UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
DEPARTAMENTO DE INFORMÁTICA
MESTRADO EM INFORMÁTICA

VANESSA BATTESTIN NUNES

INTEGRANDO GERÊNCIA DE CONFIGURAÇÃO DE SOFTWARE, DOCUMENTAÇÃO E GERÊNCIA DE CONHECIMENTO EM UM AMBIENTE DE DESENVOLVIMENTO DE SOFTWARE

VITÓRIA
2005

VANESSA BATTESTIN NUNES

INTEGRANDO GERÊNCIA DE CONFIGURAÇÃO DE SOFTWARE, DOCUMENTAÇÃO E GERÊNCIA DE CONHECIMENTO EM UM AMBIENTE DE DESENVOLVIMENTO DE SOFTWARE

Dissertação apresentada ao Mestrado em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Orientador: Prof. Dr. Ricardo de Almeida Falbo.

VITÓRIA
2005

VANESSA BATTESTIN NUNES

INTEGRANDO GERÊNCIA DE CONFIGURAÇÃO DE SOFTWARE, DOCUMENTAÇÃO E GERÊNCIA DE CONHECIMENTO EM UM AMBIENTE DE DESENVOLVIMENTO DE SOFTWARE

COMISSÃO EXAMINADORA

_____

Prof. Ricardo de Almeida Falbo, D. Sc.
Orientador

_____

Prof. Davidson Cury, D.Sc.
UFES

_____

Prof. Guilherme Horta Travassos, D.Sc.
COPPE/UFRJ

Vitória, 27 de Abril de 2005.

“Tentar e falhar é, pelo menos, aprender. Não chegar a tentar é sofrer a inestimável perda do que poderia ter sido.”

(Geraldo Eustáquio)

A minha filha Jasmine, luz da minha vida, que nasceu em meio a este trabalho enchendo de alegria e esperança os corações de todos que nos rodeiam.

A meu marido Rudnei, por todo amor, compreensão e companheirismo durante todos estes anos juntos. Por sempre ter me ouvido, entendido as minhas ausências e estado ao meu lado tanto nos momentos de alegria quanto nos momentos difíceis.

AGRADECIMENTOS

A Deus, que me deu a força necessária para que eu vencesse todos os obstáculos e persistisse nesta caminhada, não me deixando desistir mesmo nos momentos mais difíceis.

A Rudnei, meus pais, meu irmão, meus sogros, cunhados e cunhadas por todo apoio, incentivo e pelas tantas vezes que me ajudaram a cuidar da Jasmine para que eu pudesse dar continuidade ao mestrado.

A meu orientador e ídolo Ricardo, por seu profissionalismo, sua paciência e por todo aprendizado que me proporcionou. Por todas as vezes que me compreendeu e que esteve disponível para sanar as minhas dúvidas. Por sempre ter acreditado no meu trabalho e me feito ver o lado bom das coisas, principalmente quando eu não as via.

À Accenture, em especial ao meu atual coordenador Márcio Gonçalves e ao meu ex-coordenador André Marinho e a CVRD, principalmente a Antônio Vizeu e Christina Vello, por sempre terem compreendido minhas ausências referentes ao Mestrado e nunca terem feito disto empecilho a meu crescimento profissional.

Aos meus amigos Renzo, Tathyanna, Célia, André, Cláudio, Admila, Alessandra, Flávia, Luciana, Adriano e Ayrton e a todos outros que de alguma forma contribuíram para esta vitória, seja através de torcida, de apoio ou de compreensão.

Aos professores da UFES e aos colegas do Projeto ODE, principalmente ao Fabiano e ao Silvano, pela ajuda, discussões e trocas de idéias.

RESUMO

A importância de Ambientes de Desenvolvimento de Software (ADSs) tem sido, cada vez mais, destacada, por apoiarem o desenvolvimento durante todo o ciclo de vida do software, auxiliando a transferência de informação entre suas ferramentas. Porém, um dos principais problemas dos ADSs é a integração de ferramentas. Para tentar solucionar o problema de integração, uma potencial solução é estabelecer uma infra-estrutura em que a Gerência de Configuração de Software (GCS) esteja no núcleo do ADS. A GCS deve controlar os artefatos produzidos e compartilhados pelas diversas ferramentas, suportar controle de mudanças e controle de versão e prover relatos de estado do processo de alteração. Além disso, uma vez que os artefatos são os elementos básicos da documentação de software e importantes itens de conhecimento, a GCS em um ADS tem de estar integrada à documentação e à gerência de conhecimento do ambiente.

Assim, para uma integração efetiva e consistente, os artefatos de software devem possuir uma estrutura bem estabelecida, definida com base em um vocabulário comum e sem ambigüidades. Neste contexto, ontologias passam a ser consideradas por darem um entendimento compartilhado de um domínio de interesse, resolvendo problemas de comunicação entre as ferramentas e os recursos humanos envolvidos.

O uso de ontologias facilita a atuação da Gerência de Conhecimento que, integrada à GCS, permite um melhor tratamento dos artefatos de software como itens de conhecimento, beneficiando seus mecanismos de busca e disseminação de conhecimento.

Esta dissertação apresenta uma infra-estrutura de gerência de configuração que define uma ferramenta de GCS construída e integrada ao ambiente ODE (*Ontology-based software Development Environment*) e os procedimentos de integração da GCS às demais ferramentas de ODE, em especial à Gerência de Conhecimento e à documentação. Para apoiar essa integração, a infra-estrutura proposta foi construída tomando por base uma ontologia de artefatos de software desenvolvida também no contexto deste trabalho

Palavras-chave: Gerência de Configuração de Software, Gerência de Conhecimento, Ambientes de Desenvolvimento de Software, Ontologias, Documentação.

ABSTRACT

The importance of Software Development Environments (SDEs) has been more and more highlighted, because they support the software process during all software life cycle, assisting information interchange among theirs tools. However, one of the main problems in SDEs is regarding tool integration. To try to solve this problem, a potential solution is to establish an infrastructure where the Software Configuration Management (SCM) is the SDE's core. The SCM must control the artifacts produced and shared by the several tools, support change and version control, and provide status reports. Moreover, since artifacts are the basic elements of software documentation and important knowledge items to be shared, SCM in a SDE must be integrated to the environment's documentation facilities and knowledge management.

Thus, for an effective and consistent integration, the software artifacts must share a well-defined structure, based on a common vocabulary, without ambiguities. In this context, ontologies should be considered, because they establish a shared understanding about a domain of interest, solving communication problems between the tools and the human resources involved.

The use of ontologies is also essential for Knowledge Management that, integrated with SCM, allows better treatment of software artifacts as knowledge items, improving its search and dissemination mechanisms.

This work presents a configuration management infrastructure that defines a SCM tool, built and integrated to ODE (Ontology-based software Development Environment) and the procedures for integrating SCM to the others ODE's tools, in special to Knowledge Management and Documentation. To support this integration, the proposed infrastructure was built based on an ontology of software artifact that was also developed in the context of this work.

Keywords: Software Configuration Management, Knowledge Management, Software Development Environments, Ontologies, Documentation.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo 1

Introdução

Com a crescente demanda no setor de desenvolvimento de software, aumenta a necessidade de se dispor de ambientes automatizados que apóiem todo o processo de desenvolvimento. Essa necessidade se deve também à grande complexidade dos sistemas atuais e à expectativa de que eles tenham alta qualidade e que sejam desenvolvidos respeitando os prazos estabelecidos, sem a necessidade de alocação de mais recursos.

Ambientes de Desenvolvimento de Software (ADSs) surgem, então, com o intuito de integrar diferentes ferramentas, construídas para finalidades específicas, mas que operam juntas para apoiar a construção de um produto de software. Tal integração é fundamental para os ADSs e envolve diversas dimensões, tais como apresentação, dados, processo, controle e conhecimento (TRAVASSOS, 1994).

Dentre as dimensões de integração, a integração de conhecimento tem merecido destaque, pois possibilita às ferramentas um melhor entendimento da semântica das informações e dos métodos existentes para tratá-las, além de tornar disponíveis os serviços básicos de armazenamento, gerenciamento e utilização do conhecimento descrito e adquirido ao longo do processo de desenvolvimento (FALBO et al., 1995). Neste sentido, sistemas de gerência de conhecimento são bastante úteis, pois objetivam documentar o conhecimento envolvido no desenvolvimento de sistemas, ao invés de deixá-lo apenas na cabeça dos desenvolvedores (NATALI, 2003).

Porém, à medida que o conhecimento em um ADS cresce, a sua utilização pode se tornar uma tarefa árdua, muitas vezes inclusive, não sendo possível coletar em tempo hábil a informação necessária para apoiar a tomada de decisão. ADS Semânticos (FALBO et al., 2004a) surgem para tentar solucionar esse problema, incluindo semântica na busca por recursos realmente relevantes para o assunto em mãos.

Neste contexto, ontologias tornam-se essenciais, pois podem ser usadas como uma estrutura unificadora para dar semântica e uma representação comum à informação (FALBO et al., 2004a). Além disso, melhoram a comunicação entre desenvolvedores e ferramentas em um ADS, evitando problemas de interpretação e ajudando a minimizar um dos maiores problemas recorrentes em ADSs - a integração de ferramentas.

Segundo PRESSMAN (2001), uma boa alternativa para ajudar na solução do problema da integração é a utilização de uma estrutura em que o principal mecanismo de integração de ferramentas é a Gerência de Configuração de Software (GCS).

A GCS deve ficar no núcleo do ADS e prover mecanismos para, dentre outras coisas, identificar os artefatos a serem gerenciados, prover controle de versão e controle de mudanças, apoiar auditorias e fornecer relatos de configuração dos artefatos gerenciados. Deve estar integrada a todo o ambiente e acessível a todas as ferramentas. Sua finalidade é estabelecer e manter a integridade dos produtos de software ao longo de todo o ciclo de vida, objetivando a maximização da produtividade através da minimização de erros (FIORINI et al., 1998).

1.1 Objetivo do Trabalho

Como a integração de ferramentas ainda é um dos maiores desafios na construção de Ambientes de Desenvolvimento de Software, o uso de ontologias pode trazer muitos benefícios. No ADS em que este trabalho está inserido – ODE (*Ontology based software Development Environment*) (FALBO et al., 2003), as ferramentas internas são desenvolvidas baseadas na ontologia de processo de software definida em (FALBO, 1998) ou em ontologias integradas a ela. Essa base ontológica facilita a integração de ferramentas, uma vez que os conceitos da ontologia estão bem definidos e são compartilhados entre as ferramentas integradas ao ambiente (NATALI, 2003).

Porém, o problema da integração de ferramentas ainda não está totalmente solucionado em ODE. As ferramentas de ODE compartilham artefatos e a integração desses artefatos ainda não estava sendo feita a contento. Como artefatos são compartilhados entre as ferramentas, antes que essa integração ocorresse, era necessário definir um vocabulário comum, para facilitar a comunicação não apenas entre as ferramentas, mas também entre os desenvolvedores de ODE. Este trabalho teve, então, como um de seus objetivos, construir uma ontologia de artefato para atender a tal propósito.

No entanto, a construção de uma ontologia única para artefato descartaria características importantes de seus tipos (documentos, diagramas, artefatos de código, componentes de software e produtos de software). Para solucionar esse problema, a construção da ontologia de artefato se estendeu para a construção de sub-ontologias para tratar alguns tipos específicos: documento, diagrama e artefato de código. A partir de então,

estabeleceu-se uma estrutura comum à qual cada ferramenta de ODE tem de estar aderente para criar artefatos. A avaliação de tais ontologias ocorreu no decorrer deste e de outros trabalhos que necessitavam sua utilização, como é o caso da ferramenta de apoio à documentação de ODE, XMLDoc (SILVA, 2004).

No momento em que artefatos passaram a ser o foco em ODE, outras questões tiveram que ser tratadas, dentre elas: Como esses artefatos serão controlados? Como evitar que um desenvolvedor altere um artefato que estiver sendo utilizado por outro desenvolvedor? Como controlar as diversas versões de um artefato? A Gerência de Configuração de Software passou, então, a ser considerada como aspecto prioritário e a ontologia de artefato foi novamente estendida, porém agora para dar espaço para criação de uma sub-ontologia para GCS.

Ainda no intuito de solucionar o problema de integração e tendo em vista a importância assumida pela GCS, outro objetivo deste trabalho foi a definição de uma infra-estrutura para GCS. Feitos estudos levando em conta, principalmente, aspectos de integração, chegou-se a uma abordagem que considera uma única base de dados e arquivos XML. Nessa infra-estrutura, é criado um arquivo XML para cada variação (versão ou variante) de um artefato submetido à Gerência de Configuração. Ao se tentar abrir um artefato sob GCS, o arquivo XML referente à variação mais recente do artefato é exibido pela ferramenta XMLDoc. Isso só não ocorre nos casos em que o artefato estiver em alteração, sendo o desenvolvedor solicitante um dos responsáveis por ela, ou quando o artefato não estiver sob gerência de configuração. Nesses dois casos, é disponibilizado o artefato da base de dados e o desenvolvedor pode realizar alterações sobre ele. Sempre que uma nova versão ou variante do artefato for criada, um arquivo XML correspondente também é criado e passa a ser o mais atual.

Essa infra-estrutura facilita o acesso à estrutura interna dos artefatos, uma vez que os artefatos em alteração estão disponíveis em base de dados, facilita a integração com as demais ferramentas do ADS e, ao mesmo tempo, não prejudica a atuação da Gerência de Conhecimento de ODE, cuja infra-estrutura foi proposta em (NATALI, 2003). A única mudança na infra-estrutura da Gerência de Conhecimento é o fato da memória organizacional passar a ser constituída, para o caso de artefatos, pelos arquivos XML atuais e não mais pelos artefatos da base de dados de ODE. Assim, a Gerência de Conhecimento passa a não considerar os artefatos que estão em alteração, os que não estão sob Gerência de Configuração e os não-atuais (versões ou variantes antigas dos artefatos).

Assim, a integração da GCS às demais ferramentas do ambiente, em especial à Gerência de Conhecimento e à Documentação, também foi um dos objetivos deste trabalho. A GCS teve que disponibilizar algumas funcionalidades que pudessem ser utilizadas pelas outras ferramentas do ambiente. Um procedimento básico para utilização dessas funcionalidades foi definido. Esse procedimento engloba ainda a utilização de funcionalidades da ferramenta de Documentação de ODE – XMLDoc (SILVA, 2004), tanto pela GCS quanto pelas outras ferramentas.

Com base na ontologia de artefato, na infra-estrutura de GCS e nos procedimento de integração definidos, construiu-se uma ferramenta de apoio à GCS em ODE, que tem a finalidade de identificar itens de configuração, controlar versões e controlar alterações dos artefatos de ODE.

Por fim, a ferramenta de GCS foi integrada a ODE, à Gerência de Conhecimento, à Documentação e a uma das ferramentas do ambiente, a ferramenta de Gerência de Riscos – GeRis (FALBO et al., 2004c).

1.2 Metodologia

Este trabalho teve início com uma revisão de literatura sobre os principais temas que o compõem. Foram avaliados e discutidos artigos científicos, relatórios técnicos, livros e trabalhos acadêmicos que tratam sobre Ontologias, Ambientes de Desenvolvimento de Software, Documentação, Gerência de Configuração de Software e Gerência de Conhecimento.

A partir desse estudo, foi desenvolvida uma ontologia de artefato, com ontologias específicas para os artefatos de código, documentos e diagramas, para dar apoio à integração e servir como uma linguagem única e padronizada para comunicação entre as ferramentas e os desenvolvedores de ODE. Essa ontologia foi posteriormente complementada com uma ontologia de gerência de configuração de software.

A seguir, ferramentas de GCS existentes foram pesquisadas e algumas abordagens para infra-estruturas de GCS foram analisadas, dentre as quais foi selecionada a que é composta de apenas uma base de dados e arquivos em XML, pois, além de gerar poucas alterações em ODE, facilitava a integração com as outras ferramentas e não possuia grande dificuldade técnica.

O passo seguinte foi estabelecer um procedimento de integração da GCS com as demais ferramentas de ODE. Foi dada ênfase especial à Documentação e à Gerência de Conhecimento. A ferramenta de Documentação é extremamente necessária, pois é utilizada para exibir os artefatos em XML e devido às funcionalidades que disponibiliza para conversão de artefatos da base de dados para arquivos em XML e vice-versa, ou seja, para conversão de arquivos XML para objetos. A Gerência de Conhecimento, por sua vez, teve de ser considerada para que seus serviços não fossem negativamente impactados pela atuação da GCS. Ao contrário, a idéia básica era fazer com que as ferramentas trabalhassem em conjunto e cada uma conseguisse atingir seus objetivos.

De posse da infra-estrutura definida, da ontologia de GCS e dos procedimentos para integração de ferramentas foi possível adaptar as especificações de casos de uso e os modelos de análise e projeto da ferramenta de GCS, proposta em (REZENDE, 2001). Tal ferramenta não estava integrada ao ODE e não foi construída com base na ontologia de artefato e na infra-estrutura de GCS desenvolvidas neste trabalho.

A infra-estrutura definida foi efetivamente implantada e uma nova ferramenta foi, então, desenvolvida e integrada ao ODE e às suas ferramentas, em especial à Documentação e à Gerência de Conhecimento. Como exemplo, a ferramenta de GCS foi utilizada pela ferramenta de Gerência de Riscos – Geris.

Parte do trabalho realizado foi publicado nos anais do VII Workshop Íbero-Americano de Engenharia de Requisitos e Ambientes de Software (IDEAS'2004), realizado em Arequipa, Peru, de 03 a 05 de maio de 2004, no artigo intitulado "Apoio à Documentação em um Ambiente de Desenvolvimento de Software" (NUNES et al., 2004). Além disso, o texto da dissertação foi elaborado à medida que o trabalho avançava, documentando os resultados dos estudos realizados, as técnicas utilizadas e as soluções adotadas, servindo, inclusive de base para outros trabalhos.

Em um trabalho paralelo a este, SILVA (2004) utilizou a ontologia de artefato desenvolvida aqui com o intuito de aperfeiçoar a ferramenta de apoio à Documentação de ODE – XMLDoc, adequando-a a ela, uma vez que, em sua versão inicial (SOARES, 2002), tal ferramenta não era baseada em ontologias.

1.3 Organização do Trabalho

Nesta dissertação, além deste capítulo que apresenta a *Introdução*, há mais cinco capítulos e um anexo.

O Capítulo 2 – *Gerência de Configuração de Software, Documentação e Gerência de Conhecimento no Apoio Automatizado ao Processo de Software* – apresenta os conceitos de Ambientes de Desenvolvimento de Software, enfocando a evolução dos ADS, e apresenta o ambiente ODE. Além disso, discute os principais aspectos relacionados à Gerência de Conhecimento, à Documentação e à Gerência de Configuração e as relações existentes entre elas no contexto de um ADS, em especial de ODE.

No capítulo 3 – *Uma Ontologia de Artefato de Software* – é apresentada a ontologia desenvolvida neste trabalho e suas sub-ontologias, a saber, Artefato, Documento, Diagrama, Artefato de Código e GCS. As ontologias são apresentadas por meio de questões de competência, diagramas de ontologias usando uma extensão da UML como linguagem de modelagem de ontologias, dicionários de termos e restrições, essas últimas escritas na forma de axiomas em lógica de primeira ordem.

O capítulo 4 – *Integrando Gerência de Configuração de Software, Documentação e Gerência de Conhecimento em ODE* – discute as abordagens de infra-estrutura de GCS analisadas e a escolhida neste trabalho. Discute, ainda, como integrar a GCS às demais ferramentas de ODE, em especial à Documentação e à Gerência de Conhecimento.

O capítulo 5 – *Uma Ferramenta de Apoio à Gerência de Configuração de Software em ODE* – apresenta a ferramenta de apoio à Gerência de Configuração desenvolvida no contexto deste trabalho. São apresentadas suas funcionalidades, sua estrutura interna e suas principais características. Além disso, é exemplificada a utilização da GCS pela ferramenta de Gerência de Riscos – GeRis.

O capítulo 6 – *Conclusões e Perspectivas Futuras* – contém as considerações finais sobre o trabalho aqui desenvolvido, apresentando suas contribuições e propostas para trabalhos futuros.

Finalmente, o anexo A – *Funcionalidades da Gerência de Configuração de Software de ODE* – apresenta o modelo de casos de uso completo da especificação de requisitos da ferramenta desenvolvida neste trabalho.

Capítulo 2

Gerência de Configuração de Software, Documentação e Gerência de Conhecimento no Apoio Automatizado ao Processo de Software

Um dos principais objetivos da Engenharia de Software é desenvolver sistemas com qualidade, dentro dos prazos estabelecidos, sem necessidade de alocação de mais recursos. Para que tal objetivo seja alcançado, o foco não deve estar apenas nos produtos gerados, mas também no processo de desenvolvimento de tais produtos.

À medida que aumenta a complexidade dos sistemas a serem desenvolvidos, o processo de software torna-se também mais complexo e cresce a necessidade de automatizar o desenvolvimento de software, através da utilização de ferramentas. Porém, verificou-se que ferramentas isoladas podem oferecer apenas soluções parciais, enquanto o que se deseja é utilizar ferramentas de apoio ao longo de todo o processo de desenvolvimento de software (TRAVASSOS, 1994).

Surgem, então, os Ambientes de Desenvolvimento de Software (ADSs), que podem ser descritos como coleções de ferramentas integradas que facilitam as atividades da engenharia de software, durante todo o ciclo de vida do processo ou pelo menos em porções significativas dele (HARRISON et al., 2000).

A integração de ferramentas é uma questão fundamental para ADSs. Integração em ADSs envolve diversas dimensões, tais como apresentação, dados, processo, controle e conhecimento (TRAVASSOS, 1994).

PRESSMAN (2001) propõe uma solução para o problema de integração, através de uma estrutura, em que o principal mecanismo de integração de ferramentas é a Gerência de Configuração de Software (GCS). A gerência de configuração deve identificar os artefatos a serem gerenciados, prover controle de versão e controle de mudanças, apoiar auditorias e fornecer relatos de configuração dos artefatos gerenciados, ficando assim, no núcleo de todo ADS.

Vale destacar ainda que dentre as dimensões de integração, recentemente a integração de conhecimento tem merecido destacada consideração, tendo em vista que, com o aumento da complexidade dos processos de software, torna-se necessário considerar informações de natureza semântica. Como o desenvolvimento de software é uma atividade de conhecimento intenso, tal conhecimento deve estar disponível aos usuários, assim como deve ser deslocado de volta ao ambiente, tanto quanto possível (NATALI, 2003).

A integração de conhecimento possibilita às ferramentas um melhor entendimento da semântica das informações e dos métodos existentes para tratar a informação, além de tornar disponíveis os serviços básicos de armazenamento, gerenciamento e utilização do conhecimento descrito e adquirido ao longo do processo de desenvolvimento (TRAVASSOS, 1994). Assim, deve-se combinar os benefícios do apoio automatizado provido pelos ADSs com os benefícios da gerência de conhecimento, evitando que erros já cometidos venham a ser repetidos e possibilitando o reuso de soluções já aprovadas na execução de tarefas similares, buscando melhorar a produtividade e a qualidade e diminuir custos (NATALI, 2003).

Dentre os itens de conhecimento que podem ser gerenciados, destacam-se os documentos produzidos ao longo dos processos de software. Uma documentação de qualidade é extremamente necessária para o sucesso do desenvolvimento e, uma vez que implica diretamente na qualidade do processo e do produto gerado, deve ser construída e acessada de forma única e padronizada, necessitando de procedimentos para seu controle.

Este capítulo procura dar uma visão geral dos conceitos relativos a ambientes de desenvolvimento de software, gerência de conhecimento, documentação e gerência de configuração de software. A seção 2.1 discute o que é um ambiente de desenvolvimento de software e a evolução por que tem passado essa classe de sistemas. Apresenta ainda, o ambiente ODE (*Ontology-based software Development Environment*) (FALBO et al., 2003), ambiente no qual este trabalho foi desenvolvido. Na seção 2.2, o enfoque recai sobre a gerência de conhecimento, sendo apresentada também, a infra-estrutura de gerência de conhecimento de ODE. A seção 2.3 versa sobre o tema documentação de software, sendo discutido também como a documentação é tratada em ODE. A seção 2.4 discute aspectos da gerência de configuração de software, inclusive apresentando brevemente alguns sistemas de gerência de configuração existentes. A seção 2.5 trata da relação estreita entre a gerência de configuração, documentação e a gerência de conhecimento, sobretudo no contexto de um ADS. Finalmente, a seção 2.6 apresenta as conclusões deste capítulo.

2.1 Ambientes de Desenvolvimento de Software

Com o aumento da complexidade dos sistemas de software a serem desenvolvidos, os engenheiros de software sentiram a necessidade de apoio de ferramentas e ambientes que automatizassem o processo de desenvolvimento de software, visando à qualidade do software e a uma maior produtividade na sua construção. Ferramentas CASE (*Computer Aided Software Engineering*) passaram, então a ser utilizadas, uma vez que provêem a habilidade de automatizar atividades manuais e aumentar a percepção do processo de desenvolvimento, ajudando a garantir que a qualidade é projetada antes do produto ser construído (PRESSMAN, 2001).

Apesar dos benefícios do uso de ferramentas CASE individuais, atualmente, o número e a variedade de ferramentas têm crescido a tal ponto que levou os engenheiros de software a pensarem não apenas em automatizar os seus processos, mas sim em trabalhar com diversas ferramentas que interajam entre si e forneçam suporte a todo ciclo de vida do desenvolvimento. O objetivo principal é atender a grande demanda de software, com uma produção rápida e em custo razoável, sem perder de vista a qualidade (HARRISON et al., 2000).

Para atender a tal propósito, surgiram os Ambientes de Desenvolvimento de Software (ADSs) que buscam combinar técnicas, métodos e ferramentas para apoiar o engenheiro de software na construção de produtos de software, abrangendo todas as atividades inerentes ao processo, tais como planejamento, gerência, desenvolvimento e controle da qualidade (FALBO, 1998).

Os benefícios da utilização de ADSs incluem (PRESSMAN, 2001):

- Facilitar a transferência de informação entre ferramentas e, consequentemente, entre passos do processo de Engenharia de Software;
- Reduzir os esforços para efetuar atividades de gerência de configuração, garantia de qualidade e produção de documentação;
- Aumentar o controle do projeto, através de melhor planejamento, monitoramento e comunicação;
- Prover coordenação entre os recursos humanos que estão trabalhando em um grande projeto de software.

Segundo TRAVASSOS (1994), um ADS deve se preocupar com o apoio às atividades individuais e ao trabalho em grupo, o gerenciamento de projeto, o aumento da qualidade geral

dos produtos e o aumento da produtividade, permitindo ao desenvolvedor acompanhar o projeto e medir a sua evolução.

A integração de ferramentas ainda é um dos maiores desafios dos ADSs, pois demanda representação consistente da informação, interfaces padronizadas, significados homogêneos da comunicação entre engenheiros de software e ferramentas e uma efetiva abordagem que possibilite aos ADSs mudar entre várias plataformas (PRESSMAN, 2001).

THOMAS e NEJMEH (1992), TRAVASSOS (1994) e PFLEEGER (2001), dentre outros, propõem que a integração de ferramentas em ADSs seja tratada através de uma infra-estrutura que contemple várias dimensões, dentre elas:

- *Integração de Dados*: suportada por serviços de repositório de dados, que provê armazenamento e gerência de objetos, e de integração de dados, compartilhando informações entre as ferramentas.
- *Integração de Controle*: suportada por serviços de gerência de processos e mensagens, controlando os eventos ocorridos e compartilhando funcionalidades.
- *Integração de Processo*: suportada pela ligação explícita entre ferramentas e o processo de desenvolvimento de software.
- *Integração de Apresentação*: suportada por serviços de interface com o usuário. Tem o objetivo de criar uma uniformidade entre as interfaces do ambiente, aumentando a sua usabilidade.
- *Integração de Conhecimento*: suportada por serviços de gerência de conhecimento, permitindo capturar conhecimento durante os projetos de software e oferecendo apoio baseado em conhecimento aos engenheiros de software durante a realização de atividades do processo.
- *Integração de Plataforma*: suportada pela independência da plataforma sobre a qual funcionará a ferramenta.

O processo de desenvolvimento ou manutenção de um produto indica uma regra crucial na determinação do nível de qualidade e do custo envolvido em tal desenvolvimento ou manutenção. Sua importância é reconhecida há décadas, mas há pouco tempo tem se tornado uma prioridade no desenvolvimento de software e nos serviços de alta tecnologia (ARBAOUI et al., 2002).

Porém, devido às dificuldades inerentes ao controle de processo de software, a integração de processo tem se mostrado tão importante que deu origem a uma nova classe de ambientes, os Ambientes de Desenvolvimento de Software Centrados em Processo

(ADSCPs). ADSCPs integram ferramentas para apoiar o desenvolvimento do software e para apoiar a modelagem e a execução do processo de software que desenvolve este produto (HARRISON et al., 2000; FUGGETTA, 2000). Os modelos de processo de software tipicamente descrevem as atividades a serem realizadas, os responsáveis por cada uma delas e as ferramentas de software a serem utilizadas. São criados para vários propósitos, dentre eles: documentação do processo de software, melhoria do processo de software, gerenciamento do fluxo de trabalho e automatização (GRUHN, 2002).

Desta forma, ao prover meios mais poderosos de descrever e implementar processos de engenharia de software, ADSCPs provêem um poderoso mecanismo de integração de processo e ferramentas e, parcialmente, de automatização de tarefas (HARRISON et al., 2000).

Alguns exemplos de ADSCPs citados em (HARRISON et al.,2000) incluem: *Adele, Argo, PCTE e SPADE*. Em (ARBAOUI et al., 2002) são citados *PML TEMPO* e *APEL*, construídos a partir do *Adele*, *LEU*, *PEACE*, *PIE* e *OZ*. HOLZ et al. (2001) e RICHTER e MAURER (2003) enfocam o ADSCP *MILOS (Modeling Language and Operational Support for Software Processes)*, que provê meios para definir um modelo de processo genérico e configurar planos de projeto concretos baseados neste modelo. Além disso, em (FALBO, 1998) foi proposta uma infra-estrutura para definir processos na Estação TABA, permitindo, assim, a instanciação de ADSCPs no ambiente.

Mas não é apenas a melhoria do processo que influi na qualidade de sistemas computacionais. Durante o desenvolvimento de sistemas, os desenvolvedores se deparam com uma série de atividades não triviais, dentre elas, identificação e descrição corretas do que o sistema de software se propõe a fazer. Isso é particularmente difícil quando a equipe envolvida não tem o conhecimento necessário nem experiência sobre o domínio do problema, o que afeta a produtividade no desenvolvimento (OLIVEIRA et al., 2004).

Ambientes de Desenvolvimento de Software Orientados a Domínio (ADSODs) (OLIVEIRA et al., 2004) surgiram como uma extensão dos ADSs tradicionais, pois além de conter um repositório e um conjunto de serviços e ferramentas, um ADSOD incorpora conhecimento sobre o domínio de aplicação e viabiliza o uso deste conhecimento ao longo do processo de software. Assim, desenvolvedores de software podem trabalhar diretamente com o domínio do problema e ter assistência específica no contexto do seu trabalho (OLIVEIRA, 1999).

As principais questões para a definição de um ADSOD referem-se a: (i) qual conhecimento deve estar disponível no ambiente, (ii) como este conhecimento deve estar

organizado e representado e (iii) quando e como utilizar este conhecimento definido durante o desenvolvimento (OLIVEIRA et al., 2004; MIAN, 2003).

Porém, além do conhecimento de domínio, outros tipos de conhecimento também são importantes para apoiar o desenvolvimento de software, tais como relativo às diretrizes e melhores práticas organizacionais, às lições aprendidas e aos métodos e técnicas de software. Tal conhecimento, entretanto, varia de uma organização para outra (LIMA et al., 2004). Surgiu, então, o conceito de Ambientes de Desenvolvimento de Software Orientados a Organização (ADSOrg), cujo objetivo é fornecer aos desenvolvedores conhecimento organizacional necessário à realização das atividades do processo de desenvolvimento e manutenção de software.

Para apoiar o gerenciamento do conhecimento em organizações, um ADSOrg deve considerar alguns requisitos, dentre eles: (i) ter a representação da infra-estrutura da organização, (ii) reter conhecimento especializado sobre desenvolvimento e manutenção de software, (iii) permitir a utilização deste conhecimento em projetos, (iv) apoiar a atualização constante do conhecimento armazenado no ambiente e (v) facilitar a localização de especialistas da organização que podem ser úteis em um projeto (LIMA, 2004).

A Estação TABA, sendo um meta-ambiente, é capaz de gerar tanto ADSODs (OLIVEIRA, 1999) quanto ADSOrg (LIMA, 2004). O ambiente ODE (FALBO et al., 2003) também é um exemplo de ADSOD (MIAN et al., 2002) ou ADSOrg (NATALI, 2003).

Em meio a evolução dos ADS, pode-se observar que assim como a falta de informação foi constatada como um problema grave, o excesso de recursos de informação também o é, já que, em última instância, pode não ser possível coletar em tempo hábil a informação necessária para apoiar a tomada de decisão. Esse problema está associado ao excesso de recursos de informação e à falta de semântica para guiar uma busca por recursos realmente relevantes para o contexto em mãos. Para tratar esse problema, uma potencial abordagem, que inclusive vem sendo usada em outras áreas, tal como a *Web* Semântica (*Semantic Web*), consiste em associar conhecimento do significado aos recursos de informação, tipicamente através da utilização de (meta) dados e ontologias (FALBO et al., 2004a).

Apesar do escopo dos ADSs ser menos abrangente que o da *Web* Semântica, à medida que eles crescem e oferecem mais funcionalidades de apoio ao desenvolvimento de software, principalmente com a incorporação de facilidades de gerência de conhecimento, problemas deste tipo tornam-se preocupantes. Assim, da mesma forma que a *Web* tem avançado para *Web* Semântica, os ADSs buscam evoluir para ADSs Semânticos. Neste contexto, ontologias

tornam-se essenciais, pois podem ser usadas como uma estrutura unificadora para dar semântica e uma representação comum à informação (FALBO et al., 2004a).

Ontologias têm sido larga e sistematicamente utilizadas em ODE, visando torná-lo um ADS Semântico. Uma vez que o presente trabalho foi desenvolvido no contexto de ODE, maiores detalhes sobre esse ADS são fornecidos na próxima seção.

2.1.1 O Ambiente ODE

ODE (*Ontology-based software Development Environment*) (FALBO et al., 2003; FALBO et al., 2004a) é um Ambiente de Desenvolvimento de Software desenvolvido no Laboratório de Engenharia de Software da Universidade Federal do Espírito Santo (LabES/UFES), tendo por base ontologias e buscando tratar características de um ADS Semântico.

O propósito fundamental de ODE é ser um ambiente integrado, que forneça serviços de infra-estrutura, permitindo integrar ferramentas ao longo das dimensões de integração discutidas na seção 2.1: dados, controle, processo, apresentação, conhecimento e plataforma (FALBO et al., 2004a).

A sua característica marcante, que o distingue de outros ADSs, é o fato de ser desenvolvido baseado em ontologias. Uma ontologia é a especificação de uma conceituação (GRUBER, 1995), que inclui um vocabulário de termos e a especificação de seu significado. Inclui também definições e uma indicação de como os conceitos são inter-relacionados, que coletivamente impõem a estrutura em um domínio e limita as possíveis interpretações dos termos (JASPER et al., 1999).

A premissa do projeto ODE está baseada no argumento de que, se as ferramentas em um ADS são construídas baseadas em ontologias, a integração delas pode ser mais facilmente obtida. A mesma ontologia pode ser usada para construir ferramentas diferentes que apóiam atividades de engenharia de software correlacionadas. Além disso, se as ontologias são integradas, a integração de ferramentas construídas com base nelas pode ser bastante facilitada (FALBO et al., 2003).

ODE iniciou como um Ambiente de Desenvolvimento de Software Centrado em Processo, porém posteriormente evoluiu para um ADSOD, quando passou a considerar o conhecimento do domínio, para prover apoio de gerência de conhecimento. Este objetivo foi alcançado quando ODE passou a prover suporte ao desenvolvimento de ontologias de

domínio, como forma de permitir a descrição do conhecimento de domínio no ambiente, através de um editor de ontologias proposto em (MIAN, 2003).

Além do apoio à captura do conhecimento sobre o domínio, foi visto em ODE a necessidade de apoiar a captura do conhecimento gerado sobre desenvolvimento de software. NATALI (2003) desenvolveu uma infra-estrutura de gerência de conhecimento que foi incorporada ao ODE, tornando-o um ambiente capaz de prover aos desenvolvedores conhecimento acumulado no contexto do desenvolvimento e da manutenção de software. A infra-estrutura desenvolvida consiste de uma memória organizacional, que apóia o armazenamento e compartilhamento do conhecimento, além de serviços de gerência de conhecimento, que provêem ativamente informações úteis a usuários que estejam trabalhando em atividade que exijam conhecimento.

Por ser baseado em ontologias, ODE realiza grande parte de suas tarefas utilizando e respeitando os modelos e restrições impostos pelas ontologias sobre as quais se fundamenta. Além disso, como ODE utiliza a tecnologia de objetos, é seguida uma abordagem sistemática que permite que os elementos definidos em uma ontologia (conceitos, relações, propriedades e restrições definidas como axiomas) sejam mapeados para um modelo de objetos que é, então, integrado como parte fundamental da estrutura do ambiente (FALBO et al., 2004a).

Com o intuito de manter a amarração semântica entre os objetos de ODE e agregar ontologias ao ambiente, sua arquitetura conceitual[1] foi projetada em três níveis, como mostra a figura 2.1, discutidos a seguir (FALBO et al., 2004a):

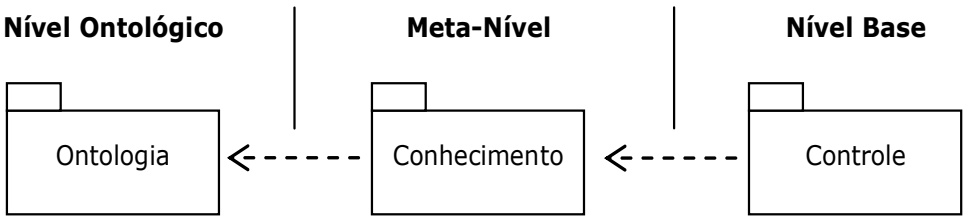


Figura 2.1 – Os três níveis da arquitetura conceitual de ODE.

- O Nível Ontológico, ou pacote *Ontologia*, é responsável pela descrição das ontologias de ODE. Nela encontram-se as classes que definem explicitamente a ontologia de um certo domínio e seus elementos, não sendo de sua responsabilidade prover detalhes de implementação de sistemas nesse domínio. Contudo, vale ressaltar que as instâncias do nível ontológico guiam a definição

[1] Está-se usando o termo "arquitetura conceitual" para indicar uma decomposição de alto nível dos pacotes do ambiente, em contraste à arquitetura de software em camadas utilizada para implementá-lo efetivamente.

das classes dos outros níveis, originando as principais classes tanto do meta-nível (ou nível de conhecimento) quanto do nível base.

- O Meta-nível, ou pacote *Conhecimento*, abriga as classes que descrevem o conhecimento em relação a um domínio de aplicação. Suas classes são derivadas das ontologias e as instâncias dessas classes atuam no papel de conhecimento sobre os objetos do nível base. Elas constituem o conhecimento do ambiente, que pode ser utilizado tanto pelo ambiente quanto pelas ferramentas que o compõem.
- O Nível Base, ou de Aplicação (pacote *Controle*), define as classes responsáveis por implementar as aplicações no contexto do ambiente (funcionalidades da infra-estrutura do ambiente e suas ferramentas). Essas classes também são derivadas das ontologias, mas tipicamente incorporam detalhes não descritos por elas, necessários para implementar as aplicações do ambiente. Muitas vezes é necessária a criação de novas classes, associações, atributos e operações com o intuito de se tratar decisões específicas do nível de aplicação.

Essa divisão arquitetural facilita o estabelecimento de uma correlação entre objetos dos diferentes níveis de ODE, permitindo anotar os objetos com informação semântica, dada efetivamente pelas ontologias, o que contribui para que ODE possa evoluir para um ADS semântico.

Dessa forma, em diversas tarefas do ambiente, os objetos dos níveis base ou mesmo de conhecimento podem ter sua anotação ontológica utilizada para realização de tarefas de forma mais inteligente e consistente, segundo uma visão ontológica.

Os três pacotes citados acima fazem parte da camada de domínio do problema (CDP) da arquitetura de software real de ODE. Essa arquitetura é composta, ainda, de mais três camadas, a saber: a camada de interação humana (CIH), que é responsável pelas classes de interface com o usuário, a camada de gerência de tarefas (CGT), que é responsável pelo controle das funcionalidades (classes de aplicação), e a camada de gerência de dados (CGD), responsável pela persistência de objetos em um banco de dados relacional, como mostra a figura 2.2 (MORO et al., 2005).

