the ontology. Furthermore ontology is the center of all artifacts, and by using the vocabulary's ontology on them it is easy to trace associated artifacts and to extract semantic meaning of what the artifact "is all about."

In this context the use of Software Engineering Ontology (SWEO) helps in discovering the link between artifacts. Therefore it is possible to infer which type of document needs to be modified first. Otherwise, the Business Ontology represents the domain and helps, when a concept is changed, to know other related concepts and, consequently, other artifacts that also need to be modified.

2.3 Awareness

Awareness is knowledge created through interaction between an agent and its environment – in simple terms, "*knowing what is going on*" (Endsley, M., 1995).

As artifacts are always in constant change in SDP, members of the development team must be aware of these changes in order to complete their tasks and keep project artifacts complete and consistent.

There are several groupware tools that provide awareness, allowing team members to work together from different places. E-mail and bulletin-boards that promote asynchronous remote communication, and instant-messaging, video-conference and virtual collaborative environments, which promote synchronous remote communication.

Our interest is to automatically detect project artifacts changes, providing awareness to team members and consistence among all project artifacts. There is no need for team members to search for what is going on, avoiding delay of their work.

2.4 Software development process

The initial phase in some SDP is business process analysis, where business requirements are elicited through business process modeling. For the sake of clarity, a generic SDP model will be used to exemplify the use of ARARA. In its initial phase, team members can prepare UML diagrams (activity

diagrams, use-case diagrams and descriptions, etc). It is advised that the Business Ontology of the domain be prepared before this phase, or in parallel with the preparation of such diagrams, for the terms defined in the ontology will be used in the UML artifacts to link them with each other.

In the next phase, team members prepare the conceptual model of the system, where classes, relations between them, and properties are created to define the domain.

The set of artifacts is composed by different types of documents, and as one artifact is derived from another, it is important to link them to ease the traceability process. One of the best practices to create traceable artifacts is to use a well-defined project glossary; another is to incorporate domain knowledge into the traceability infrastructure (Cleland-Huang, J. et al, 2007).

We can provide traceability with the use of domain's ontology that defines the correct terms to represent the domain's concepts. These terms are propagated to all other artifacts, e.g., software requirements specifications, use-case diagrams and descriptions, and class diagrams. Using SWEO we can specify the link between the artifacts.

2.5 Related works

Substantial work has been done for awareness support in collaborative systems. The "Big Watch Framework" (BW Framework) is an example of this effort (Pinheiro, M. Et al, 2003). It aims to supporting past events awareness and has been design to be flexible enough to be used to improve existent groupware applications and also to build new ones.

This framework is based on three-layer structure: registering, monitoring and notifying. Different from ARARA (that provides awareness supported by MAS.), BW Framework adopted an event-based awareness mechanism. In the first phase the groupware registers in the framework what events are interesting for awareness purposes. In the second phase the activities are happening inside the groupware and once one of these activities is executed, the groupware can pass to the BW framework the event related to this activity. The last phase consists of informing the user about what has happened inside the group work and the framework executes a filtering of the available information, based on profiles, that specifies the user's or role's preferences about which activities, among the group activities, should be notified and the time interval in which they are interesting. A similar filtering mechanism has been proposed by David (David, J. Et al, 2001).

3. IMPLEMENTING THE SOLUTION

ARARA is a P2P application that is being implemented on the COPPEER framework, which is a Multi-Agent System (MAS) framework designed to support P2P cooperative applications under a complex adaptive system paradigm (Miranda, M. et al, 2006). Figure 2 represents the COPPEER framework.

A P2P application developed in COPPEER is defined by its environment, which is a set of interconnected cells sharing a common namespace. Each computer that wants to access this application must create an agency related to that application running its agents. An agent is a piece of software associated with an environment, which access cells and moves across agencies to perform distributed computations. A cell offers agents an interface containing operations to write and read entries, subscribes for notification about the writing of entries and establishes or terminates connections to other known cells.

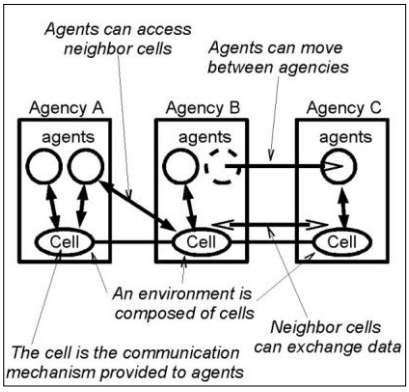

Figure 2. COPPEER framework

In order to determine the behavior of agents, application developers must implement behavior objects and pass it to the agency during agent creation. A behavior object contains methods that will be invoked when relevant events, such as agent creating, agent moving, or cell notification, occur.

Although COPPEER technology is wide-open to any type of strategy for implementing agents, our approach is to implement single-function and specialized agents that cooperate through entry exchange, which should be uncoupled. In this way, any agent can be changed or overloaded without need to change other agents.