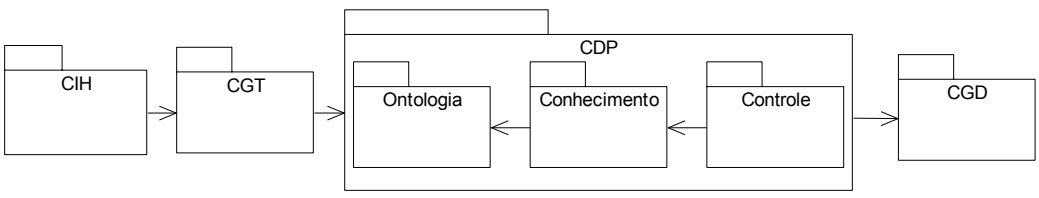


Figura 2.2. Arquitetura Base de ODE

ODE e suas ferramentas internas têm sido implementados em plataforma livre, utilizando Java como linguagem de programação. As informações ficam armazenadas em um banco de dados relacional – o PostgreSQL e, em alguns casos específicos, em arquivos XML.

Dentre os principais módulos e ferramentas integrados atualmente ao ambiente ODE pode-se citar: Controle de Projetos (Manutenção e Caracterização), Cadastro de Conhecimento (Processos, Riscos e Qualidade), Cadastro de Recursos Organizacionais, Definição de Processos, Acompanhamento de Projetos, Alocação de Recursos Humanos, Alocação de Ferramentas de Software, GeRis - Gerência de Riscos, Agenda e Registro de Esforço, EstimaODE - Estimativas de Tamanho e Esforço, Gerência de Conhecimento, Definição de Processos em Níveis, Manipulação de Conhecimento Organizacional, XMLDoc - Documentação Automatizada e OODE – Modelagem UML.

2.2 Gerência de Conhecimento

O diferencial competitivo de uma empresa depende essencialmente da sua capacidade em oferecer soluções para as demandas da sociedade. Essa capacitação advém dos conhecimentos e das experiências acumuladas pelos funcionários ao longo do tempo (REZENDE, 2003). No entanto, para essa vantagem competitiva ser sustentável, o conhecimento não pode estar no nível de indivíduo, uma vez que este conhecimento é perdido quando o indivíduo sai da empresa (LIMA, 2004). Conhecimento organizacional registrado em papel também representa um problema, pois não pode ser facilmente acessado, compartilhado e atualizado (O'LEARY, 1998a). Associado a isso, o crescimento na complexidade dos produtos, a globalização, a emergência de organizações virtuais e o crescimento no foco de orientação ao cliente, demandam métodos mais sistemáticos para gerenciar o conhecimento (STAAB, 2001).

Desta forma, as organizações têm adotado a Gerência de Conhecimento para melhorar a troca de conhecimento entre seus funcionários, capturar conhecimento no nível organizacional e promover o surgimento de conhecimento novo (LIMA, 2004).

A Gerência de Conhecimento consiste no gerenciamento formal que facilita a criação, o acesso e o reuso do conhecimento, tipicamente usando tecnologia avançada (O'LEARY, 1998b). É a administração, de forma sistemática e ativa, dos recursos de conhecimento de uma organização, utilizando tecnologia apropriada e visando fornecer benefícios à

organização (LIMA, 2004). Envolve gerenciamento de recursos humanos e organização e cultura da empresa, assim como métodos e ferramentas da tecnologia da informação que possam apoiá-la (O'LEARY et al., 2001).

A gerência de conhecimento tem seu foco voltado para a solução de algumas questões, tais como, a forma como as organizações podem tirar maior proveito do conhecimento existente dentro delas, como os membros da organização podem distribuir o conhecimento, como registrar as soluções adotadas para tratar problemas, como reter o conhecimento de seus especialistas mesmo quando esses deixam a organização, além de discutir formas de se gerar novo conhecimento a partir do conhecimento existente dentro da organização ou a partir de fontes externas (NATALI, 2003).

Uma das preocupações dos pesquisadores na área da Gerência de Conhecimento foi definir os tipos de conhecimento existentes. Apesar da grande variedade de visões, pode-se afirmar que todas elas estão baseadas em se assumir que há um conhecimento difícil de se articular através da linguagem e existe apenas em forma de experiências - *conhecimento tácito.* O outro tipo de conhecimento pode ser expresso verbalmente, coletado em livros ou manuais, por exemplo, e distribuído eletronicamente - *conhecimento explícito* (NATALI, 2003; RUS et al., 2002).

A gerência de conhecimento é um processo cíclico com atividades relacionadas entre si (FISCHER et al., 2001). NATALI (2003) faz um comparativo entre os processos de gerência de conhecimento propostos por vários autores e visualiza que eles compartilham várias similaridades. A maioria dos processos se inicia com a atividade de "criação" de conhecimento. A segunda atividade gira em torno do armazenamento e organização do conhecimento capturado, integrando-o à organização. A terceira atividade lida com o acesso ao conhecimento, que envolve distribuição pró-ativa e acesso e recuperação do conhecimento. As últimas atividades dos processos tratam do uso e da manutenção do conhecimento.

Em se tratando da gerência de conhecimento no desenvolvimento de software, a realidade não é diferente. Uma abordagem de desenvolvimento baseada em reuso parte da premissa que esforços e experiências adquiridas em projetos anteriores são de grande valia para novos projetos. Para viabilizar a reutilização da informação, é preciso retê-la e armazená-la de forma a minimizar o esforço para obtê-la no futuro (NATALI, 2003).

As organizações têm passado a enxergar a Gerência de Conhecimento como uma estratégia de prevenção e mitigação de riscos, uma vez que trata riscos frequentemente ignorados, tais como repetição de erros, retrabalho, perda do conhecimento etc (RUS et al.,

2002). Entretanto, implementar a Gerência de Conhecimento envolve muitos desafios e obstáculos. Três questões são particularmente importantes (RUS et al., 2002):

- *Questões tecnológicas* – Nem sempre é possível integrar diferentes subsistemas e ferramentas para alcançar o nível de compartilhamento desejado. Além disso, segurança é um requisito que as tecnologias disponíveis não provêem satisfatoriamente.
- *Questões organizacionais* – Um erro cometido por muitas organizações é focar apenas na tecnologia e não na metodologia a ser utilizada.
- *Questões individuais* – As pessoas frequentemente não têm tempo para registrar ou procurar conhecimento, não querem compartilhar o conhecimento adquirido ou não querem reutilizar o conhecimento de outros.

2.2.1 Sistemas de Gerência de Conhecimento

O conhecimento de uma organização pode ser gerenciado de maneira mais eficaz através de um sistema de gerência de conhecimento. Um sistema de gerência de conhecimento interage com outros sistemas da organização para facilitar aspectos do processamento do conhecimento, auxiliando na solução de problemas e facilitando a tomada de decisões. Esses sistemas têm sido utilizados nas mais diversas áreas, tais como medicina, engenharia, finanças, construção, dentre outras (NATALI, 2003).

Um sistema de gerência de conhecimento é um sistema de suporte à gerência do conhecimento em uma organização, o que implica uma infra-estrutura formada por uma dimensão técnica e outra organizacional. É uma solução híbrida que envolve pessoas e tecnologia e propõe a memória organizacional como o núcleo da dimensão técnica e, ao redor desta memória, os vários serviços para apoiar as atividades de gerência de conhecimento (LIMA, 2004).

A memória organizacional é uma representação explícita e persistente do conhecimento e das informações cruciais para uma organização, cuja finalidade é facilitar seu acesso, compartilhamento e reuso pelos diversos membros da organização (NATALI, 2003). Pode ser considerada um repositório do conhecimento disponível na organização, cuja finalidade é assegurar que o conhecimento desejado possa ser recuperado no tempo e no lugar correto. No entanto, esse repositório não deve ser visto como um repositório físico único,

monolítico e de um tipo específico, uma vez que nem mesmo as organizações são entidades monolíticas (LIMA, 2004).

Os repositórios de conhecimento representam o conteúdo da infra-estrutura de conhecimento e, de forma mais específica, da memória organizacional. Esta, por sua vez, pode conter vários repositórios de conhecimento, que variam de acordo com negócio e domínio específicos no qual a organização está inserida (O'LEARY, 1998a). Atributos descritivos do conhecimento armazenado nos repositórios de experiência são essenciais para o uso e a manutenção do conhecimento. Repositórios de conhecimento tipicamente utilizados por empresas de consultoria, por exemplo, incluem bases de propostas, bases de projetos, bases de melhores práticas, bases de especialistas, bases de notícias, dentre outras (O'LEARY, 1998a).

Para se construir um sistema de gerência de conhecimento, as atividades típicas de um processo de gerência de conhecimento devem ser consideradas. Para tal, os seguintes serviços devem estar presentes no sistema a ser implementado (NATALI, 2003):

- Criação ou importação: o sistema de gerência de conhecimento não possui todo o conhecimento necessário para a solução de problemas. Assim, deve-se permitir que membros da organização criem novo conhecimento e que este seja capturado pelo sistema (FISCHER et al., 2001);
- Integração e Captura: tem o objetivo de não atrapalhar o fluxo de trabalho normal do usuário, pois caso contrário não conseguiria sua aceitação. Para tal, a memória organizacional deve ser ativamente integrada ao processo de trabalho e às práticas da organização que constroem o conhecimento, capturando o novo conhecimento gerado (FISCHER et al., 2001);
- Busca e Acesso: é necessário apoiar a busca por itens de conhecimento existentes na memória organizacional, permitindo o seu acesso (STAAB et al., 2001). Máquinas de busca, agentes inteligentes e modelos de visualização estão entre as tecnologias dominantes neste aspecto (O'LEARY, 1998a);
- Disseminação: refere-se à apresentação pró-ativa de itens de conhecimento. Serviços dessa natureza são importantes, uma vez que os membros das organizações, freqüentemente, estão muito ocupados para procurar conhecimento ou nem mesmo sabem da existência de conhecimento pertinente (NATALI, 2003);
- Uso: os dados sobre o uso do sistema de gerência de conhecimento são fundamentais para a identificação de novos conhecimentos relevantes ou itens

obsoletos, sendo a base para a evolução da memória organizacional (STAAB et al., 2001);

- Preservação e evolução: uma vez que o sistema de gerência de conhecimento tem que lidar com informação incompleta, potencialmente incorreta e em freqüente atualização, deve haver serviços para apoiar a manutenção e a evolução da memória organizacional (STAAB et al., 2001);

Várias tecnologias estão sendo utilizadas no desenvolvimento de sistemas de gerência de conhecimento. Pode-se destacar: bases de conhecimento, ontologias, *browsers e* máquinas de busca, *Intranets e Internet, XML,* agentes inteligentes, modelos de visualização, bases de dados e *data warehouses, groupware, workflow* e modelos de negócio, gerência de documento, sistemas baseados em conhecimento e sistemas especialistas (O'LEARY, 1998a; O'LEARY, 1998b; LIMA, 2004; NATALI, 2003).

2.2.2 Gerência de Conhecimento em ODE

Conforme discutido na seção 2.1, uma das preocupações centrais do projeto ODE é a integração de conhecimento. Assim, é necessário apoiar a captura do conhecimento gerado sobre o desenvolvimento de software em ODE. Cada novo projeto de software desenvolvido no ambiente gera conhecimento relevante e útil a novos projetos. Para atender a esse requisito, foi desenvolvida uma infra-estrutura de gerência de conhecimento para ODE, de forma a torná-lo um ambiente capaz de prover aos desenvolvedores conhecimento acumulado no contexto de desenvolvimento e manutenção de software (NATALI et al., 2003).

Na infra-estrutura proposta, mostrada na Figura 2.3, a memória organizacional é posicionada em seu centro, apoiando o compartilhamento e reuso de conhecimento. Em torno da memória organizacional, se localizam os serviços que apóiam cada uma das atividades do processo de gerência de conhecimento.

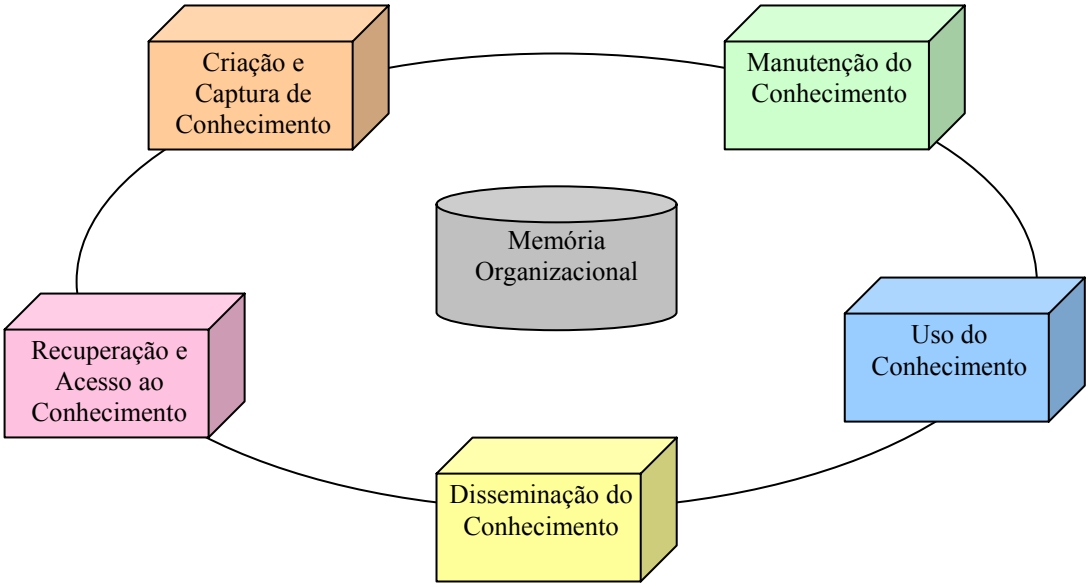


Figura 2.3 - Infra-estrutura da Gerência de Conhecimento de ODE.

A memória organizacional é uma representação explícita e persistente do conhecimento e das informações cruciais para uma organização. Tem a finalidade de facilitar o acesso, compartilhamento e reuso pelos diversos membros da organização e deve gerenciar todos os tipos de conhecimento necessários ao desenvolvimento de software (NATALI et al., 2003).

Em ODE, os itens de conhecimento podem ser classificados em itens de conhecimento formais e informais. Os itens de conhecimento formais são os diversos tipos de artefatos gerados pelas ferramentas do ambiente. Já os itens de conhecimento informais compreendem, atualmente, lições aprendidas e pacotes de mensagens. As lições aprendidas são relatos de sucesso ou oportunidades de melhoria detectadas durante o desenvolvimento de um produto de software, utilizando o processo definido para o projeto. Uma lição sempre está associada a um objeto de conhecimento, tal como um conhecimento sobre uma atividade, um artefato, um procedimento, um método, um recurso, uma ferramenta etc, que são instâncias de conceitos das ontologias utilizadas (NATALI et al., 2003). Pacotes de mensagens, por sua vez, são obtidos pela avaliação e adaptação das mensagens trocadas nas ferramentas de *groupware* de ODE. Tais ferramentas incluem uma ferramenta de mensagens instantâneas, uma ferramenta de *email* com grupos de discussão e uma ferramenta de fórum. A figura 2.4 mostra a estrutura da memória organizacional de ODE (FALBO et al., 2004a).

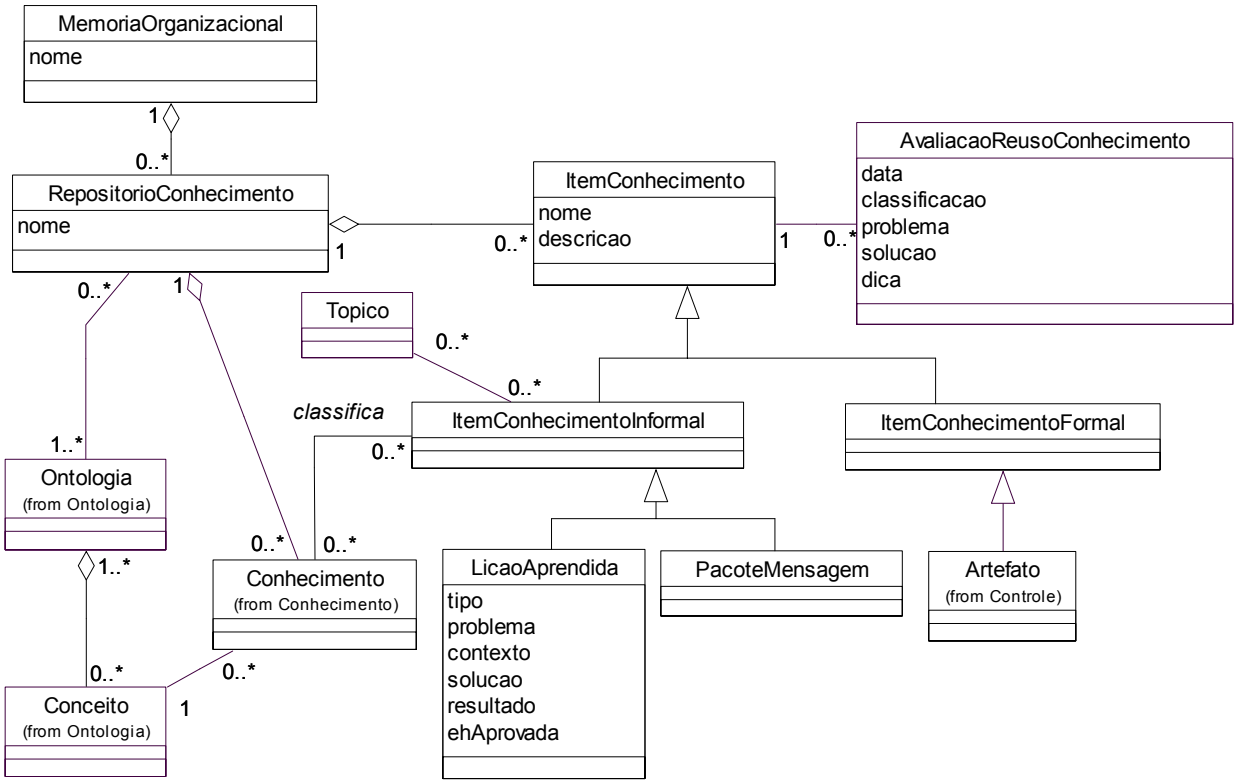


Figura 2.4 - Estrutura da Memória Organizacional de ODE.

Tanto os itens de conhecimento quanto as instâncias de ontologias (objetos da classe Conhecimento) compõem os repositórios de conhecimento do ambiente, que formam a memória organizacional. De fato, os objetos da classe Conhecimento podem ser vistos como um tipo de item de conhecimento formal, já que são itens de conhecimento utilizados em várias situações no ambiente e são formalmente definidos a partir de ontologias. Contudo, uma vez que não são utilizados diretamente por todos os serviços de gerência de conhecimento (em especial recuperação e uso de conhecimento), eles não são tratados como tal.

Pode-se notar que os itens de conhecimento, formais ou informais, são anotados segundo conceitos de ontologias. Considerando-se os itens de conhecimento formais, os artefatos, que fazem parte do nível base, eles possuem uma correspondência com a classe *KArtefato* do nível de conhecimento, que, por sua vez, está ligada ao conceito *Artefato* no nível ontológico. Já os itens de conhecimento informais (lições aprendidas e pacotes de mensagens) são classificados usando instâncias da classe *Conhecimento*, correlacionando-se, assim, indiretamente a conceitos das ontologias (FALBO et al., 2004a).

Como mostra a figura 2.3, a infra-estrutura de gerência de conhecimento de ODE possui, ainda, um conjunto de serviços que se apóia na memória organizacional comum, incluindo serviços para:

- **Criação e Captura de Conhecimento**

Como ODE lida com três tipos de conhecimento em sua memória organizacional (artefatos, lições aprendidas e pacotes de mensagens), deve haver facilidades para apoiar a captura de cada um destes itens de conhecimento.

Artefatos de software produzidos pelas ferramentas do ambiente, durante o processo de desenvolvimento, são armazenados no repositório central de ODE e, conforme proposto por NATALI (2003), deveriam ser submetidos à gerência de configuração. Vale destacar, contudo, que na versão desenvolvida em (NATALI, 2003) ainda não havia um sistema de gerência de configuração de software integrado a ODE.

Pacotes de mensagens são criados a partir da classificação das mensagens trocadas nas aplicações de *groupware* de ODE. Inicialmente as mensagens são trocadas entre os desenvolvedores de ODE. Posteriormente, tais mensagens são classificadas para compor os pacotes de mensagens. A partir daí, esse conhecimento pode ser pesquisado e reutilizado por outros desenvolvedores. Percebe-se que na integração da gerência de conhecimento com *groupware*, o conhecimento contido nas mensagens, que usualmente estão dispersas em caixas de *email* e ferramentas isoladas ou perdidas em ferramentas de mensagens instantâneas, pode ser capturado, armazenado, disseminado e reutilizado (FALBO et al., 2004b).

A infra-estrutura de ODE oferece também um serviço para registrar as lições aprendidas. Neste momento, esse conhecimento ainda não fica disponível para outros desenvolvedores. O gerente de conhecimento é responsável por avaliar e adaptar a lição aprendida para que ela possa ser considerada um conhecimento de nível organizacional. Uma vez aprovada, a lição aprendida fica disponível como um item de conhecimento para todos na organização.

- **Recuperação e Acesso ao Conhecimento**

O acesso ao conhecimento em ODE pode se dar através da iniciativa do desenvolvedor em buscar itens de conhecimento armazenados na memória organizacional. Neste caso, cabe ao usuário definir quais são as suas necessidades, ou seja, qual o tipo de conhecimento ele

deseja buscar. Essas necessidades do usuário se tornam uma consulta à memória organizacional e os itens de conhecimento recuperados são apresentados.

- **Disseminação do Conhecimento**

Ao contrário da busca, em que o usuário deve ter a iniciativa de procurar conhecimento, na disseminação, esta iniciativa é do próprio sistema. Os itens de conhecimento que o sistema julgar relevantes são apresentados ao usuário como uma sugestão de ajuda na realização de uma atividade. O serviço de disseminação é implementado para cada ferramenta, uma vez que para se oferecer ajuda pró-ativa é necessário conhecer detalhes sobre a tarefa que está sendo realizada e, conseqüentemente, sobre a ferramenta que apóia a atividade.

Em ODE, enquanto um usuário utiliza uma determinada ferramenta, agentes de software monitoram suas ações, identificando suas necessidades de conhecimento e recuperando experiências anteriores. Esta disseminação de conhecimento é particularmente importante quando os usuários não estão motivados a buscar uma informação ou não sabem da existência de conhecimento relevante à tarefa que estão executando (NATALI et al., 2003).

- **Uso do Conhecimento**

Os itens de conhecimento recuperados são apresentados ao usuário de ODE, para que ele possa navegar por este conjunto de itens de conhecimento e escolher qual item deseja reutilizar. Se o usuário selecionar um item de conhecimento, ele deve avaliar seu conteúdo para auxiliar a manutenção da memória organizacional. Esta avaliação inclui informações como: se o item foi útil, problemas que surgiram ao reutilizá-lo e as soluções aplicadas.

- **Manutenção do Conhecimento**

A manutenção do conhecimento em ODE é realizada levando-se em conta o *feedback* do usuário. Com isso, o gerente de conhecimento pode verificar quais itens de conhecimento estão obsoletos e quais nunca foram reutilizados, podendo decidir quais devem ser excluídos.

Para a exclusão de itens de conhecimento, o gerente de conhecimento pode escolher um repositório específico para realizar a manutenção ou operar sobre todos os repositórios, ou seja, a memória organizacional. Após definir o local da manutenção, é preciso estabelecer os critérios para exclusão, que podem ser baseados nas caracterizações de reuso dos itens de conhecimento feitas pelo usuário. A partir das caracterizações de reuso de um item, é possível determinar, por exemplo, sua importância (dada pela classificação feita pelos usuários),

freqüência de uso ou data do último uso. Com base nos critérios definidos, os itens candidatos à exclusão são recuperados, apresentados e, caso selecionados, excluídos (NATALI, 2003).

2.3 Documentação

Devido à complexidade dos sistemas atuais e à contínua necessidade de armazenamento e utilização do conhecimento nas organizações, uma boa documentação tem se mostrado essencial.

No desenvolvimento de sistemas de software, deve existir, dentre outros, documentação para descrever processos (plano de projeto, plano de testes etc.), descrever a estrutura do sistema a ser construído (modelos de análise e projeto, código fonte etc), descrever o produto gerado (manual do usuário, ajuda online, tutoriais etc) e descrever a organização em si (habilidades dos recursos humanos, recursos de hardware e software disponíveis, documentos de problemas e soluções etc).

Uma documentação de qualidade propicia uma maior organização durante o desenvolvimento de um sistema, facilitando modificações e futuras manutenções no mesmo. Além disso, reduz o impacto da perda de membros da equipe, reduz o tempo de desenvolvimento de fases posteriores, reduz o tempo de manutenção e contribui para redução de erros, aumentando assim, a qualidade do processo e do produto gerado. Dessa forma, a criação da documentação é tão importante quanto a criação do software em si (SANCHES, 2001a). Não é por menos que é um item exigido e extremamente necessário quando se deseja alcançar qualquer padrão de qualidade, tal como tratam a ISO 9001 (ISO, 2000), a ISO/IEC 12207 (NBR ISO, 1998), CMM (*Capability Maturity Model*) (FIORINI et al., 1998) e CMMI (*Capability Maturity Model Integrated*) (AHERN et al., 2004), dentre outros.

Há ainda a necessidade de se definir um procedimento para controlar a documentação de uma organização, através de atividades de análise, aprovação ou reprovação, identificação de alterações, situação da revisão atual, disponibilidade das versões pertinentes de documentos aplicáveis, dentre outras (ISO, 2000).

Entretanto, a documentação tem sido o calcanhar de Aquiles de muitas organizações de Tecnologia da Informação, que a tem deixado de lado, seja por falta de uma política organizacional, por prazos curtos ou inexistência de uma ferramenta específica para essa atividade.

A seguir é brevemente apresentada a ferramenta de apoio à documentação de ODE, XMLDoc (SILVA, 2004), mostrando como a documentação é tratada no contexto desse ambiente e a tecnologia utilizada para tal.

2.3.1 Apoio à Documentação em ODE

Com o intuito de fornecer um meio único e padronizado de documentação em ODE, foi desenvolvida a ferramenta XMLDoc (SOARES, 2002; SILVA, 2004).

A primeira versão de XMLDoc foi desenvolvida por (SOARES, 2002), com o intuito de fornecer um meio padronizado de documentação para ODE, sendo seus objetivos:

- Fazer com que os desenvolvedores tenham acesso a documentos atualizados e consistentes;
- Utilizar modelos para estabelecer como os documentos devem ser elaborados, de forma a auxiliar os desenvolvedores na realização dessa atividade e assegurar que questões importantes sejam sempre tratadas;
- Assegurar a consistência entre documentos e diagramas.

A tecnologia escolhida para representar os documentos de ODE foi XML (*eXtensible Markup Language* - Linguagem de Marcação Extensível), por ser uma linguagem poderosa e elegante para se pensar em dados, trocá-los e apresentá-los de forma independente de plataforma. XML permite representar metadados, isto é, informações sobre um conjunto de dados (MARTIN et al., 2001), o que está bastante em linha com a filosofia de anotações ontológicas de ODE. Dentre as vantagens na sua utilização, merecem destaque: a facilidade que oferece para resolver diversos problemas de integração entre ferramentas; a facilidade de leitura por humanos e máquinas; a facilidade de processamento dos documentos por algumas ferramentas, como as de busca por um conteúdo específico e a portabilidade.

XMLDoc faz uso das facilidades de definição da linguagem XML, o que permite a criação de modelos de documentos que asseguram uma apresentação padronizada e a presença de estruturas obrigatórias no documento baseado naquele modelo (SOARES, 2002).

Como XML preza que conteúdo e forma de apresentação devem estar relacionados, mas não presos um ao outro, XMLDoc utiliza folhas de estilo para a apresentação. Uma folha de estilo é um documento em que o desenvolvedor especifica os estilos de formatação e as normas que determinam quando certo estilo deve ser aplicado.

O conteúdo, por sua vez, é obtido diretamente da base de dados de ODE que, por ser uma base única, é a mesma base utilizada por todas as suas ferramentas, onde estão armazenados os mais diversos artefatos, tais como documentos, modelos, códigos fonte, dentre outros.

Em um trabalho posterior, SILVA (2004) adaptou XMLDoc aos novos componentes e padrões de ODE, em especial aos conceitos ontológicos relacionados a artefatos. Assim, a criação de artefatos, tanto na base de dados de ODE como em XML, passou a obedecer à ontologia de artefato, definida como parte deste trabalho (NUNES et al., 2004) e apresentada no próximo capítulo.

Além da adequação de XMLDoc às ontologias de artefato, SILVA (2004) separou a persistência de artefatos em XML, que era feita internamente à ferramenta, uma vez que ela é necessária a outras ferramentas de ODE. Assim, foram disponibilizadas, na classe Artefato, funcionalidades para gerar arquivos em XML a partir de artefatos armazenados na base de dados de ODE e para a situação oposta, ou seja, para gerar artefatos na base de dados de ODE, a partir de arquivos XML.

As transações disponíveis em XMLDoc permitem definir modelos de documento, criar documentos de acordo com os modelos previamente definidos, verificar a consistência dos documentos e gerá-los em XML. A figura 2.5 mostra um documento de Plano de Riscos, exibido na ferramenta XMLDoc usando o formato HTML.

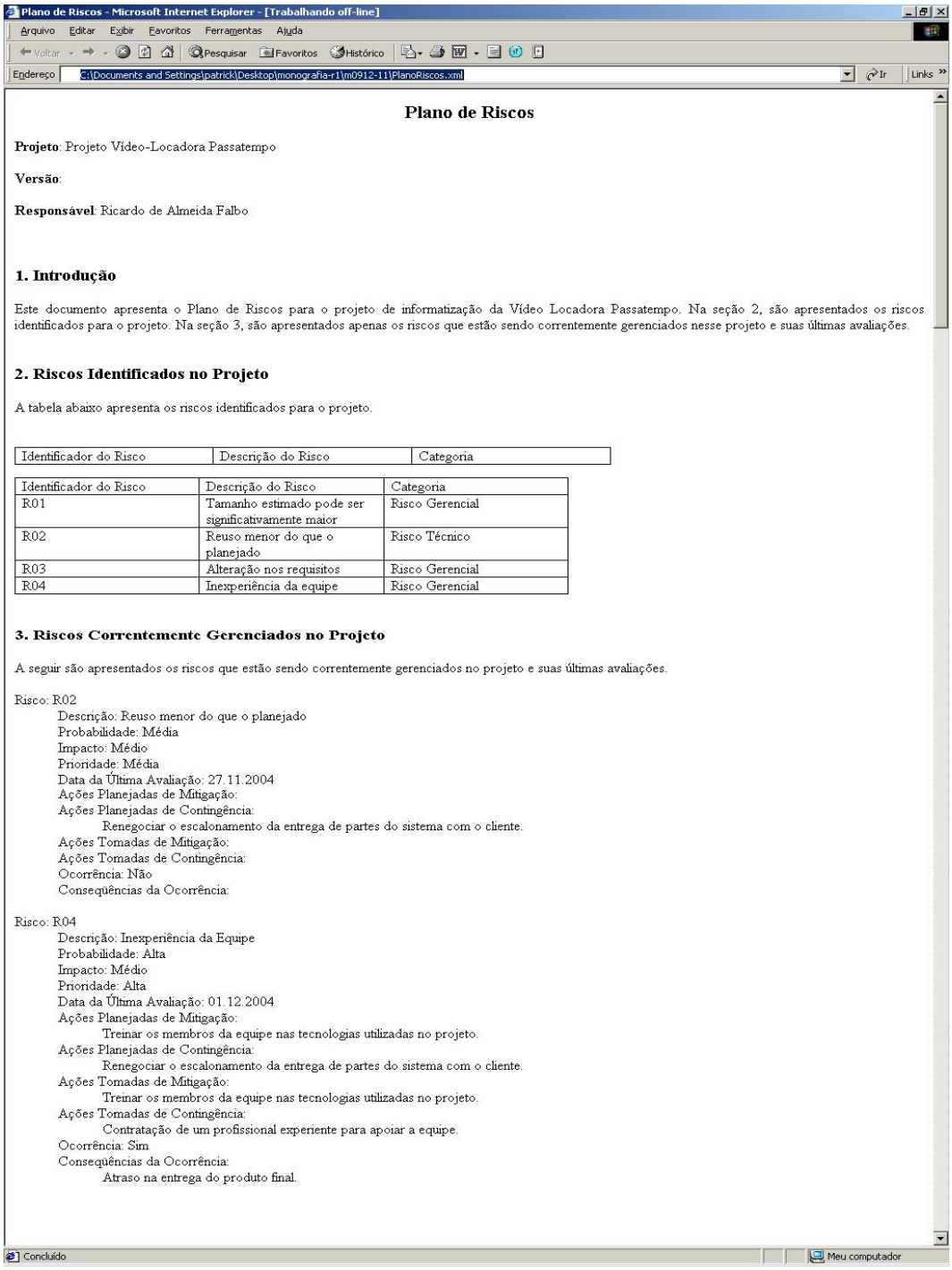


Figura 2.5 – Visualização de documento em XMLDoc.

2.4 Gerência de Configuração de Software

Durante o processo de desenvolvimento de software, vários artefatos são produzidos e alterados constantemente, evoluindo até que seus propósitos fundamentais sejam atendidos (CAETANO, 2004). Ferramentas de software, tais como compiladores e editores de texto,

também podem ser substituídas por versões mais recentes de seus fabricantes ou mesmo por outras.

Porém, caso essas mudanças não sejam devidamente documentadas e comunicadas, poderão acarretar diversos problemas, tais como: dois ou mais desenvolvedores estarem alterando um mesmo artefato ao mesmo tempo, não se saber qual a versão mais atual de um artefato, não refletir alterações nos artefatos impactados por um artefato em alteração, dentre outros. Esses problemas podem gerar vários transtornos como: incompatibilidade entre os grupos de desenvolvimento, inconsistências, retrabalho, atraso na entrega e insatisfação do cliente.

Assim, para que esses transtornos sejam evitados, é de suma importância o acompanhamento e controle de artefatos e ferramentas, através de um processo de gerência de configuração de software, durante todo o ciclo de vida do software (SANCHES, 2001b).

A Gerência de Configuração de Software (GCS) é uma disciplina para controlar a evolução dos sistemas de software (ESTUBLIER, 2000; DART, 1991). Deve identificar e documentar os produtos de trabalho que podem ser modificados, através de características físicas e funcionais, estabelecer as relações entre eles e os mecanismos para administrar suas diferentes versões ou variantes, controlar as modificações, além de fazer auditoria e preparar relatórios sobre tais modificações (PRESSMAN, 2001; SWEBOK, 2001).

A GCS tem, então, a finalidade de estabelecer e manter a integridade dos itens de software ao longo de todo o ciclo de vida (FIORINI et al., 1998; ISO/IEC, 1998), garantir a completeza, a consistência e a correção de tais itens, e controlar o armazenamento, a manipulação e a distribuição dos mesmos (NBR ISO/IEC, 1998). Objetiva a maximização da produtividade pela minimização de erros (PRESSMAN, 2001).

A GCS está diretamente relacionada com as atividades de Garantia da Qualidade de Software (*Software Quality Assurance - SQA*), cujo objetivo principal é monitorar o software e seu processo de desenvolvimento, assegurando a utilização de padrões e procedimentos (SWEBOK, 2001). Todos os modelos de referência de qualidade reconhecidos internacionalmente abordam a GCS. O CMM (*Capability Maturity Model*) (FIORINI et al., 1998), por exemplo, determina como cada uma das atividades principais deve ser executada para que haja qualidade.

Em (PRESSMAN, 2001), (SWEBOK, 2001), (DART, 1991), (FIORINI et al., 1998), (ISO/IEC, 1998) e (NBR ISO/IEC, 1998), são citadas as atividades da GCS, que podem ser resumidas em:

1) Implementação do Processo de GCS – Um plano deve ser desenvolvido descrevendo: as atividades da gerência de configuração, procedimentos e cronogramas para executar as atividades, as organizações responsáveis pela execução dessas atividades e seu relacionamento com outras organizações.
2) Identificação da Configuração - Identificação dos itens de software e suas versões ou variantes a serem controladas, estabelecendo uma linha básica.
3) Controle de Versão – Combina procedimentos e ferramentas para administrar diferentes versões dos itens de configuração criados durante o processo de software.
4) Controle de Modificação – Combina procedimentos humanos e ferramentas automatizadas para controlar as alterações feitas em itens de software, definindo um processo para tratar alterações que envolve: solicitação de mudança, aprovação ou rejeição da solicitação, registro de retirada para alteração (*check-out*), análise, avaliação e realização das alterações e registro da realização das alterações (*check-in*).
5) Auditoria de Configuração – Avalia um item de configuração quanto a características não consideradas nas revisões técnicas formais, tais como se os padrões de Engenharia de Software foram utilizados ou se os itens relacionados aos solicitados foram devidamente atualizados. Valida a completeza de um produto e mantém consistência entre os componentes, assegurando que o produto é uma coleção bem definida de componentes.
6) Relato da situação da configuração – Devem ser preparados registros de gerenciamento e relatórios que mostrem a situação e o histórico dos itens de software controlados. Tais relatórios devem incluir, dentre outras coisas, o número de alterações em um projeto, as últimas versões do item de software, identificadores de liberação e as comparações entre elas.

O primeiro passo do processo de gerência de configuração de software é a confecção de um plano de gerência de configuração, que inicia com a identificação dos itens que serão colocados sob gerência de configuração, chamados itens de configuração. Os itens mais relevantes para serem submetidos à gerência de configuração são aqueles mais usados durante o ciclo de vida, os mais genéricos, os mais importantes para segurança, os projetados para reutilização e os que podem ser modificados por vários desenvolvedores ao mesmo tempo

(SANCHES, 2001b). Os itens não colocados sob gerência de configuração podem ser alterados livremente.

Após a seleção dos itens, deve-se descrever como eles se relacionam. Isso será muito importante para as futuras manutenções, pois permite identificar de maneira eficaz os itens afetados em decorrência de uma alteração (SWEBOK, 2001; SANCHES, 2001b).

A seguir, deve-se criar um esquema de identificação dos itens de configuração, com atribuição de nomes exclusivos, para que seja possível estabelecer a evolução de cada versão ou variante dos itens[2] (PRESSMAN, 2001; SANCHES, 2001b).

Após a identificação dos itens de configuração, devem ser planejadas as linhas-base dentro do ciclo de vida do projeto. Uma linha-base (ou *baseline*) é uma versão estável de um sistema contendo todos os componentes que constituem este sistema em um determinado momento (FIORINI et al., 1998). Nos pontos estabelecidos pelas linhas-base, os itens de configuração devem ser identificados, analisados, corrigidos, aprovados e armazenados em um local sob controle de acesso, denominado repositório central, base de dados de projeto ou biblioteca de projeto. Assim, quaisquer alterações nos itens daí em diante só poderão ser realizadas através de procedimentos formais de controle de modificação (SANCHES, 2001b; PRESSMAN, 2001).

O passo seguinte do processo de GCS é o controle de versão, que combina procedimentos e ferramentas para identificar, armazenar e administrar diferentes variações dos itens de configuração que são criados durante o processo de software (PRESSMAN, 2001; SANCHES, 2001b). A idéia é que a cada modificação que ocorra em um item de configuração, uma nova versão ou variante seja criada (ESTUBLIER, 2000). Versões de um item são as revisões geradas pelas diversas alterações, enquanto variantes são as diferentes formas de um item, que existem simultaneamente, e atendem a requisitos similares (SANCHES, 2001b).

Outra, e talvez a mais importante, atividade do processo de GCS é o controle de modificação, alteração ou mudanças, que combina procedimentos humanos e ferramentas automatizadas para o controle das alterações realizadas nos itens de configuração (PRESSMAN, 2001).

Assim que uma alteração é solicitada, o impacto em outros itens de configuração e o custo para a modificação devem ser avaliados (SWEBOK, 2001). Um responsável deve decidir se a alteração poderá ou não ser realizada. Caso a alteração seja liberada, pessoas são

[2] Versões e variantes de itens de configuração serão tratados genericamente neste texto como variações de itens de configuração.

indicadas para sua execução. Assim que não houver ninguém utilizando os itens de configuração envolvidos, cópias deles são retiradas do repositório central e colocadas em uma área de trabalho do desenvolvedor, através de um procedimento denominado *check-out*. A partir deste momento, nenhum outro desenvolvedor poderá alterar esses itens. Os desenvolvedores designados fazem as alterações necessárias e, assim que essas forem concluídas, os itens são submetidos a uma revisão. Se as alterações forem aprovadas, os itens são devolvidos ao repositório central, estabelecendo uma nova linha-base, em um procedimento chamado *check-in* (SANCHES, 2001b; PRESSMAN, 2001).

Porém, mesmo com os mecanismos de controle mais bem sucedidos, não é possível garantir que as modificações foram corretamente implementadas. Assim, revisões técnicas formais e auditorias de configuração de software são necessárias no processo de GCS (PRESSMAN, 2001). Essas verificações tentam descobrir omissões ou erros na configuração e se os procedimentos, padrões, regulamentações ou guias foram devidamente aplicados no processo e no produto (SANCHES, 2001b; SWEBOK, 2001).

Enfim, o último passo do processo de GCS é a preparação de relatórios, que é uma tarefa que tem como objetivo relatar a todas as pessoas envolvidas no desenvolvimento e manutenção do software as seguintes questões: (i) O que aconteceu? (ii) Quem fez? (iii) Quando aconteceu? (iv) O que mais será afetado? O acesso rápido às informações agiliza o processo de desenvolvimento e melhora a comunicação entre as pessoas, evitando, assim, muitos problemas de alterações do mesmo item de configuração, com intenções diferentes e, às vezes, conflitantes (SANCHES, 2001b).