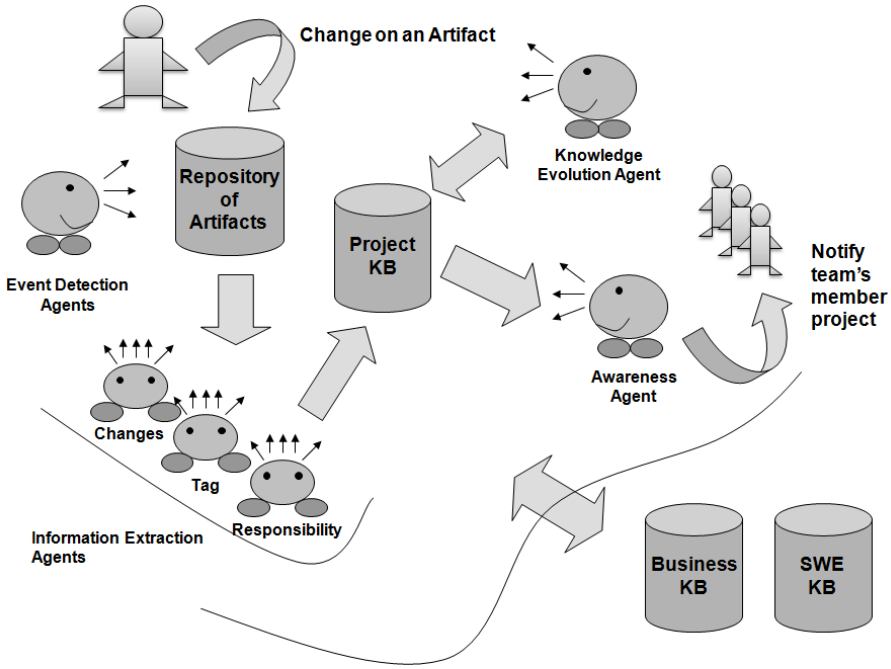

Figure 3. Description of ARARA

ARARA is under development for deployment in a large software project of a large company, where requirements constantly change and lack of awareness is a fact, as members of the development team are all dispersed geographically within the company.

The solution being implemented is shown in Figure 3. Tasks of ARARA are:

- To identify artifacts changes;
- To describe the changes and to identify the portion of the artifact that was changed;
- To identify to whom a change should be reported; and
- To suggest proper changes to the reference ontologies if there are new concepts or if the existing concepts have been changed.

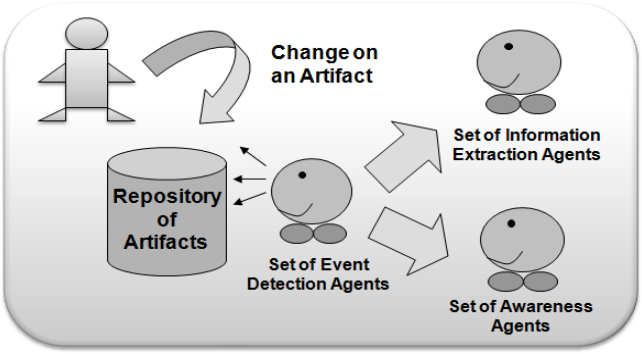

Figure 4. Identifying Changes on Artifacts

All artifacts are stored on the **Repository of Artifacts** (RA), in our case, a version control tool repository. As the Repository of Artifacts is populated by different types of files, e.g. word files (.doc), model files (.emx), and Java classes, for each type we define an **Event-Detection Agent** (EDA).

As shown in Figure 4, a set of EDA's is responsible to detect every change on repository's artifacts, whether it is the inclusion of new artifacts or modification to existing ones. EDA's are capable to detect when an artifact has changed and what kind of change it was: inclusion, update, deletion of large granularity artifacts, etc. The use of a version control system can ease the task of the EDA's.

These agents then communicate the event to a set of agents called **Information Extraction Agents** (IEA's) that, after doing their job, dispatch a set of agents called **Awareness Agents** (AA's).

Change Agents, **Tag Agents** and **Responsibility Agents** compose the set of IEA's. These three types of agents act on the RA and on **Project Knowledge Base** (PKB) to know the exact semantic of change, the set of artifacts that can be affected and to whom deliver the notice. Figure 5 illustrates this second task of ARARA.

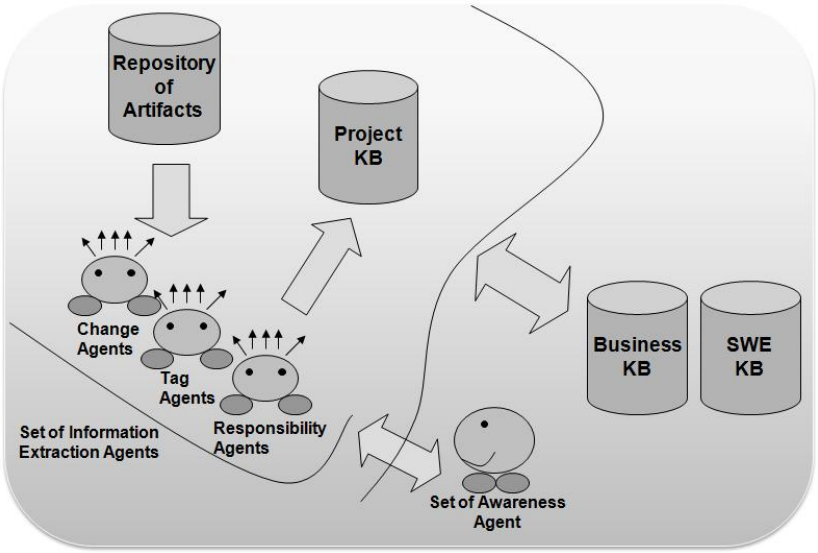

Figure 5. Describing the Semantic of Changes

The PKB is composed by references to all the artifacts of the project residing on the RA, together with tags regarding their relation to the concepts defined on the Software Engineering KB (SWEKB) and the Business KB (BKB). When a Version Management system is not available, it may be necessary to store the last version of the artifact and owner information on the PKB.

On the SWEKB, the SDP ontology is defined as the artifact types are related to each other according to their role in the SDP. This records the characteristics of a SDP, independent of business aspects, e.g., a *sequence diagram* implements a *use case scenario* using *objects* defined in a *class diagram/model*. In addition, the characteristics that allow for the identification of a file or its contents as a specific artifact are stored there. As an example, a file with a .ucs extension is there identified as a Clear Case file, that has information about use cases and requirements.

The BKB holds the Business Ontology. The artifacts receive tags that relate them to this ontology based on the information contained in this KB.

Whenever a change is detected by an EDA, it notifies IEA's (Change Agents, Tag Agents and Responsibility Agents). **Change Agents** are responsible to analyze changes to artifacts. They match the tags of available ontologies identifying the semantics of the changes. **Tag Agents** scan the artifact that was changed looking for modification on its set of tags, using the SWEKB and BKB. This check aims at detecting new tags that shall be applied to the artifact's existing set of tags, or the removal of tags not related to the artifact anymore. **Responsibility Agents** relate the set of artifacts that have to be verified due to the detected change, to interested or responsible project members. This is done by discovering the owner of the artifact and other peers that have worked on it.

Once the set of artifacts to be changed, and related project members, are discovered, the IEA's report the AA's, which start their work.

The set of AA's is dedicated to search the workspace for team members responsible for artifacts, to determine the way to communicate with them, and to do the communication effectively. AA's are usually specialized to the various kinds of online communication, e.g., e-mail or instant-messaging.

When the EDA's detect a change on an artifact they report the change to the AA's that shall in turn notify team members responsible for the artifact that has been modified after search for them in the PKB (See Figure 4). In another moment, AA's are notified by Responsibility Agents about the set of artifacts that may have been affected by the changes made (See Figure 5).

Finally, the **Knowledge Evolution Agents** (KEA's) act on the PKB, since new terms found can be new concepts that are not described on the ontology.

They are responsible to detect new terms and their relevance. KEA's also work on discovering other potentially affected artifacts, navigating the available ontologies. This is done by relating the concepts changed with others using the Business Ontology, on the BKB, and by relating the artifact with others using the Software Engineering Ontology, on the SWEKB.

4. CONCLUSIONS AND FUTURE WORK

ARARA is an agent-based system that facilitates software-development process. It is a system to be used in software development projects, where requirements constantly change, the development team is all dispersed geographically and awareness improvement is desired.

It provides artifacts and requirements changes awareness, based on knowledge bases, some of them constructed as ontologies. With workspace awareness, where all team members are notified of artifacts changes, it is guaranteed that requirements and artifacts are always consistent, avoiding rework.

Best practices in software-development process can improve the results, e.g., prepare the domain ontology before any other artifact is constructed (or, at least, during construction), use a conceptual object-oriented architecture similar to the business ontology, and use ontology terms on the artifacts to link them to each other and ease the traceability process.

We choose the COPPEER framework because it has been designed to provide an environment for developing P2P applications. FoxPeer is an example of a successfully collaborative tool developed using COPPEER framework (Vivacqua, A. et al, 2007). Although COPPEER technology supports any type of strategy for implementing agents, our approach is to implement single function and specialized agents that cooperate through entry exchange, which should be uncoupled.

A first prototype has already been implemented showing the adequacy of the ARARA setting. A new version is now under development and will be deployed on a large development project, where actually the need for such a tool was detected. This project will allow us to evaluate the performance of the system since the members of the development team are all dispersed geographically within the company.

The first phase of this project is being developed without the support of ARARA and it's a fact that artifact's changes propagation is delayed more than the expected, time is lost to gather information and, despite the team efforts, it is usual to find inconsistent artifacts, which demands rework by team members. Delays on the schedule are also common too. The second phase of this project will be supported by ARARA and a comparison in response time between these two phases will measure the improvement on team's work.

To measure the effectiveness of awareness the results will be evaluated using two metrics: the recall and the precision metric. This will be done with the support of a manual traceability matrix where we know the link between artifacts and so the correct artifacts that should be affected by a change in a specific one and by monitoring the agents' reports to team members that will provide the agents' retrieved artifacts.

$$\mathrm{Re}call = \frac{(CorrectArtifacts \cap \mathrm{Re}trievedArtifacts)}{CorrectArtifacts} \qquad \mathrm{Pr}ecision = \frac{(CorrectArtifacts \cap \mathrm{Re}trievedArtifacts)}{\mathrm{Re}trievedArtifacts}$$

As future work, besides finishing its complete implementation, we intend to improve KEA's functioning, since now it is based on very simple heuristics, and to provide mechanisms that allow the tuning of sensibility, i.e., how far from the original affected concept artifacts shall be considered for inspection.

REFERENCES

Cleland-Huang, J. et al, 2007. Best Practices for Automated Traceability. *In IEEE Computer*, Vol. 40, Issue 6, pp. 27-35.

Chandrasekaran, B. et al, 1999. What Are Ontologies, and Why Do We Need Them? *In IEEE Intelligent Systems and Their Applications*, Vol. 14, Issue1, pp. 20-26.

David, J. et al, 2001. Improving the Selectivity of Awareness Information in Groupware Applications. *In Computer Supported Cooperative Work in Design, the Sixth International Conference on 2001*. London, Canada, pp. 41-46.

Dix, A. et al, 1998. *Human Computer Interaction.* Prentice Hall.

Dourish, P. and Bellotti, V., 1992. Awareness and Coordination in Shared Workspace. *In Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work*. Toronto, Canada, pp. 107-114.

Endsley M., 1995. Toward a Theory of Situation Awareness in Dynamic Systems. *In Human Factors: The Journal of the Human Factors and Ergonomics Society*, Vol. 37, No. 1, pp. 32-64.

Gruber, T.R., 1993. Toward principles for the design of ontologies used for knowledge sharing. *In International Journal of Human-Computer Studies*, Vol. 43, Issue 5-6, pp. 907-928.

Gutwin, C. et al, 1996. Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation. *In Proceedings of HCI on People and Computer XI*, pp. 281-298.

Gutwin, C. et al, 2004. Group Awareness in Distributed Software Development. *In Proceedings of* 2004 ACM Conference on *Computer Supported Cooperative Work*. Chicago, USA, pp. 72-81.

Gutwin, C. and Greenberg, S., 2004. The Importance of Awareness for Team Cognition in Distributed Collaboration. *In E.Salas and S. M. Fiore (Editors) Team Cognition: Understanding the Factors that Drive Process and Performance*, pp. 177-201.