2.4.1 Sistemas de Gerência de Configuração de Software

Uma vez identificadas as boas práticas e definido um processo de GCS, o uso de uma ferramenta auxilia em muito a realização desse processo. Porém, não há um conceito universal do que constitui um sistema de GCS.

Idealmente, um sistema de GCS deve prover funcionalidades para apoiar todas as atividades do processo de GCS, citadas na seção anterior, mas, de forma mais prática, qualquer sistema que provê alguma forma de controle de versão, identificação da configuração e tem o intuito de prover gerência de configuração (em algum grau) é considerado pela comunidade de engenharia de software como um sistema de GCS. Pode-se notar que os sistemas de GCS existentes provêem suas próprias combinações de funcionalidades, ao invés de um conjunto padrão (DART, 1991).

Nos anos 80, foram construídos os primeiros sistemas de GCS, cujo controle era através de arquivos. Podem ser citados: DSEE (*Domain Software Engeneering Environment*), que introduziu o conceito de modelo de sistema, antecessor da linguagem de descrição de arquitetura; NSE (*Network Software Environment*), que introduziu o controle de espaço de trabalho e trabalho cooperativo; Adele, que introduziu um modelo de produto especializado, com a construção de configuração automática e ADC (*Aides de Camp*), que introduziu o conjunto de mudanças (ESTUBLIER, 2000).

Os primeiros produtos reais de GCS surgiram nos anos 90, muito melhores do que os anteriores. Muitos usam bancos de dados relacionais, mas continuam baseados em controle de arquivos. Eles provêem suporte a espaço de trabalho e ao processo. Essa geração inclui o ClearCase (sucessor do DSEE), que introduziu o sistema de arquivo virtual e Continuus, que introduziu com Adele o suporte explícito ao processo (ESTUBLIER, 2000).

Na segunda metade dos anos 90, suporte ao processo foi adicionado e a maioria dos produtos ganhou mais maturidade. Esse período foi a consagração da GCS como uma tecnologia madura, confiável e essencial para o sucesso do desenvolvimento de software. Algumas pessoas consideram a GCS como um dos poucos sucessos da Engenharia de Software (ESTUBLIER, 2000).

Dentre as várias ferramentas de GCS atuais, duas merecem destaque devido a sua vasta utilização: a ferramenta CVS (*Concurrent Version System*) e a ferramenta Rational ClearCase. A seguir, essas ferramentas são brevemente apresentadas a título de ilustração. Outros sistemas de GCS existentes podem ser encontrados em (DART, 1991), que evidenciou os aspectos mais marcantes de cada um. Foram objeto de estudo, as ferramentas: Adele, da Universidade de Grenoble; ADC (*Aides de Camp*), da *Software Maintenance and Development Systems Inc.;* CCC (*Change and Configuration Control*), da Softool; CMA (*Configuration Management Assistant*), dos laboratórios Tartan; DMS (*Design Management System*), da *Sherpa Corporation;* DSEE (*Domain Software Engeneering Environment*), da Apollo; ISTAR, da *Imperial Software Technology Ltd*; Jasmine, da *Xerox information Systems Division*; LIFESPAN, da *Yard Software Systems;* NSE (*Network Software Environment*), da *Sun Microsystems*; PowerFrame, da *EDA Systems Inc.*; Rational, da própria; RCS (*Revision Control System*), de W. Tichy; Shape, da Universidade de Berlin e SMS (*Software Management System*), da Biin.

CVS (*Concurrent Version System*)

O CVS é uma ferramenta de código aberto (*open source*) e multiplataforma que implementa as principais funções da gerência de configuração de software e é amplamente utilizada por diversos projetos em todo o mundo. Assim como várias outras ferramentas de GCS, utiliza os conceitos de Repositório e Área de Trabalho (CAETANO, 2004).

O Repositório nada mais é que um diretório que tem por objetivo abrigar todos os arquivos de um projeto sob o controle de versões. Tais arquivos são armazenados na forma de Delta – uma versão completa do arquivo e suas modificações, para minimizar o uso de espaço em disco. O repositório pode estar situado local ou remotamente e precisa ser inicializado antes de sua utilização. A área de trabalho, por sua vez, é o espaço, ou um diretório, que o usuário utilizará para fazer suas alterações, criar novos arquivos ou excluir arquivos existentes.

O funcionamento do CVS pode ser resumido pela figura 2.6

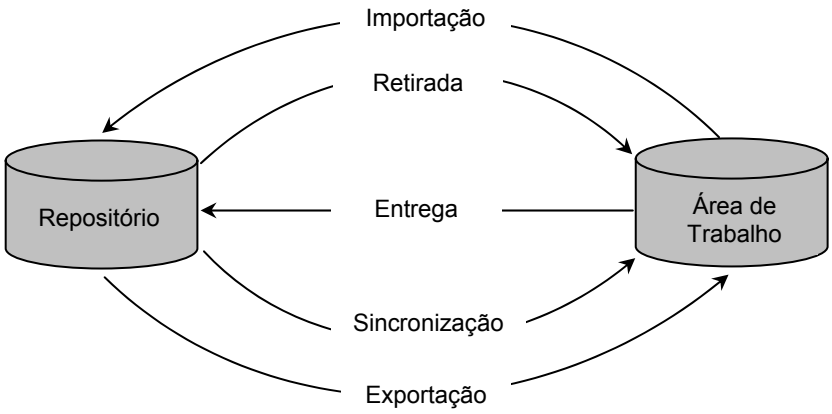


Figura 2.6 – Principais operações realizadas pelo CVS (CAETANO, 2004).

No CVS, para iniciar o controle de versões, o primeiro passo é a **importação** ("*import*"), em que os arquivos de um projeto são transferidos da área de trabalho para o repositório.

Uma vez que os arquivos de um projeto tenham sido importados com sucesso, é possível iniciar efetivamente as operações de controle de versão. Para tal, deve-se retirar os arquivos desejados do repositório e transferi-los para a área de trabalho ("c*heckout*"), para que as alterações possam ser realizadas. Podem ser retirados arquivos em sua versão mais atual ou em versões anteriores. É importante frisar que sempre que arquivos forem criados ou excluídos, tal fato deve ser informado ao CVS, através dos comandos "*add*" e "*remove*",

respectivamente, uma vez que tais informações não são reconhecidas automaticamente por ele (CAETANO, 2004).

Após a conclusão das modificações, torna-se necessário que elas sejam submetidas ao repositório, para que estejam disponíveis a todos. Nesse processo de entrega ("*commit*"), o CVS examina todos os arquivos em todos os subdiretórios, em busca de alterações. Os arquivos que não sofreram alteração não são transferidos ao repositório. Para cada arquivo criado é atribuído um número de revisão, enquanto que para os arquivos alterados, as respectivas revisões são incrementadas.

Após as modificações terem sido submetidas ao repositório, a área de trabalho deve ser liberada ("*release*") para que os arquivos mantidos nela sejam eliminados e para que o CVS seja informado de que não se irá trabalhar mais naquele projeto. A liberação da área de trabalho pode ainda ocorrer antes da entrega, quando se deseja descartar todas as alterações realizadas nos arquivos do projeto.

Como atualmente há uma grande tendência na utilização de equipes dispersas geograficamente, o controle de versões é visto como uma extensão natural do processo de desenvolvimento colaborativo. Para controlar tal processo, o CVS dispõe de três métodos (CAETANO, 2004):

- Bloqueio – torna a operação de entrega de um arquivo restrita ao usuário que realizou o bloqueio ("*admin –l*"), impedindo que outros usuários submetam modificações deste mesmo arquivo ao repositório, enquanto ele estiver em alteração.
- Bloqueio fraco – usado para observar ou vigiar um arquivo ("*watch*"). Assim, o usuário que estiver vigiando o arquivo será avisado sempre que tal arquivo sofrer alguma alteração. Quando algum outro usuário retirar o arquivo, ele virá como somente para leitura, para que ele saiba que o arquivo não deve ser alterado. Porém, ainda assim, caso ele deseje alterar o arquivo, poderá fazê-lo ("*edit*").
- Sincronização – é o método mais utilizado por permitir que alterações paralelas sejam feitas em um mesmo arquivo. É realizada por meio de uma operação de atualização ("*update*"), que verifica se alguma outra atualização foi submetida ao repositório. Neste caso, o CVS indica a necessidade de sincronização ("*needs merge*"), porém, cabe ao usuário examinar as diferenças entre as versões ("*diff*") e solucionar os possíveis conflitos.

Frequentemente, os arquivos de um projeto devem ser entregues ao cliente ou disponibilizados para uma comunidade de desenvolvedores. Porém, é natural que eles não utilizem o CVS ou nenhuma outra ferramenta de suporte ao desenvolvimento colaborativo. Em tais circunstâncias, os arquivos devem ser entregues em seu formato nativo, ou seja, sem os recursos ou arquivos utilizados pelo CVS durante a realização do controle de versões, como por exemplo, os subdiretórios administrativos ou as palavras-chave. Para solucionar esse problema, o CVS oferece um mecanismo chamado exportação ("*export*"), que retira os arquivos de um ponto do tempo determinado, ignorando alguns de seus recursos característicos (CAETANO, 2004).

O CVS oferece, ainda, algumas outras funcionalidades, tais como funções para examinar o estado dos arquivos da área de trabalho em relação ao repositório ("*status*"), para rastrear o histórico de modificações realizadas em um arquivo ("*log*") e para examinar linha a linha tais modificações ("*annotate*").

Apesar de sua vasta utilização, o CVS também possui suas limitações, dentre as quais, quatro merecem destaque:

- O CVS foi concebido para lidar com arquivos texto, em virtude de suas operações alterarem o conteúdo dos arquivos. Mesmo assim, é possível controlar versões de arquivos binários, porém, nem todas as funcionalidades podem ser utilizadas.
- Não é possível estabelecer relações de composição ou dependência entre os arquivos. Esses são tratados como itens isolados. Com isso, dentre outras coisas, não é possível identificar quais arquivos impactados pela alteração em um determinado arquivo ou se certo diagrama faz parte da composição de um documento.
- Não é possível identificar a que tipo de artefato corresponde o arquivo, ou seja, se é um documento, um código fonte, um diagrama etc.
- O fato do CVS ser uma ferramenta de linha de comando faz com que ele não possua uma interface muito amigável. Em virtude desse problema, hoje há disponíveis diversos *front-ends* gráficos, a fim de tornar o uso do CVS mais fácil e intuitivo (CAETANO, 2004). Em algumas dessas ferramentas, tais como o TortoiseCVS, o WinCVS, o JCVS, o SandWeb, o Cervisia e o MacCVS, é possível utilizar várias funcionalidades do CVS, tais como "*chekout*" e "*commit*". Outras, porém, são ferramentas auxiliares, tais como o CvsPlot, o StatCvs e o CVS Monitor, que geram gráficos estatísticos; o Bonsai, que é uma ferramenta *web* para

consultas aos projetos do repositório; o Cvs2html e o Cvs2cl, que têm por objetivo exportar os dados gerados nos *logs* para HTML e GNU-style, respectivamente; o WinMerge e o CSDiff, que comparam as modificações entre dois arquivos; o Palantír, que disponibiliza funcionalidades que permitem que os desenvolvedores tenham consciência de tudo que está em desenvolvimento sem se ater unicamente ao que se está fazendo (SARMA, et al., 2003) e o CVSNT, que é um Password Server para Windows.

Rational ClearCase

ClearCase é a ferramenta de GCS da IBM - Rational, líder de mercado em soluções de gerência de configuração. Ela oferece uma família de produtos cuja utilização varia desde pequenos grupos de trabalhos até empresas globalmente distribuídas, possibilitando o desenvolvimento paralelo. Seu processo permite que se inicie a utilização da ferramenta e que este seja adaptado à medida que o projeto cresce.

Usando ClearCase é possível controlar requisitos, modelos, códigos-fonte, documentação e scripts de teste. Ele provê controle de versão, desenvolvimento paralelo, gerência do espaço de trabalho, processo configurável, gerência do produto, auditoria e ainda uma interface *web* (WAHLI et al., 2004).

O ClearCase chama de "elementos" quaisquer itens que possam ser controlados pelo processo de GCS. Esses elementos podem ser arquivos armazenados em um sistema de arquivos, como, por exemplo, documentos e códigos-fonte, ou diretórios, que podem conter outros elementos.

As alterações realizadas sobre um elemento geram para ele seqüências de versões, chamadas de *branches*. Cada elemento possui um *branch* principal, que representa a linha principal de desenvolvimento, e *subbranches*, que representam outras linhas separadas de desenvolvimento, como mostra a figura 2.7. Em geral, o *branch* principal é reservado para os marcos de referência mais significativos e para versões de entrega, enquanto os demais *branches* são utilizados para o trabalho de desenvolvimento (WAHLI et al., 2004).

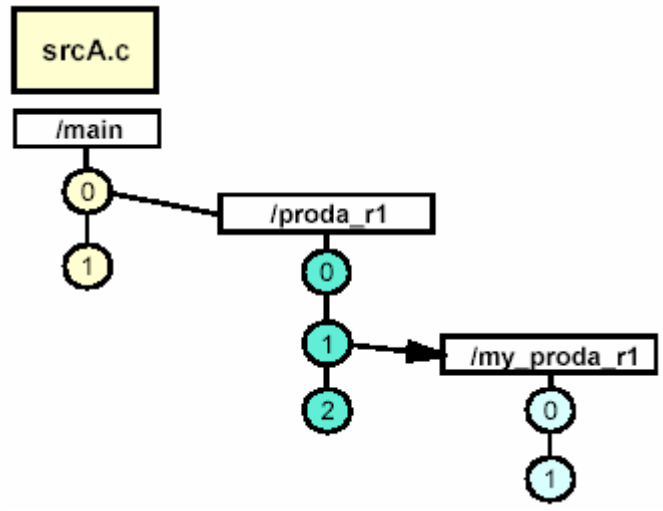


Figura 2.7 – Branches do elemento srcA.c.

Como os números das versões podem não ser muito sugestivos, um rótulo (*label*) pode ser anexado a cada versão de um elemento para identificá-lo de uma maneira mais fácil de ser lembrado. Em geral, *labels* são aplicados a um conjunto de elementos para marcar importantes marcos de referência do projeto ou o ponto de início de um *branch*, como mostra a figura 2.8 (WAHLI et al., 2004).

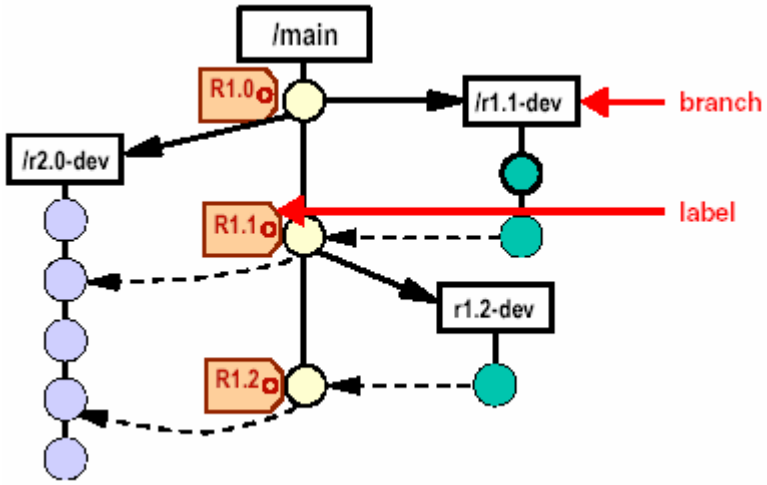


Figura 2.8 – Labels de versões.

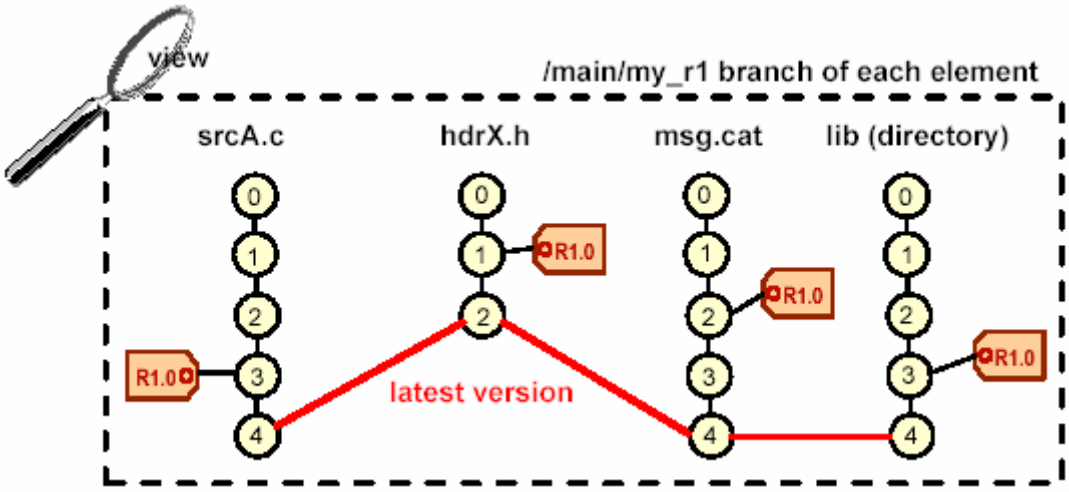
Todas as versões de todos os elementos de um projeto, assim como seus meta-dados, são armazenadas em repositórios chamados VOB ("*Versioned Object Base*").

Para realizar alterações sobre os elementos armazenados em uma VOB, utilizam-se visões (*views*), que são representadas como diretórios e provêem acesso a versões específicas de um ou mais elementos da VOB. Uma *view* provê um espaço de trabalho no qual um

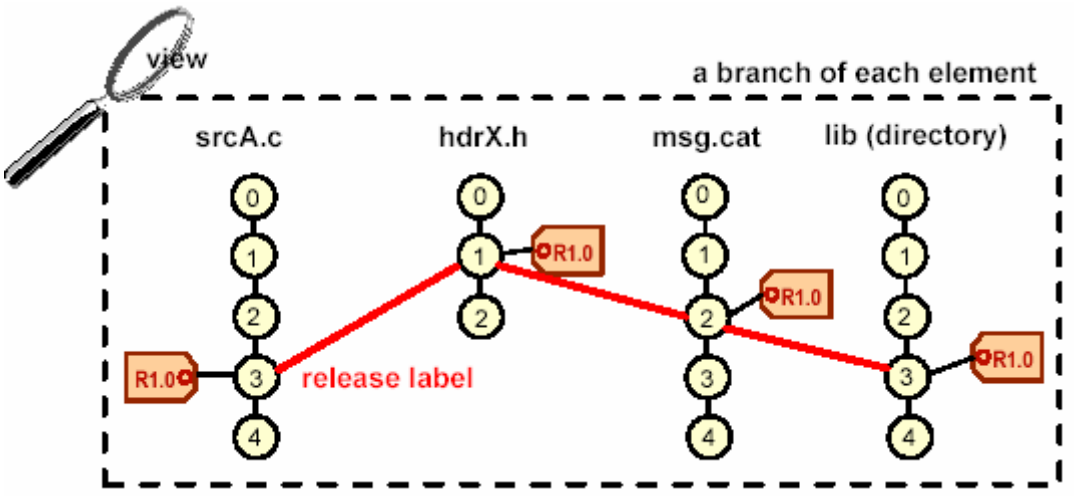
desenvolvedor pode trabalhar isolado de outros desenvolvedores. Assim, ninguém verá as alterações que estão sendo feitas por esse desenvolvedor (WAHLI et al., 2004).

O ClearCase oferece dois conceitos de *view*: "*snapshot view*" e "*dynamic view*". A *snapshot view* é um espaço de trabalho criado em um computador local, onde são copiadas versões dos elementos das VOBs. Como essas *views* ficam isoladas das VOBs, é necessário atualizá-las periodicamente para que as últimas versões dos elementos sejam copiadas para a *snapshot view*. A *dynamic view*, por outro lado, provê acesso imediato aos dados armazenados nas VOBs, baseado em regras de seleção. Neste caso, não é necessário copiar os dados das VOBs para a *view* e sempre é possível ver as versões mais atuais dos elementos. Isso significa que se deve ter ciência do impacto que o trabalho pode sofrer devido às mudanças feitas em outros elementos (WAHLI et al., 2004).

As regras utilizadas por uma *view* para selecionar versões dos elementos das VOBs são chamadas de "*configuration specification*" ou "*config spec*". Por exemplo, a *config spec* da *view* da figura 2.9(a) contém regras para selecionar as últimas versões do *branch* de desenvolvimento my_r1, enquanto a *config spec* da figura 2.9(b) contém regras para selecionar as versões com um *label* específico – R1.0 (WAHLI et al., 2004).



(a) Selecionando as últimas versões



(b) Selecionando as versões com o label R1.0.

Figura 2.9 – Configuration specification.

Porém, as alterações nos elementos de uma *view* não podem ser feitas deliberadamente. Antes disso, deve-se passar por um processo formal de retirada para alteração ("*checkout*"), que permite obter uma cópia privada e editável de um elemento específico, conforme o exemplo mostrado na figura 2.10. Quando as alterações são finalizadas, uma nova versão do elemento é adicionada à VOB, através do processo de entrega ("*checkin*").

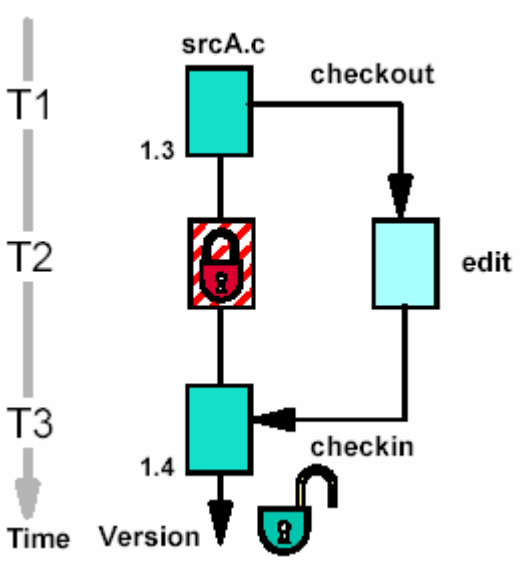


Figura 2.10 – Processo de alteração da versão 1.3 do elemento srcA.c.

Muitas ferramentas de controle de versão bloqueiam os elementos após ser realizado o *checkout,* conforme figura 2.10. Assim, nenhum outro usuário pode alterar um elemento que esteja em alteração. Porém, para um desenvolvimento concorrente mais efetivo, o ClearCase

permite modificações paralelas em um mesmo elemento através de dois tipos de *checkout*: "*non-optimistic checkout"* e "*optimistic checkout"* (WAHLI et al., 2004).

No *non-optimistic checkout*, a ordem do *checkout* determina a ordem do *checkin*. No exemplo da figura 2.11, o desenvolvedor Adam fez o *checkout* na versão 1.3 do elemento libA.c e logo após, o desenvolvedor Joe fez o mesmo. Após Adam concluir suas alterações e realizar o *checkin*, é gerada a versão 1.4 do elemento em questão. Agora que Adam já fez o checkin, Joe está liberado para fazer o seu. Porém, como foram feitos dois *checkouts* da mesma versão de um elemento, a ferramenta detecta uma situação de colisão e força Joe a executar uma reconciliação (*merge*) da sua versão com a versão 1.4 gerada por Adam (WAHLI et al., 2004).

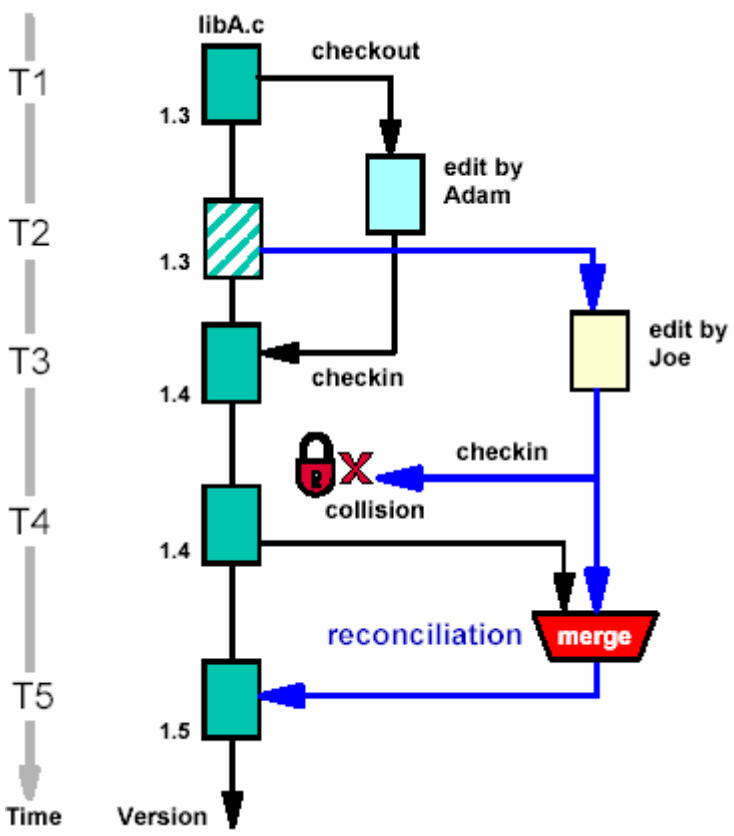


Figura 2.11 – Processo de alterações paralelas da versão 1.3 do elemento libA.c.

No *optimistic checkout*, não há restrição quanto à ordem do *checkout*. Assim, quem chegar primeiro faz o *checkin*. Desta forma, no exemplo anterior, Joe poderia ter feito o *checkin* antes de Adam, apesar de Adam ter feito o *checkout* primeiro. De qualquer forma, assim que houvesse o segundo *checkin*, a colisão seria detectada e uma reconciliação seria necessária (WAHLI et al., 2004).

O ClearCase possui ainda duas outras variações: o ClearCase LT e o ClearCase MultSite. O ClearCase LT é uma versão mais simples, que roda em um servidor único e

suporta funções básicas de GCS. O ClearCase MultSite permite o desenvolvimento paralelo através de equipes de desenvolvimento distribuídas geograficamente (WAHLI et al., 2004).

De forma geral, o ClearCase cobre controle de versão, controle de configuração e parte das áreas de gerência de processos do domínio da GCS. Porém, ele pode ser integrado a duas outras tecnologias da Rational, a fim de aumentar o seu espectro de atuação: o UCM e o ClearQuest.

UCM (*Unified Change Management*) define as melhores práticas de processo para o gerenciamento de mudanças nos requisitos, baseado em atividades. Estrutura o esforço da equipe de desenvolvimento em um processo definido e repetível. Associado ao ClearCase, aumenta o nível de abstração para gerenciar mudanças em termos de atividades ao invés do rastreamento de arquivos individuais. Cada atividade é associada ao seu conjunto de mudanças, que encapsula todas as versões dos artefatos utilizados em um projeto para implementar tal atividade. Isso possibilita que se identifiquem facilmente as atividades incluídas em cada produto de software e em cada linha base (WAHLI et al., 2004).

ClearQuest é uma ferramenta que cobre a gerência de processos e a busca de problemas. Utilizada junto com o ClearCase e o UCM, é uma solução bastante completa de GCS. Provê mudanças baseadas em atividades, busca de defeitos e pode gerenciar todos os tipos de requisição de mudanças, com um processo de trabalho flexível, que pode ser adaptado às necessidades específicas de cada organização e às várias fases do processo de desenvolvimento (WAHLI et al., 2004).

Apesar de suas inúmeras vantagens, principalmente se aplicado em conjunto com outras ferramentas da Rational, O ClearCase possui a grande desvantagem de não ser uma ferramenta livre, possuindo um custo elevado. No caso da utilização em conjunto com outras ferramentas, como o ClearQuest, o custo fica ainda mais elevado. Outra desvantagem vista na ferramenta diz respeito à alteração de versões passadas de artefatos. Neste caso, só é possível determinar quais as versões dos artefatos em determinado ponto do tempo se, naquele momento, lhes foi atribuído um *label*. Caso um *label* não tenha sido criado, ainda assim é possível alterar um artefato de versão antiga, porém, não se saberá a quais outros artefatos ele correspondia. Por último, assim como várias outras ferramentas, ele trata os artefatos, ou elementos, como arquivos, não estabelecendo relações entre as partes que os compõem.

GConf (TABA)

Com o intuito de apoiar a abordagem para a gerência de configuração fundamentada no conceito de ADSOrgs, foi definida a ferramenta *GConf,* que está presente nos ADSOrg instanciados pela Estação TABA (FIGUEIREDO, 2005). Essa abordagem é baseada nas abordagens de gerência de configuração propostas pela ISO/IEC 12207 (NBR ISO, 1998), pelo SWEBOK (SWEBOK, 2001) e suporta as atividades do nível de maturidade 2 da GCS do CMMI (AHERN et al., 2004).

GConf apóia todas as atividades do processo de gerência de configuração: identificação da configuração, controle da configuração, relato da situação, auditoria da configuração e gerência de liberação e entrega.

Esta ferramenta possui três módulos de execução de acordo com o usuário que a está executando, uma vez que algumas atividades só podem ser executadas por determinados usuários. As atividades de cada módulo são (FIGUEIREDO, 2005):

- ***Módulo do Gerente do Projeto***: Identificar Responsáveis, Visualizar Plano de GCS, Identificar Itens de Configuração, Identificar Responsáveis, Analisar Pedido de Alteração, Verificar Item de Software Alterado, Relatar Situação da Configuração, Auditoria da Configuração, Criar *Baseline*, Entregar *Baseline*;
- ***Módulo dos Responsáveis pelo Item de Configuração***: Analisar Pedido de Alteração, Verificar Item de Software Alterado, Relatar Situação da Configuração;
- ***Módulo da Equipe do Projeto***: Solicitar Alteração, Implementar Alteração, Relatar

Sua interface básica é exibida na figura 2.12. No lado esquerdo pode-se visualizar o processo de GCS que guia a ferramenta e no lado direito identifica-se a atividade que está sendo realizada pelo usuário (FIGUEIREDO, 2005).

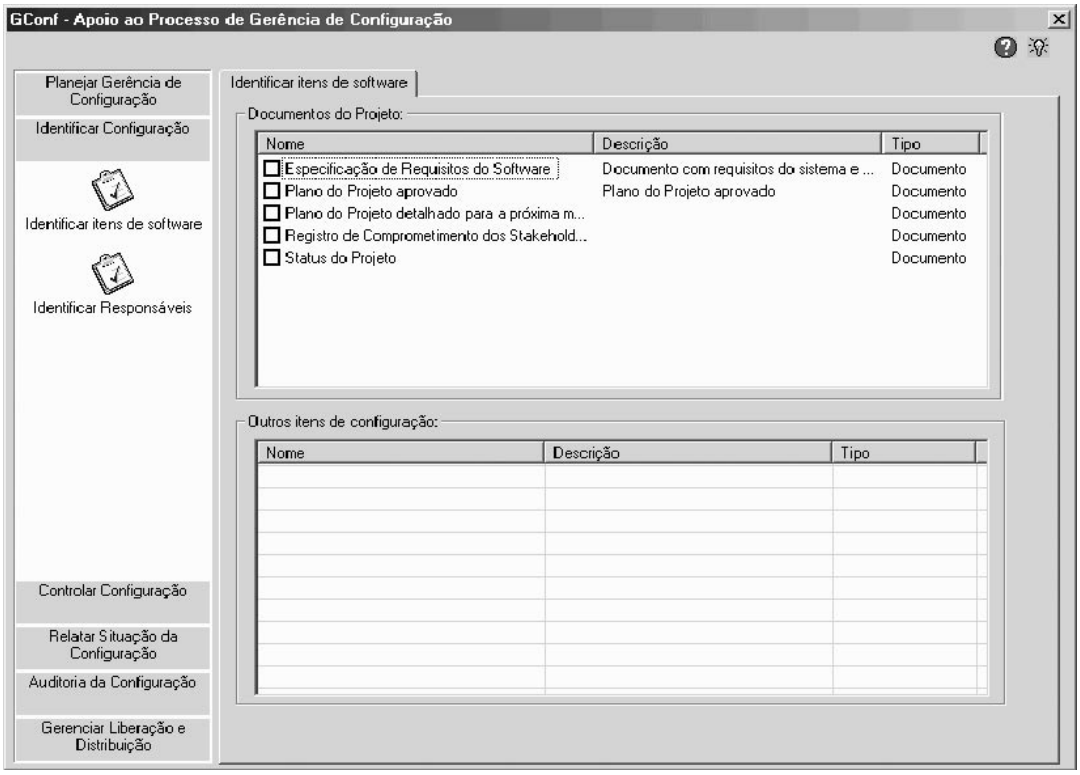


Figura 2.12 – Interface da ferramenta GConf.

Nesta tela é ilustrada a atividade *Identificar Itens de Configuraçã*o, na qual o gerente do projeto identifica os itens de informação que ficarão sob controle da gerência de configuração. Na lista superior são apresentados os artefatos que são produzidos pelas atividades do processo de desenvolvimento do projeto em questão. O gerente pode, então, selecionar os itens que desejar ou então incluir algum item não previsto no processo, a partir da lista na parte inferior da tela (FIGUEIREDO, 2005).

Uma informação importante a ser destacada é que a partir de cada tela da ferramenta é possível buscar conhecimento no que diz respeito às atividades do processo, através da interação com uma ferramenta de Aquisição de Conhecimento – *Acknowledge* (MONTONI et al., 2004). É permitido o registro de idéias, dúvidas, problemas encontrados, diretrizes e lições aprendidas durante a execução das atividades do processo de gerência de configuração. Após serem registradas, essas informações são, então, filtradas e empacotadas para armazenamento no repositório da organização e vinculação ao processo e atividade adequados. A partir desse momento, o novo conhecimento (agora explícito) estará acessível aos usuários da ferramenta *GConf*. A figura 2.13 apresenta a tela de consulta de conhecimento com a descrição da atividade "Solicitar Alteração" (FIGUEIREDO, 2005).

Figura 2.13 – Consulta de Conhecimento – interface com a ferramenta *Acknowledge*.

A figura 2.14 ilustra a tela *Analisar Pedido de Alteração*, que pode ser executada pelo gerente do projeto ou pelo responsável de algum item de configuração com pedido de alteração pendente. A diferença entre os dois modos de execução está no fato de que o parecer dado pelo gerente do projeto é o parecer final, em que utiliza os outros pareceres para poder apoiar sua decisão. Além disso, é o gerente do projeto quem seleciona o responsável pela alteração. Sendo assim, a primeira coisa a ser feita ao executar esta atividade é selecionar o item de configuração para o qual se deseja fornecer um parecer, na lista presente na parte superior da tela. Em seguida, o usuário pode analisar os outros pareceres através da caixa de texto existente em baixo da lista de itens de configuração. Nessa caixa de texto, os relatórios dos pareceres fornecidos e os detalhes da alteração que o solicitante deseja realizar podem ser visualizados na forma de uma página HTML.

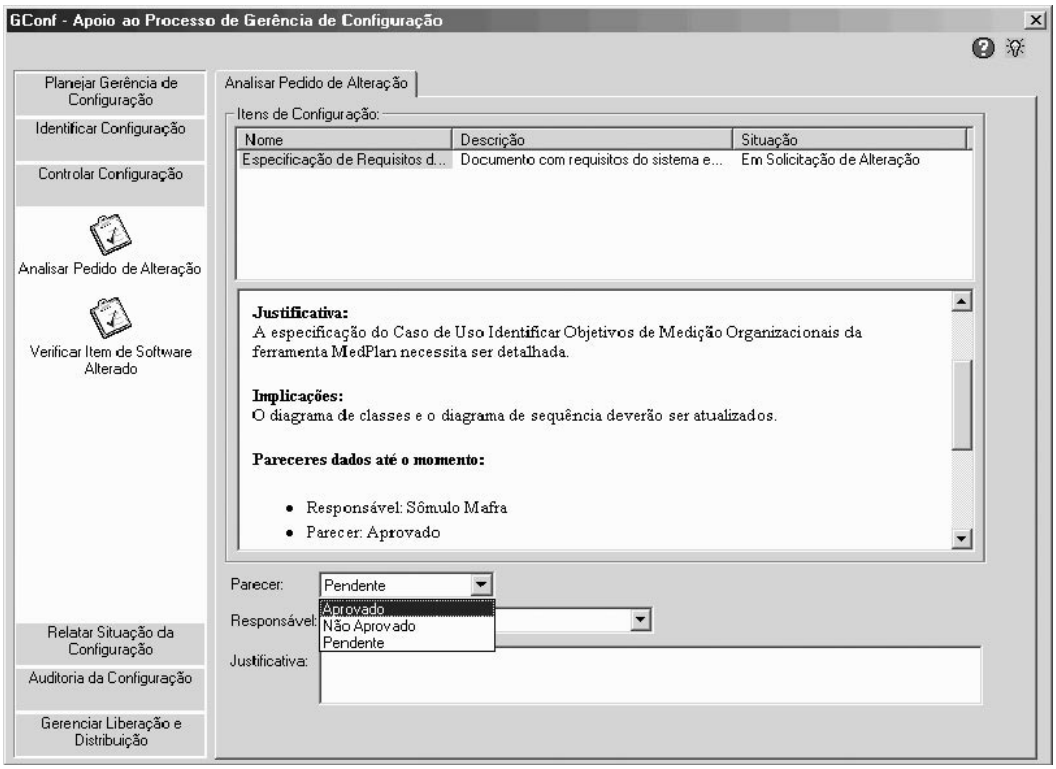


Figura 2.14 – Atividade Analisar Pedido de Alteração.

Através da atividade *Relatar Situação da Configuraçã*o, apresentada na figura 2.15, pode-se visualizar um relatório sobre as modificações realizadas nos itens de configuração. Ao selecionar o item desejado na lista da parte superior da tela, um relatório das alterações realizadas neste item será apresentado.

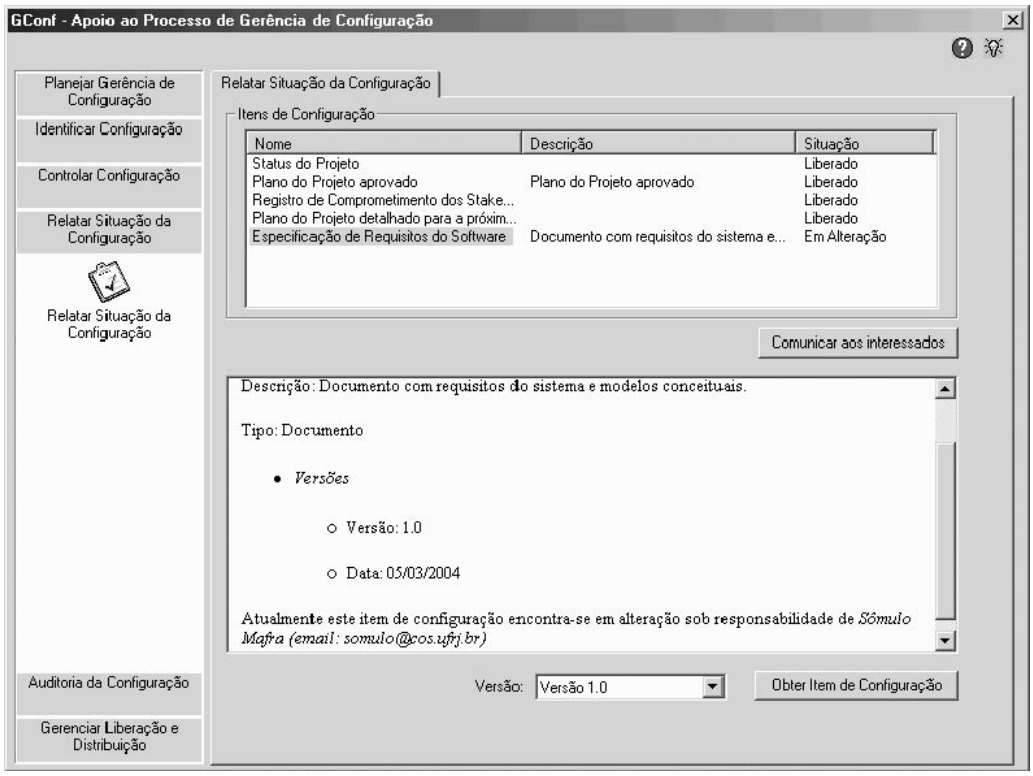


Figura 2.15 – Atividade Relatar Situação da Configuração.

Esta tela apresenta algumas funcionalidades adicionais, como, por exemplo, obter qualquer versão de um item de configuração ou copiar uma determinada versão do item de configuração. Outra funcionalidade existente é a notificação por e-mail da situação de um determinado item de configuração. Este e-mail é enviado aos "interessados pelo item", ou seja, as pessoas que executam alguma atividade que manipule o item em questão.

Na figura 2.16 é apresentada a atividade *Entregar Baseline*. Nesta atividade, o gerente do projeto seleciona, empacota e envia um *baseline* para o receptor desejado. Tal *baseline* foi previamente criado na atividade *Criar Baseline*. Na parte superior da figura pode-se visualizar a lista da qual o *baseline* deve ser selecionado, que contém todos os *baselines* do projeto. Existem duas formas de entregar o *baseline* selecionado: por e-mail ou colocando o *baseline* numa pasta onde o receptor possa pegá-lo.