Institute of Electrical and Electronic Engineers, 1998. IEEE std. 830-1998 *IEEE recommended practice for software requirements specification*, IEEE-SA.

Jennings, N. A., 2001. An Agent-Based Approach for Building Complex Software Systems. *Communications of the ACM*, Vol. 44, No. 4

Miranda, M. et al, 2006. Building Tools for Emergent Design with COPPEER. *In Proceedings of the 10th International Conference on CSCW in Design*, Vol.1, pp. 550-555.

Pinheiro, M. et al, 2003. A Framework for Awareness Support in Groupware Systems. *In Computers in Industry*. Vol. 52, No. 1, pp. 47-57.

Rodrigues, N, et al, 2006. *A P2P Approach for Business Process Modelling and Reuse.* In Business Process Management Workshops. Springer Publisher, Berlin, Heidelberg.

Roussopoulos, M. et al, 2005. *2 P2P or Not 2 P2P?* In Proceedings of the 3rd International Workshop on Peer-to-Peer Systems. Springer Publishers, Berlin, Heidelberg.

Software Engineering Institute, 2006. CMU-SEI-2006-TR-008 *CMMI® for Development*, Version 1.2. Pittsburg, PA: Software Engineering Institute.

Vivacqua, A. et al, 2007. FoxPeer: Navigating the Web with Community Recommendations. *In Web Based Communitie, Salamanca. In Proceedings of the IADIS International Conference Web Based Communities*. Lisboa, Portugal, pp.118-125.

Wang, A. et al, 1999. A Multi-Agent Architecture for Cooperative Software Engineering. *In Proceedings of Eleventh International Conference on Software Engineering and Knowledge Engineering. Kaiserslautern, Germany.*

Wooldridge, M., 1997. Agent-Based Software Engineering. *In IEEE Proceedings of Software Engineering*, Vol. 144, No.1, pp.26-37

ARARA – A Collaborative Tool to Requirement Change Awareness

Ester J. C. de Lima[1], José A. Rodrigues Nt.[1], Geraldo B. Xexéo[1,2], Jano M. de Souza[1,2]
[1] *COPPE/UFRJ, Graduate School of Engineering*
[2] *DCC-IM Computer Science Department - Institute of Mathematics*
Federal University of Rio de Janeiro, Brazil
{esterlima, rneto, xexeo, jano}@cos.ufrj.br

Abstract— **This paper describes a collaborative tool, ARARA (Artifacts and Requirements Awareness Reinforcement Agents), that provides awareness in Software Development Process (SDP). Ontologies support this application for they are used by project team members when preparing the artifacts, propagating the concepts through them, and used by agents as knowledge base to retrieve information, tag the artifacts, detect changes, and search for linked artifacts that may be affected by a change on a previous artifact. This collaborative tool can improve team members awareness automatically no matter where they are geographically working, once agents are responsible for tracking and notifying requirements, or any other changes, to team members, including consequent candidates for modifications.**

Keywords-tools for CSCW; awareness; agents; ontology; traceability

Introduction

The lack of traceability and communication demands that project members have full knowledge about artifact relations (be them explicit or implicit) and knowledge about other members' work, i.e., what they are doing and when. Both types of knowledge are necessary for correctly introducing changes to the project, without compromising its integrity. If members are not aware of what others are doing, it is difficult to detect changes and their resulting implications. In nowadays, since project team members can be geographically distributed over a large area, this problem is even more relevant.

We have been working on a three-year project for an Oil Company, modeling and developing a system that assesses and reports the company reserves to the company board, regulatory agencies and stock markets. Since the beginning of the project, we knew that team members would change frequently and would be geographically distributed over the company, and team size would increase throughout the development. These facts could lead to some weakness on member interaction, compromising the detection of changes and their resulting implications, e.g., the propagation of changes among distinct artifacts. In addition, since SDP is a cooperative work, communication is recognized as a critical task [5]. However, although SDP artifacts referred by the documents spawned by the Software Requirements Specification are all related, forward traceability is still a problem, as the relation among them are not always explicit and proper communication to team members is not always effective.

Based on these facts, we realized the need for a tool that could improve members awareness, reducing the impact of project artifacts changes. The size and complexity of the application, in terms of business coverage and capillarity, led us to design a collaborative tool to automate traceability in software engineering projects.

When the project started, we created a Business Ontology in parallel with the Business Process Model, to support and facilitate the development of the latter, and the System Design Model. Some of the concepts where new to a great part of the development team, and some terms had different meaning for different stakeholders, which justified the creation of an appropriate ontology. Cappelli [7] proposes ontology as a complement of business-process models, for domain understanding, to explicit and better understand the business, with the purpose of identifying software requirements through business requirements. Taking this into consideration, we extended the use of the ontology to all artifacts of the SDP. This way, all software artifacts were related to the proper business ontology concepts, to facilitate understanding, to ease communication among team members and stakeholders, and to promote requirements traceability.

Additionally, most SDP artifacts can be related using an adequate software engineering ontology or metamodel.

Considering all of the above, we decided to develop a SDP support tool, using ontologies, to promote the discovery of software artifacts relationships, mixing the knowledge offered by the business ontology and the software engineering ontology. Therefore, if an artifact is modified, the tool can improve awareness, providing project team members proper information about the set of artifacts that may have been affected by another artifact modification.

ARARA is based on agents that can track changes in artifacts and report to team members those changes and their effects on other artifacts. To allow for traceability information recovery, we use the available ontologies to prepare the artifacts [6]. We believe this is an effective way to link all the artifacts and to promote traceability among them.

The rationale is to prepare the system model using the associated business concepts extracted from the business ontology, tagging all UML artifacts produced (activity diagrams, use-case diagrams, class diagrams, etc). Later on, the tool can benefit from the business ontology, to retrieve the artifacts that are connected to the one that was changed,

navigating it and finding related business concepts, as well as using the software engineering ontology to find related artifacts

The paper is organized as follows. In section II we introduce traceability in SDP. In section III we provide a basic explanation of ARARA. In section IV we discuss the principles and mechanisms of ARARA and in Section V we present the results we achieved. Finally, section VI presents the state of our work, next steps and offers some conclusions.

Traceability in Software Development Process

Traceability is one of the IEEE Recommended Practice for Software Requirements Specifications (SRS) [8]. Gotel [9] defines requirement traceability as the ability to describe and follow the life of a requirement, in both forward and backward directions, i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases.

Edwards [10] defines traceability as a technique used to provide a relationship between the requirements, the design, and the final implementation of a system. During *design*, traceability allows designers and maintainers to keep track of what happens when a change request is implemented, throwing light on what shall be redesigned.

Palmer [11] states that traceability gives essential assistance in understanding the relationships that exist within and across software requirements, design, and implementation. Watkins [12] found that traceability aids project managers in verification, cost reduction, accountability, and change management. For each change, it is easy to determine which related elements of the design are affected. This helps to keep documentation up to date as the implementation progresses.

Several issues make artifacts' links recovery difficult. The connection between artifacts is rarely represented explicitly and the artifacts themselves are represented at different abstraction levels and using diverse formalisms. Once SDP produces a large set of artifacts, which is composed by different types of documents, and as one artifact is derived from others, it is important to link them to ease the traceability process, and it is, at least, desirable to adopt automatic techniques to recover those links. Manual recovery can be prohibitive for a large set of artifacts.

There are other works focusing on the recovery of traceability links between software documentation and source code. In [13], "the methodology is based on the extraction, analysis, and mathematical representation of the comments and identifiers from the source code". In [14], "the method uses the identifiers extracted from source code component as a query to retrieve the documents relevant to the component, assuming that programmers use meaningful names for their identifiers and consequently "words" are chosen as indexing feature.

ARARA uses ontology to provide a vocabulary to represent a Business domain and to serve as identifiers to be used in all artifacts. Object-Oriented design of software systems is somewhat similar to domain ontology, due to existence of objects, their attributes, and relations among them. It mirrors aspects of the domain that are relevant to the application [2]. In this case, using the ontology (vocabulary, concepts, relations, etc) it is possible to link artifacts of an SDP and better recover traceability between them.

ARARA Conception

We implemented ARARA to be deployed on a project we have been working, where requirements constantly change and lack of awareness is a fact, as members of the development team are all dispersed geographically within the company and team changes in size and constituents during the project, with professionals coming and going.

As artifacts are always in constant change in SDP, team members must be aware of these changes in order to complete their tasks and keep project artifacts complete and consistent.

ARARA communicates to team members the SDP artifacts that may be affected by the change done by someone in another specific artifact. As team members are notified by ARARA about artifacts that shall be verified, we reduce the chance of inconsistencies.

Figure 1 illustrates the behavior of ARARA's agents. As a collaborative tool, ARARA's basic tasks are:

- Detect changes on artifacts;
- Identify other artifacts that shall be changed;
- Identify to whom a change should be reported to; and
- Notify proper project team members.

In our setting, all artifacts of the project are stored on a version control repository. Therefore, whenever an artifact is written on the repository a set of **Event-Detection Agents** are responsible to detect this event and identify the change. These agents then communicate with another set of agents, called **Information Extraction Agents,** which are composed by three subsets of agents: **Tag Agents**, **Change Agents** and **Responsibility Agents**. They are responsible to analyze the changes, track the artifacts that may be affected due to the change initially detected and find whom to notify.

Project Knowledge Base is composed by references to all the artifacts of the project together with tags regarding their relation to the concepts defined on the **Business Knowledge Base**.

After the **Information Extraction Agents** have done their job, they communicate with the **Awareness Agents** that shall report team members about the detected change and the artifacts that may be also affected.