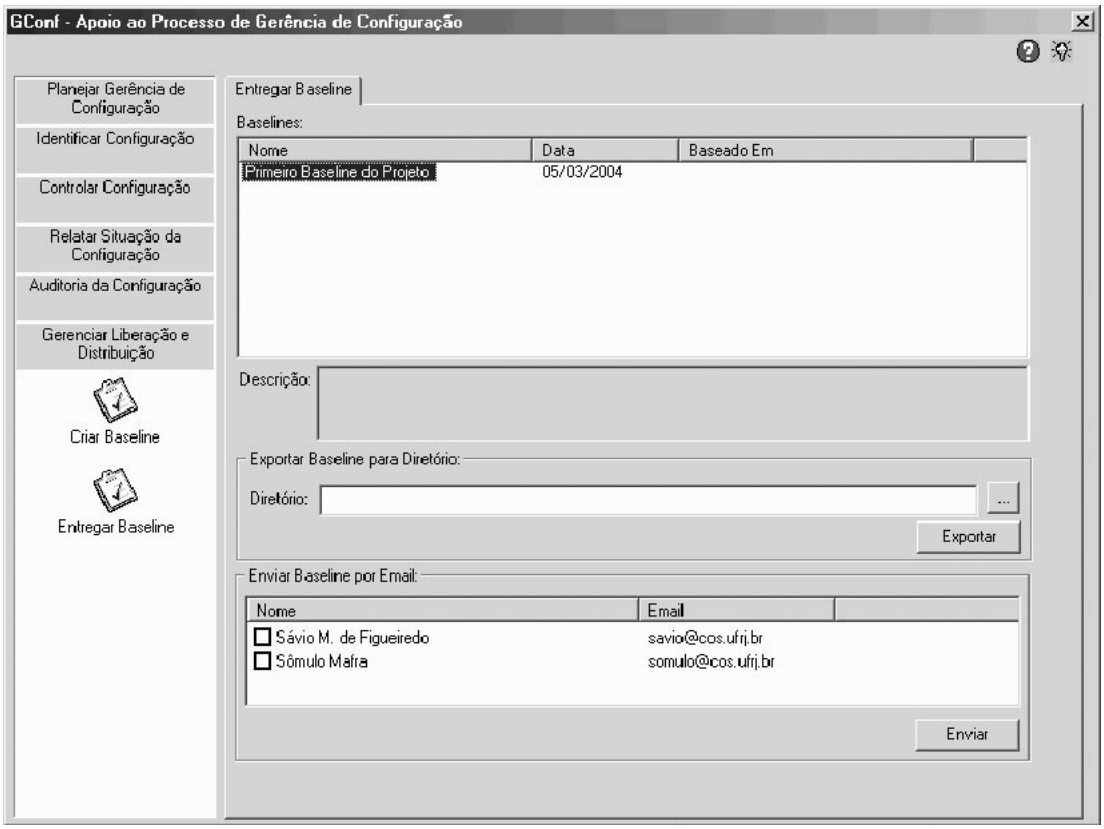


Figura 2.16 – Atividade Entregar *Baseline*.

Para colocar o *baseline* numa pasta que o receptor tenha acesso, basta selecionar o *baseline* na lista, entrar com o caminho do diretório e em seguida exportar. Caso o gerente prefira enviá-lo por e-mail, basta selecionar o *baseline*, identificar as pessoas que irão recebê-lo e enviar.

Em ambos os casos, o processo de empacotamento dos itens de configuração que formam o *baseline* é transparente ao usuário. Os arquivos referentes às versões dos itens de configuração pertencentes ao *baseline* são compactados e colocados dentro de um único arquivo, sem que o usuário perceba. Ao receber o *baseline*, a única coisa a fazer é descompactar os itens de configuração de dentro deste arquivo.

Considerações sobre os Sistemas de GCS

Apesar das diferenças entre as ferramentas, cada qual com suas particularidades e seu conjunto de funcionalidades disponíveis, uma característica marcante nelas é o fato de serem baseadas em arquivos. Ou seja, sob esta filosofia, um artefato sob gerência de configuração é tratado como um todo (um arquivo) e não por partes, não sendo possível, assim, identificar a sua estrutura intrínseca. Por exemplo, no caso do artefato ser um documento, com a estrutura proposta em (NUNES et al., 2004), constituído de seções, em que cada seção pode ser constituída de subseções, e quaisquer delas podem ser compostas por outros artefatos, não seria possível ter acesso a tais elementos, uma vez que o acesso seria apenas ao artefato e não às suas partes.

Instala-se uma situação de contradição: se por um lado, para construir um software de alta qualidade é crucial a habilidade para se gerenciar a evolução de abstrações lógicas, objetos e estruturas durante o processo de projeto e implementação, por outro lado, a maioria das ferramentas de GCS existentes trata os artefatos como arquivos. Existe, então, um obstáculo entre o domínio de projeto e implementação (nível arquitetural) e o domínio da GCS (nível de arquivo), pois eles requerem modelos mentais diferentes. Isso pode fazer com que os sistemas de GCS se tornem um incômodo durante o desenvolvimento, uma vez que a conexão semântica entre as entidades arquiteturais e o código fonte é difícil de ser gerenciada. (NGUYEN et al., 2004).

Outra característica encontrada em algumas ferramentas é permitir desenvolvimentos concorrentes. Essa abordagem possui a grande vantagem de permitir que múltiplos desenvolvedores alterem um mesmo artefato, ao mesmo tempo. Por exemplo, enquanto um desenvolvedor está trabalhando em uma manutenção evolutiva, que levará muitas horas para ser concluída, um outro desenvolvedor poderá efetuar uma manutenção corretiva e devolvê-la (*checkin*) antes do primeiro desenvolvedor ter concluído suas alterações. Porém, em desenvolvimentos paralelos, quando ocorre uma colisão na ocasião do *checkin*, torna-se necessária uma reconciliação (*merge*) e este processo pode não ser trivial. Algumas

ferramentas oferecem recursos para automatizar parte dessa tarefa, porém esta automatização é apenas para arquivos texto e possui várias limitações. A reconciliação acaba ficando, na maioria das vezes, a cargo do usuário, ou seja, torna-se um processo manual. Muitas vezes a reconciliação não é possível ou é inviável e a única alternativa é descartar as alterações de um dos artefatos, o que causa prejuízos no tempo de desenvolvimento e no custo de forma geral.

A ferramenta GConf possui um diferencial em relação à maioria das ferramentas pois possui um controle mais intenso sobre as alterações a serem realizadas, uma vez que antes de realizar uma mudança, o desenvolvedor deve fazer um pedido formal de alteração, que deve ser aprovado. O mesmo ocorre com as alterações implementadas, que só se tornam uma nova versão do item se forem aprovadas após serem analisadas pelos responsáveis pela configuração, o que não ocorre com as demais ferramentas.

Outra vantagem de GConf é o fato de estar integrada ao ambiente. Assim, todos os artefatos previstos no processo podem ser diretamente controlados a partir da ferramenta. Além disso, ela permite através da interface com a ferramenta *Acknowledge* a consulta e registro de conhecimento no que diz respeito às atividades do processo.

2.5 Gerência de Conhecimento, Documentação e Gerência de Configuração

Conforme discutido na seção 2.2, em um sistema de gerência de conhecimento, a memória organizacional deve armazenar o conhecimento disponível na organização, permitindo que ele possa ser recuperado no tempo e no lugar corretos (LIMA, 2004). Ela deve gerenciar todos os tipos de itens de conhecimento relevantes para o desenvolvimento de software (NATALI et al., 2003), dentre os quais merecem destaque os artefatos de software.

Artefatos são elementos produzidos ou consumidos por atividades durante a sua realização, tais como documentos, diagramas, artefatos de código, componentes de software ou produtos de software. Ou seja, constituem a documentação do sistema, do processo de desenvolvimento e da organização. Em se tratando de um Ambiente de Desenvolvimento de Software, os artefatos podem ser considerados o elo de comunicação entre as várias ferramentas que o compõem.

Sabe-se que a integração de ferramentas é um dos maiores desafios dos ADSs. Segundo PRESSMAN (2001), uma potencial abordagem para tratar o problema da integração seria a criação de uma estrutura, em que o principal mecanismo de integração de ferramentas

é a Gerência de Configuração de Software (GCS). A gerência de configuração deve identificar os artefatos a serem gerenciados, prover controle de versão e controle de mudanças, apoiar auditorias e fornecer relatos de configuração dos artefatos gerenciados, ficando, assim, no núcleo de todo ADS.

A Gerência de Configuração deve atuar, então, no controle dos artefatos, durante o desenvolvimento e manutenção de um sistema, podendo ser visto como o procedimento necessário para permitir um controle efetivo da documentação do sistema.

Desta forma, a Gerência de Conhecimento, a Documentação e a Gerência de Configuração devem trabalhar juntas, de modo que, a partir do momento que um artefato, que nada mais é que um item de documentação[3], for colocado sob Gerência de Configuração, ele deve se tornar, também, um item de conhecimento. Para que isso seja possível, deve ser estabelecida uma relação entre as bases da Gerência de Configuração, da Gerência de Conhecimento e da Documentação, ou seja, entre o repositório tratado pela gerência de configuração, com seus itens de configuração, entre a memória organizacional, com seus itens de conhecimento e entre os artefatos apresentados como itens de documentação.

Na base de dados de um Ambiente de Desenvolvimento de Software existem vários artefatos: os que foram criados, mas ainda não estão sob GCS; os que estão sob GCS e são os mais atuais e os que estão sob GCS, porém, são versões antigas. O repositório da GCS contém apenas os artefatos que se enquadram nos dois últimos casos, ou seja, os artefatos que estão sob GCS, seja a versão mais atual ou versões anteriores. Deve-se, então, tomar a decisão de quais artefatos comporão a memória organizacional. Caso se opte por utilizar todos os artefatos do ADS, os itens de conhecimento englobarão tanto os artefatos sob GCS, os itens de configuração, quanto os artefatos que não estão sob GCS. Caso se opte por utilizar apenas itens sob GCS, os itens de configuração se equivalerão aos itens de conhecimento. Por último, caso se opte por utilizar apenas as versões mais atuais dos artefatos, os itens de conhecimento serão um subconjunto dos itens de configuração. Contudo, deve-se apontar que, de modo geral, não é adequado que a memória organizacional contenha artefatos que não estejam sob GCS, uma vez que, provavelmente, ainda não estão maduros o suficiente para serem reutilizados.

Deve-se considerar, ainda, que o repositório central de um ADS não possui apenas itens de configuração ou itens de conhecimento, mas também informações relativas ao

[3] Está sendo utilizado o termo item de documentação ao invés de documento para deixar claro que se está referindo a qualquer item que componha uma documentação, seja ele um código fonte, um diagrama ou mesmo um documento. Ou seja, um item de documentação pode ou não ser um documento.

processo de desenvolvimento, à organização e outras informações relevantes. Assim, as bases de dados da Gerência de Conhecimento e da Gerência de Configuração são subconjuntos do repositório central do ADS.

Além da necessidade de uma relação entre as bases da GCS, da Gerência de Conhecimento e do ADS, outra decisão importante a ser tomada é definir um padrão para os artefatos, pois, como eles serão compartilhados entre várias ferramentas do ADS, devem possuir uma estrutura comum, passível de entendimento tanto pelas ferramentas quanto pelos desenvolvedores.

Uma solução para tal é a criação de uma ontologia de artefatos de software, conforme discutido no capítulo3. Uma ontologia é um entendimento compartilhado de algum domínio de interesse, que pode ser usado para resolver diversos problemas, tais como: comunicação precária entre as pessoas de uma organização, dificuldades na identificação de requisitos e na definição da especificação de sistemas, interoperabilidade entre sistemas e reuso e compartilhamento de conhecimento (USCHOLD et al., 1996).

No caso deste trabalho em especial, o domínio de interesse são os artefatos de software, que compõem a documentação dos sistemas, são controlados pela Gerência de Configuração e utilizados como itens de conhecimento pela Gerência de Conhecimento. Um artefato, por exemplo, pode ser criado por certa ferramenta, submetido à GCS, tratado como item de conhecimento pela Gerência de Conhecimento, utilizado por outras ferramentas do ADS e apresentado como um item da documentação de um sistema. Assim, o estabelecimento de um entendimento comum deste domínio é necessário para que não haja ambigüidades ou discrepância de conceitos. Devido à forte utilização dos artefatos pela Gerência de Configuração, pela Gerência de Conhecimento e pela Documentação em um ADS, uma ontologia de artefatos passa a ser essencial em um ADS Semântico, pois pode ser vista como a “cola” que mantém ligadas essas atividades.

2.6 Conclusões do Capítulo

Com a crescente demanda no setor de desenvolvimento de software, aumenta a necessidade de se dispor de ambientes automatizados que apóiem todo o processo de desenvolvimento. Os Ambientes de Desenvolvimento de Software (ADSs) surgem, então, com a finalidade de integrar diferentes ferramentas, construídas para finalidades específicas, mas que operam juntas para a construção de um produto final (TRAVASSOS,1994).

Muitas pesquisas têm focado neste assunto e, como resultado, surgiram novos tipos de ADS: os ADSs centrados em processo (ADSCPs), os ADSs orientados a domínio (ADSODs), os ADSs centrados na organização (ADSOrg) e os ADSs Semânticos. Nesta última classe, se enquadra ODE (*Ontology-based software Development Environment*), o ADS foco deste trabalho, que tem sido desenvolvido tendo por base ontologias.

Em um ADS Semântico, assim como nos ADSODs e nos ADSOrg, a gerência de conhecimento tem muita importância. Durante o desenvolvimento de software, muito conhecimento é criado. Porém, esse conhecimento não pode estar no nível de indivíduo, uma vez que ele será perdido quando o indivíduo sair da empresa (LIMA, 2004), nem apenas registrado em papel, pois não pode ser facilmente acessado, compartilhado e atualizado (O'LEARY, 1998a). Para minimizar tais problemas, as organizações podem adotar uma abordagem de Gerência de Conhecimento integrada a um ADS. Neste contexto, NATALI (2003) definiu uma infra-estrutura de gerência de conhecimento para ODE, que posiciona ao centro a memória organizacional e ao seu redor os serviços que apóiam cada uma das atividades do processo de gerência de conhecimento.

Um ADS necessita, ainda, de uma forma única e padronizada de se acessar e apresentar os documentos dos projetos de software. Com este intuito, SOARES (2002) definiu a ferramenta XMLDoc, para apoiar a documentação em ODE, que foi posteriormente adaptada por SILVA (2004).

Além da preocupação com o conhecimento, o ADS deve evitar que, durante o desenvolvimento de um software, a criação, a alteração e a exclusão de artefatos ocorram de forma indiscriminada. Para tal, ele deve dispor de ferramentas que estabeleçam e mantenham a integridade de tais artefatos ao longo de seu ciclo de vida (REZENDE, 2001). Entra em ação a Gerência de Configuração de Software (GCS), que consiste de um conjunto de atividades de controle e rastreamento que começa quando um projeto de desenvolvimento de software se inicia e termina somente quando o software é tirado de operação (PRESSMAN, 2001).

Atualmente, várias ferramentas são utilizadas para este fim. Apesar das particularidades das ferramentas de GCS existentes, em geral elas realizam as atividades de identificação dos itens de software a serem controlados, controle de versão e controle de modificação, dentre outras. Além disso, uma característica marcante nelas é o fato de serem baseadas em arquivos.

Um sistema de apoio à Gerência de Configuração de Software deve estar posicionado no núcleo de um ADS (PRESSMAN, 2001) e deve estar diretamente relacionado à Gerência de Conhecimento e à Documentação, estabelecendo-se uma relação clara entre repositório de

itens de configuração, memória organizacional e itens de documentação. Assim, é possível integrar artefatos em um ADS. Para que essa integração seja bem sucedida, é necessário, contudo, que o ADS, assim como suas ferramentas, incluindo os sistemas de Gerência de Configuração, Gerência de Conhecimento e Documentação, compartilhem um entendimento comum a respeito dos artefatos, seus tipos, composição e dependência. Para prover esse entendimento compartilhado no ambiente ODE, uma ontologia de artefatos foi desenvolvida e é apresentada no próximo capítulo. Finalmente, o capítulo 4 apresenta a abordagem proposta neste trabalho que integra Gerência de Configuração, Gerência de Conhecimento e Documentação no ambiente ODE, utilizando a ontologia proposta.

Capítulo 3

Uma Ontologia de Artefato de Software

Pessoas, organizações e sistemas de software têm necessidade de se comunicar constantemente. Entretanto, pelo fato de possuírem diferentes níveis de conhecimento, podem possuir pontos de vistas e concepções extremamente variadas sobre um mesmo assunto. Isso pode ocasionar uma sobreposição ou ambigüidade de conceitos e levar a uma comunicação pobre, confusa e sem consenso. No caso de desenvolvimento de sistemas, gera dificuldades na identificação de requisitos e limita a inter-operabilidade e o potencial de reuso e compartilhamento (USCHOLD et al., 1996).

Neste contexto, ontologias podem ter um papel importante, pois podem promover o entendimento comum entre desenvolvedores e podem ser usadas como uma base para especificação e desenvolvimento de software. Em um ADS Semântico, elas exercem papel fundamental no compartilhamento de informações entre diferentes ferramentas.

Quando se trata de compartilhamento de informações entre ferramentas de um ADS, entram em cena os artefatos de software. Uma ontologia para definir os conceitos e relações envolvidos neste contexto torna-se necessária para uma efetiva integração de ferramentas, o que inclui tratar artefatos, sua taxonomia, decomposição, dependências e aspectos relacionados à gerência de sua configuração.

Sabe-se, porém, que independente do domínio, a construção de ontologias é uma tarefa complexa que requer a adoção de práticas de engenharia de software. Assim, para a construção da ontologia de artefato, foi utilizado o método denominado SABiO (***S****ystematic* ***A****pproach for* ***B****ui****l****ding* ***O****ntologies*) (FALBO, 2004) (FALBO, 1998).

Este capítulo apresenta conceitos de ontologias, discute como elas são empregadas em ODE e apresenta a ontologia desenvolvida neste trabalho. A seção 3.1 apresenta brevemente os principais conceitos e aplicações de ontologias e a metodologia usada neste trabalho para a construção da ontologia de artefato. A seção 3.2 aponta as ontologias utilizadas em ODE. A seção 3.3 apresenta a ontologia de artefato proposta neste trabalho. Tal ontologia teve a necessidade de ser sub-dividida para contemplar características específicas de certos tipos de artefatos, no caso, documentos, artefatos de código e diagramas. Além disso, aspectos

relacionados à gerência de configuração também foram tratados. As seções 3.4, 3.5, 3.6 e 3.7 apresentam, respectivamente, cada uma dessas sub-ontologias. Por fim, a seção 3.8 apresenta as conclusões do capítulo.

3.1 Ontologias: Conceitos, Aplicações e Metodologias

Uma ontologia é uma especificação de uma conceituação, isto é, uma descrição de conceitos e relações e suas definições, propriedades e restrições. Tenta resolver problemas como falta de comunicação dentro das organizações, evitando dificuldades na identificação dos requisitos de um sistema. Além disso, ontologias não descrevem apenas conhecimento imediato, isto é, conhecimento factual que pode ser obtido diretamente a partir da observação do domínio, mas também conhecimento derivado, ou seja, aquele obtido através de inferência sobre o conhecimento imediato disponível (FALBO, 1998).

Ontologias são utilizadas para se referir a um entendimento compartilhado de algum domínio de interesse, que pode ser usado para resolver diversos problemas como: comunicação precária entre as pessoas de uma organização; dificuldade na identificação de requisitos e na definição da especificação de sistemas de informação; problemas com interoperabilidade, reuso e compartilhamento de métodos, paradigmas, linguagens e ferramentas de software; além de muito esforço perdido "reinventando a roda" (USCHOLD et al., 1996).

O termo ontologia foi adotado inicialmente na Filosofia, para tentar descrever domínios naturais (as coisas naturais do mundo) e a existência dos seres e coisas em si (FALBO, 1998). Porém, está agora ganhando regras específicas na comunidade de ciência da computação. Em particular, sua importância está sendo reconhecida em diversos campos de pesquisa, tais como na gerência do conhecimento, lingüística computacional, bancos de dados, recuperação de informação, projetos de sistemas baseados em agentes, análise orientada a objetos e, principalmente, na Inteligência Artificial (GUARINO, 1998).

Na ciência da computação, ontologias são usadas para descrever domínios já consagrados, como Medicina, Engenharia e Direito, onde é possível saber o significado projetado das coisas. Assim, o que se busca é firmar um acordo sobre o vocabulário do domínio de interesse, a ser partilhado por agentes que conversam sobre ele (FALBO, 1998).

Com o passar do tempo, as ontologias têm sido utilizadas de diferentes maneiras. Com base no seu conteúdo, elas podem ser classificadas em (GUARINO, 1997) (GUARINO, 1998) (FALBO, 1998):

- *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 conceituações de domínios particulares, descrevendo o vocabulário relacionado a um domínio, tal como Medicina, Direito, Engenharia de Software etc.
- *ontologias de tarefas*: expressam conceituações sobre a resolução de problemas, independentemente do domínio em que ocorram, isto é, descrevem o vocabulário relacionado a uma atividade ou tarefa genérica, tal como, diagnose ou vendas;
- *ontologias de aplicação*: descrevem conceitos dependentes do domínio e da tarefa particulares. Esses conceitos freqüentemente correspondem a papéis desempenhados por entidades do domínio quando da realização de certa atividade;
- *ontologias de representação*: explicam as conceituações que fundamentam os formalismos de representação de conhecimento.

Ontologias de domínio são construídas para serem utilizadas em um micro-mundo e são as mais comumente desenvolvidas. Diversos trabalhos são encontrados na literatura, enfocando áreas como química, manufatura, modelagem de empreendimento, medicina, física, direito, biologia, bioquímica, ciência dos materiais, engenharia de software, entre outros (FALBO, 1998). O uso de ontologias neste trabalho é para o domínio da Engenharia de Software, mais especificamente Artefatos de Software.

Uma ontologia de domínio é uma representação formal e declarativa que estabelece o vocabulário (conceitos) para os elementos no domínio, as declarações lógicas que descrevem o que são estes elementos e as relações existentes entre eles. Assim, basicamente, uma ontologia consiste de um *vocabulário* específico usado para descrever uma certa realidade, além de um conjunto de suposições explícitas que dizem respeito ao *significado planejado* dos termos do vocabulário (FALBO, 1998). Este conjunto de suposições tem, usualmente, a forma de uma teoria lógica de primeira ordem, onde os termos do vocabulário aparecem como nomes de predicados unários ou binários, representando conceitos e relações. Em casos mais simples, uma ontologia descreve uma hierarquia de conceitos relacionados por associações de especialização e generalização; em casos mais complexos, axiomas apropriados são

adicionados para expressar outros relacionamentos entre conceitos ou para restringir sua interpretação (GUARINO, 1998).

Em suma, uma ontologia consiste de conceitos e relações - o vocabulário - e suas definições, propriedades e restrições, descritas na forma de axiomas. Os conceitos são normalmente organizados em taxonomias, porém, os conceitos primitivos, não podem ser expressos em termos de outros conceitos e, desta forma, necessitam ser definidos textualmente, ou explicados através de exemplos. Entretanto, uma ontologia não deve simplesmente ser composta por um conjunto de termos básicos e sim utilizar axiomas para definir a semântica de tais termos. Axiomas são um meio de definir e representar outras informações sobre conceitos e suas relações, incluindo restrições sobre propriedades e seus valores. Idealmente, uma ontologia não deve ser uma simples hierarquia de termos, e sim uma infra-estrutura teórica que verse sobre o domínio (FALBO, 1998).

O uso de ontologias é uma ferramenta útil para apoiar a especificação e implementação de qualquer sistema de computação complexo. A modelagem dentro do domínio do problema apresenta as seguintes vantagens: (i) independência técnica, (ii) conhecimento do domínio reutilizável para diferentes aplicações e (iii) representação mais facilmente compreendida pelo especialista do domínio (MIAN, 2003).

Porém, deve-se lembrar que ontologias de domínio não devem ter como objetivo a especificação dos termos mais comuns, mas a especificação das *categorias básicas* de conhecimento do domínio, uma vez que termos comumente usados podem ter significados contraditórios ou estar mais relacionados à comunicação do que à essência do conhecimento. Além disso, não é objetivo de uma ontologia de domínio descrever todo o conhecimento a ser codificado em uma base de conhecimento. Algum conhecimento empírico, compilado ou prático, que é dependente da tarefa ou aplicação particular, pode encontrar lugar apenas em uma ontologia de aplicação e não devem fazer parte de uma ontologia de domínio (GUARINO, 1998).

JASPER et al. (1999) classificam aplicações de ontologias em quatro categorias principais, sendo que uma aplicação pode integrar mais de uma destas categorias:

- Autoria neutra: uma ontologia é desenvolvida em uma única linguagem e é traduzida em formatos diferentes e usada em aplicações de múltiplos objetivos;
- Ontologia como Especificação: uma ontologia de um determinado domínio é criada e provê um vocabulário para especificar requisitos para uma ou mais aplicações-alvo. A ontologia é usada como uma base para especificação e desenvolvimento de software, permitindo reuso de conhecimento;

- Acesso Comum à Informação: uma ontologia é usada para permitir que múltiplas aplicações-alvo (ou pessoas) tenham acesso a fontes heterogêneas de informação que são expressas usando vocabulário diverso ou formato inacessível;
- Busca Baseada em Ontologias: uma ontologia é usada para procurar, em um repositório de informação, por recursos desejados, melhorando a precisão e reduzindo a quantidade de tempo gasto na busca.

Fundamentalmente, ontologias são utilizadas para melhorar a comunicação entre humanos e computadores, e sua utilização se classifica nas seguintes categorias (JASPER et al., 1999):

- Auxiliar a comunicação entre agentes humanos – através de uma linguagem sem ambigüidades, ontologias reduzem confusões terminológicas e conceituais dentro da organização, capacitam o entendimento compartilhado e a comunicação entre pessoas com diferentes necessidades e pontos de vista.
- Alcançar Inter-Operabilidade entre sistemas de computadores – refere-se à necessidade de diferentes usuários trocarem dados ou utilizarem diferentes ferramentas. Ontologias podem ser usadas como uma inter-língua para apoiar a tradução entre diferentes linguagens e representações. Deve-se levar em consideração a natureza dos sistemas (internos ou externos a uma organização), a necessidade de integrar ontologias em diferentes domínios para apoiar alguma tarefa específica e, ainda, a necessidade de integrar diferentes ontologias dentro de um mesmo domínio (OLIVEIRA, 1999).
- Melhorar o processo e/ou a qualidade dos sistemas de software – ontologias podem ser benéficas na especificação, confiabilidade e reutilização durante a *engenharia de sistemas*. Na *especificação*, podem ser utilizadas para promover o entendimento compartilhado de um problema ou tarefa, facilitando o processo de identificação dos requisitos do sistema e o entendimento das relações entre os componentes do sistema. No que se refere à *confiabilidade*, ontologias podem servir para verificação manual do projeto em relação à sua especificação ou uma verificação semi-automatizada, para o caso de ontologias formais. Outro benefício do uso de ontologias no desenvolvimento de software é a *reutilização* de especificações de domínio (OLIVEIRA, 1999).

No desenvolvimento tradicional, para cada nova aplicação a ser construída, é desenvolvida uma conceituação nova. Porém, esta abordagem é extremamente cara, uma vez

que a elicitação de requisitos é uma das atividades que consome mais tempo e os peritos, que são essenciais a esta atividade, são recursos escassos e caros. Portanto, é importante compartilhar e reutilizar o conhecimento capturado (FALBO et al., 1998).

Em uma abordagem baseada em ontologias, a elicitação e a modelagem de requisitos podem ser realizadas em duas fases. Primeiro, o conhecimento de domínio geral deve ser extraído e especificado na forma de ontologias. Estas ontologias são usadas para guiar a segunda fase da análise de requisitos, quando são consideradas as particularidades de uma aplicação específica. Deste modo, a mesma ontologia pode ser usada para guiar o desenvolvimento de várias aplicações, diminuindo os custos da primeira fase e permitindo o compartilhamento e reuso do conhecimento (FALBO et al., 1998).

3.1.1 Construção de Ontologias

De maneira geral, qualquer que seja o domínio, a complexidade envolvida na construção de ontologias é grande e, portanto, algum mecanismo de decomposição deve ser usado para facilitar o processo de construção. Uma abordagem interessante é considerar *sub-ontologias*, isto é, dividir o domínio em sub-domínios e tratar cada um deles separadamente, mas levando em conta que alguns conceitos são comuns aos vários sub-domínios.

Uma vez que ontologias são projetadas, a escolha de seus compromissos ontológicos torna-se uma decisão de projeto, em que são necessários critérios de qualidade objetivos. GRUBER (1995) enumerou um conjunto de critérios para avaliar a qualidade de ontologias. Estes critérios, relacionados a seguir, devem nortear o processo de construção de uma ontologia em todas as suas etapas.

- *Clareza*: Uma ontologia deve comunicar efetivamente o significado projetado dos termos definidos, com definições objetivas.
- *Coerência*: Uma ontologia deve comportar apenas inferências consistentes com as definições.
- *Extensibilidade*: Uma ontologia deve ser projetada para antecipar usos do vocabulário compartilhado, tornando possível a definição de novos termos, com base no vocabulário existente, sem haver necessidade de rever definições existentes.
- *Compromissos de codificação mínimos*: A conceituação deve ser especificada no nível de conhecimento sem depender de uma tecnologia particular de representação de conhecimento. Uma tendência de codificação deve ser

minimizada, já que agentes de conhecimento podem ser implementados em diferentes sistemas e paradigmas.

- *Compromissos ontológicos mínimos*: O conjunto de compromissos ontológicos de uma ontologia deve ser o menor possível, capaz de suportar as atividades planejadas de compartilhamento de conhecimento, permitindo que as partes comprometidas com a ontologia fiquem livres para possíveis especializações e instanciações. Compromissos ontológicos podem ser minimizados através da especificação de uma teoria mais fraca (que admita um maior número de modelos), contendo definições restritas apenas para os termos essenciais à comunicação consistente do conhecimento da teoria.

Um último aspecto a ser considerado sobre a construção de ontologias é a escolha de uma linguagem para expressá-las. A princípio, qualquer linguagem de representação de conhecimento formal, ou até mesmo informal, poderia ser usada (FALBO, 1998). Na prática, contudo, algumas linguagens têm sido mais utilizadas, tais como Lógica de Primeira Ordem (VALENTE, 1995), KIF (GRUBER, 1992), Ontolingua (GRUBER, 1992), RDF (LASSILA et al., 1999), DAML+OIL (FENSEL et al., 2000) e OWL (BECHHOFER et al., 2004). Além dessas, extensões da UML (*Unified Modeling Language*) (BOOCH et al., 2000), tal como a proposta em (MIAN et al., 2003), têm sido utilizadas como linguagens gráficas para definir ontologias. Em particular, são utilizados os diagramas de classes, que provêem uma notação para definição de classes, seus atributos e relacionamentos (CRANEFIELD et al., 1999).

Uma grande quantidade de ontologias tem sido desenvolvida por diferentes grupos, sob diferentes abordagens, e usando diferentes métodos e técnicas. FALBO (1998, 2004) propôs uma abordagem, denominada SABiO (***S****ystematic* ***A****pproach for* ***Bui****lding* ***O****ntologies*), que define as atividades do processo de construção de ontologias, com algumas orientações de como proceder na sua realização. Além disso, é proposto um ciclo de vida, mostrando as interações entre as várias atividades, mostrado na figura 3.1. Tendo em vista que essa abordagem foi utilizada neste trabalho, a seguir suas atividades são brevemente descritas (FALBO, 2004):

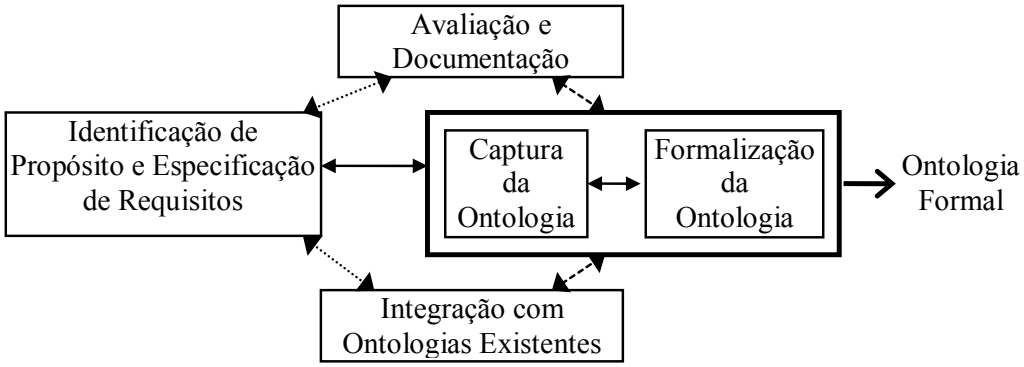


Figura 3.1 - Etapas do desenvolvimento de ontologias e suas interdependências (FALBO, 1998).

- **Identificação de Propósito e Especificação de Requisitos:** Visa identificar claramente o propósito da ontologia e os usos esperados para ela, i.e., sua competência. Para sua realização, questões de competência são utilizadas.
- **Captura da Ontologia:** O objetivo é capturar a conceituação do domínio, com base na competência da ontologia. Conceitos, relações, propriedades e axiomas relevantes devem ser identificados e organizados. Um modelo usando uma linguagem gráfica e um dicionário de termos deve ser utilizado para facilitar a comunicação com os especialistas do domínio.
- **Formalização da Ontologia:** Visa representar explicitamente a conceituação capturada pela ontologia em uma linguagem formal.
- **Integração com Ontologias Existentes:** Durante a captura ou formalização da ontologia, pode ser necessário integrar a ontologia em questão com outras já existentes, para utilizar conceituações previamente estabelecidas.
- **Avaliação da Ontologia:** A ontologia deve ser avaliada a fim de verificar se os requisitos estabelecidos na especificação estão sendo satisfeitos. Deve ser avaliada em relação à competência da ontologia e a alguns critérios de qualidade, como aqueles definidos por GRUBER (1995).
- **Documentação:** Todo o desenvolvimento da ontologia deve ser documentado.

SABiO propõe o uso de uma linguagem gráfica para expressar ontologias e de uma linguagem formal, tal como a lógica de primeira ordem, para formalizá-las. A linguagem gráfica originalmente adotada era LINGO (FALBO, 1998). LINGO possui primitivas para representar conceitos, relações e propriedades, e alguns tipos de relações que possuem uma semântica bem-estabelecida, tais como relações todo-parte e de generalização / especialização.

Posteriormente, decidiu-se por permitir também a captura de ontologias utilizando-se a UML (MIAN, 2003), uma vez que a UML também tem sido utilizada como linguagem de modelagem de ontologias e não se pode ignorar que ela é um padrão e seu uso está vastamente difundido. Assim, foi definido um subconjunto de elementos da UML, que exerce o mesmo papel da notação de LINGO, ou seja, os elementos de modelo da UML são aplicados usando a mesma semântica imposta pelos correspondentes elementos em LINGO, conforme mostra a figura 3.2 (MIAN, 2003).

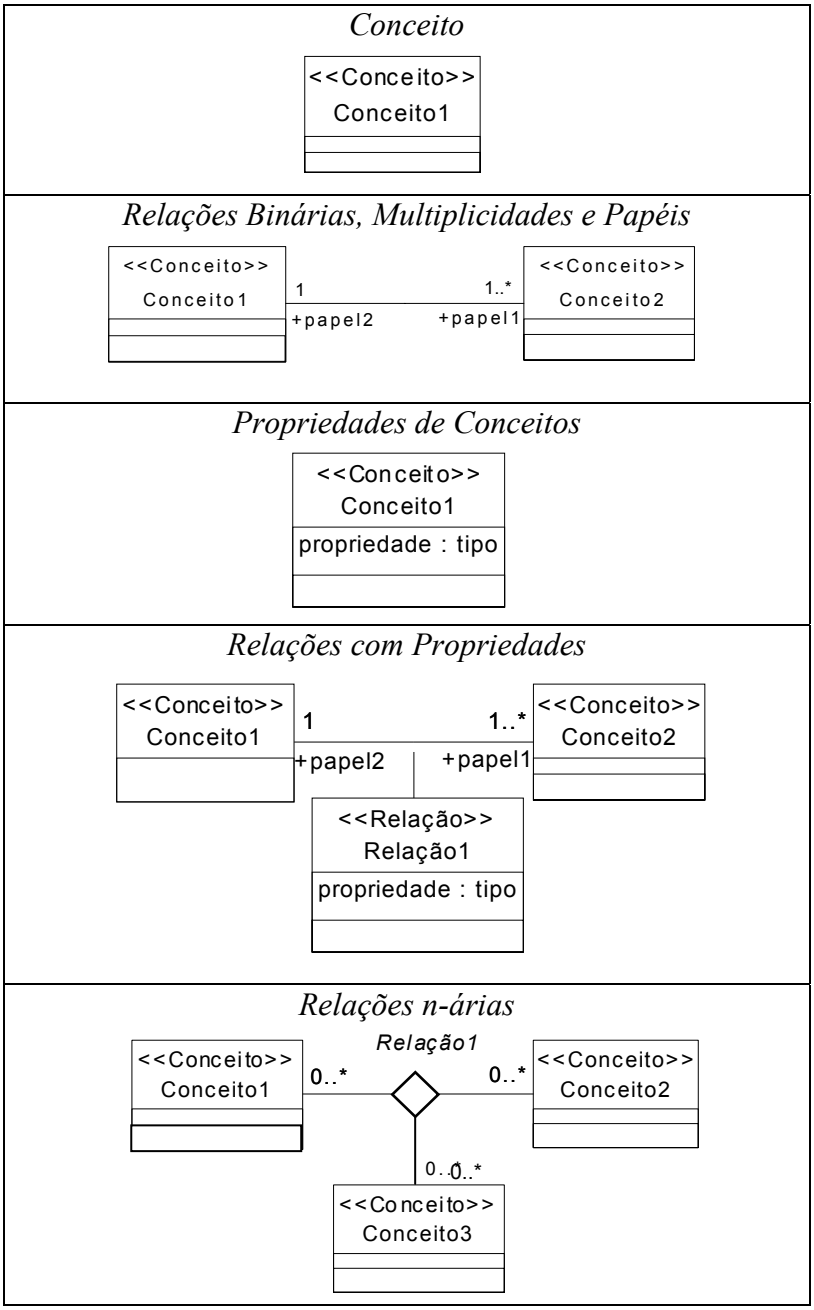


Figura 3.2 - Subconjunto da UML para representar ontologias.

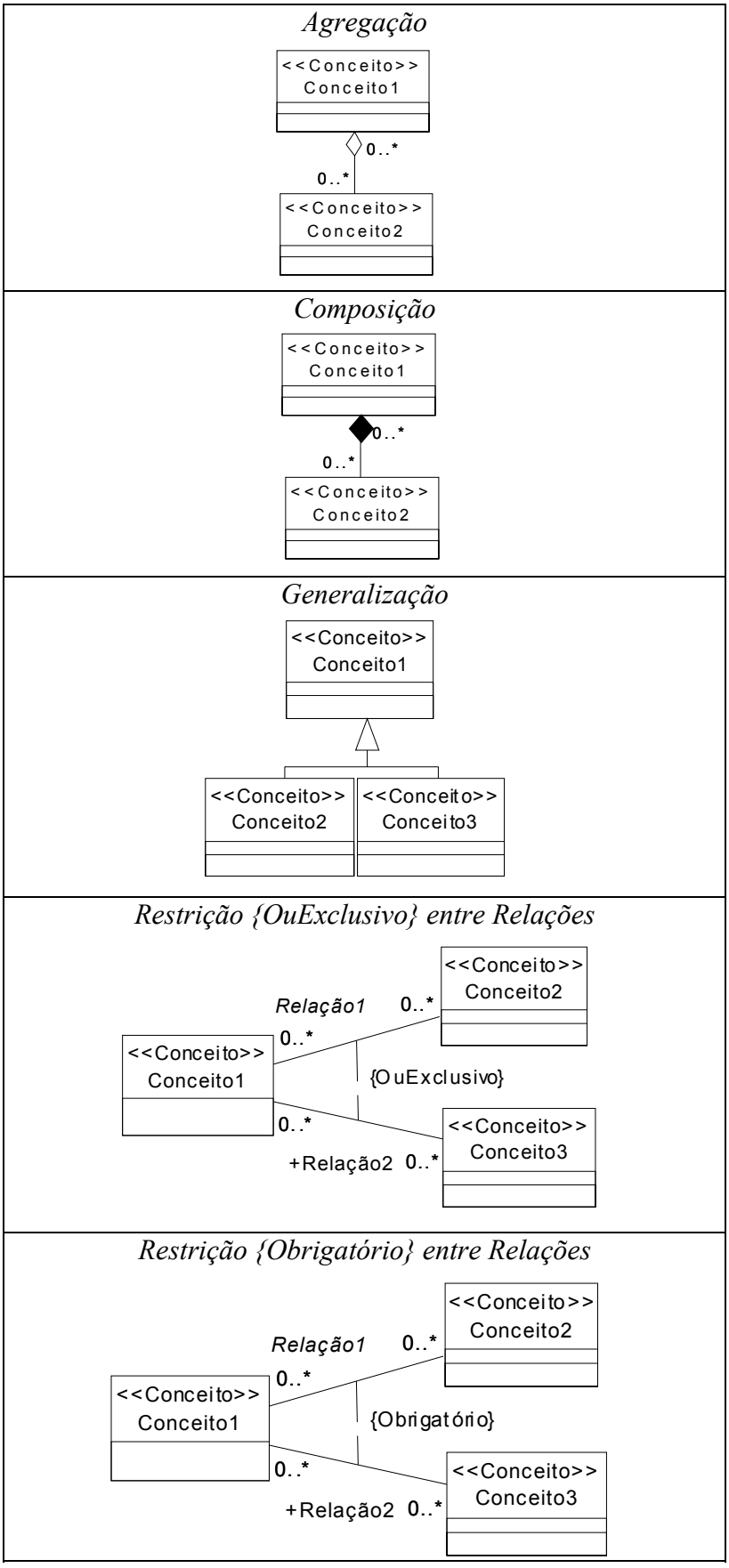


Figura 3.2 - Subconjunto da UML para representar ontologias (continuação).