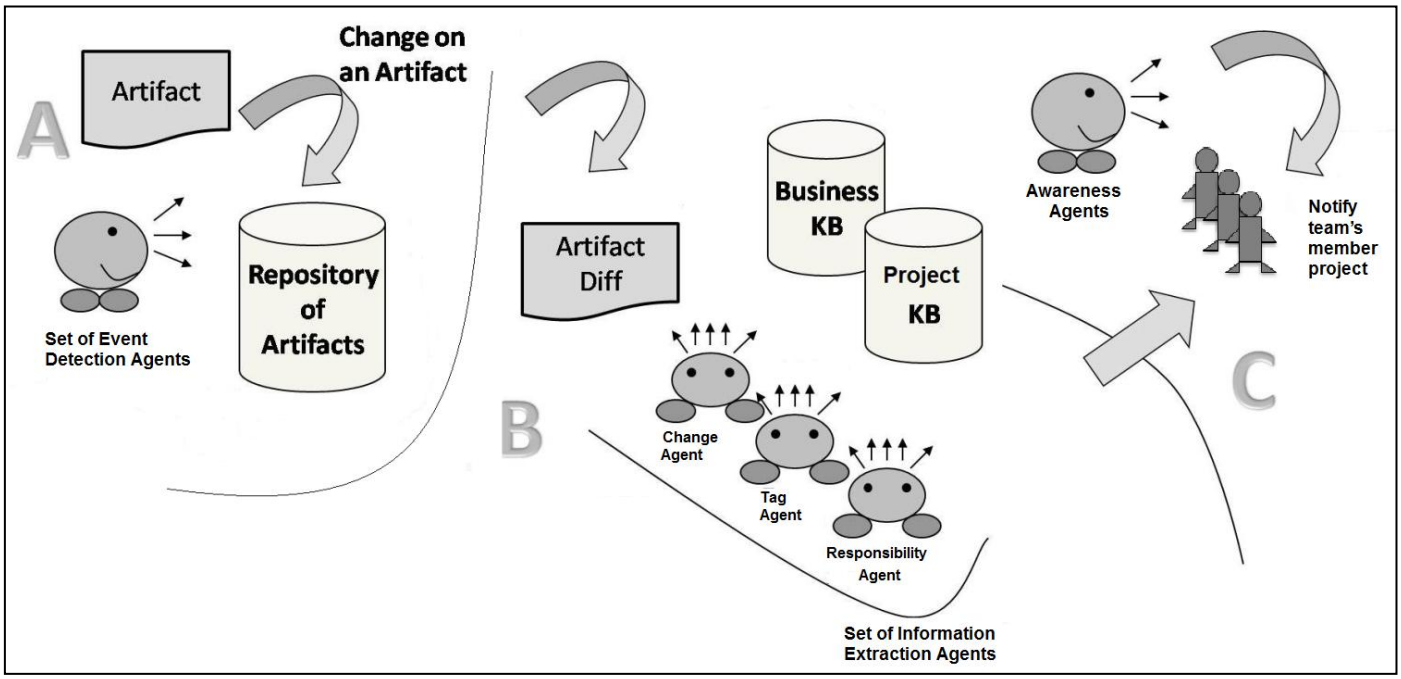

Figure 1. Illustration of the behavior of ARARA's agents

Principles and Mechanisms

Ontology Supporting Traceability

In this work we propose to apply the ontology, created during the business modeling process, to the artifacts produced in the system development. The idea is to link the artifacts by using ontology and to use ontology as a knowledge base to provide ARARA with automatic tracing capabilities.

One of the best practices for traceability recommended by [1] is to create traceable artifacts using a well defined project glossary to construct a meaningful hierarchy of information. All these can be achieved using ontology. Oliveira [3] uses ontology to promote software-tool integration. Whenever we have an ontology that is derived from a business process model, the concepts mapped onto the ontology can be used in all the artifacts.

Edgington [4] points out that one goal of ontology is to provide a faithful specification of a knowledge unit, i.e., that the resulting conceptual model adequately represents the domain or contextual knowledge of interest. Thereby, ontology is one way to clarify knowledge structure and to formalize it, to represent adequately the domain, and to enable better communication between humans and machines.

Contextualizing, as we orient system development to business process models, all the concepts mapped onto the business ontology were used on the artifacts. We have tried not to abbreviate the ontology terms and not to suppress the use of them. In model building, we follow standard conventions of uppercase and lowercase usage to facilitate the algorithm that identifies tokens and compare them with ontology terms, e.g., an interface of a component of the system architecture, "ICreateDissociatedField" that allows for the identification of the ontology Class "Field" and the Data Type Property "Dissociated".

With all the artifacts prepared using ontology terms and based on those concepts, we support traceability through tagging the artifacts and searching for artifacts that are related, by having some similar tags and tags that are linked by ontology, and consequently may have been affected.

Tagging the Artifact – the Project Knowledge Base

Each artifact, after it has been written on the repository, is retrieved by a **Tag Agent** to create a representation of the artifact on the **Project Knowledge Base**. The process consists of:

- Searching ontology terms on it, creating Business Tags set that best represents this artifact;
- Defining which type of UML diagram or element the artifact represents; and
- Identifying the team member that committed the artifact's change to the repository.

Such process uses the concepts mapped on the Business Ontology to create the Business Tags:

- Ontology Classes – used to match the artifact and form the conjunction of Business Tags that represent it (*example of* Ontology Classes: ***Field***, ***Project***).
- Object Properties – do not appear explicitly in the artifacts. Represent the association between ontology's

classes, linking an individual to another individual.[11] Object Properties are used by ARARA to relate concepts in artifacts that are not explicitly linked but have an implicit link provided by ontology (*e.g.*, *Field* ***has Projects*** or *Project* ***has EconomicValues***). So, artifacts that deal with ***Field*** are in some way related to the ones that deal with ***Project***.

- Datatype Properties – are the attributes of Ontology's Classes. As Datatype Properties link an individual to an XML Schema Datatype value or an RDF literal [15], it contributes to find the Classes that can represent that artifact (e.g., ***Field*** *has exactly* ***1 partner*** *and can be* ***dissociated*** *or* ***Project*** *has exactly* ***1 description*** *and can be* ***dissociated***).

During the retrieving process, the **Tag Agents** remove stop words on the artifacts and stem the rest of the words to their root form to get the ontology term.

As we are using a CASE tool – RSA (Rational Software Architect) [16] – to model the system, ARARA has to parse its XMI files (XML Model Interchange) [17] to get elements that compose the artifacts and to search them for Business Tags.

The XMI files themselves provide us the type of UML Diagram being described. We limited our scope to Activity diagrams, Use-case diagrams, Class diagrams, and Component diagrams. They also provide a mechanism for specifying references within and across documents, but we did not use it in our setting. This mechanism uses the XMI *href* attribute to locate XML elements in another XML document by its XMI id. The value of *href* is a URI reference in the form *URI#id_value*, where *URI* locates the XML file containing the XML element to link to, and *id_value* is the value of the XML element's XMI id [17].

This mechanism helps link artifacts, but it does not link artifacts that are in the same context and do not have a reference within and across documents. On the other hand, with ontology, we can retrieve all artifacts that represent the same concepts or are strongly related to it.

In the Version Control Tool Repository that we use in the project – *SubVersion* [18] –, the *svn log* command provides records of who made changes to a file or directory, in which revision it was changed, the time and date of the revision, and – if it was provided – the log message that accompanied the version commit. We infer the responsible for an artifact as the one that made the last change on the artifact.

Before the **Tag Agent** substitutes the new representation created for an artifact changed on the **Project Knowledge Base,** the agent communicates with the **Change Agents**.

Searching for Artifacts that may be Affected

Change Agents compare two representations of an artifact to find the differences between them. It also uses the information that *SubVersion* makes available to discover if the artifact committed on the repository is an inclusion, a deletion, or a modification. **Change Agents** treat each action in a different way.

- Inclusion – as the artifact is new it does not have a previous representation of it on the Project Knowledge Base. Therefore, the agents need only to find the group of artifacts that are linked to it by the same business context (artifacts that represent the same concepts and are strongly related to it).
- Deletion – as the artifact is not anymore part of the project repository, the agents use the last representation of this artifact to find the group of artifacts that was linked to it by the same business context.
- Modification – in this case, the agents need to compare the two representations of the artifact on the Project Knowledge Base (the previous and the new) to check: 1) if the set of Business Tags continues the same; or 2) if there are new Business Tags or if some are not anymore Business Tags that represent the artifact. In the first case, the agents need to find the group of other artifacts that are linked by the same business context to the changed artifact. In the second case, the agents need to find the new group of artifacts that are linked to it and find the other group that was linked to it before.

To find the artifacts that are linked by the same business context to a changed one is to search for artifacts that may be affected by the change. To get these artifacts we use the Jaccard Similarity Coefficient [19] to measure similarity between the artifact that has been changed and all others that compose the Project Knowledge Base. Jaccard Similarity Coefficient definition is:

$$JS_{(ArtfTag_1, ArtfTag_2)} = \left[\frac{ArtfTag_1 \cap ArtfTag_2}{Artf_{Tag1} \cup Artf_{Tag2}}\right], \text{ where:}$$

JS represents the Jaccard's similarity between artifacts;

$ArtfTag_T$ – stands for the set of business tags of an artifact *T* ($Artf_T$).

$$ArtfTag_T = \{BT_1, BT_2, \ldots, BT_n\}; \text{ and}$$

BT_i – stands for the *ith* business tag that represents an artifact on the Project Knowledge Base.

We rank the results, based on artifacts' similarity, starting with the artifacts that are more similar to the changed one.

Agents Supporting ARARA

ARARA is a P2P application that is being implemented on the COPPEER framework, which is a multi-Agent System (MAS) framework, i.e., an environment for developing and running agent-based collaborative P2P applications [20], and that has successfully been used on the implementation of other collaborative tools like FoxPeer [21] and BPCE [22].

[11] Individuals can be referred to as being instances of classes.

Although COPPEER technology supports any type of strategy for implementing agents, our approach is to implement single function and specialized agents that cooperate through entry exchange, which should be uncoupled.

ARARA agents can directly communicate with the artifacts' repository through the *SubVersion API*, after being started by a repository hook. A repository hook is a program triggered by the occurrence of an appropriate repository event, e.g. a commit event.

The hook used to start ARARA process is called *post-commit* hook, because it runs after the completion of a *commit* event and it hands enough information about the repository changes [17].

ARARA Results

We have been also experimenting the best way to list the artifacts to notify team members. Table 1 presents the ranking of the artifacts that may be affected by changes on the artifacts $Artf_1$ and $Artf_2$.

The column named **JS(ArtfTag$_1$, ArtfTag$_x$)** shows the Jaccard's Similarity Coefficient between $Artf_1$ and the artifacts listed on column named **RelatedArtifacts,** which shows the artifacts related to the one changed. The next two columns present the same, but for $Artf_2$.

Table 1. Ranking of artifacts that may be affected by a change

JS(ArtfTag$_1$, ArtfTag$_x$)	Related Artifacts	JS(Artftag$_2$, ArtfTag$_x$)	Related Artifacts
0,75	$Artf_2$	0,75	$Artf_1$
0,66	$Artf_3$	0,66	$Artf_5$
0,6	$Artf_4$	0,66	$Artf_6$
0,5	$Artf_5$	0,66	$Artf_7$
0,5	$Artf_6$	0,66	$Artf_8$
0,5	$Artf_7$	0,5	$Artf_{11}$
0,5	$Artf_8$	0,5	$Artf_{12}$
0,5	$Artf_9$	0,5	$Artf_{13}$
0,5	$Artf_{10}$	0,5	$Artf_{14}$

We can see in the results presented above that a propagation of the change on $Artf_1$ to $Artf_2$ leads us to rank another set of artifacts candidate to be modified. Some of the artifacts listed on the second rank are also listed in the first rank, e.g., the set of artifacts ($Artf_5$, $Artf_6$, $Artf_7$ and $Artf_8$).

Comparing with the first rank these artifacts are less similar to $Artf_1$ then to $Artf_2$. In this particular case, maybe it's not necessary to propagate changes on $Artf_1$ to $Artf_5$, $Artf_6$, $Artf_7$ and $Artf_8$. It's possible to wait till propagation changes on $Artf_1$ occur on $Artf_2$. Since $Artf_2$ are consistent after the modification reflecting changes on $Artf_1$, changes on $Artf_5$, $Artf_6$, $Artf_7$ and $Artf_8$ will reflect changes on $Artf_1$ and $Artf_2$.

Considering this specific case, it is not necessary to rank all artifacts related to the one that has been changed. The process of propagating modifications covers all the artifacts and we avoid information overload. Suppressing from the first ranking the artifacts $Artf_5$ to $Artf_{10}$ would not impact on the process of propagating changes.

In this work, the feedback of team members is necessary to evaluate the implemented algorithm. We are now conducting experiments with team members to get their view of:

- The artifacts they consider shall be prioritized when propagating change;
- The ones that they consider necessary to propagate change, but in a second moment (after changes are made to the ones initially prioritized);
- And the ones that they consider not necessary to propagate change;

Conclusion

ARARA is a system to be used in software development projects where requirements constantly change; the development team is dispersed geographically, new team members arrive during the project; and awareness improvement is desired.

It provides awareness of artifact and requirement changes, based on knowledge bases, constructed as ontologies. A Business ontology is useful in this context, as it provides a business vocabulary and the relations among the represented concepts. This business vocabulary can be used by all artifacts produced in a software development process, linking them to each other and supporting traceability.