Segundo essa representação, classes com estereótipo <<Conceito>> representam conceitos da ontologia. Relações são definidas como associações nomeadas. Propriedades de conceitos e relações são representadas como atributos de classes. Relações que contêm propriedades ou que possuem aridade maior que dois são representados como classes associativas com estereótipo <<Relação>>. Relações de supertipo e todo-parte são representadas como relações de generalização/especialização e de agregação/composição, respectivamente. Condicionantes entre relações são representados por restrições entre associações (MIAN, 2003).

3.2 As Ontologias utilizadas em ODE

Conforme discutido no capítulo 2, ODE (*Ontology-based software Development Environment*) (FALBO et al., 2003) busca evoluir para um Ambiente de Desenvolvimento de Software Semântico, tendo por base ontologias. Uma mesma ontologia pode ser usada para construir ferramentas diferentes que apóiam atividades de engenharia de software correlacionadas. Além disso, busca que suas ontologias sejam integradas, o que facilita a integração das ferramentas construídas com base nelas (FALBO et al., 2003).

Dentre as ontologias que compõem a base ontológica de ODE, encontram-se ontologias de processo de software (FALBO, 1998), qualidade de software (DUARTE et al., 2000) e riscos de software (FALBO et al., 2004c). Os critérios de qualidade para a construção de ontologias, descritos na seção 3.1.1, foram seguidos, com destaque para o critério de comprometimento ontológico mínimo, com o intuito de tornar as ontologias amplamente reutilizáveis e extensíveis.

A ontologia principal é a ontologia de processo de software, já que ela trata dos principais conceitos envolvidos na área de Engenharia de Software. Todas as demais estão de alguma forma integradas a ela. Mesmo considerando apenas os aspectos gerais de processos de software, esse domínio mostrou-se extremamente complexo e foi necessário, então, utilizar um mecanismo de decomposição, que permitiu construir a ontologia por partes. Assim, a estratégia adotada consistiu em se definir sub-domínios do domínio de processos de software, e construir ontologias básicas sobre esses sub-domínios, a saber, atividade, procedimento e recurso. Uma vez definidas as ontologias básicas, estas foram utilizadas de forma integrada para construir a ontologia de processo de software (FALBO, 1998).

Como a integração de ferramentas é um dos maiores desafios para os Ambientes de Desenvolvimento de Software, o uso de ontologias pode trazer muitos benefícios. Em ODE, todas as ferramentas são desenvolvidas baseadas na ontologia de processo de software ou em ontologias integradas a ela. Porém, no momento em que se começou a tratar efetivamente as atividades de documentação, gerência de configuração de software e gerência de conhecimento em ODE, percebeu-se que as ontologias disponíveis ainda não eram suficientes para lidar com a complexidade inerente aos artefatos de software produzidos e consumidos em um processo. Como a maioria das ferramentas de ODE gera artefatos e novas funcionalidades para apoiar documentação, gerência de configuração de software e gerência de conhecimento também fariam uso intensivo de artefatos, era necessário definir um vocabulário comum, para facilitar a comunicação entre elas e entre os seus desenvolvedores. Assim, visando preencher essa lacuna, foi desenvolvida neste trabalho uma ontologia de artefato de software, apresentada a seguir. A partir da definição dessa ontologia, cada ferramenta de ODE deve passar a criar seus artefatos de acordo com ela.

3.3 Ontologia de Artefato de Software

Artefatos de software são peças fundamentais em um processo de desenvolvimento de software, uma vez que são produzidos ou consumidos por atividades durante a sua realização. Desta forma, artefatos estão presentes em todo o ciclo de vida de desenvolvimento do software.

A necessidade de criação de uma ontologia de artefato vem do intuito de poder compartilhar os diversos artefatos em um ADS, através de um vocabulário comum, facilitando, assim, a comunicação entre os desenvolvedores e entre as ferramentas do ADS.

Porém, não é interessante construir uma ontologia única para artefatos de software, uma vez que, para cada tipo de artefato (documento, diagrama, artefato de código etc), pode-se vislumbrar características diferentes. Subdividindo-se essa ontologia em sub-ontologias mais específicas, podem ser adicionados outros elementos, garantindo o critério de comprometimento ontológico mínimo. Além disso, uma vez que, durante o desenvolvimento, artefatos devem ser submetidos à gerência de configuração, uma sub-ontologia tratando aspectos relacionados a essa atividade foi também desenvolvida.

Desta forma, nesta seção está sendo apresentada uma ontologia de artefato que se aplica a todos artefatos de forma geral e nas próximas seções são apresentadas ontologias para alguns tipos específicos de artefatos, a saber, documento, diagrama e artefato de código.

Conforme citado anteriormente, para desenvolver a ontologia de artefato de software, foi adotada SABiO em sua versão mais recente, que inclui o uso de uma extensão da UML como linguagem gráfica para representação de ontologias. Além disso, para formalizar os axiomas da ontologia, optou-se pela lógica de primeira ordem. Foram identificados as constantes e os predicados, compondo o alfabeto da linguagem e, a partir dele, os axiomas da ontologia foram formalizados. Na seqüência são discutidas as principais atividades do processo de SABiO (identificação do propósito e especificação de requisitos, captura e formalização), como forma de apresentar a ontologia desenvolvida.

3.3.1 Identificação de Propósito e Especificação de Requisitos

De modo geral, uma ontologia de artefato deve ser capaz de responder às seguintes questões de competência:

1. Qual a natureza de um artefato?
2. Como um artefato pode ser decomposto?
3. Que artefatos são consumidos por uma determinada atividade?
4. Que artefatos são produzidos por uma determinada atividade?
5. Quais recursos humanos podem aprovar um determinado artefato?
6. Que artefatos dependem de um determinado artefato, isto é, mudanças neste artefato podem provocar impactos em seus dependentes?

3.3.2 Captura e Formalização da Ontologia

Analisando as questões de competência anteriormente relacionadas, identificamos alguns aspectos relevantes:

- taxonomia de artefatos (questão 1);
- decomposição de artefatos (questão 2);
- atividades como primitivas de transformação (questões 3 e 4);
- aprovação de artefatos (questão 5); e
- dependências entre artefatos (questão 6).

O terceiro aspecto mostra a interação entre a ontologia de artefato e a ontologia de atividade (FALBO, 1998) e o quinto aspecto mostra a interação entre a ontologia de artefato e a

ontologia de recursos (FALBO, 1998), indicando a integração da ontologia aqui desenvolvida com a ontologia de processo de software.

Taxonomia de Artefatos

No contexto de desenvolvimento de software, artefatos, quanto a sua natureza, podem ser classificados em: documento, diagrama, artefato de código, componente de software e produto de software. A figura 3.3 mostra a taxonomia de artefatos.

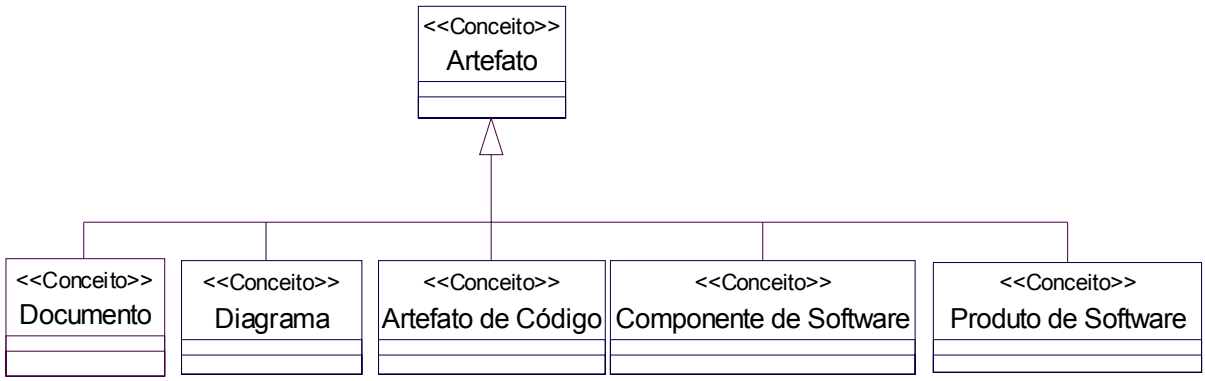


Figura 3.3 - Taxonomia de Artefatos.

Documentos são artefatos de software não passíveis de execução, constituídos tipicamente de declarações textuais, normalmente associados a padrões organizacionais (roteiros) que definem a forma como eles devem ser produzidos. Exemplos de documentos incluem: documento de especificação de requisitos, plano de projeto, plano de qualidade, relatório de avaliação da qualidade, entre outros.

Diagramas são artefatos gráficos que consistem em apresentações gráficas de um conjunto de elementos de modelo, em geral representadas como um gráfico conectado de vértices (elementos de modelo) e arcos (relacionamentos). Exemplos de diagramas incluem: diagrama de casos de uso, diagrama de classes, diagrama de entidades e relacionamentos, diagrama relacional, entre outros.

Artefatos de Código são porções de código, passíveis de execução, geradas no próprio desenvolvimento. Exemplos de artefatos de código incluem: programas, sub-programas, classes, rotinas, funções, entre outros.

Componentes de Software são artefatos de software que provêem um conjunto bem definido de serviços, desenvolvidos de forma independente e manipulados (usados, vendidos, mantidos em uma biblioteca etc.) como unidades coerentes, que podem ser combinados para

formar artefatos maiores, fornecendo uma interface bem definida e explícita. Exemplos de componentes de software incluem: classes, frameworks, padrões gerativos, entre outros.

Produtos de Software são os artefatos resultantes de um processo de desenvolvimento de software concluído e entregues ao usuário, ou seja, colocados em operação. Incluem o conjunto de todos artefatos gerados no desenvolvimento, além das ferramentas e softwares necessários para colocar os produtos em operação. Em um ciclo de vida seqüencial, qualquer alteração necessária em um produto de software se dará em uma atividade de manutenção e não mais em uma atividade de desenvolvimento do software. Em um ciclo de vida iterativo, porém, alterações podem significar a criação de uma nova versão ou variante de um produto de software.

Para formalizar a existência de diferentes tipos de artefatos, definimos os seguintes predicados, representando cada um dos tipos identificados na taxonomia: *artefato(a)*, indicando que *a* é um artefato; *documento(d)*, indicando que *d* é um documento; *diagrama(di)*, indicando que *di* é um diagrama; *artcódigo(ac)*, indicando que *ac* é um artefato de código; *componentesoftware(c)*, indicando que *c* é um componente de software e *produtosoftware(p)*, indicando que *p* é um produto de software.

A notação de sub-tipo empregada na figura 3.3 reflete os seguintes axiomas epistemológicos[4]:

$(\forall d)\ (documento(d) \rightarrow artefato(d))$	(AE1)
$(\forall di)\ (diagrama(di) \rightarrow artefato(di))$	(AE2)
$(\forall ac)\ (artcódigo(ac) \rightarrow artefato(ac))$	(AE3)
$(\forall c)\ (componentesoftware(c) \rightarrow artefato(c))$	(AE4)
$(\forall p)\ (produtosoftware(p) \rightarrow artefato(p))$	(AE5)

Uma vez que axiomas desta natureza são implicitamente descritos pela notação da extensão da UML proposta com base em LINGO, eles não serão mais apresentados neste trabalho. Sempre que a notação para sub-tipos for empregada, assumimos que existem axiomas deste tipo.

Decomposição de Artefatos

Artefatos não apenas se caracterizam como peças elementares em um processo de software, como também podem ser decompostos em outros artefatos, sendo, assim,

[4] Axiomas epistemológicos são derivados simplesmente da estrutura dos conceitos e não de seus significados particulares. Portanto, podem ser derivados automaticamente por uma ferramenta para edição de ontologias,

importante identificar os artefatos que compõem um outro artefato. Para tal, foram introduzidos os conceitos de super-artefato e sub-artefato (FALBO, 1998), como mostram os papéis na figura 3.4.

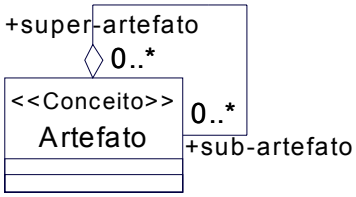


Figura 3.4 - Decomposição de Artefatos.

Um *super-artefato* é um artefato composto de outros artefatos, enquanto um *sub-artefato* é um artefato que compõe um outro artefato.

O predicado *subartefato(a1, a2)* indica que o artefato *a1* é um sub-artefato (ou é parte) do artefato *a2* , enquanto o predicado *superartefato (a2 ,a1)* indica que *a2* é um super-artefato de (ou é um todo, em que uma das partes é) *a1*. Esses predicados estão relacionados segundo a seguinte sentença:

$(\forall a1, a2)\ (subartefato(a1, a2) \leftrightarrow superartefato(a2, a1))$ (A1)

Como a lógica de primeira ordem não é uma lógica poli-sortida, torna-se necessário definir axiomas de consolidação5 que estabeleçam que tipos de objetos podem ser utilizados como argumentos em um predicado. Assim, o seguinte axioma de consolidação deve ser observado (FALBO, 1998):

$(\forall a1, a2)\ (subartefato(a1, a2) \rightarrow artefato(a1) \wedge artefato(a2))$ (AC1)

Este axioma estabelece que os argumentos a1 e a2 do predicado *subartefato* devem ser artefatos.

A decomposição de artefatos, assim como qualquer relação todo-parte, é transitiva, isto é, se um artefato a_1 é um sub-artefato do artefato a_2 e a_2 é um sub-artefato do artefato a_3, então a_1 é também um sub-artefato de a_3. O axioma epistemológico abaixo formaliza esta propriedade da relação de agregação.

$(\forall a1, a2, a3)\ (subartefato(a1, a2) \wedge subartefato(a2, a3) \rightarrow subartefato(a1, a3))$ (AE6)

quando tais estruturas forem utilizadas (FALBO, 1998). A numeração desses axiomas será precedida das letras AE.

[5] Axiomas de consolidação têm por objetivo verificar a coerência das informações existentes e não representam conseqüências lógicas, isto é, não derivam nova informação (FALBO, 1998). Para diferenciá-los de outros tipos de axiomas, a numeração dos axiomas de consolidação será precedida pelas letras AC.

A decomposição de artefatos é também assimétrica, isto é, se um artefato a_1 é um sub-artefato do artefato a_2, então a_2 não pode ser um sub-artefato de a_1.

$(\forall a1, a2)\ (subartefato(a1, a2) \rightarrow \neg\ subartefato(a2, a1))$	(AE7)

As propriedades de transitividade e assimetria são válidas para quaisquer relações todo-parte e, assim, da mesma forma que as hierarquias de sub-tipo, não terão seus axiomas mais apresentados neste trabalho, uma vez que são implicitamente descritos pela notação utilizada.

Porém, existem artefatos que não são decompostos em outros. Surge, então, o conceito de *artefato elementar*. Seu equivalente inverso é o conceito de *macro-artefato*, que é um artefato que não é parte de nenhum outro artefato.

Os predicados *artefatoelementar(a)* e *macroartefato(a)* indicam, respectivamente, que um artefato *a* é um artefato elementar ou um macro-artefato. Esses predicados são definidos em termos dos conceitos de sub-artefato e super-artefato, como mostram as sentenças abaixo:

$(\forall a)\ (artefatoelementar(a) \leftrightarrow \neg\ (\exists a1)\ (subartefato(a1, a)))$	(A2)
$(\forall a)\ (macroartefato(a) \leftrightarrow \neg\ (\exists a1)\ (superartefato(a1, a)))$	(A3)

Uma propriedade particular dos produtos de software é não poder ser sub-artefato de nenhum outro artefato, uma vez que são resultantes de um processo de desenvolvimento de software concluído. Sendo assim, não faria sentido a existência de um artefato composto em que um de seus sub-artefatos fosse um produto de software. Este fato é mostrado pela sentença abaixo:

$(\forall p)\ (produtosoftware(p) \rightarrow \neg\ (\exists a)\ (superartefato(a, p)))$	(A4)

Uma outra propriedade particular, porém desta vez de documentos e diagramas, é o fato de não poderem ser sub-artefatos de artefatos de código, uma vez que artefatos de código são passíveis de execução e não suportam estes tipos de sub-artefato.

$(\forall di)\ (diagrama(di) \rightarrow \neg\ (\exists a)\ (subartefato(di, a) \wedge artcodigo(a)))$	(A5)
$(\forall d)\ (documento(d) \rightarrow \neg\ (\exists a)\ (subartefato(d, a) \wedge artcodigo(a)))$	(A6)

Atividades como Primitivas de Transformação

As atividades de um processo de desenvolvimento de software podem consumir artefatos (insumos) e, através da realização das mesmas, transformá-los em artefatos de saída

(produtos). Desta forma, os artefatos de entrada são, de alguma forma, incorporados aos artefatos de saída (FALBO, 1998). A figura 3.5 mostra essas relações.

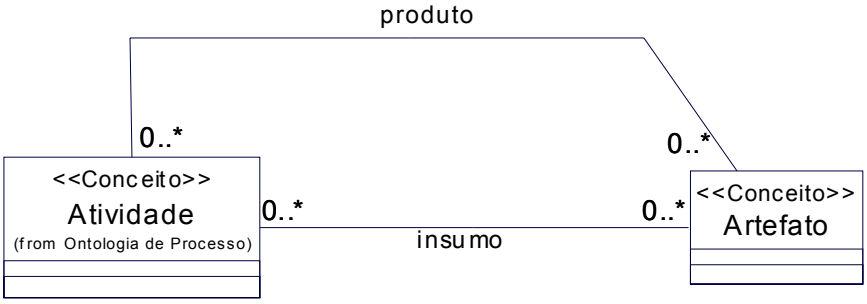


Figura 3.5 - Atividades como Primitivas de Transformação.

As relações *insumo* e *produto*, mostradas na figura 3.5, foram formalizadas pelos predicados *insumo(s,a)*, denotando que o artefato *s* é um insumo para a atividade *a*, e *produto(s,a)*, denotando que o artefato *s* é um produto da atividade *a* (FALBO, 1998). Para garantir sua integridade, os seguintes axiomas de consolidação devem ser observados:

$(\forall a, s)\ (insumo(s, a) \rightarrow artefato(s) \wedge atividade(a))$	(AC2)
$(\forall a, s)\ (produto(s, a) \rightarrow artefato(s) \wedge atividade(a))$	(AC3)

Aprovação de Artefatos

Os artefatos de uma organização evoluem constantemente, através de contínuas alterações. Porém, essas alterações podem fazer com que os artefatos deixem de atender a requisitos e tragam futuros problemas à organização. Assim, no intuito de aprovar as condições de um determinado artefato, são definidos os recursos humanos responsáveis por tal tarefa, como mostrado na figura 3.6.

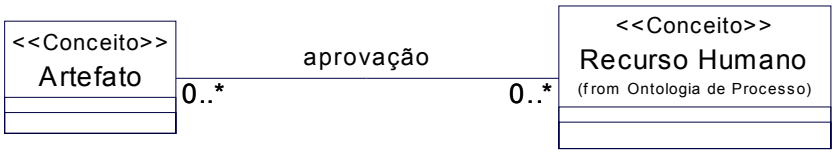


Figura 3.6 - Aprovação de artefatos.

O predicado *aprovação(a, rh)* indica que o recurso humano *rh* é responsável por aprovar o artefato *a* e o seguinte axioma de consolidação tem de ser observado:

$(\forall a, rh)\ (aprovação(a, rh) \rightarrow artefato(a) \wedge recursohumano(rh))$	(AC4)

Dependência entre Artefatos

Um artefato pode estar diretamente relacionado a outros artefatos, estabelecendo, assim, uma relação de dependência do primeiro com os demais. Através dessa relação de dependência, pode-se identificar quais artefatos podem ser impactados quando da ocorrência de mudanças em um artefato específico. A figura 3.7 mostra a relação de dependência entre artefatos.

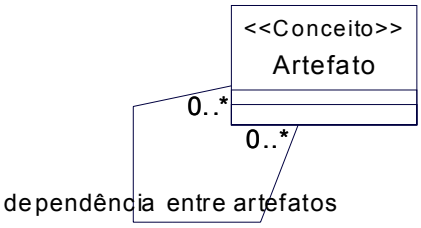


Figura 3.7 - Dependência entre artefatos.

O predicado *art_dependência(a1, a2)* indica que um artefato *a1* depende de um artefato *a2* e o seguinte axioma de consolidação tem de ser observado:

$$(\forall a1, a2)\ (art\_dependência(a1, a2) \rightarrow artefato(a1) \wedge artefato(a2)) \qquad \text{(AC5)}$$

Além disso, é importante notar a validade da transitividade, ou seja, se um artefato depende de um segundo artefato e este segundo artefato depende de um terceiro, então o primeiro depende do terceiro. Essa relação é expressa na sentença abaixo:

$$(\forall a1, a2, a3)\ (art\_dependência(a1, a2) \wedge art\_dependência(a2, a3) \rightarrow art\_dependência(a1, a3)) \qquad \text{(A8)}$$

É importante destacar que, se um artefato depende de outro artefato, seus super-artefatos também dependem de tal artefato. Além disso, um super-artefato depende sempre de seus sub-artefatos.

$$(\forall a1, a2, a3)\ (art\_dependência(a1, a2) \wedge super\text{-}artefato(a3, a1) \rightarrow art\_dependência(a3, a2)) \qquad \text{(A9)}$$

$$(\forall a1, a2)\ (super\text{-}artefato(a2, a1) \rightarrow art\_dependência(a2, a1)) \qquad \text{(A10)}$$

A figura 3.8 mostra o modelo completo da ontologia de artefato e a tabela 3.1 apresenta o dicionário dos termos correspondentes.

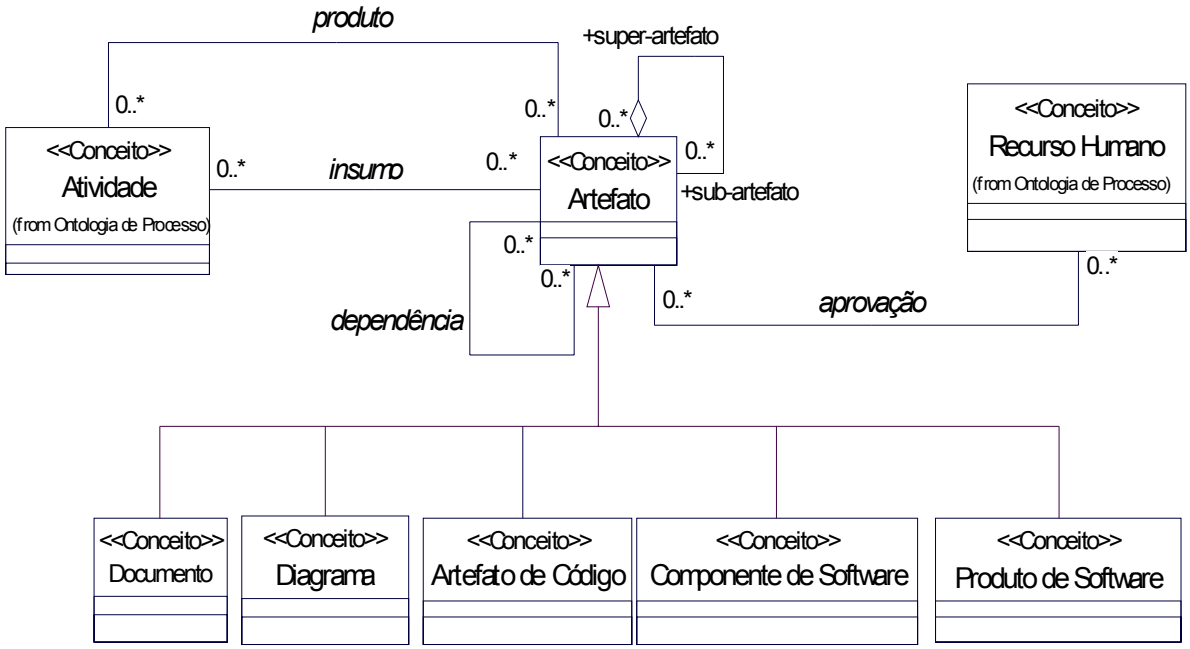


Figura 3.8 - Ontologia de Artefato.

Tabela 3.1 - Dicionário de Termos da Ontologia de Artefato.

Aprovação	Relação entre **recurso humano** e **artefato**, indicando quais **recursos humanos** podem aprovar um determinado **artefato**.
Artefato	Um **insumo** para, ou um **produto** de, uma **atividade**, no sentido de ser um objeto de transformação da atividade. Em função de sua natureza, artefatos podem ser classificados em: **documentos**, **diagramas**, **artefatos de código**, **componentes de software** ou **produtos de software**.
Artefato de Código	Porção de código-fonte, passível de execução, gerada no próprio desenvolvimento. Ex: programas, sub-programas, classes, rotinas, funções etc.
Atividade	Ação que transforma **artefatos** de entrada (**insumos**) em **artefatos** de saída (**produtos**). Ex.: atividade de especificação de requisitos, atividades de testes etc.
Componente de Software	**Artefato** de software que provê um conjunto bem definido de serviços, desenvolvido de forma independente e manipulado (usado, vendido, mantido em uma biblioteca, etc.) como uma unidade coerente, que pode ser combinado para formar artefatos maiores, oferecendo uma interface bem definida e explícita. Ex.: classes, frameworks, padrões gerativos etc.
Dependência entre artefatos	Relação entre **artefatos**, indicando quais **artefatos** dependem de um determinado **artefato**. Assim, pode-se identificar quais **artefatos** podem ser impactados quando da ocorrência de mudanças em um **artefato** específico.

Tabela 3.1 - Dicionário de Termos da Ontologia de Artefato (Continuação).

Diagrama	**Artefato** gráfico, não passível de execução, que consiste em uma apresentação gráfica de um conjunto de **elementos de modelo**, em geral representada como um gráfico conectado de vértices (**elementos de modelo**) e arcos (**relacionamentos**). Ex.: diagrama de casos de uso, diagrama de classes, diagrama de entidades e relacionamentos, diagrama relacional etc.
Documento	**Artefato** de software não passível de execução, constituído tipicamente de declarações textuais, normalmente associado a padrões organizacionais (**roteiros**) que definem a forma em que eles devem ser produzidos. Ex.: documento de especificação de requisitos, plano de projeto, plano de qualidade, relatório de avaliação da qualidade etc.
Insumo	Relação entre um **artefato** e uma **atividade**, indicando que o **artefato** é utilizado como "matéria-prima" pela **atividade**, sendo de alguma forma incorporado a outro **artefato**, o **produto** da **atividade**.
Produto	Relação entre um **artefato** e uma **atividade**, indicando que o **artefato** é produzido pela **atividade**.
Produto de software	**Artefato** resultante de um processo de desenvolvimento de software concluído e entregue ao usuário, ou seja, colocado em operação. Inclui o conjunto de todos **artefatos** gerados no desenvolvimento, além das ferramentas e softwares necessários para colocar o produto em operação. Em um ciclo de vida seqüencial, qualquer alteração necessária em um produto de software se dará em uma **atividade** de manutenção e não mais em uma **atividade** de desenvolvimento do software. Em um ciclo de vida iterativo, porém, alterações podem significar a criação de **variações** de um produto de software.
Recurso Humano	Agente humano necessário para a realização de uma **atividade**. Ex: engenheiro de software, programador, especialista de domínio etc.
Sub-Artefato	Papel da relação de agregação entre dois **artefatos** s_1 e s_2. Se s_2 é parte de s_1, então s_2 é dito um sub-artefato de s_1.
Super-Artefato	Papel da relação de agregação entre dois **artefatos** s_1 e s_2. Se s_1 é decomposto em outros artefatos, dentre eles s_2, então s_1 é dito um super-artefato de s_2.

3.4 Ontologia de Documento

Documentos são muito necessários em uma organização, uma vez que descrevem textualmente diversos aspectos relevantes no desenvolvimento de software, vindo a servir como fonte de consulta em eventos futuros, como forma de aprovação de uma determinada atividade, para maiores esclarecimentos acerca de um certo assunto, diminuindo, assim, divergências, ou mesmo como comprovação de uma certa questão.

Porém, documentos produzidos sem um critério ou padrão de organização são também um problema, pois a consulta, interpretação ou reutilização dos mesmos se torna muito difícil

e dispendiosa. Desta forma, documentos devem estar associados a padrões organizacionais que definam a forma como eles devem ser produzidos, facilitando futuras referências a estes.

Para tratar estes aspectos específicos de documentos, uma ontologia de documentos foi desenvolvida.

3.4.1 Identificação de Propósito e Especificação de Requisitos

Para tratar aspectos relacionados à documentação de software, uma ontologia de documento deve ser capaz de responder às seguintes questões de competência:

1. A quais documentos um roteiro se aplica?
2. Qual a estrutura definida por um modelo de documento?
3. Qual a estrutura de um documento?
4. Um documento está aderente a um modelo de documento?

3.4.2 Captura e Formalização da Ontologia

As questões de competência anteriormente relacionadas nos levam aos seguintes aspectos a serem tratados pela ontologia de documentos:

- estrutura de documentos (questão 3);
- aplicação de roteiros e estrutura de modelos de documentos (questões 1 e 2);
- aderência de documentos a modelos de documento (questões 4).

Todos os aspectos acima relacionados mostram a interação entre a sub-ontologia de documento e a sub-ontologia de procedimentos, definida como parte da ontologia de processo de software (FALBO, 1998), indicando, novamente, a integração da ontologia aqui desenvolvida com a ontologia de processo de software.

Estrutura de documentos

Documentos normalmente não se apresentam apenas constituídos de descrições textuais contínuas e sim através de uma estrutura bem definida, composta por seções, como mostra a figura 3.9.

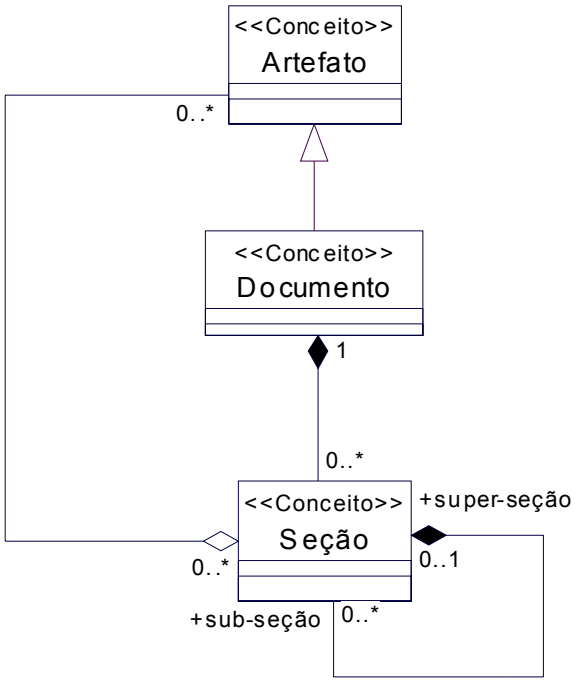


Figura 3.9 – Estrutura de documentos.

Seções são tipicamente constituídas de declarações textuais, de outras seções ou de outros artefatos.

Como uma seção pode ser decomposta em outras seções, surgem os papéis de super-seção e sub-seção. Uma *super-seção* é uma seção composta de outras seções, enquanto uma *sub-seção* é uma seção que compõe outra seção.

Para formalizar os conceitos de seção e dos seus papéis na decomposição de seções, definimos os predicados *seção(s)*, denotando que *s* é uma seção, *subseção(s2, s1),* denotando que *s2* é uma sub-seção de *s1*, e *superseção(s2, s1)*, denotando que *s2* é uma super-seção de *s1*. Vale ressaltar que axiomas análogos aos definidos na agregação de artefatos concernentes a transitividade (AE6) e assimetria (AE7) são válidos para seção, já que a decomposição de seções, assim como toda relação todo-parte, é transitiva e assimétrica.

É importante destacar que, se uma seção pertence a um determinado documento, suas sub-seções também pertencem a tal documento. E que, além disso, se um artefato é parte de uma seção, tal artefato também é parte das super-seções de tal seção. Os axiomas referentes a tais afirmações são mostrados abaixo, em que o predicado *doc_composição(d, s)* indica que o documento *d* é composto pela seção *s e* o predicado *sec_art_composição(s, a)*, indica que a seção *s* é composta pelo artefato *a*:

$$(\forall s1, s2, d)\ (doc\_composição(d, s1) \wedge subseção(s2, s1) \rightarrow doc\_composição(d, s2)) \quad \text{(A11)}$$

$$(\forall a, s1, s2)\ (sec\_art\_composição(s1, a) \wedge superseção(s2, s1) \rightarrow sec\_art\_composição(s2,a)) \quad \text{(A12)}$$

Além disso, se uma seção *s* de um documento *d* é composta de um artefato *a*, então o artefato *a* é sub-artefato do documento *d*, conforme mostra o seguinte axioma:

$$(\forall s,d, a)\ (doc\_composição(d, s) \wedge sec\_art\_composicao(s, a) \rightarrow subartefato(a, d)) \quad \text{(A13)}$$

Aplicação de roteiros e estrutura de modelos de documentos

Roteiros são utilizados como diretrizes para a construção de documentos, almejando, assim, uma padronização e facilitando o acesso a tais documentos. A figura 3.10 mostra este fato.

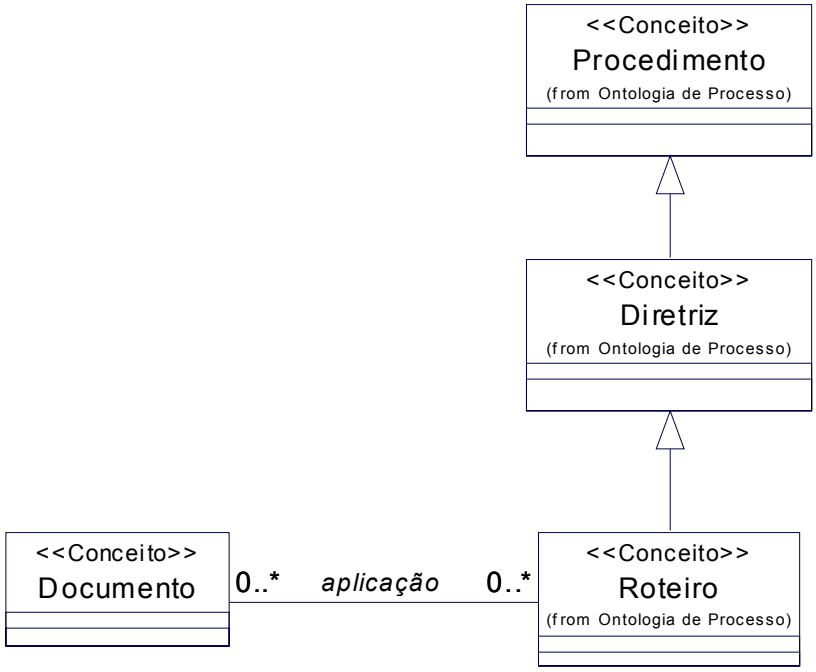


Figura 3.10 – Aplicação de roteiros.

Para formalizar o conceito de roteiro e a relação entre documento e roteiro, foram definidos, respectivamente, o predicado *roteiro(r)*, denotando que *r* é um roteiro (FALBO, 1998) e o predicado *aplicação(r, d)*, denotando que o roteiro *r* se aplica ao documento *d*. O seguinte axioma de consolidação tem de ser observado:

$(\forall r, d)$ (aplicação(r, d) $\rightarrow$ roteiro(r) $\wedge$ documento(d))	(AC6)

A construção de documentos pode ser feita com base em roteiros, porém existe um tipo especial de roteiro que estabelece a estrutura de um documento e instruções para sua elaboração, que são os modelos de documentos. Os modelos de documentos são estruturados com base em modelos de seção, como mostra a figura 3.11.

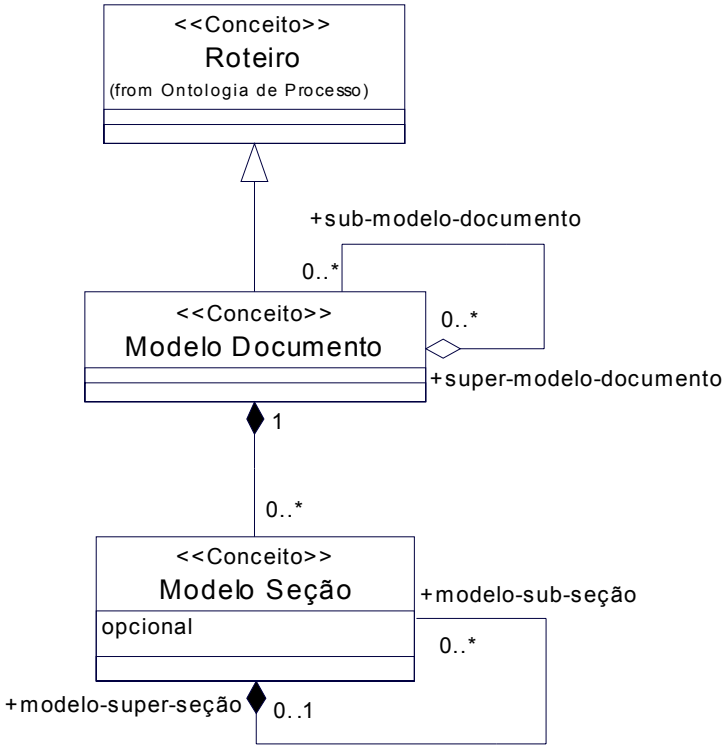


Figura 3.11 – Estrutura de modelo de documento.

Documentos são artefatos que podem compostos por outros artefatos. Analogamente, um modelo de documento pode ser composto de outros modelos de documentos. Assim, surgem os papéis *super-modelo-documento*, que caracteriza um modelo de documento constituído de outros modelos de documento, e *sub-modelo-documento*, que caracteriza um modelo de documento que compõe outro modelo de documento.

Para formalizar os conceitos de modelo de documento e dos seus referentes papéis nessa relação, foram definidos os predicados *modelodocumento(md)*, denotando que *md* é um modelo de documento, *submodelodocumento(md2, md1),* denotando que *md2* é um sub-modelo-documento de *md1*, e *super-modelo-documento(md2, md1)*, denotando que *md2* é um super-modelo-documento de *md1*. Vale lembrar que os axiomas da relação todo-parte do tipo agregação são válidos para essa relação.

Modelos de seção são parte da estrutura de modelos de documento e estabelecem um padrão para construção de seções de tais modelos de documento. Desta forma, definem instruções para a elaboração de seções de documentos aderentes aos respectivos modelos de documento.

Um modelo de seção possui a característica de poder ser, ou não, opcional. Um modelo de seção opcional é aquele que não necessita de uma seção correspondente nos documentos que aplicam o modelo de documento ao qual pertence, enquanto um modelo de seção obrigatório necessita de uma seção correspondente para ele.

Da mesma forma que as seções, um modelo de seção pode ser decomposto em outros modelos de seções. Assim, surgem os papéis *modelo-super-seção*, que caracteriza um modelo de seção constituído de outros modelos de seção e *modelo-sub-seção*, que caracteriza um modelo de seção que compõe outro modelos de seção.

Para formalizar os conceitos de modelo de seção e dos seus referentes papéis de decomposição, foram definidos os predicados *modeloseção(ms)*, denotando que *ms* é um modelo de seção, *submodeloseção(ms2, ms1),* denotando que *ms2* é um modelo-sub-seção de *ms1*, e *supermodeloseção(ms2, ms1)*, denotando que *ms2* é um modelo-super-seção de *ms1*.

É importante destacar que, se um modelo de seção pertence a um determinado modelo de documento, seus modelos-sub-seções também pertencem a tal modelo de documento. O axioma referente a tal afirmação é mostrado abaixo, onde o predicado *mdoc_composição(md, ms)* indica que o modelo de documento *md* é composto pelo modelo de seção *ms*:

$$(\forall ms1, ms2, md)\ (mdoc\_composição(md, ms1) \wedge (submodeloseção(ms2, ms1) \rightarrow mdoc\_composição(md, ms2)) \quad \text{(A14)}$$

Aderência de documentos a modelos de documento

Apesar dos modelos de documentos serem roteiros que estabelecem a estrutura de um documento e instruções para sua elaboração, nada garante que um documento criado com base em um determinado modelo de documento esteja aderente a ele.

Para que um documento esteja aderente a um modelo de documento, deve possuir uma seção correspondente para cada modelo de seção não opcional do modelo de documento. Tal correspondência é mostrada na figura 3.12 através da relação entre seção e modelo-seção.

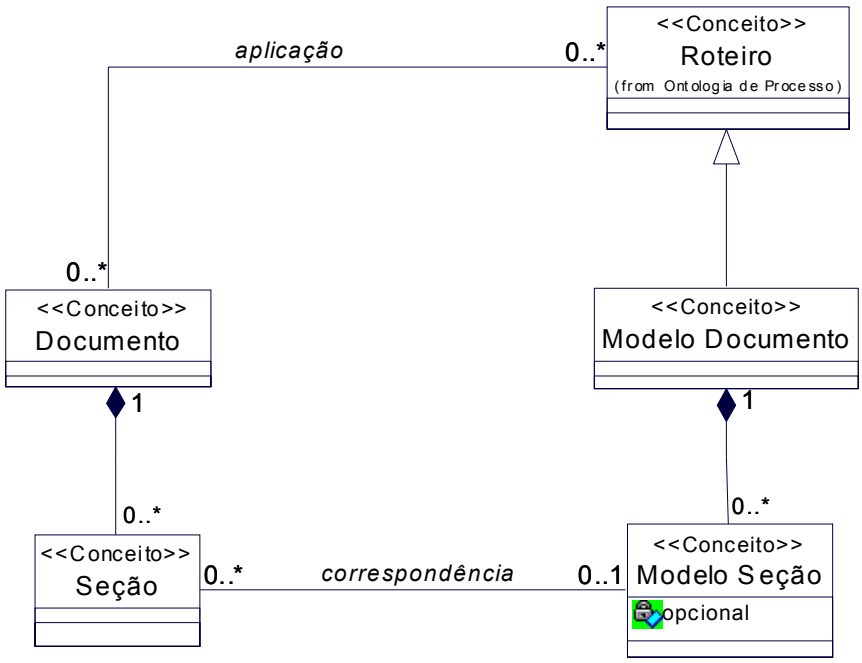


Figura 3.12 – Aderência de documentos.

Tal situação pode ser expressa através do axioma abaixo, em que são utilizados os predicados *correspondência(s, ms)*, que denota que a seção *s* corresponde ao modelo de seção *ms*, *opcional(ms)*, que denota que o modelo de seção *ms* é opcional, e *aderente(d, md)*, que denota que o documento *d* está aderente ao modelo de documento *md*.

$$(\forall d, md)\ aderente(d, md) \leftrightarrow ((\forall ms)(\ mdoc\_composição(md, ms) \wedge \neg opcional(ms)) \rightarrow ((\exists s)\ (doc\_sec\_composição(d, s) \wedge correspondência(s, ms)) \quad \text{(A15)}$$

Deve-se observar que a relação de correspondência tem de satisfazer o seguinte axioma de consolidação: Uma seção só pode corresponder a um modelo de seção, se fizer parte de um documento que está aplicando o modelo de documento do qual faz parte o correspondente modelo de seção.

$$(\forall s, ms)\ correspondência(s, ms) \rightarrow (\exists d, md)\ (doc\_sec\_composição(d, s) \wedge mdoc\_composição(md, ms) \wedge\ aplicação(d, md))) \quad \text{(AC7)}$$

A figura 3.13 mostra o modelo completo da ontologia de documento e a tabela 3.2 apresenta o dicionário dos termos básicos correspondentes.

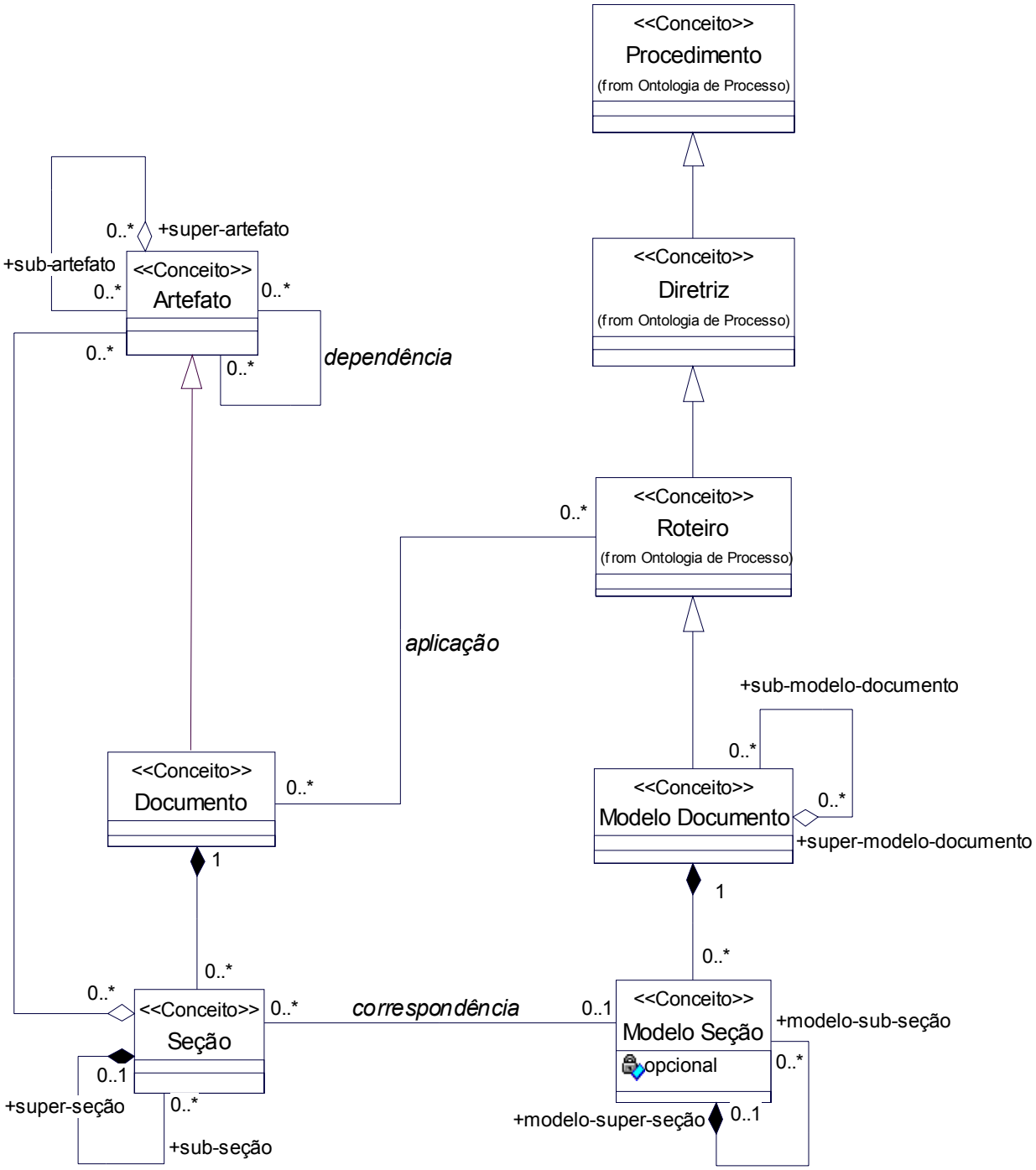


Figura 3.13 – Ontologia de documentos.

Tabela 3.2 - Dicionário de Termos da Ontologia de Documentos.

Aplicação	Relação entre **documento** e **roteiro**, indicando quais **roteiros** podem ser utilizados na elaboração de um determinado **documento**.
Correspondência	Relação entre **seção** e **modelo de seção**, indicando o **modelo de seção** que corresponde a uma determinada **seção**.
Diretriz	**Procedimento** que visa estabelecer um padrão para a realização de **atividades**. Diretrizes podem ser divididas em **roteiros** e **normas**.
Modelo Documento	Tipo de **roteiro** que estabelece a estrutura de um **documento** e instruções para sua elaboração. A estrutura é definida na forma de **modelos de seção**. Ex.: modelo de documento de plano de projeto, modelo de relatório de avaliação da qualidade etc.
Modelo Seção	Parte da estrutura de um **modelo de documento**, que estabelece um padrão para construção de **seções** desse **modelo de documento**. Define instruções para a elaboração de uma **seção** de um **documento** aderente a ele. Um modelo de seção pode ser **opcional**, quando não necessitar de uma **seção** correspondente nos **documentos** que aplicam o **modelo de documento**, ou obrigatório, quando há necessidade de uma **seção** correspondente para ele. Ex.: Modelo de Introdução de um **modelo de documento** para elaboração de plano de projeto.
Modelo-sub-seção	Papel da relação de composição entre dois **modelos de seção** m_1 e m_2. Se m_2 é parte de m_1 então m_2 é dito um modelo-sub-seção de m_1.
Modelo-super-seção	Papel da relação de composição entre dois **modelos de seção** m_1 e m_2. Se m_1 é decomposto em outros **modelos de seção**, dentre eles m_2, então m_1 é dito um modelo-super-seção de m_2.
Procedimento	Conduta bem estabelecida e ordenada para a realização de uma **atividade**. Quanto à sua natureza, procedimentos podem ser classificados em **métodos**, **técnicas** e **diretrizes**.
Roteiro	**Diretriz** para a elaboração de **documentos**. Ex.: roteiro de plano de projeto.
Seção	Parte da estrutura de um **documento**, constituída tipicamente de declarações textuais, de outras **seções** ou de **diagramas**. Ex.: seção de introdução, seção de referências bibliográficas etc.
Sub-modelo-documento	Papel da relação de composição entre dois **modelos de documento** d_1 e d_2. Se d_2 é parte de d_1, então d_2 é dito um sub-modelo-documento de d_1.
Super-modelo-documento	Papel da relação de composição entre dois **modelos de documento** d_1 e d_2. Se d_1 é decomposto em outros **modelos de documento**, dentre eles d_2, então d_1 é dito um super-modelos-documento de d_2.
Sub-seção	Papel da relação de composição entre duas **seções** s_1 e s_2. Se s_2 é parte de s_1, então s_2 é dita uma sub-seção de s_1.
Super-seção	Papel da relação de composição entre duas **seções** s_1 e s_2. Se s_1 é decomposta em outras **seções**, dentre eles s_2, então s_1 é dita uma super-seção de s_2.

3.5 Ontologia de Artefato de Código

Em qualquer organização em que ocorre a produção de sistemas computacionais, estão presentes os artefatos de código, que são porções de código-fonte, passíveis de execução, geradas no próprio desenvolvimento. Exemplos de artefatos de código incluem funções, classes, procedimentos, rotinas e programas. Assim, é interessante definir uma sub-ontologia de artefato de código que trate dos aspectos específicos dessa categoria de artefatos.

3.5.1 Identificação de Propósito e Especificação de Requisitos

Podemos enumerar as seguintes questões de competência relativas a artefatos de código:

1. Qual a linguagem adotada na confecção de um artefato de código?
2. Quais os paradigmas de uma determinada linguagem?

3.5.2 Captura e Formalização da Ontologia

As questões de competência anteriormente relacionadas nos levam aos seguintes aspectos a serem tratados pela ontologia de código:

- Adoção de linguagem de programação (questão 1);
- Conformidade em relação a paradigma (questão 2).

O segundo aspecto mostra a interação entre a sub-ontologia de artefato de código e a ontologia de processo de software (FALBO, 1998), que define o conceito de paradigma.

Adoção de linguagem de programação

Artefatos de código são escritos em determinadas linguagens de programação, das quais se pode citar como exemplos Java, C, SQL e Script Unix. A adoção de linguagens de programação por artefatos de código é mostrada na figura 3.14.

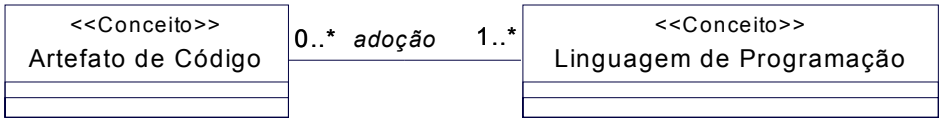


Figura 3.14 – Adoção de linguagem de programação.

Para formalizar o conceito de linguagem de programação e a relação entre artefato de código e linguagem de programação, foram definidos, respectivamente, o predicado *lingprogramação(lp)*, denotando que *lp* é uma linguagem de programação e o predicado *adoção(a, lp)*, denotando que o artefato de código *a* está escrito na linguagem de programação *lp*. O seguinte axioma de consolidação tem de ser observado:

$$(\forall a, lp)\ (adoção(a, lp) \rightarrow artcódigo(a) \wedge lingprogramação(lp)) \qquad \text{(AC8)}$$

Conformidade em relação a paradigma

Cada linguagem de programação adota certas filosofias na construção de um software, que abrangem um conjunto de princípios e conceitos que norteiam o desenvolvimento. Tais filosofias são chamadas de paradigmas e alguns exemplos incluem o paradigma estruturado e o paradigma orientado a objetos. A figura 3.15 mostra esta relação entre paradigma e linguagem de programação.

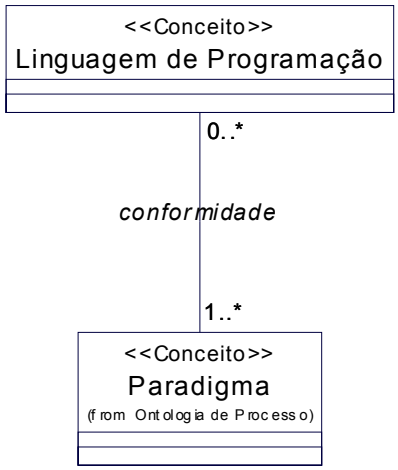


Figura 3.15 – Conformidade em relação a paradigma.

Para formalizar o conceito de paradigma e sua relação com linguagem de programação, foram definidos, respectivamente, o predicado *paradigma(p)* (FALBO, 1998), denotando que *p* é um paradigma e o predicado *conformidade(lp, p)*, denotando que a linguagem de programação *lp* está em conformidade com o paradigma *p*. O seguinte axioma de consolidação tem de ser observado:

$$(\forall lp, p)\ (conformidade(lp, p) \rightarrow lingprogramação(lp)) \wedge paradigma(p)) \qquad \text{(AC9)}$$

A figura 3.16 mostra o modelo completo da ontologia de artefato de código e a tabela 3.3 apresenta o dicionário dos termos básicos correspondentes.

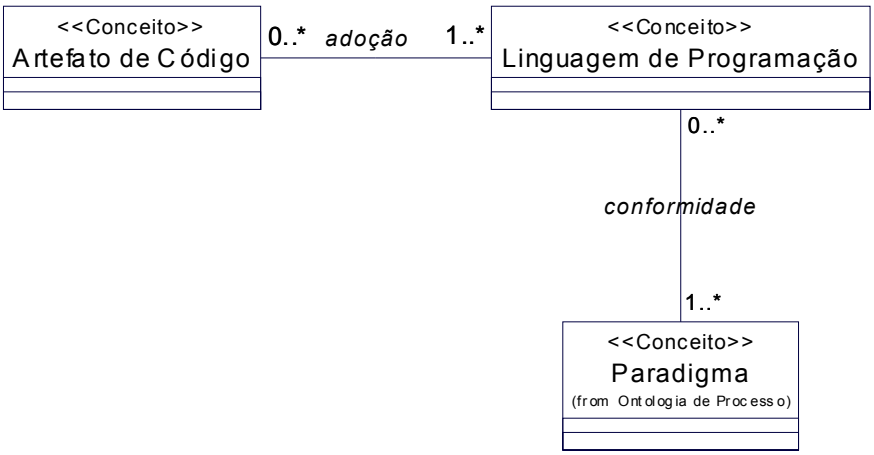


Figura 3.16 – Ontologia de artefato de código.

Tabela 3.3 - Dicionário de Termos da Ontologia de Código.

Adoção	Relação entre **artefato de código** e **linguagem de programação**, indicando em quais **linguagens** um determinado **artefato de código** foi escrito.
Conformidade	Relação entre **linguagem de programação** e **paradigma**, indicando quais os **paradigmas** suportados por uma linguagem de programação.
Linguagem de programação	Linguagem de programação que pode ser utilizada na construção de um **artefato de código**. Ex.: Java, C, SQL, Script Unix, etc.
Paradigma	Filosofia adotada na construção do software, abrangendo um conjunto de princípios e conceitos que norteiam o desenvolvimento. Ex.: paradigma estruturado, paradigma orientado a objetos etc (FALBO, 1998).

3.6 Ontologia de Diagrama

Certamente, a confecção de um sistema computacional não se dá apenas através de artefatos de códigos e de documentos. Apesar deles serem de fundamental importância, são igualmente importantes os diagramas, usados para auxiliar diversas atividades do ciclo de vida do desenvolvimento. Nas atividades de análise e projeto de sistemas, por exemplo, a modelagem através de diagramas é extremamente relevante, uma vez que leva a uma abstração do mundo real, tornando mais fácil a compreensão de determinado assunto. Além disso, diagramas utilizam um formalismo maior do que a linguagem natural, diminuindo, assim, divergências a respeito de um assunto, além de facilitar a comunicação com clientes e usuários finais.

Como já definido, diagramas são artefatos gráficos, não passíveis de execução, que consistem em apresentações gráficas de um conjunto de elementos de modelo, em geral representadas como um gráfico conectado de vértices (elementos de modelo) e arcos

(relacionamentos). Exemplos de diagramas incluem: diagrama de casos de uso, diagrama de classes, diagrama de entidades e relacionamentos, diagrama relacional, entre outros.

3.6.1 Identificação de Propósito e Especificação de Requisitos

Uma ontologia de diagrama tem o objetivo de relacionar as principais características presentes em qualquer diagrama utilizado em desenvolvimento de software e, para tal, foi utilizado como base um subconjunto do meta-modelo da UML (*Unified Modeling Language*) (OMG, 2001), tanto para a definição de conceitos como para a descrição dos termos básicos.

Foram relacionadas as seguintes questões de competência:

1. Quais elementos de modelo fazem parte de um diagrama?
2. Qual a natureza de um elemento de modelo?
3. Como os diversos tipos de elementos de modelo se relacionam?

3.6.2 Captura e Formalização da Ontologia

Analisando as questões de competência anteriormente relacionadas, identificamos os seguintes aspectos:

- estrutura de diagramas (questão 1)
- taxonomia de elementos de modelo (questão 2)
- relações entre elementos de modelos (questão 3);

Composição de diagramas

Para construir diagramas é utilizado um tipo especial de elemento chamado elemento de modelo. Elementos de modelos são abstrações em forma de desenhos que têm o intuito de facilitar a compreensão humana (OMG, 2001). A figura 3.17 mostra esta relação entre diagramas e elementos de modelo.

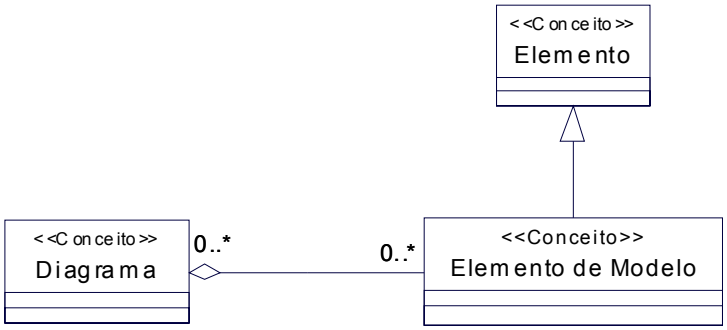


Figura 3.17 – Composição de diagramas.

Os axiomas derivados das relações *todo-parte* e *sub-tipo de*, naturalmente, aplicam-se a este caso, ainda que não tenham sido representados.

Taxonomia de elementos de modelo

Existem vários tipos de elementos de modelos que são utilizados para construir diagramas, dos quais se podem citar os elementos generalizáveis, as propriedades, os relacionamentos e seus respectivos subtipos, como mostra a figura 3.18.

Um elemento generalizável é um elemento de modelo que pode participar em um relacionamento de generalização. Seu tipo principal é um classificador, que permite a descrição de propriedades, como, por exemplo, classes, interfaces, tipos de dados, etc (OMG, 2001).

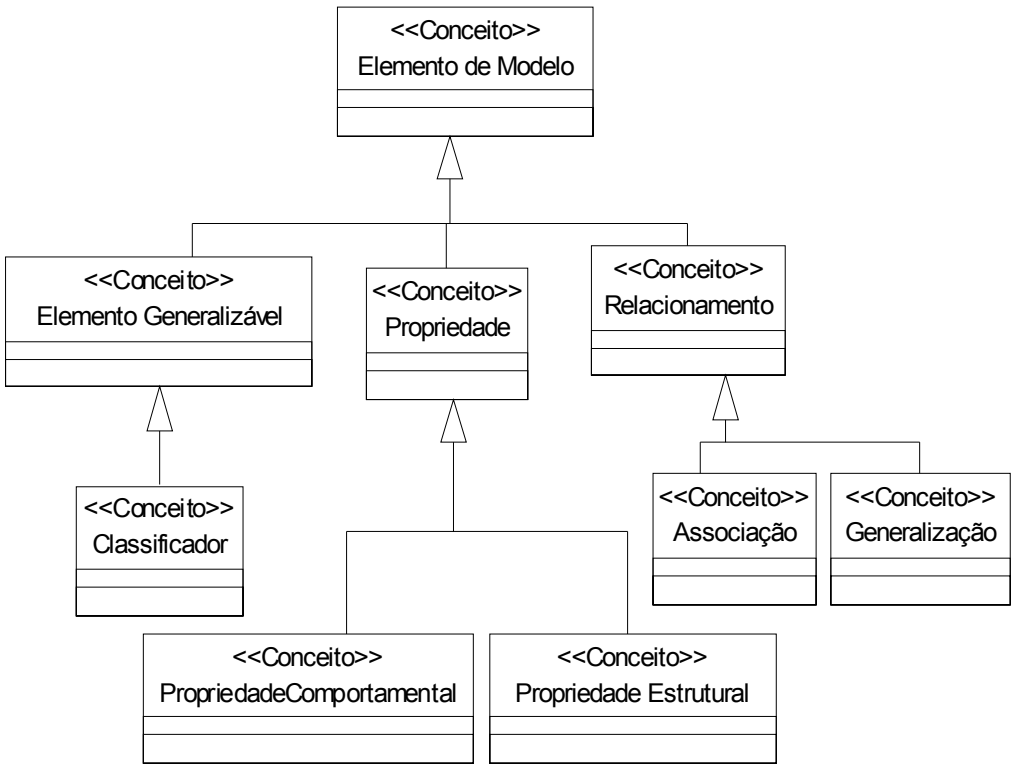


Figura 3.18 – Taxonomia de elemento de modelo.

Uma propriedade pode ser uma propriedade comportamental ou propriedade estrutural. Propriedades comportamentais referem-se às propriedades dinâmicas de um elemento de modelo, como, por exemplo, uma operação de uma classe. Propriedades estruturais referem-se às propriedades estáticas de um elemento de modelo, como, por exemplo, um atributo de uma classe (OMG, 2001).

Um relacionamento é uma conexão semântica entre elementos de modelo e, basicamente, pode ser de dois tipos: generalização e associação. Uma generalização é um relacionamento taxonômico entre um elemento de modelo mais geral e um mais específico. O elemento de modelo mais específico é completamente consistente com o elemento de modelo mais geral (ele possui todas suas propriedades, membros e relacionamentos) e pode conter informações adicionais (OMG, 2001).

Uma associação define um relacionamento semântico entre dois ou mais classificadores. O conjunto de instâncias de uma associação é o conjunto de tuplas relacionando instâncias dos classificadores envolvidos na associação (OMG, 2001).

Relações entre elementos de modelos

Diversos elementos de modelo que constituem um diagrama não se encontram isolados e sim conectados a outros elementos de modelo. Porém cada tipo de elemento de modelo se relaciona, tipicamente, com determinados tipos de elemento de modelo.

A figura 3.19 mostra que uma propriedade é uma característica de um classificador. Pode-se citar como exemplo, um atributo (propriedade) de uma classe (classificador). Além disso, toda propriedade estrutural tem um classificador como tipo. Pode-se citar como exemplo, um atributo (propriedade estrutural) cujo tipo seja uma classe (classificador).

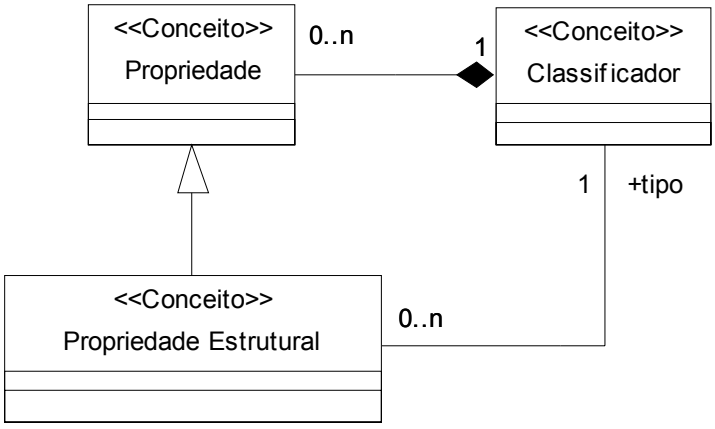


Figura 3.19 – Relações entre propriedade e classificador.

A figura 3.20 mostra que toda generalização possui um elemento generalizável como pai e outro como filho. Filho é o papel da relação entre elemento generalizável e generalização, designando que o elemento generalizável é uma especialização de outro elemento generalizável. Pai é o papel da relação entre elemento generalizável e generalização, designando que o elemento generalizável é uma generalização de outro elemento generalizável.

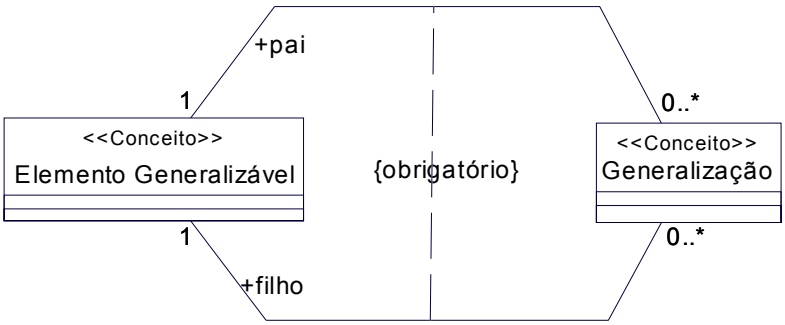


Figura 3.20 – Relações entre elemento generalizável e generalização.

Para formalizar os conceitos de pai e filho, foram definidos os predicados *pai(e, g)*, denotando que *e* é o elemento generalizável pai de *g* e *filho(e, g)*, denotando que *e* é o elemento generalizável filho de *g*.

Deve-se lembrar que um elemento generalizável não pode ser ao mesmo tempo pai e filho de uma mesma generalização, como mostra o axioma abaixo.

$(\forall e, g)\ (pai(e, g) \rightarrow \neg filho(e, g))$	(A16)
$(\forall e, g)\ (filho(e, g) \rightarrow \neg pai(e, g))$	(A17)

Neste momento pode-se introduzir o conceito de subtipo, formalizado pelo predicado *subtipo(e2, e1)*, que indica que *e2* é subtipo de *e1*. Assim, se um elemento generalizável *e1* desempenha o papel de pai em uma generalização *g* e outro elemento generalizável *e2* desempenha o papel de filho nessa mesma generalização *g*, então *e2* é dito subtipo de *e1*. Além disso, se *e2* é subtipo de *e1* e *e3* é subtipo de e2, tem-se que *e3* é subtipo de *e1*. Essas restrições são mostradas através dos seguintes axiomas:

$(\forall e1, e2, g)\ pai(e1, g) \wedge filho(e2, g) \rightarrow subtipo(e2, e1)$	(A18)
$(\forall e1, e2, e3)\ subtipo(e2, e1) \wedge subtipo(e3, e2) \rightarrow subtipo(e3, e1)$	(A19)

Por fim, a figura 3.21 mostra que uma associação se dá entre classificadores, indicando que em uma associação, há no mínimo dois classificadores em suas extremidades. Pode-se citar como exemplo, o relacionamento (associação) entre duas entidades (classificadores) em um modelo de entidades e relacionamentos.

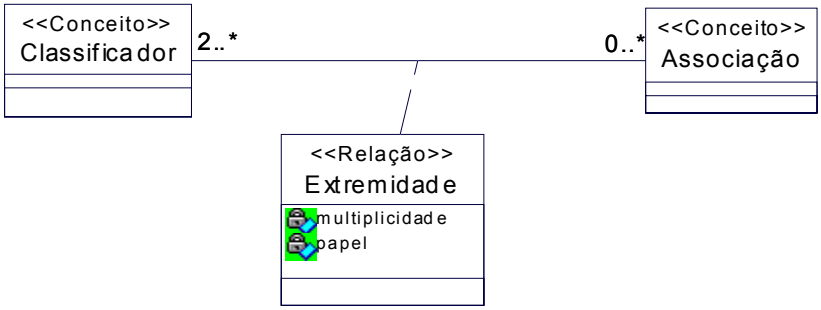


Figura 3.21 – Relação entre classificador e associação.

A figura 3.22 mostra o modelo completo da ontologia de diagrama e a tabela 3.4 apresenta o dicionário dos termos básicos correspondentes.

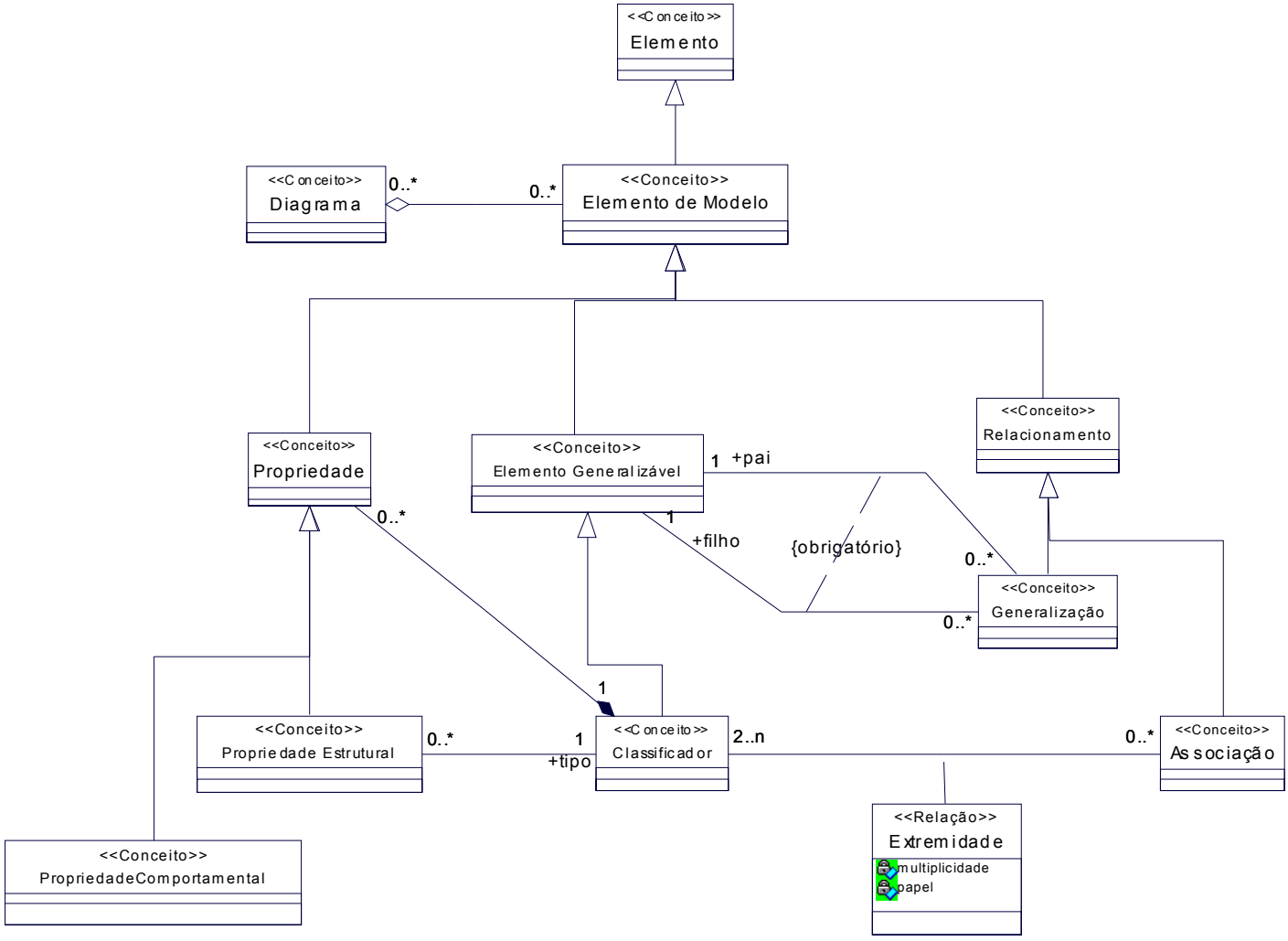


Figura 3.22 – Ontologia de diagrama.

Tabela 3.4 - Dicionário de Termos da Ontologia de Diagrama.

Associação	Define um **relacionamento** semântico entre dois ou mais **classificadores**. O conjunto de instâncias de uma associação é o conjunto de tuplas relacionando instâncias dos **classificadores** envolvidos na associação.
Classificador	**Elemento generalizável** que descreve **propriedades**. Ex.: classe, interface, tipo de dado etc.
Elemento	Constituinte atômico de um modelo.
Elemento de modelo	**Elemento** que é uma abstração em forma de um desenho para facilitar a compreensão humana.
Elemento Generalizável	**Elemento de modelo** que pode participar em de um **relacionamento** de **generalização**.
Extremidade	Relação entre **classificador** e **associação** indicando os **classificadores** que se conectam através de uma **associação**. A **multiplicidade** de uma **extremidade** indica com quantos elementos do **classificador** a **associação** pode estar relacionada. O **papel** de uma **extremidade** indica a regra de associação com o **classificador**. Ex.: No contexto de uma locadora de vídeo, extremidade da associação "alugar" com a classe (classificador) "cliente", que possui multiplicidade "0..n" e papel "locatário".
Filho	Papel da relação entre **elemento generalizável** e **generalização**, designando que o **elemento generalizável** é uma especialização de outro **elemento generalizável**.
Generalização	É um **relacionamento** taxonômico entre um **elemento de modelo** mais geral e um mais específico. O **elemento de modelo** mais específico é completamente consistente com o **elemento de modelo** mais geral (ele possui todas suas propriedades, membros e relacionamentos) e pode conter informações adicionais.
Pai	Papel da relação entre **elemento generalizável** e **generalização**, designando que o **elemento generalizável** é uma generalização de outro **elemento generalizável**.
Propriedade	É uma característica de um **classificador**.
Propriedade Comportamental	Refere-se a uma **propriedade** dinâmica de um **elemento de modelo**. Ex.: operação de uma classe.
Propriedade Estrutural	Refere-se a uma **propriedade** estática de um **elemento de modelo**. Ex.: atributo de uma classe.
Relacionamento	É uma conexão semântica entre **elementos de modelo**.
Tipo	Papel da relação entre **classificador** e **propriedade estrutural** que designa o **classificador** de quem instâncias são valores das **propriedades estruturais**.

3.7 Ontologia de Gerência de Configuração de Software

Durante o processo de desenvolvimento de um software, vários artefatos são produzidos e alterados constantemente. Ferramentas de software, tais como compiladores e editores de texto, também podem ser substituídos por versões mais recentes de seus fabricantes ou mesmo por outras. Assim, para que não haja inconsistência nos artefatos e ferramentas utilizados, é de suma importância o acompanhamento e controle de tais itens,

através de um processo de gerência de configuração de software, durante todo o ciclo de vida do software (SANCHES, 2001b).

Conforme discutido no capítulo 2, a Gerência de Configuração de Software (GCS) compreende o conjunto de atividades para controlar modificações, identificando os produtos de trabalho que podem ser modificados, estabelecendo as relações entre eles e os mecanismos para administrar suas diferentes versões ou variantes, controlando as modificações, fazendo auditoria e preparando relatórios sobre tais modificações (PRESSMAN, 2001).

3.7.1 Identificação de Propósito e Especificação de Requisitos

Tendo em vista o escopo da gerência de configuração de software, pode-se listar as seguintes questões de competência para a ontologia de gerência de configuração de software:

1. Quais itens (artefatos ou ferramentas de software) estão sob gerência de configuração?
2. Quais as variações (versões/variantes) de um item de configuração?
3. Como a variação de um item de configuração se decompõe?
4. Em quais outras variações uma alteração de uma determinada variação poderá provocar impactos?
5. Uma variação de um item de configuração derivado de um artefato está aderente à estrutura (decomposição e dependências) de tal artefato?
6. Quais as variações sujeitas a modificação em uma determinada alteração?
7. Quais as variações produzidas por uma determinada alteração?
8. De que variações de itens de configuração é composta uma determinada linha base?
9. A quais itens de configuração um determinado recurso humano tem acesso? E qual o tipo de acesso?
10. Quem é o responsável por uma determinada alteração?
11. Uma determinada variação de um item de configuração está disponível para ser alterada?

3.7.2 Captura e Formalização da Ontologia

Analisando as questões de competência anteriormente relacionadas, identificamos os seguintes aspectos a serem tratados pela ontologia de GCS:

- itens sob gerência de configuração (questão 1);
- variações de itens de configuração (questão 2);

- decomposição e dependência entre variações (questões 3 a 5);
- alteração de variações (questões 6, 7 e 11);
- formação de linha-base (questão 8);
- responsabilidade de alteração (questão 10);
- acesso a itens de configuração (questão 9).

O primeiro e os dois últimos aspectos da sub-ontologia de GCS estão relacionados à sub-ontologia de recursos da ontologia de processo de software (FALBO, 1998), que define os conceitos de ferramenta de software e recurso humano.

Itens sob Gerência de Configuração

Durante o processo de desenvolvimento de software, vários itens de informação são constantemente alterados. Assim, para que não haja inconsistência nos itens mais importantes para o projeto, eles devem ser acompanhados e controlados por um processo de gerência de configuração. Desta forma, esses itens passam a estar sob gerência de configuração e são chamados de itens de configuração de software.

Um item de configuração de software é, então, um item de software que está sob gerência de configuração e, assim, só pode ser alterado segundo um procedimento de controle de alteração formalmente estabelecido e documentado. Pode ser uma ferramenta de software ou um artefato, como, por exemplo, um determinado plano de projeto ou um certo artefato de código. A figura 3.23 mostra o modelo ontológico correspondente.

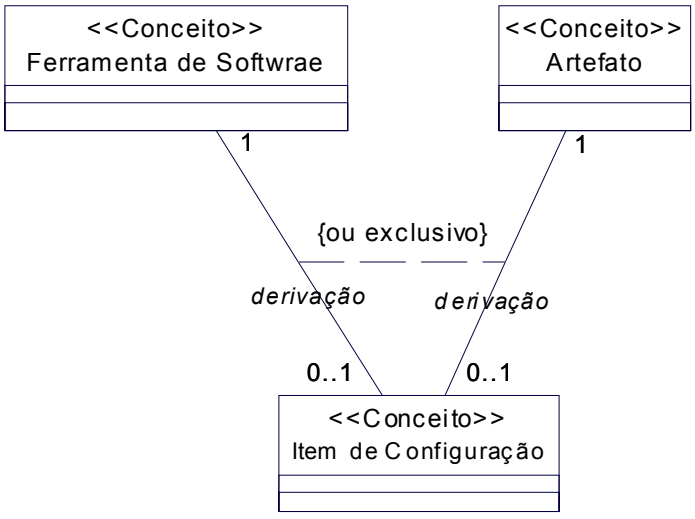


Figura 3.23 – Itens sob gerência de configuração.

Ferramentas de software são recursos de software utilizados para (semi-) automatizar procedimentos adotados na realização de atividades, como por exemplo, editores de textos,

planilhas eletrônicas, compiladores, ferramentas CASE etc (FALBO, 1998). São muito importantes de serem consideradas como itens de configuração, pois são utilizadas para produzir os artefatos de um projeto e precisam estar disponíveis quando forem feitas alterações nesses artefatos.

Para formalizar o conceito de derivação, é definido o predicado *derivação(i, af)*, que denota que *i* é um item de configuração derivado de um item *af*, que pode ser uma ferramenta de software ou um artefato. Ou, em outras palavras, denota que o item *af* está sob gerência de configuração, se tornando, assim, o item de configuração *i*. O seguinte axioma de consolidação tem de ser observado, onde *artefato(af)* indica que *af* é um artefato e *itemconfiguração(af)* indica que *af* é um item de configuração:

$$(\forall af, i)\ (derivação(i,af) \rightarrow itemconfiguração(i) \wedge (artefato(af) \vee ferramentasoftware(af)) \quad \text{(AC10)}$$

Variações de Itens de Configuração

Quando um item de configuração é desenvolvido, ocorre uma evolução natural decorrente das diversas alterações, até que se atinja um estado em que esse atenda aos propósitos para o qual foi criado. A cada estado, é gerada uma nova versão do item. Para estabelecer o controle sobre as diversas versões, elas devem ser armazenadas e identificadas (SANCHES, 2001b).

Quando, porém, um item existe simultaneamente em duas ou mais formas diferentes que atendam a requisitos similares, tem-se *variantes* deste item (SANCHES, 2001b).

Uma variação de um item de configuração indica, então, a versão ou variante desse item de configuração. É caracterizada por um número único para cada variação de um mesmo item de configuração. Como exemplo, um diagrama de casos de uso submetido à gerência de configuração pode possuir três variações (1.0, 2.0 e 2.0.1). A figura 5.2 mostra as variações de itens de configuração e seus tipos. Vale ressaltar que todos os axiomas válidos para relações *sub-tipo de* aplicam-se para a taxonomia de variação.