We intend to deploy ARARA in this large development project, where actually the need for a collaborative tool to notify requirement changes exists. This project will allow us to evaluate the effectiveness of ARARA.

The first phase of this project was conducted without the support of ARARA, and it is a fact that propagation of an artifact change was delayed more than expected or even suppressed, time is lost gathering information, and despite the team efforts, it is usual to find inconsistent artifacts, which demands rework. Delays on the schedule are also common.

The second phase of this project will be supported by ARARA and a comparison in response time between these two phases will measure improvement in work productivity, since we expect not have artifacts inconsistent though the constantly changes during SDP. This second phase will also provide us feedback about information overload. Although ARARA provides awareness on requirements/design changes, it is about the team members the responsibility to analyze the

artifacts suggested to be changed and to do the necessary changes on the artifacts that need to be changed, ensuring the consistence of the requirements. This feedback will allow for the improvement of ARARA.

Although ARARA is a collaborative tool, we focused on the implementation of the algorithm to rank the artifacts that are more susceptible to have been impacted by the detected change. We are presently implementing the agents that automate the process.

The first prototype that was implemented was to evaluate the use of ontology in support of information retrieval techniques. ARARA development efforts are currently focused on ranking the artifacts that may be affected. We still need more time to refine the mechanism and define the actual algorithm that is to be used.

REFERENCES

[1] Cleland-Huang, J., Settimi, R. and Romanova, E. (2007) "Best practices for automated traceability", *IEEE Computer*, pp. 27-35.

[2] Chandrasekaran, B., Josephson, J. R. and Benjamins, R., 1999. What are ontologies, and why do we need them? *In IEEE Intelligent Systems and Their Applications*, Vol. 14, Issue1, pp. 20-26.

[3] Oliveira, F., Antunes J., Guizzardi, R. (2007) "Towards a collaboration ontology", *II Workshop on Ontologies and Metamodeling in Software and Data Engineering*, pp. 97-108.

[4] Edgington, T, raghu, T. S. and Vinze, A. (2005) "Knowledge ontology: a method for empirical identification of 'As-Is' Contextual Knowledge". In: *Proceedings of the 38th Hawaii Internantiona Conference on System Sciences.*

[5] Pressman, R. (2006). Engenharia de Software, McGrawHill.

[6] Lima, E. J. C, Rodrigues Nt., Xexéo, G., and Souza, J. M., (2008) "ARARA – Artifacts and Requirements Awareness Reiforcement Agents". In: *Proceedings of the IADIS International Conference ISA (part of MCCSIS 2008)*, p.p. 92-99.

[7] Cappelli, C., Baião, F., Santoro, F., Iendrike, H., Lopes, M. and Nunes, V. (2007) "Uma abordagem de construção de ontologia de domínio a partir do modelo de processo de Negócio", *II Workshop on Ontologies and Metamodeling in Software and Data Engineering*, pp. 85-96.

[8] IEEE (1998) "IEEE std. 830-1998 IEEE recommended practice for software requirements specification", *IEEE-SA.*

[9] Gotel, O. and Finkelstein, A. (1994) "An analysis of the requirements traceability problem". In: *1st IEEE International Conference on Requirements Engineering*, pp. 94-101.

[10] Edwards, M. and Howell, S. (1991) "A methodology for systems requirements specification and traceability for large real time complex systems", *Technical report, Naval Surface Warfare Center*.

[11] Palmer, J. (1997). "Traceability", In: Software Requirements Engineering, Edited by R.H. Thayer and M. Dorfman, IEEE Computer Society Press, pp. 364-374.

[12] Watkins, R. and Nea, M. (1994) "Why and how of requirements tracing", *IEEE Software*, pp. 104-106.

[13] Marcus, A., Maletic, J. and Sergeyev, A. (2005) "Recovery of traceability links between software documentation and source code". In: *International Journal of Software Engineering and Knowledge Engineering*, pp. 811-836.

[14] Antoniol, G., Canfora, G., Casazza, G., Lucia, A. D. and Merlo, E. (2002) "Recovering traceability links between code and documentation", *IEEE Transactions on Software Engineering*, pp. 970-983.

[15] Horridge, M., KnuBlauch, H., Rector, A., Stevens, R. and Wroe, C. (2004). A practical guide to building OWL ontologies using the protégé-OWL plugin and CO-ODE tools, The University of Manchester, Ed. 1.0.

[16] Rational Software Architect (RSA) [Online]. Available: http://www-01.ibm.com/software/awdtools/swarchitect/websphere/

[17] XMI Mapping (2007) "MOF 2.0/XMI Mapping, Version 2.1.1", OMG.

[18] Collins-Sussman, B., Fitzpatrick, B. W. and Pilato C. M. (2008) "Version Control with SubVersion", 2nd Edition, O'Reilly Media.

[19] Jain, A. K. and Dubes, R. C. (1998). Algorithms for clustering data, Prentice Hall Inc, Upper Saddle, NJ, USA.

[20] Miranda, M. Xexéo, G. B. and Souza, J. M. (2006) "Building tools for emergent design with COPPEER". In: *Proceedings of the 10th International Conference on Computer Supported Cooperative Work in Design*, pp. 550-555.

[21] Vivacqua, A., Rodrigues Nt., J. A., Machado, M., Padula, R., Paes, M., Barros, P., Xexéo, G., Souza, J. and Miranda, M. (2007) "FoxPeer: navigating the web with community recommendations". In: *Web Based Communitie, Salamanca. In Proceedings of the IADIS International Conference Web Based Communities*, pp. 118-125.

[22] Rodrigues Nt., J. A., Souza, J. M., Zimbrão, G., Xexéo, G. and Miranda, M. (2008) "Business process reuse and standardization with P2P technologies, in *Handbook of Research on Virtual Workplaces and the New Nature of Business Practices.*.