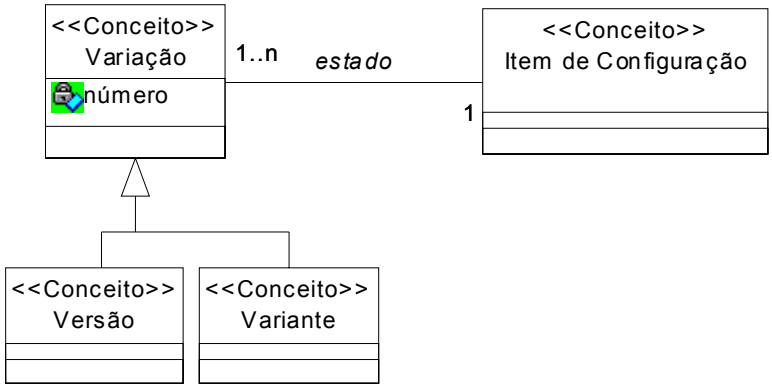


Figura 3.24 – Variações de itens de configuração.

Para formalizar o conceito de estado, é definido o predicado *estado(v, i)*, indicando que *v* é uma variação do item de configuração *i*. O seguinte axioma de consolidação tem de ser observado, onde *variação(v)* indica que *v* é uma variação:

$$(\forall v, i)\ (estado(v,i) \rightarrow variação(v) \wedge itemconfiguração(i)) \qquad \text{(AC11)}$$

Decomposição e Dependência entre Variações

Da mesma forma que ocorre com os artefatos, variações podem ser decompostas em outras variações, surgindo os conceitos de sub-variação (variação que compõe outra variação) e super-variação (variação que é composta de outras variações).

Assim, pode-se ter um artefato que esteja sob gerência de configuração, em que os seus sub-artefatos correspondam a variações diferentes. Por exemplo, a versão 3.0 (uma variação) de uma especificação de requisitos (um artefato) possui como sub-variação a variante 1.01 de um diagrama de casos de uso (sub-artefato). Os papéis de sub-variação e super-variação são mostrados na figura 3.25.

O predicado *subvariação(v1, v2)* indica que a variação *v1* é uma sub-variação (ou é parte) da variação *v2* , enquanto o predicado *supervariação (v2 ,v1)* indica que *v2* é uma super-variação de (ou é um todo, em que uma das partes é) *v1*. Vale ressaltar que todos os axiomas válidos para relações todo-parte do tipo agregação aplicam-se para esta relação.

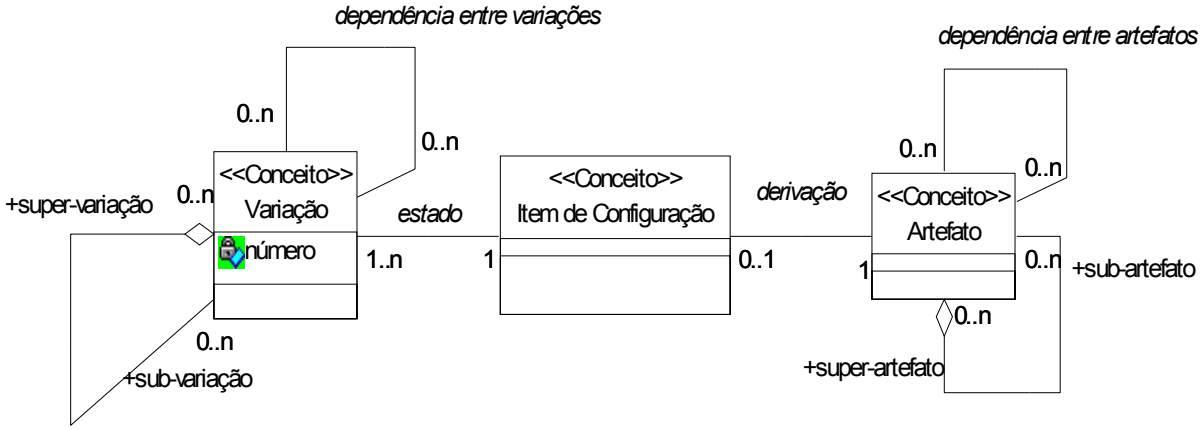


Figura 3.25 – Decomposição e dependência entre variações.

De maneira análoga, como ocorre com artefatos, uma variação pode estar diretamente relacionada a outras variações, estabelecendo, assim, uma relação de dependência da primeira com as demais. Através desta relação de dependência, pode-se identificar quais variações podem ser impactadas quando da ocorrência de mudanças em uma variação específica. A figura 3.25 mostra, também, esta relação de dependência entre variações.

O predicado *var_dependência(v1, v2)* indica que uma variação *v1* depende de uma variação *v2*.

É importante destacar que se uma variação depende de outra variação, suas super-variações também dependem de tal variação.

$$(\forall v1, v2, v3)\ (var\_dependência(v1, v2) \wedge super\text{-}variação(v3, v1) \rightarrow var\_dependência(v3, v2)) \qquad \text{(A20)}$$

Além disso, pode-se desejar saber se uma variação de um item de configuração derivado de um artefato está aderente à estrutura (decomposição e dependências) de tal artefato.

Para que ocorra a aderência de uma variação a seu respectivo artefato, esta variação deve possuir uma sub-variação para cada sub-artefato do artefato. Deve possuir, também, dependências com todas variações que correspondam aos artefatos dependentes do artefato em questão.

Tal situação pode ser expressa através do axioma abaixo, em que é utilizado o predicado *variação_aderente(v, a)*, que denota que a variação *v* está aderente ao artefato *a*.

$$(\forall v, a)\ variação\_aderente(v, a) \leftrightarrow (((\forall a1)\ (subartefato(a1, a))) \rightarrow ((\exists v1)\ (sub\text{-}variação(v1,v)))) \wedge$$

(((∀ a2) (art_dependência(a2, a))) *→ ((∃ v2) (var_dependência(v2, v))))*	(A21)

Alteração de variações

Em grandes projetos de engenharia de software, alterações sem controle levam rapidamente ao caos. Tais projetos necessitam de um controle de modificação que combine procedimentos humanos e ferramentas automatizadas.

Quando é solicitada uma alteração de uma determinada variação de um item de configuração, é feita uma análise de custo e impacto para se certificar da necessidade da alteração. Além disso, deve-se verificar se as variações desejadas estão disponíveis para alteração, ou seja, se já não há algum outro desenvolvedor que as esteja alterando.

Se a alteração for liberada, é dito que houve uma submissão de alteração e se pode realizar o *check-out* das variações envolvidas na alteração. No *check-out*, são registradas a data e a hora em que foram retiradas para alteração as variações submetidas na solicitação e essas variações são bloqueadas no repositório central para que nenhum outro desenvolvedor tente alterá-las ao mesmo tempo.

Assim que as alterações forem concluídas, faz-se o *check-in* das variações envolvidas, registrando a data e a hora de liberação das variações envolvidas na alteração, e essas variações, ou seja, as variações resultantes das alterações, são liberadas para modificação no repositório central.

Caso não tenha ocorrido alterações nas variações submetidas à alteração, o *check-in* funcionará da mesma forma, porém não haverá variações resultantes das alterações. A figura 3.26 mostra o modelo ontológico correspondente às alterações de variações.

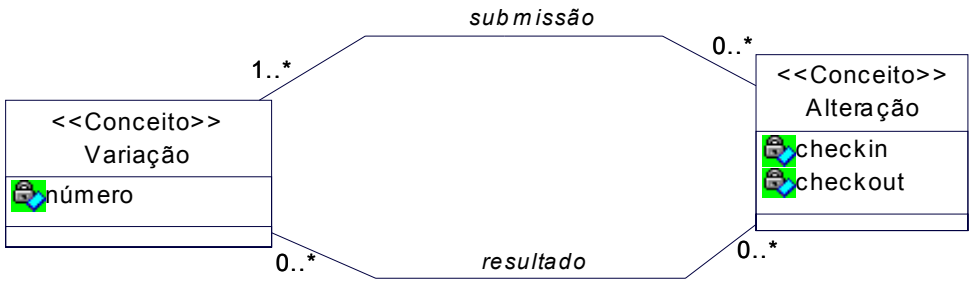


Figura 3.26 – Alterações de variações.

Para formalizar o conceito de alteração, com suas propriedades de *check-in* e *check-out*, bem como as relações submissão e resultado, foram definidos os seguintes predicados: *alteração(a)*, que denota que *a* é uma alteração; *checkin(ci, a)*, que denota que *ci* é o *check-in*

da alteração *a*; *checkout(co, a)*, que denota que *co* é o *check-out* da alteração *a*; *submissão(v, a)*, que denota que *v* é uma das variações submetidas para a alteração *a* e *resultado(v, a)*, que denota que *v* é uma das variações resultantes da alteração *a*.

Tanto ferramentas quanto artefatos podem ser colocados sob gerência de configuração, porém, apenas variações de itens de configuração derivados de artefatos podem ser submetidos a alteração, como mostra o seguinte axioma de consolidação:

$$(\forall a, v, i, ar)\ (submissão(v, a) \rightarrow estado(v, i) \wedge derivação(i, ar) \wedge artefato(ar)) \quad \text{(AC12)}$$

As variações submetidas a uma alteração podem possuir sub-variações. Assim, suas sub-variações também são submetidas a tal alteração.

$$(\forall v1, v2, a)\ (submissão(v1, a) \wedge sub\text{-}variação(v2, v1) \rightarrow submissão(v2, a)) \quad \text{(A22)}$$

Como mencionado anteriormente, quando uma alteração de uma variação é solicitada, deve-se analisar se ela está disponível para alteração. Assim, não pode haver mais de uma alteração para uma mesma variação cujos *check-out* e *check-in* se interceptem.

$$(\forall a, ci, co, v)\ (checkin(ci, a) \wedge checkout(co, a) \wedge submissão(v, a) \rightarrow \neg (\exists a1, ci1, co2)\ (checkin(ci1, a1) \wedge checkout(co1, a1) \wedge submissão(v, a1) \wedge ((co1 < co \wedge ci1 > co) \vee (co1 > co \wedge co1 < ci)))) \quad \text{(AC13)}$$

Deve-se lembrar, ainda, que o *check-in* de uma alteração concluída deve sempre ser posterior ao *check-out*.

$$(\forall a, ci, co)\ (alteração(a) \wedge checkin(ci, a) \wedge checkout(co, a) \rightarrow (ci > co)) \quad \text{(AC14)}$$

Decomposição de linha-base

Linha-base é o conjunto de itens de configuração em determinadas variações, que serve de base para o desenvolvimento ulterior, como, por exemplo, uma linha base formada pela porção de código X (variação 2.0), pelo documento de plano de projeto Z (variação 1.1.1), pelo diagrama de casos de uso (variação 1.0) etc. A figura 3.27 mostra o modelo ontológico correspondente à decomposição de linhas-base em variações de itens de configuração.

Figura 3.27 – Decomposição de linha-base.

Como variações podem possuir sub-variações, se uma variação de item de configuração faz parte de uma linha base, todas as suas sub-variações também fazem parte de tal linha base, como define o axioma abaixo.

$$(\forall v1, v2, l)\ (linhabase\_agregação(l, v1) \wedge sub\text{-}variação(v2, v1) \rightarrow linhabase\_agregação(l, v2)) \qquad \text{(A23)}$$

Responsabilidade de alteração

Quando uma alteração é solicitada, deve-se saber quem a solicitou, bem como quem autorizou a sua realização e quais os recursos humanos responsáveis pela sua execução, como mostra o modelo da figura 3.28.

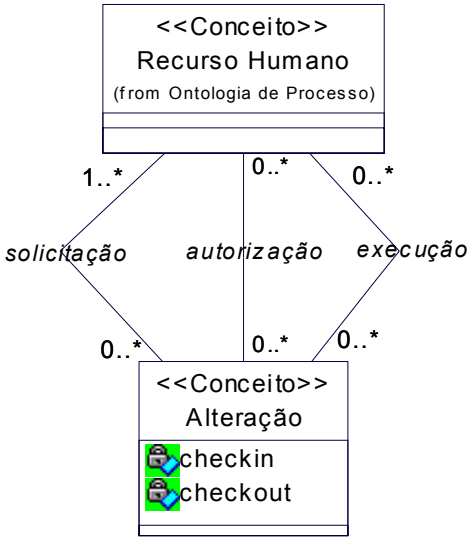


Figura 3.28 – Responsabilidade de alteração.

Para formalizar as relações descritas na figura 3.28, foram definidos os predicados *solicitação(rh, a)*, que indica que o recurso humano *rh* solicitou a alteração *a*, *autorização(rh, a)*, que indica que o recurso humano *rh* autorizou a alteração *a*, e *execução(rh, a)*, que indica que o recurso humano *rh* é responsável por executar a alteração *a*.

Acesso a itens de configuração

Para um controle efetivo das alterações realizadas sobre variações de itens de configuração, é necessário os estabelecer o tipo de acesso que cada recurso humano tem aos itens de configuração, como mostra a figura 3.29.

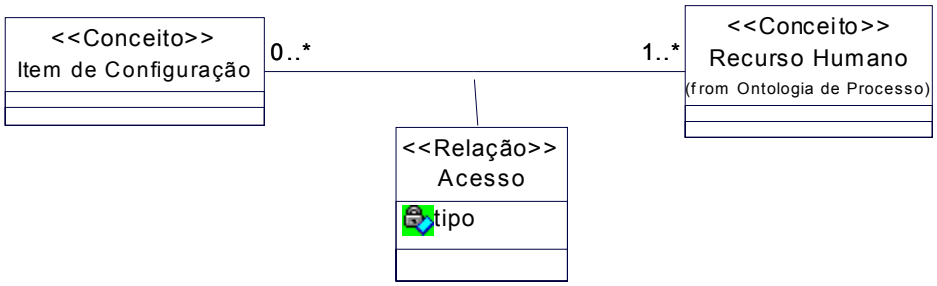


Figura 3.29 – Acesso a itens de configuração.

Para formalizar o conceito de acesso, é definido o predicado *acesso(rh, i, t)*, denotando que o recurso humano *rh* tem acesso do tipo *t* ao item de configuração *i*.

Uma restrição importante neste caso diz que, se um recurso humano for designado para executar uma alteração, ele deve ter acesso de escrita às variações desta alteração, como mostra o axioma abaixo:

$$(\forall rh, a, v, i)\ (execução(rh, a) \wedge submissão(v, a) \wedge estado(v, i) \rightarrow acesso(rh, i, \text{“escrita”})) \quad \text{(A24)}$$

A figura 3.30 mostra o modelo completo da ontologia de gerência de configuração e a tabela 3.5 apresenta o dicionário dos termos correspondente.

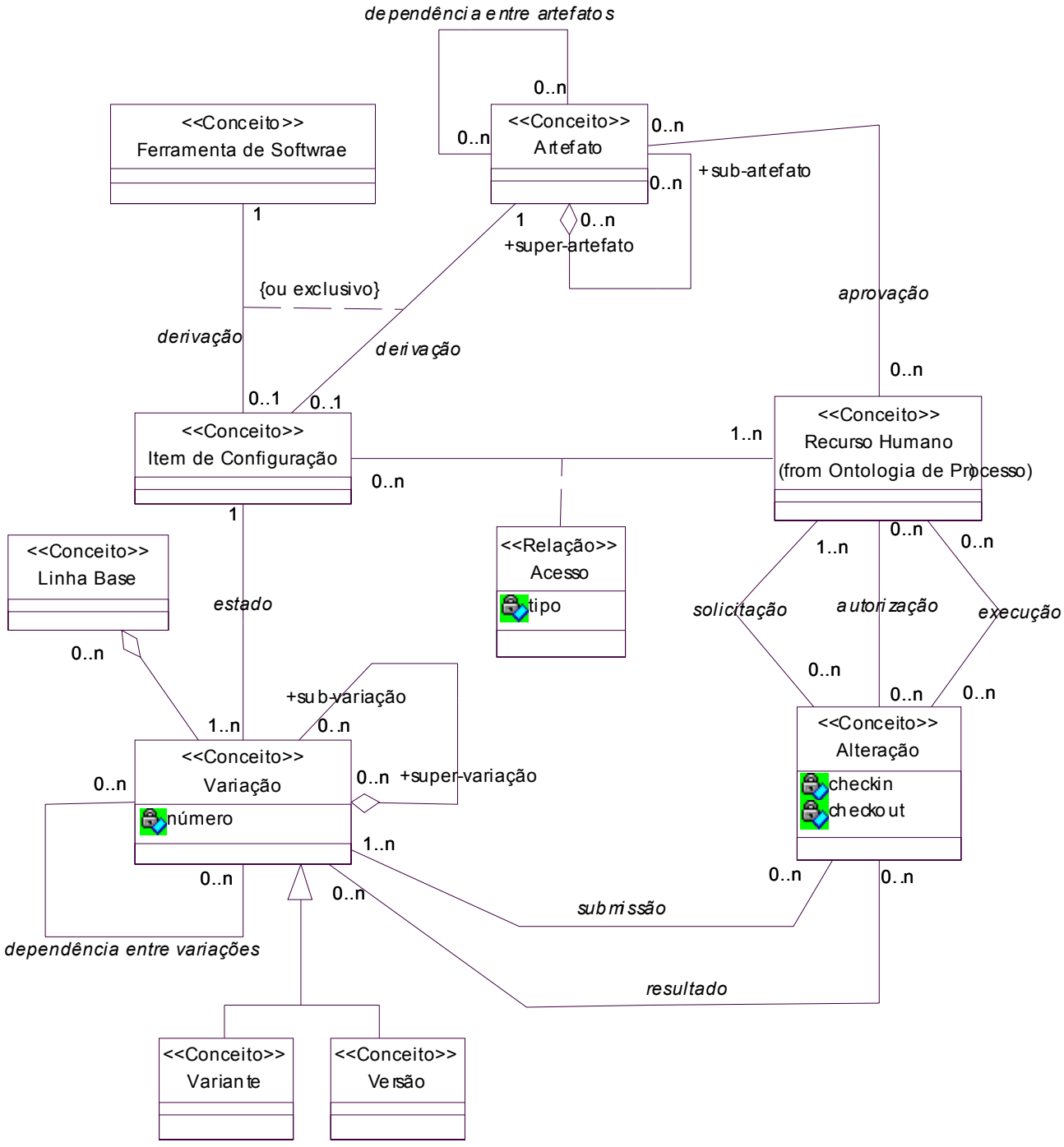


Figura 3.30 – Ontologia de gerência de configuração.

Tabela 3.5 - Dicionário de Termos da Ontologia de Gerência de Configuração.

Acesso	Relação entre **item de configuração** e **recurso humano**, indicando qual **tipo** de acesso (leitura, alteração, exclusão, etc.) determinado **recurso humano** tem sobre determinado **item de configuração**.
Alteração	Representa uma solicitação de alteração efetuada por um **recurso humano**, indicando as **variações de itens de configuração** que poderão ser alteradas. Esta alteração, sendo aprovada, deverá ser executada por **recursos humanos**, dando origem a novas **variações**. Para um controle formal, deverão ser executados procedimentos de ***check-out*** e ***check-in***.
Autorização	Relação entre **recurso humano** e **alteração**, indicando quais **recursos humanos** são os responsáveis por autorizar uma determinada **alteração**.
Check-in	Propriedade de **alteração** que indica quando as **variações** submetidas para uma **alteração** foram devolvidas ao repositório central, estando disponíveis para novas **alterações**. Contém a data/hora em que as **variações** tornaram-se disponíveis novamente.
Check-out	Propriedade de **alteração** que indica quando cópias das **variações** submetidas para **alteração** foram retiradas do repositório central e disponibilizadas na área de trabalho do desenvolvedor, para que sejam manipuladas. Contém a data/hora em que as **variações** foram retiradas do repositório central. A partir deste momento, nenhum outro **recurso humano** poderá alterar estas **variações** até que se tenha realizado um procedimento de ***check-in***.
Dependência entre variações	Relação entre **variações** indicando as **variações** que são dependentes de uma determinada **variação**. Assim, é possível identificar todas as **variações** que podem ser afetadas em uma determinada **alteração** e não apenas as solicitadas.
Derivação	Relação entre **item de configuração** e **artefato** (ou entre **item de configuração** e **ferramenta de software**), indicando que um determinado **artefato** (ou **ferramenta de software**) está sob gerência de configuração, ou seja, torna-se um **item de configuração**.
Estado	Relação entre **item de configuração** e **variação** que indica quais as **variações** de um certo **item de configuração**.
Execução	Relação entre **recurso humano** e **alteração,** indicando quais **recursos humanos** são responsáveis pela execução de uma determinada **alteração**, ou seja, quais **recursos humanos** estão (ou estiveram) de posse de determinadas **variações** para, possivelmente, realizar **alterações**.
Ferramenta de Software	**Recurso de software** utilizado para (semi-)automatizar um **procedimento** adotado na realização de uma **atividade**. Ex.: Editor de textos, Planilha eletrônica, ferramentas CASE etc.
Item de Configuração	Item que está sob gerência de configuração e, assim, só pode ser alterado segundo um procedimento de controle de alteração formalmente estabelecido e documentado. Pode possuir várias **variações**. Pode ser uma **ferramenta de software** ou um **artefato**, como, por exemplo, um determinado plano de projeto ou um certo artefato de código.
Linha Base	Conjunto de **itens de configuração** em determinadas **variações**, que serve de base para o desenvolvimento ulterior. Ex.: uma linha base formada pela porção de código X (variação 2.0), pelo documento de plano de projeto Z (variação 1.1.1), pelo diagrama de casos de uso (variação 1.0) etc.
Resultado	Relação entre **variação** e **alteração**, indicando quais **variações** foram produzidas em uma determinada **alteração**, ou seja, quais as **variações** resultantes de uma determinada **alteração**.
Solicitação	Relação entre **recurso humano** e **alteração**, indicando quais **recursos humanos** fizeram a solicitação de uma determinada **alteração**.

Tabela 3.5 - Dicionário de Termos da Ontologia de Gerência de Configuração (Continuação).

Item de Configuração	Item que está sob gerência de configuração e, assim, só pode ser alterado segundo um procedimento de controle de alteração formalmente estabelecido e documentado. Pode possuir várias **variações**. Pode ser uma **ferramenta de software** ou um **artefato**, como, por exemplo, um determinado plano de projeto ou um certo artefato de código.
Linha Base	Conjunto de **itens de configuração** em determinadas **variações**, que serve de base para o desenvolvimento ulterior. Ex.: uma linha base formada pela porção de código X (variação 2.0), pelo documento de plano de projeto Z (variação 1.1.1), pelo diagrama de casos de uso (variação 1.0) etc.
Resultado	Relação entre **variação** e **alteração**, indicando quais **variações** foram produzidas em uma determinada **alteração**, ou seja, quais as **variações** resultantes de uma determinada **alteração**.
Solicitação	Relação entre **recurso humano** e **alteração**, indicando quais **recursos humanos** fizeram a solicitação de uma determinada **alteração**.
Submissão	Relação entre **variação** e **alteração**, indicando quais **variações** foram submetidas para modificação em uma determinada **alteração**.
Sub-variação	Papel da relação de agregação entre duas **variações** v_1 e v_2. Se v_2 é parte de v_1, então v_2 é dita uma sub-variação de v_1.
Super-variação	Papel da relação de agregação entre duas **variações** v_1 e v_2. Se v_1 é decomposta em outras **variações**, dentre eles v_2, então v_1 é dita um super-variação de v_2.
Variação	Indica qual a **versão** ou **variante** de um determinado **item de configuração**. É caracterizada por um **número** único para cada **variação** de um mesmo **item de configuração**.
Variante	Estado em que um **item de configuração** existe simultaneamente em duas ou mais formas diferentes que atendam a requisitos similares. Ex.: Um documento estava na versão 1.0 e após alterações passou para a versão 1.1. Porém, foi criada a variante 2.0 que, apesar de atender aos mesmos requisitos que a versão 1.1, foi construída de forma diferente.
Versão	Estado em que se encontra um **item de configuração**. Cada **alteração** realizada em um **item de configuração** gera uma nova versão deste **item**. Ex.: Um documento que estava na versão 1.0 e após algumas alterações passou para versão 1.1.

3.8 Conclusões do Capítulo

Neste capítulo foram apresentados conceitos de ontologias, sua classificação quanto ao conteúdo, aplicações de ontologias e aspectos importantes referentes à construção de ontologias.

A seguir, foram discutidas, brevemente, as ontologias utilizadas pelo ambiente ODE, procurando ressaltar a importância de se definir uma ontologia de artefato de software, apresentada no restante do capítulo, dividida em sub-ontologias para tratar aspectos específicos de documentos, diagramas, artefatos de código e Gerência de Configuração de Software (GCS). Para cada uma das sub-ontologias, foi apresentada a identificação de

propósito e especificação de requisitos, contemplando as questões de competência e a captura e formalização da ontologia, através de diagramas, descrições textuais, axiomas formais em lógica de primeira ordem e dicionários de termos.

No estágio atual, a ontologia de artefatos definida está sendo incorporada ao ambiente ODE por meio da redefinição dos artefatos produzidos pelas ferramentas de ODE, pela definição de uma ferramenta de apoio à documentação (XMLDoc), descrita em (SILVA, 2004) e por meio do desenvolvimento de um sistema de GCS, integrado à infra-estrutura de gerência de conhecimento de ODE, apresentado no próximo capítulo.

Capítulo 4

Integrando Gerência de Configuração de Software, Documentação e Gerência de Conhecimento em ODE

Conforme discutido no capítulo 2, a Gerência de Configuração de Software é um processo de aplicação de procedimentos administrativos e técnicos, por todo o ciclo de vida de desenvolvimento do software. É destinada a identificar e definir os itens de software em um sistema, estabelecer suas linhas básicas, controlar as modificações e liberações dos itens, registrar e apresentar a situação dos itens e dos pedidos de modificação, garantir a completeza, a consistência e a correção dos itens, e controlar o armazenamento, a manipulação e a distribuição dos itens (NBR ISO/IEC, 1998).

Porém, antes de se pensar na criação de uma ferramenta para apoiar a Gerência de Configuração em um ADS, em especial em ODE, deve-se definir uma infra-estrutura para ela, que impacte o mínimo possível no ambiente existente, preservando suas principais características. No caso de ODE, é essencial que a integração de ferramentas não seja prejudicada, que seja possível enxergar as partes que compõem um artefato, e não simplesmente o artefato como um todo, e que a Gerência de Conhecimento possa continuar atuando no auxílio ao desenvolvimento de sistemas.

Ao se definir, então, uma infra-estrutura de GCS, deve-se levar em conta como ocorrerá a integração da Gerência de Configuração com as demais ferramentas de ODE, em especial com a de Documentação e com a Gerência de Conhecimento.

Este capítulo discute os requisitos de integração que devem ser considerados na construção de uma ferramenta de apoio à GCS. Na seção 4.1 são apresentadas diferentes abordagens para tratar essa questão e a escolhida para este trabalho. A seção 4.2 discute os procedimentos para integração da GCS à documentação e às demais ferramentas de ODE. Na

seção 4.3 é apresentada a adaptação da Gerência de Conhecimento à GCS, em ODE. Por fim, a seção 4.4 traz as conclusões do capítulo.

4.1 Infra-estrutura para Gerência de Configuração em ODE

Para que a Gerência de Configuração de Software (GCS) seja mais efetiva em uma organização, uma ferramenta para apoiar o seu processo torna-se necessária, pois automatiza suas atividades, permitindo um maior controle e diminuindo a possibilidade de erros. Porém, antes de se criar uma ferramenta para este fim, deve ser estabelecida uma infra-estrutura que acomode as características desejadas para ela.

Primeiramente, deve-se focalizar no ambiente em que a GCS será inserida. Trata-se de um ambiente de desenvolvimento de software? Ou de um ambiente que não chega a ser um ADS, mas possui ferramentas CASE que criam elementos que serão controlados? Ou de um ambiente em que as ferramentas trabalham desconectadas? Em outras palavras, integração é um problema a ser considerado? Esse raciocínio deve ser levado em conta, pois pode ser o ponto chave na escolha entre diversas abordagens. No caso deste trabalho, trata-se de um ADS no qual a questão da integração é fundamental. Assim, aspectos de integração de ferramentas, em suas várias dimensões, têm de ser considerados como base da infra-estrutura sobre a qual será implementada a GCS em ODE.

O próximo passo visa a estabelecer quais serão os tipos de itens de configuração gerenciados, ou seja, se serão gerenciados documentos, códigos fonte, artefatos de forma geral, ferramentas de software etc. A definição da infra-estrutura deve ser baseada nos itens que a ferramenta vai acomodar para que não ocorra o problema da ferramenta necessitar de mais recursos do que a infra-estrutura permite ou de se disponibilizar recursos demais que não serão utilizados, em outras palavras, como diz o ditado popular, “matar uma formiga com uma bazuca”.

Neste trabalho, os itens de configuração enfocados são os artefatos de software, em sua maioria gerados pelas ferramentas de ODE. Porém, a idéia aqui foi estabelecer uma infra-estrutura mais geral, de forma que pudesse acomodar no futuro outros tipos de itens de configuração, tais como ferramentas de software ou mesmo outros elementos de um ADS, como processos, atividades etc.

Outro ponto fundamental a ser tratado é com relação ao armazenamento dos itens de configuração, ou seja, o repositório. No caso de um ambiente de desenvolvimento de software existente em que se deseja implantar a Gerência de Configuração, surgem algumas questões: o repositório da GCS será uma nova base, para onde os dados existentes serão migrados? Ou a base continuará sendo a mesma, ou seja, a implantação da GCS será transparente ao restante do ambiente? Onde ficarão os itens não gerenciados, que podem ser alterados livremente, uma vez que a GCS bloqueia alterações nos itens que não tiverem prévia solicitação de alteração? Onde ficarão os itens sob GCS de versões antigas? Qual o impacto, em termos de espaço de armazenamento, eficiência de execução e tempo de resposta, se for utilizada uma mesma base para armazenar todas as versões existentes de um artefato (atuais e antigas) sob GCS ou não?

Neste ponto surge uma nova questão: Onde ficarão os itens que estão sendo alterados, uma vez que eles devem ser visualizados apenas pelos responsáveis pela alteração? Os demais desenvolvedores devem visualizar a última versão, sem as alterações em andamento, uma vez que artefatos em alteração podem não estar em um formato final, ou seja, podem conter erros que não são interessantes de serem exibidos aos demais desenvolvedores. Para resolver este problema, será criada uma nova base para as alterações serem realizadas? Manter-se-á apenas uma única base compartilhada?

A estratégia a ser utilizada é de suma importância, pois poderá acarretar uma nova filosofia de armazenamento de dados. Porém, antes de defini-la, deve-se analisar o ambiente em que a GCS será inserida, para avaliar o impacto que ela causará. Algumas questões devem ser respondidas: Como o ambiente trata os itens que serão gerenciados, eles estão em bases de dados ou em arquivos? Qual a estrutura de armazenamento desses itens? Sem GCS, como são realizadas alterações nos itens controlados pelo ambiente? Que ferramentas utilizam os itens que serão gerenciados? Essas perguntas devem ser levadas em consideração para que, após a implantação de uma ferramenta de GCS, não se descubra que ela prejudica ou impede a utilização de alguma ferramenta do ambiente.

No caso do ambiente ODE, os dados são armazenados em um repositório central na forma de uma base de dados relacional, seguindo uma estrutura previamente definida. No caso dos artefatos de software, a estrutura de armazenamento é aderente à ontologia de artefato desenvolvida neste trabalho, apresentada no capítulo 3. Como artefatos são peças fundamentais na comunicação entre as ferramentas do ADS, sua manipulação deve ser cuidadosa, pois pode impactar o funcionamento dessas ferramentas. Como ODE armazena seus artefatos em um banco de dados relacional, deve-se analisar a possibilidade de manter

esse tipo de armazenamento para os itens de configuração ou de mudá-lo para uma forma de armazenamento em arquivos.

Em ODE, deve haver, ainda, uma preocupação especial com a Gerência de Conhecimento. A Gerência de Conhecimento de ODE possui uma memória organizacional, em que os artefatos são itens de conhecimento formal armazenados, conforme discutido no capítulo 2. Ou seja, a infra-estrutura da ferramenta de GCS deve se preocupar em garantir compatibilidade entre seu repositório e a memória organizacional da Gerência de Conhecimento. Além disso, deve-se estabelecer com quais elementos a Gerência de Conhecimento irá lidar: se com todos os itens do ADS (sob GCS ou não), se apenas com os itens sob GCS ou se apenas com os itens sob GCS em suas versões atuais. Outra preocupação é estabelecer se a Gerência de Conhecimento irá buscar os artefatos em alteração ou a última versão no repositório. Independentemente da estratégia adotada, deve-se ter em mente que a Gerência de Conhecimento deve continuar operando da melhor maneira possível, após a implantação da GCS. Ou seja, a GCS não deve impedir que as ferramentas usufruam das facilidades da Gerência de Conhecimento, como, por exemplo, busca e disseminação do conhecimento. Da mesma forma, deve-se buscar minimizar os impactos no restante do ambiente, advindos da introdução das funcionalidades da GCS.

Por fim, deve-se analisar o impacto técnico das possíveis soluções a serem adotadas. Algumas soluções podem levar a grandes dificuldades técnicas para o desenvolvimento, devido à sua complexidade. Outras podem forçar uma reestruturação de todo ambiente, sendo necessário fazer nele um número muito grande de alterações. Outras podem envolver funcionalidades que gerem perda de eficiência em tempo de execução, utilização exagerada de espaço em disco, problemas de segurança de acesso às informações etc.

Várias abordagens para se estabelecer uma infra-estrutura para a GCS foram analisadas neste trabalho, para se definir qual seria a abordagem adotada em ODE. Dentre elas, quatro merecem destaque: (i) tratar itens de configuração como arquivos, (ii) dispor de duas bases de dados e arquivos XML, (iii) dispor de uma única base de dados, cópias nessa base e arquivos XML, (iv) dispor de uma única base de dados e arquivos XML. A seguir, cada uma dessas abordagens é apresentada, juntamente com uma discussão do porquê ela foi rejeitada ou aceita.

Opção 1: Tratar itens de configuração como arquivos

Fazendo um levantamento na literatura, a primeira abordagem considerada foi tratar os vários itens de configuração como arquivos. Essa abordagem é utilizada pela maioria das ferramentas de GCS existentes. Os itens de configuração, em especial os artefatos, ficariam armazenados em um local único (repositório), na forma de arquivos. As versões antigas dos itens de configuração também ficariam no repositório e poderiam ser alteradas seguindo o mesmo processo de alteração aplicado às variações atuais de um artefato. Porém, os itens que não estivessem sob gerência de configuração ficariam separados do repositório central, em bases locais.

Quando um artefato fosse ser alterado, seria feita uma cópia dele, do repositório para uma base ou um diretório local, através do processo de *checkout*. Assim, o desenvolvedor efetuaria as alterações apenas na cópia, mantendo intacto o artefato que está no repositório. As demais pessoas não teriam permissão para alterar tal artefato enquanto ele estivesse em alteração e continuariam enxergando a versão do repositório, ou seja, a versão sem as alterações em andamento. Passariam a enxergar a nova versão apenas quando o artefato fosse devolvido ao repositório, através do processo de *checkin*. A figura 4.1 ilustra essa abordagem.

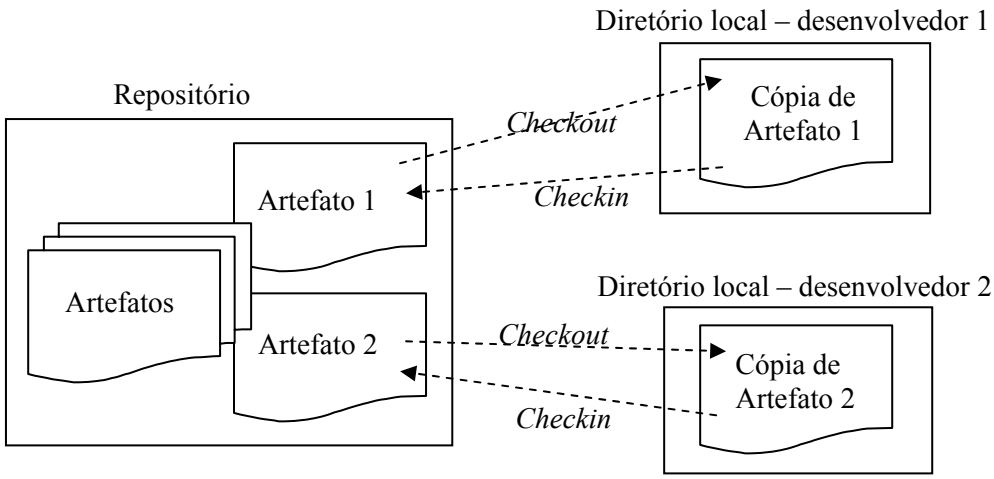


Figura 4.1 – Itens de configuração armazenados como arquivos.

A grande vantagem desta opção é a simplicidade. Pelo fato de artefatos serem encarados como arquivos, a manipulação e o controle sobre eles são facilitados, além do seu tempo de acesso não ser impactado pela quantidade de artefatos armazenados, que pode ser grande devido às versões anteriores.

Essa opção, contudo, foi logo descartada pelos diversos prejuízos que causaria à integração, sobretudo nas dimensões de dados e conhecimento. Como os dados devem estar disponíveis a todas as ferramentas do ambiente, que podem usar informações parciais contidas

em diversos artefatos, uma abordagem de armazenamento em arquivos se mostra bastante inadequada para o ambiente ODE. É exatamente por este motivo que ODE adota como repositório de dados central um banco de dados relacional, ao invés de, por exemplo, uma estrutura de arquivos.

Além disso, conforme apontado pela ontologia de GCS, apresentada no capítulo anterior, é muito importante que a GCS trate da composição e dependência entre artefatos, pois é através dessas relações que se consegue determinar o impacto que as alterações em certo artefato podem gerar em outros artefatos. Porém, como na abordagem de arquivos um artefato é visto como uma "caixa preta" e não composto de partes, as relações de composição de artefatos não são consideradas.

Opção 2: Dispor de duas bases de dados e arquivos XML

Uma vez que o uso de um banco de dados é fundamental para tratar adequadamente a dimensão de integração de dados em ODE, passou-se a explorar abordagens utilizando esse tipo de recurso. Na segunda opção levantada, os itens de configuração e, por conseguinte, os artefatos de software, em suas versões mais atuais, seriam armazenados em bancos de dados relacionais e não na forma de arquivos, como na abordagem anterior, favorecendo a integração de dados e de conhecimento. As versões antigas de cada artefato, porém, seriam armazenadas na forma de arquivos XML. A escolha de XML advém do fato dessa tecnologia já estar sendo utilizada por outras ferramentas de ODE, em especial pela ferramenta de apoio à documentação, XMLDoc, apresentada no capítulo anterior e fortemente relacionada a este trabalho.

Segundo essa concepção, haveria duas bases de dados, uma para artefatos em desenvolvimento ou alteração e a outra para artefatos sob gerência de configuração. A base de desenvolvimento conteria os itens sob GCS em alteração e os itens que não estivessem sob GCS, ou seja, que poderiam ser alterados sem passar pelo processo de Gerência de Configuração. Na outra base de dados estariam apenas os itens sob GCS que não estivessem sendo alterados. Essa base desempenharia o papel de Memória Organizacional de ODE pela correlação estabelecida com a Gerência de Conhecimento, que utilizaria apenas os artefatos dessa base como itens de conhecimento. A figura 4.2 ilustra essa abordagem.

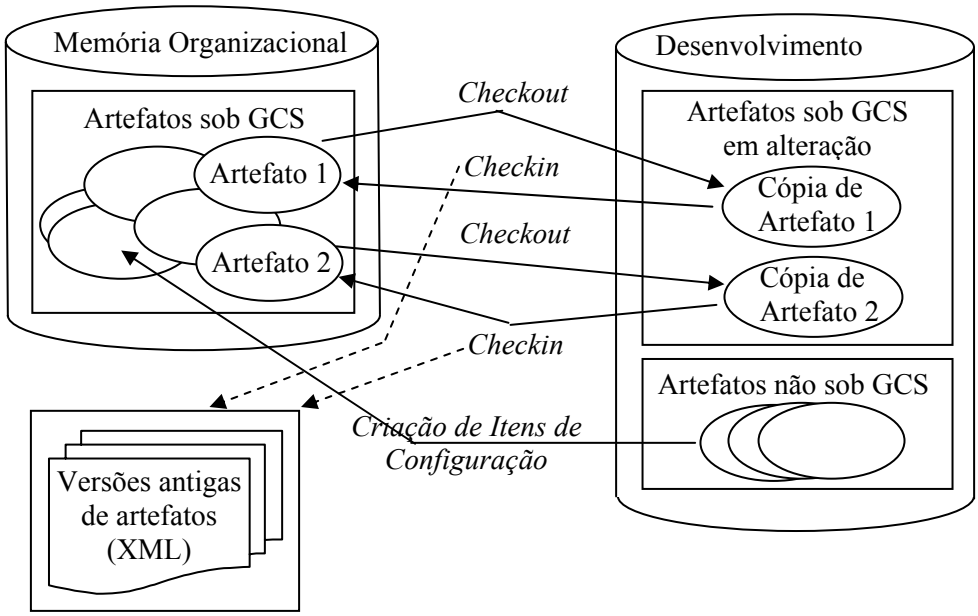


Figura 4.2 – Itens de configuração atuais armazenados em duas bases de dados (Memória Organizacional e Desenvolvimento) e antigas em arquivos XML.

Sempre que um artefato fosse submetido à GCS, ou seja, fosse criado um novo item de configuração, esse artefato seria transferido da base de desenvolvimento para a memória organizacional.

Como antes de se implantar a GCS em ODE o que existiam eram artefatos que não estavam sob GCS, eles continuariam a existir em sua base original (Desenvolvimento), que passaria a acomodar também os artefatos sob GCS em alteração. A diferença é que agora seria criada uma nova base para os artefatos sob GCS que não estivessem sendo alterados.

Sempre que se quisesse alterar um artefato, seria feita uma cópia deste da primeira base (Memória Organizacional) para a segunda (Desenvolvimento), através do processo de *checkout*. Todas as pessoas enxergariam apenas os artefatos armazenados na primeira base, com exceção dos desenvolvedores que estivessem realizando alterações, que enxergariam as versões correspondentes na base de desenvolvimento, e dos artefatos que não estivessem sob GCS, que poderiam ser alterados sempre que desejado.

Vale ressaltar que mesmo que vários artefatos sob GCS estivessem em alteração, um determinado desenvolvedor só enxergaria os artefatos sob responsabilidade dele, ou seja, os artefatos que ele estivesse alterando. Se, por exemplo, um desenvolvedor 1 estivesse alterando o artefato 1 e um outro desenvolvedor 2 estivesse alterando o artefato 2, o desenvolvedor 1 enxergaria o artefato 1 da base de desenvolvimento, porém enxergaria o artefato 2 da memória organizacional. Da mesma forma, o desenvolvedor 2 enxergaria o artefato 2 da base de desenvolvimento e enxergaria o artefato 1 da base da memória organizacional.

Terminada uma alteração, seria feita a atualização do artefato no repositório, ou seja, seria realizada a cópia do artefato da base de desenvolvimento para a memória organizacional, através do processo de *checkin*. Neste momento, a versão anterior do artefato seria convertida para um arquivo XML. Assim, na base de dados principal (a memória organizacional), ficaria apenas a versão mais recente de cada artefato.

Essa abordagem tem a grande vantagem de não prejudicar a utilização das demais ferramentas do ambiente, em especial da Gerência de Conhecimento. O fato de que as alterações em artefatos serem realizadas nas próprias bases de dados e não em um local isolado, como na abordagem em arquivos, faria com que o desenvolvedor não deixasse de usufruir das funcionalidades das demais ferramentas.

Outra grande vantagem, devido ao fato dos artefatos estarem armazenados em bases de dados e terem uma estrutura previamente definida, é que se torna possível acessar as partes que compõem um artefato e não apenas o artefato como um todo. Além disso, não demandaria uma mudança conceitual na forma de armazenamento de artefatos em ODE, uma vez que seria mantida a mesma filosofia de persistência de dados.

Vale ressaltar, ainda, a vantagem de não haver impacto no tempo de execução e de resposta das ferramentas, nem no espaço de armazenamento utilizado, uma vez que apenas os itens atuais seriam armazenados na base de dados, ficando as versões antigas armazenadas em arquivos.

Porém, tal abordagem foi descartada pelas dificuldades técnicas para que a consistência dos artefatos fosse garantida e para a comunicação e manuseio das duas bases de dados.

A dificuldade em garantir a consistência dos artefatos advém do fato de que cada ferramenta do ambiente deveria implementar um método de acesso à base de dados correta e outro método de bloqueio de escrita.

Um usuário deveria acessar um determinado artefato da base de desenvolvimento apenas se: (i) esse artefato estivesse sob GCS, estivesse em alteração e o usuário fosse um dos responsáveis pela alteração ou (ii) o artefato não estivesse sob GCS. Nas demais situações, ele deveria acessar a base da memória organizacional. Além disso, ele só poderia alterar um artefato se estivesse acessando a base de desenvolvimento. O acesso à memória organizacional seria apenas de leitura.

A dificuldade de manuseio e comunicação das duas bases está ligada ao fato de serem bases de dados separadas, necessitando esquemas de conexão diferentes e algoritmos de cópia de artefatos entre as bases de dados. No caso das conexões, haveria uma mudança drástica na

camada de persistência de ODE (RUY, 2003), a qual faz o estabelecimento das conexões com apenas um banco de dados por classe. A camada de persistência de ODE possui uma classe *Conexão* que oferece diversos serviços, dentre eles para se conectar e desconectar com um banco de dados, que foram construídos no intuito de se estabelecer conexões com apenas um banco de dado por classe. Assim, se uma determinada classe deseja acessar um banco de dados para, por exemplo, ler seus objetos, ela deve inicialmente abrir uma conexão para tal banco. Deve-se observar que todos os objetos dessa classe estão sendo buscados em um mesmo banco, aquele para o qual a classe criou uma conexão. Porém, para atender ao propósito da abordagem com duas bases de dados distintas, as conexões deveriam ser estabelecidas por objeto, uma vez que seria possível estabelecer conexões diferentes para os objetos de uma mesma classe. Por exemplo, poderia ocorrer de, em um determinado projeto, um desenvolvedor estar alterando um diagrama de classes, em que necessita ter uma conexão com a base de desenvolvimento, e estar acessando apenas para visualização outros diagramas de classe, sendo necessária conexão com a outra base (memória organizacional). Neste caso, tem-se o estabelecimento de conexões diferentes para objetos distintos de uma mesma classe.

Apesar das vantagens que se obtém na utilização de arquivos XML para armazenamento das versões antigas dos artefatos, tem-se o contratempo de ter que transformar um determinado arquivo XML em objeto, com armazenamento em banco de dados, a cada vez que se desejar alterar alguma versão antiga de um artefato. Porém, esta não é uma desvantagem significativa, pois, na maioria das vezes, os desenvolvedores estarão lidando com artefatos na sua forma atual, ou seja, alterações em versões antigas são mais uma exceção do que uma regra geral. Além disso, manter versões antigas na base de dados de ODE faria essa base de dados crescer demasiadamente, causando problemas de desempenho.

Opção 3: Dispor de uma única base de dados, cópias nessa base e arquivos XML

Visando evitar os problemas decorrentes do uso de duas bases de dados distintas, passou-se a considerar abordagens usando uma única base de dados. Na primeira abordagem considerada usando essa filosofia, os artefatos de software, em suas versões mais atuais, que estivessem ou não sob GCS, estariam armazenados em uma única base de dados relacional, enquanto suas versões antigas estariam armazenadas na forma de arquivos XML. As alterações nos artefatos sob GCS seriam feitas utilizando-se cópias, que estariam armazenadas na mesma base. Essa abordagem é ilustrada na figura 4.3.

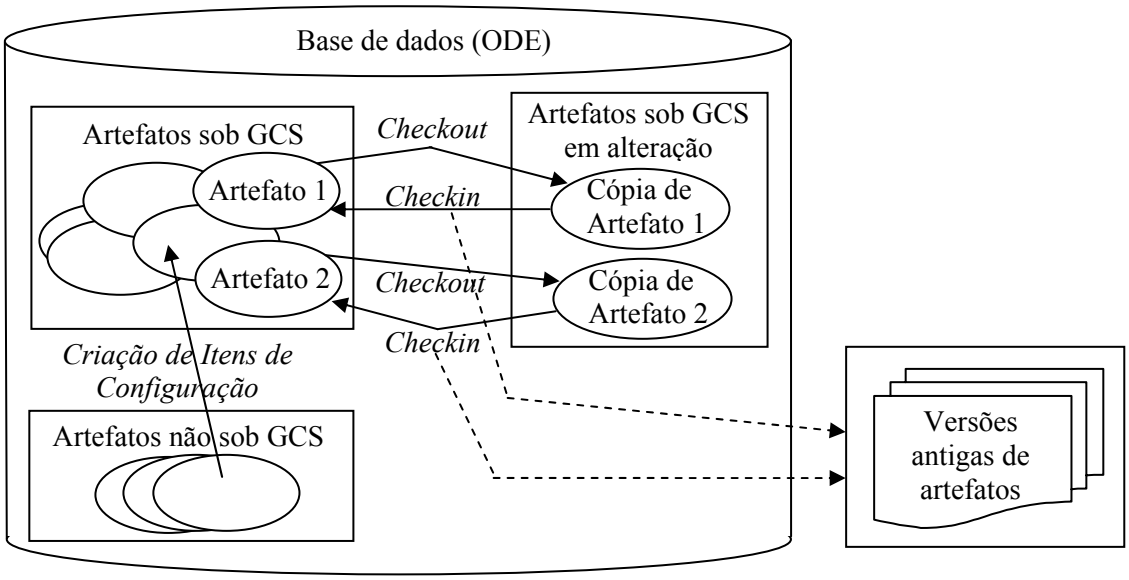


Figura 4.3 – Itens de configuração atuais e suas cópias armazenados em uma única base de dados e versões antigas em arquivos XML.

Sempre que um artefato fosse submetido à GCS, ou seja, sempre que fosse criado um novo item de configuração, o artefato correspondente deixaria de pertencer ao grupo de artefatos que não estavam sob gerência de configuração e passaria a pertencer ao grupo de artefatos sob GCS. Porém, isso seria transparente, uma vez que a base de dados seria a mesma.

Os artefatos que não estivessem sob GCS poderiam ser alterados livremente. Porém, para que um artefato sob GCS fosse alterado, seria criada uma cópia deste, através do processo de *checkout*. Todas as pessoas enxergariam o artefato original, sem as modificações que, por ventura, estivessem sendo realizadas. Porém o desenvolvedor que estivesse fazendo as alterações enxergaria a sua cópia.

Terminadas as alterações, a versão anterior do artefato seria convertida para um arquivo XML, através do processo de *checkin*. Ainda durante o *checkin*, o artefato seria atualizado com as informações advindas da cópia e a cópia seria excluída.

Essa abordagem, assim como a anterior, tem a grande vantagem de não prejudicar a utilização das ferramentas do ambiente, pelo fato de que as alterações em artefatos serem realizadas na própria base de dados e não em um local isolado, como na abordagem em arquivos.

Ainda como na abordagem anterior, devido ao fato dos artefatos estarem armazenados em uma base de dados com uma estrutura previamente definida, tornar-se-ia possível acessar as partes que os compõem e não apenas os artefatos como um todo. Além disso, não haveria

necessidade de se alterar a camada de persistência de ODE, já que o problema de conexão com as bases de dados não mais existiria, pelo fato de ser, agora, uma base única.

Vale ainda citar a vantagem de não impactar no tempo de execução e de resposta das ferramentas, nem no espaço de armazenamento utilizado, devido ao fato de apenas os itens atuais serem armazenados na base de dados, ficando as versões antigas em arquivo. Aqui, o impacto seria ainda menor, uma vez que existiria apenas uma base de dados, ao invés de duas.

Porém, apesar de solucionar alguns problemas da abordagem anterior, como os de comunicação entre as duas bases de dados, as dificuldades técnicas para que a consistência dos artefatos fosse garantida foram consideradas um problema para essa opção. Tais dificuldades advêm do fato de que cada ferramenta do ambiente deveria implementar um método de acesso ao elemento correto da base de dados e outro método de bloqueio de escrita. Assim, um usuário deveria acessar a cópia de um determinado artefato apenas se esse artefato estivesse sob GCS, estivesse em alteração e o usuário fosse um dos responsáveis pela alteração. Nas demais situações, ele deveria acessar o próprio artefato. Além disso, ele apenas poderia alterar um artefato caso: (i) estivesse acessando a sua cópia ou (ii) estivesse acessando o próprio artefato, mas este não estivesse sob GCS. Nas demais situações, o acesso seria apenas de leitura ao artefato original.

Vale destacar que essa abordagem mantém o contratempo de ter que se transformar um determinado arquivo XML em um artefato sob a forma de objetos, com armazenamento em banco de dados, a cada vez que se desejar alterar alguma versão antiga de um artefato. Porém, como já discutido, esta não é uma desvantagem significativa.

Opção 4: Dispor de uma única base de dados e arquivos XML (opção escolhida)

Finalmente, buscando minimizar o impacto de controle a ser feito por cada ferramenta e aproveitando que ODE já possui uma ferramenta de apoio à documentação (XMLDoc), que utiliza arquivos XML, buscou-se uma abordagem conciliadora das perspectivas de GCS, Gerência de Conhecimento e Documentação. Nessa abordagem, os artefatos de software, em suas versões mais atuais, que estiverem ou não sob GCS, são armazenados em uma única base de dados relacional, o repositório central de ODE. Além disso, todas as versões dos artefatos sob GCS, inclusive as atuais, são armazenadas na forma de arquivos XML, como mostra a figura 4.4.

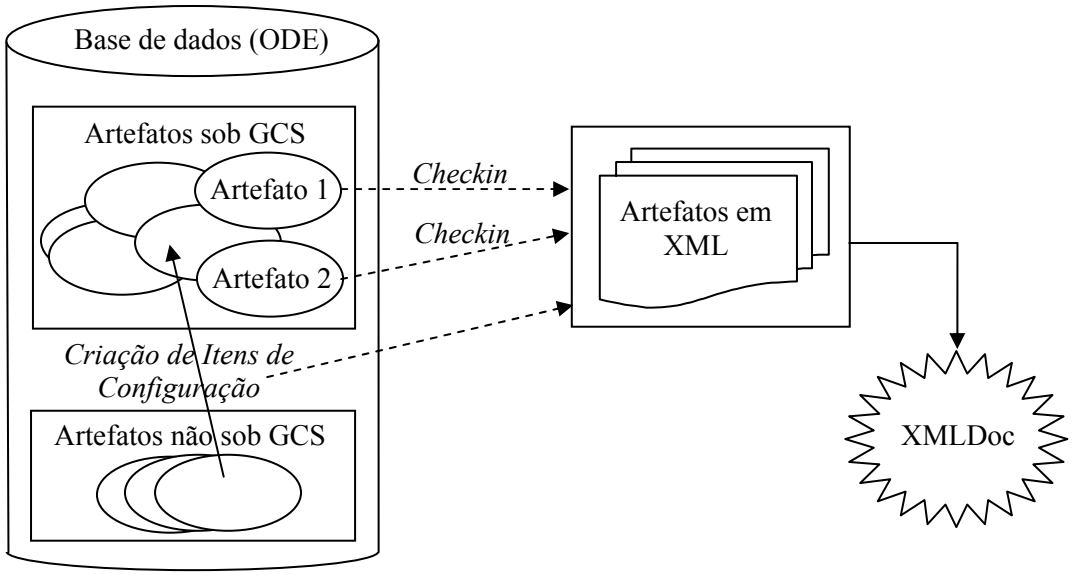


Figura 4.4 – Itens de configuração armazenados em uma única base de dados e em arquivos XML.

Ao ser colocado sob gerência de configuração, um artefato deixa de pertencer ao grupo de artefatos que não estão sob gerência de configuração e passa a pertencer ao grupo de artefatos sob GCS. Porém, isso é transparente, uma vez que a base de dados é a mesma. De fato, apenas o artefato passa a ser considerado um item de configuração. Além disso, ao se tornar um item de configuração, é criado, também, um arquivo XML correspondente ao artefato em questão.

Os artefatos que não estiverem sob GCS podem ser alterados livremente, porém, para que um artefato sob GCS possa ser alterado, ele deve estar em alteração, tendo passado previamente pelo processo de *checkout*, e o desenvolvedor acessando-o deve ser um dos responsáveis por tal alteração. Somente nesses casos as ferramentas de confecção dos respectivos artefatos seriam abertas, disponibilizando as funcionalidades para alteração dos mesmos. Nos demais casos, os usuários teriam apenas acesso de leitura a tais artefatos. Assim, seria aberta a ferramenta XMLDoc (SILVA, 2004), exibindo o arquivo XML correspondente ao artefato em questão. Deve-se observar que o arquivo XML exibido é a versão mais atual do artefato, ou uma versão anterior, caso essa tenha sido a escolha do usuário. Porém, não conteria as alterações que poderiam estar sendo realizadas por algum outro desenvolvedor, uma vez que essas alterações estariam apenas na base de dados e não nos arquivos XML.

Terminada uma alteração, é, então, criado um arquivo XML para essa nova versão do artefato, através do processo de *checkin*. Desta forma, pode-se observar que existiriam arquivos XML para todas as versões dos artefatos, inclusive para a versão mais atual, diferente das duas abordagens anteriores que possuíam arquivos XML apenas para versões

antigas. A necessidade de se ter um arquivo XML da versão atual vem do fato de algum desenvolvedor querer visualizar um artefato que esteja sob GCS e em alteração, não sendo ele um dos responsáveis. Assim, o arquivo XML atual seria exibido, ao invés das informações da base de dados.

Essa abordagem possui as vantagens herdadas das duas anteriores. Primeiramente, não prejudica a utilização das demais ferramentas do ambiente, pelo fato das alterações em artefatos serem realizadas na própria base de dados e não em um local isolado, como na abordagem em arquivos.

Em segundo lugar, é possível acessar as partes que compõem os artefatos, devido ao fato deles estarem armazenados em uma base de dados com uma estrutura previamente definida. Além disso, não demanda mudanças na camada de persistência de ODE, por existir apenas uma base de dados, ao invés de duas, e, por conseguinte, não haver o problema de conexão com bases de dados distintas.

Em terceiro lugar, tem-se a vantagem de não impactar no tempo de execução e de resposta das ferramentas, nem no espaço de armazenamento utilizado, devido ao fato de que apenas os itens atuais são armazenados na base de dados, ficando as versões antigas em arquivo.

Fora as vantagens encontradas nas abordagens anteriores, essa nova abordagem possui um grande diferencial em termos de simplicidade de implementação. Apesar da primeira opção (artefatos armazenados na forma de arquivos) ter sido a mais simples de todas, possuía desvantagens cruciais que fizeram com que não se optasse por ela, como problemas de integração com outras ferramentas, falta de acesso à estrutura interna dos artefatos, além da necessidade de alterações na filosofia de armazenamento de dados de ODE. A abordagem escolhida, de certo modo, concilia a abordagem de arquivos com as vantagens das abordagens discutidas anteriormente, facilitando bastante à implementação.

A abordagem escolhida minimiza os problemas de acesso pelas ferramentas, uma vez que é necessário apenas implementar um método que controle o acesso às ferramentas quando se for fazer uma alteração em um dado artefato. Assim, um usuário só tem acesso à ferramenta para realizar alterações em certo artefato, se esse artefato estiver sob GCS, estiver em alteração e o usuário for um dos responsáveis pela alteração. Nas demais situações, o arquivo XML atual do artefato é exibido usando XMLDoc (SILVA, 2004), não havendo, portanto, necessidade de maiores controles, uma vez que XMLDoc exibe as informações apenas para leitura, não permitindo realizar alterações.

Outra grande vantagem desta abordagem é a facilidade de integração da GCS com a Gerência de Conhecimento, em que pouquíssimas alterações seriam necessárias. A seção 4.3 apresenta os detalhes dessa integração.

Vale destacar que essa abordagem mantém o contratempo de ter que se transformar um determinado arquivo XML em um artefato sob a forma de objetos, com armazenamento em banco de dados, quando se desejar alterar alguma versão que não a mais atual de um artefato. Essa transformação de representação XML para objetos passa a ser necessária, também, quando um desenvolvedor, tendo efetuado o *checkout* de um artefato e realizado alguma alteração, desiste da alteração no momento do *checkin*. Neste caso, deve-se recuperar a última versão do artefato disponível em XML e retorná-la à base de dados, sobrepondo as possíveis alterações realizadas. Porém, essas são situações pouco usuais e, portanto, não representam desvantagens significativas para a abordagem escolhida.

Em suma, esta opção facilita a garantia da consistência dos artefatos, uma vez que precisa lidar apenas com uma base de dados e não há o conceito de cópia, além de não ter grandes dificuldades técnicas. Além disso, não se perde um grande fator motivador desta opção: acessar a estrutura interna dos artefatos e manter interoperabilidade entre as ferramentas do ambiente, em especial com a Gerência de Conhecimento. Pela relação custo/benefício, esta foi a abordagem escolhida para este trabalho, ou seja, para a infra-estrutura de Gerência de Configuração de ODE, utilizada como base para a ferramenta de GCS apresentada no capítulo 5.

4.2 Integrando a GCS à Documentação e às Demais Ferramentas de ODE

Definida a infra-estrutura de Gerência de Configuração, deve-se agora definir como ocorrerá a integração das demais ferramentas internas de ODE com a GCS, em especial a ferramenta de apoio à documentação XMLDoc (SOARES, 2002; SILVA, 2004).

A princípio, para que uma ferramenta possa utilizar a Gerência de Configuração, ela deve adaptar seus métodos de acesso a artefatos ou até mesmo de acesso à própria ferramenta. Para tal, inicialmente, devem ser listados os artefatos e, uma vez selecionado um deles, deve-se verificar se esse está acessível pelo usuário ou não. Um artefato é dito acessível a um determinado recurso humano se: (i) ele não está sob GCS ou (ii) está sob GCS, está em

alteração e o recurso humano em questão é um dos responsáveis pela alteração. Nesses casos, a ferramenta é aberta para que o usuário possa trabalhar no artefato desejado, inclusive realizando alterações. Nos demais casos, ou seja, se um artefato não estiver acessível (não há solicitação de alteração, usuário não é um dos responsáveis etc), deve ser exibido seu arquivo XML atual, através de XMLDoc.

Nos casos em que um artefato não estiver acessível, fica a cargo da ferramenta decidir como se dará o bloqueio. Caso para a atividade do processo em questão haja vários artefatos em desenvolvimento, uma solução consiste em bloquear as funcionalidades da ferramenta tomando por base o artefato selecionado. Assim, se um artefato não acessível for solicitado, ele apenas será disponibilizado para visualização em XMLDoc. Porém, para um outro artefato que esteja acessível, o usuário poderá utilizar livremente as funcionalidades de edição do artefato. Por outro lado, caso para a atividade do processo em questão haja apenas um único artefato em desenvolvimento, a solução pode ser simplificada pelo bloqueio da própria ferramenta. Assim, se o único artefato em desenvolvimento da atividade estiver acessível, a ferramenta será disponibilizada. Caso contrário, o artefato será exibido em XMLDoc, sem sequer, realizar uma chamada à ferramenta de edição, sendo esse controle feito pelo ADS.

Deve-se ressaltar ainda, que, caso a atividade sendo apoiada pela ferramenta permita a criação de mais de um artefato de um mesmo tipo, o bloqueio não deve impedir a criação de novos artefatos. Seja, por exemplo, a ferramenta OODE (BARRETO, 2002) de apoio à modelagem usando UML. Dado que na atividade de especificação de requisitos podem ser desenvolvidos vários diagramas de casos de uso, o bloqueio não pode ser feito diretamente nas funcionalidades da ferramenta nem na ferramenta como um todo, mas sim deve considerar o artefato selecionado. Ou seja, quando for solicitado um diagrama de casos de uso acessível, ele será disponibilizado na ferramenta para alteração; quando for solicitado um diagrama não acessível, ele será aberto em XMLDoc, sendo, portanto, acessível apenas para leitura.

Por outro lado, caso a atividade sendo apoiada pela ferramenta permita a criação de apenas um artefato de um mesmo tipo para o projeto, o acesso à ferramenta pode ser bloqueado. Esse é o caso da ferramenta de apoio à Gerência de Riscos de ODE – GeRis (FALBO et al., 2004c). Como só pode haver um plano de riscos para um projeto, caso esse plano de riscos esteja sob GCS e não possa ser alterado em certo momento, o acesso à ferramenta como um todo será bloqueado.

Assim que uma ferramenta for adaptada segundo a abordagem anteriormente descrita, estará apta para ser integrada à ferramenta de GCS de ODE, que é discutida no capítulo 5, assim como os detalhes técnicos de sua integração com as demais ferramentas de ODE.

4.3 Adaptação da Gerência de Conhecimento de ODE à GCS

Conforme discutido no capítulo 2, deve ser estabelecida uma relação entre as bases da Gerência de Configuração e da Gerência de Conhecimento, ou seja, entre o repositório de dados, com seus itens de configuração, e entre a memória organizacional, com seus itens de conhecimento.

De acordo com a infra-estrutura definida para a GCS de ODE, o repositório da GCS é composto de uma única base de dados, onde são armazenadas as variações atuais dos artefatos que estão sob Gerência de Configuração, e uma estrutura de arquivos, que armazena tanto as variações antigas como as atuais de tais artefatos, em formato XML. O ambiente ODE, por sua vez, possui, ainda, os artefatos que não estão sob GCS e demais informações pertinentes que não estejam na forma de artefatos, tais como informações relativas a processos, atividades, dentre outros. Pode-se dizer, então, que a base de dados da GCS e sua estrutura de arquivos XML é um subconjunto do repositório central de ODE.

No caso da Gerência de Conhecimento, conforme discutido na seção 4.1, verificou-se que não seria interessante o acesso a artefatos que não estivessem sob GCS, uma vez que estes ainda não se encontrariam em um formato maduro o suficiente a ponto de serem utilizados como itens de conhecimento. Da mesma forma, não teria muita serventia exibir as diversas variações de um mesmo artefato. Optou-se, portanto, em utilizar como itens de conhecimento de ODE apenas as variações atuais dos artefatos que estivessem sob GCS. Desta forma, a memória organizacional, no que tange a artefatos, é um subconjunto do repositório de dados da GCS. Lembrando, porém, que a memória organizacional possui, ainda, lições aprendidas e pacotes de mensagens, como mostra a figura 4.5.

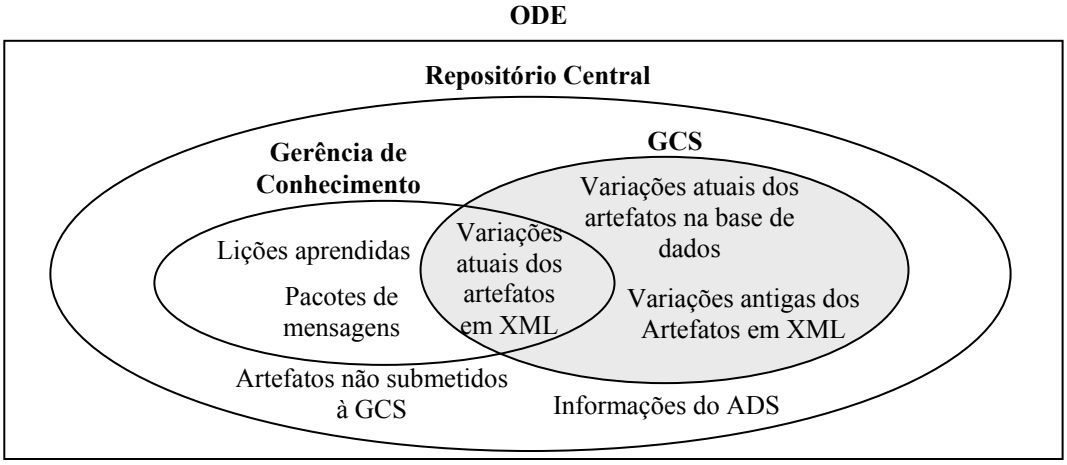


Figura 4.5 – Relação entre as bases de dados do repositório central de ODE, da GCS e da Gerência de Conhecimento.

Entretanto, as alterações que, por ventura, estejam sendo feitas em certo artefato não devem ser exibidas, uma vez que elas ainda não foram avaliadas e registradas, podendo conter erros. Como a base de dados de ODE, e consequentemente da GCS, é única e acessá-la significa visualizar as alterações em andamento em determinado artefato, decidiu-se por exibir aos usuários o arquivo XML da variação mais atual do referido artefato, ao invés de deixá-los acessar a base de dados. Desta forma, tem-se a garantia de estar visualizando sempre a variação mais atual do artefato, sem as alterações em andamento e não se corre o risco de ter o artefato alterado acidentalmente, uma vez que ele é acesso através de XMLDoc e não através da ferramenta de edição de artefatos.

Percebe-se, então, que a adaptação da Gerência de Conhecimento de ODE à infra-estrutura de GCS proposta é praticamente uma atividade trivial. Como só serão exibidos pela Gerência de Conhecimento os artefatos que estiverem sob Gerência de Configuração, o primeiro passo consiste em verificar se um artefato está ou não sob GCS. Além disso, sempre que for desejado exibir determinado artefato, deve-se obter o arquivo XML da variação mais atual do artefato e chamar XMLDoc para exibi-la.

Por fim, a última questão abordada no capítulo 2 diz respeito à definição de um padrão para compartilhamento dos artefatos. Conforme discutido no capítulo 3, a solução para tal problema foi a construção de uma ontologia de artefatos, a fim de que todas as ferramentas de ODE criem seus artefatos e os compartilhem segundo essa ontologia.

4.4 Conclusões do Capítulo

O acompanhamento e o controle dos artefatos, por meio de um processo de gerência de configuração de software, são de suma importância durante todo o ciclo de vida do software (SANCHES, 2001b) e, portanto, imprescindíveis para um ADS.

Porém, antes de se construir uma ferramenta para atender a tal propósito, deve-se avaliar o impacto que ela causará no restante do ADS. Não é interessante construir uma ferramenta que provoque uma grande mudança conceitual ou uma reestruturação de todo ambiente. Da mesma forma, não se deseja perder as facilidades de suas ferramentas e a comunicação entre elas. Deve-se pensar, então, em alterar o mínimo possível e, ainda assim, prover todos os serviços desejáveis da Gerência de Configuração.

Foram analisadas algumas abordagens para definição da infra-estrutura de ODE, das quais merecem destaque: tratar itens de configuração como arquivos; dispor de duas bases de dados e arquivos XML; dispor de uma única base de dados, cópias nessa base e arquivos XML; e dispor de uma única base de dados e arquivos XML. A primeira opção foi descartada principalmente por problemas de integração com as demais ferramentas do ambiente. O fator marcante para descarte das outras duas abordagens foram as dificuldades técnicas decorrentes da alta complexidade da solução. A última opção foi a escolhida, pois além de gerar poucas alterações no restante do ambiente, mantém a integração com as outras ferramentas e não possui grande dificuldade técnica.

Com a estrutura escolhida, foi tratada a integração das demais ferramentas de ODE à GCS, dando destaque especial à ferramenta de apoio à Documentação, XMLDoc. Foram citadas, ainda, as funcionalidades que a GCS deve prover para serem utilizadas pelas demais ferramentas.

Finalmente, foi tratada a adaptação da Gerência de Conhecimento à GCS, em ODE. Foi estabelecida uma relação entre as bases de dados da GCS e da Gerência de Conhecimento e o repositório central de ODE e foi mostrado como a GCS pode ser facilmente integrada à Gerência de Conhecimento, que trabalhará apenas com as variações atuais dos artefatos sob GCS, em formato XML.

Após terem sido definidos a infra-estrutura de Gerência de Configuração de ODE e o tratamento que a ser dado para sua integração às demais ferramentas, o passo seguinte é a construção da ferramenta propriamente dita, a ser discutida no próximo capítulo.

Capítulo 5

Uma Ferramenta de Apoio à Gerência de Configuração de Software em ODE

Uma vez definida a infra-estrutura de Gerência de Configuração e como será estabelecida sua integração com as demais ferramentas do ambiente, tem-se a base necessária para criação de uma ferramenta de apoio à GCS em ODE.

Na literatura de GCS, muitos autores fazem uma distinção entre sistemas de GCS e ferramentas de GCS. Os sistemas de GCS podem ser considerados como parte integrante de ambientes de desenvolvimento de software (ADSs), que dão o suporte à GCS, enquanto as ferramentas de GCS são ferramentas independentes (*stand-alone*), que podem ser adaptadas a quaisquer ADSs ou usadas isoladamente(DART, 1991). Um sistema de GCS tem a vantagem de ser criado especificamente para um ADS, podendo, assim, explorar muitas características internas deste. Por outro lado, não seria facilmente integrado a outro ADS. Enquanto as ferramentas de GCS apresentam a situação inversa: podem ser facilmente adaptadas a qualquer ADS, mas em compensação, não exploram particularidades de nenhum deles.

Neste trabalho, não há distinção entre tais conceitos, ainda que o sistema desenvolvido assuma, segundo essa distinção, características de um sistema de GCS. Assim, o sistema (ou ferramenta) de GCS de ODE é integrado a ele, tornando possível a comunicação desta ferramenta com as demais ferramentas do ambiente, como, por exemplo, com o sistema de Gerência de Conhecimento proposto em (NATALI, 2003). Além disso, para dar uma base sólida e conceitualmente genérica, a ferramenta foi construída com base na ontologia de artefatos, apresentada no capítulo3, mais especificamente, a ontologia de GCS, reunindo os conceitos comumente utilizados na literatura e por vários sistemas de GCS.

Como este sistema é parte integrante de ODE, ele possui a vantagem de poder lidar com seus artefatos, ou seja, ao invés de enxergar um artefato como um arquivo ou um pacote fechado, ele pode enxergar artefatos como sendo compostos de partes, tendo estrutura própria. Porém, como artefatos são utilizados por várias ferramentas de ODE, sua estrutura também é baseada na ontologia de artefatos.

Vale ressaltar que para tratar a questão de desenvolvimento paralelo, optou-se pelo bloqueio, ou seja, se um desenvolvedor estiver alterando a variação de um artefato sob GCS, nenhum outro desenvolvedor poderá alterá-lo enquanto o primeiro desenvolvedor não concluir suas alterações (*checkin*). Apesar dessa opção ter a desvantagem de não permitir alterações paralelas em um mesmo artefato, tem a grande vantagem da segurança na consistência dos dados. Isso é particularmente importante para a infra-estrutura adotada, que armazena artefatos em base de dados, com informações distribuídas em várias tabelas, e não em um sistema de arquivos.

REZENDE (2001) fez uma primeira proposta de ferramenta de GCS para ODE. Porém esta ferramenta foi construída isolada de ODE e sem fundamentação nas ontologias de artefato e na infra-estrutura definida para GCS, uma vez que estas ainda não haviam sido criadas. Os modelos de especificação de requisitos, análise e projeto de tal trabalho foram, então, adaptados, para que uma nova ferramenta de GCS pudesse ser criada, desta vez, incorporada a ODE e baseada na infra-estrutura, nos critérios de integração com outras ferramentas e nas ontologias definidas.

Este capítulo apresenta a ferramenta de apoio à Gerência de Configuração de ODE. A seção 5.1 apresenta as funcionalidades da ferramenta, através de modelos de casos de uso. Na seção 5.2 é discutida sua estrutura interna, através de diagramas de classes. A seção 5.3 mostra a ferramenta propriamente dita. Na seção 5.4 discute-se um exemplo de utilização da ferramenta de GCS por outras ferramentas, no caso por GeRis, a ferramenta de Gerência de Riscos de ODE (FALBO et al., 2004c). Finalmente, a seção 5.5 apresenta as conclusões do capítulo.

5.1 Funcionalidades da Gerência de Configuração de ODE

Nesta seção são apresentados os casos de uso necessários para modelar as funcionalidades que a Gerência de Configuração de ODE deve oferecer, elaborados com base no modelo proposto em (REZENDE, 2001). Algumas dessas funcionalidades devem estar disponíveis na ferramenta de gerência de configuração de software. Outras, porém, devem ser oferecidas pelo próprio ambiente ODE.

O diagrama de casos de uso principal, mostrado na figura 5.1, apresenta as funcionalidades que a Gerência de Configuração de ODE deve fornecer. Os casos de uso identificados foram: cadastrar os itens de configuração que fazem parte de um projeto,

controlar as solicitações de alteração sobre um item de configuração, controlar a alteração sobre um item de configuração, relatar o estado de configuração de um item de configuração, abrir um artefato e salvar um artefato como outro artefato.

Nesta seção apenas é apresentada uma descrição sucinta de cada caso de uso. A especificação de casos de uso completa encontra-se no Apêndice A.

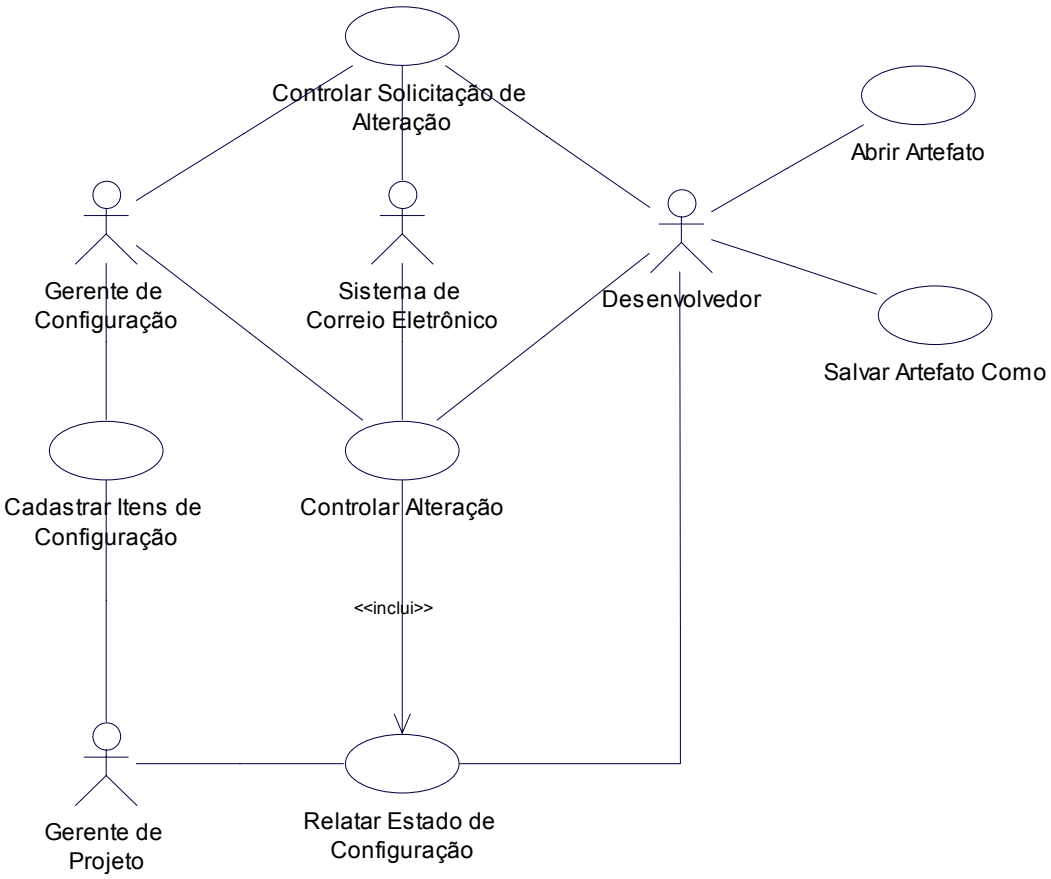


Figura 5.1 – Diagrama de caso de uso principal.

Os atores envolvidos são o Gerente de Configuração, o Desenvolvedor e o Gerente de Projeto. O Gerente de Configuração é responsável por controlar as alterações sobre os itens que estão sob gerência de configuração, o Desenvolvedor é o responsável pelas alterações em um item de configuração e o Gerente de Projeto acompanha o estado dos itens de configuração dos projetos que gerencia. O ator Sistema de Correio Eletrônico é um sistema externo que está sendo integrado ao ambiente ODE e que possui a funcionalidade de enviar e receber mensagens eletrônicas (*e-mails*) (FALBO et al., 2004b).

Os casos de uso Controlar Solicitação de Alteração e Controlar Alteração possuem diagramas próprios e, portanto são descritos posteriormente. A seguir, tem-se a descrição dos demais casos de uso.

- **Caso de Uso *Cadastrar Itens de Configuração* –** Descreve as atividades realizadas pelo Gerente de Configuração ou Gerente de Projeto para criar um novo item de configuração, ou seja, colocar um item (artefato ou ferramenta de software) sob gerência de configuração. Permite, ainda, alterar, consultar e excluir um item de configuração. Ao se criar um novo item de configuração, é criada uma variação para ele e um arquivo XML.

- **Caso de Uso *Relatar Estado de Configuração* –** Este caso de uso permite ao Desenvolvedor ou ao Gerente de Projeto verificar o estado de um artefato que esteja sob gerência de configuração, bem como ao Gerente de Projeto verificar o estado de um projeto como um todo.

- **Caso de Uso *Abrir Artefato* –** Este caso de uso ocorre quando o desenvolvedor abre um artefato em alguma ferramenta do ambiente ODE. Caso o artefato esteja disponível ao desenvolvedor, a variação, inclusive com as últimas alterações realizadas, é exibida ao desenvolvedor, que poderá realizar novas alterações. Caso contrário, o artefato mais recente é disponibilizado ao desenvolvedor, apenas para leitura, porém sem as alterações que por ventura estejam sendo realizadas.

- **Caso de Uso *Salvar Artefato Como* –** Ao abrir um artefato em alguma ferramenta do ambiente ODE, um desenvolvedor pode constatar que tal artefato possui elementos que gostaria de reutilizar. Desta forma, pode optar por salvar o artefato como outro artefato. Porém, mesmo que o artefato aberto esteja sob gerência de configuração, o novo artefato não estará.

5.1.1 Diagrama de Casos de Uso *Controlar Solicitação de Alteração*

Este caso de uso permite acompanhar todo o processo de solicitação de alteração de um item de configuração do tipo artefato, incluindo o cadastro da solicitação de alteração pelo Desenvolvedor e a aprovação da solicitação pelo Gerente de Configuração, como mostra a figura 5.2.

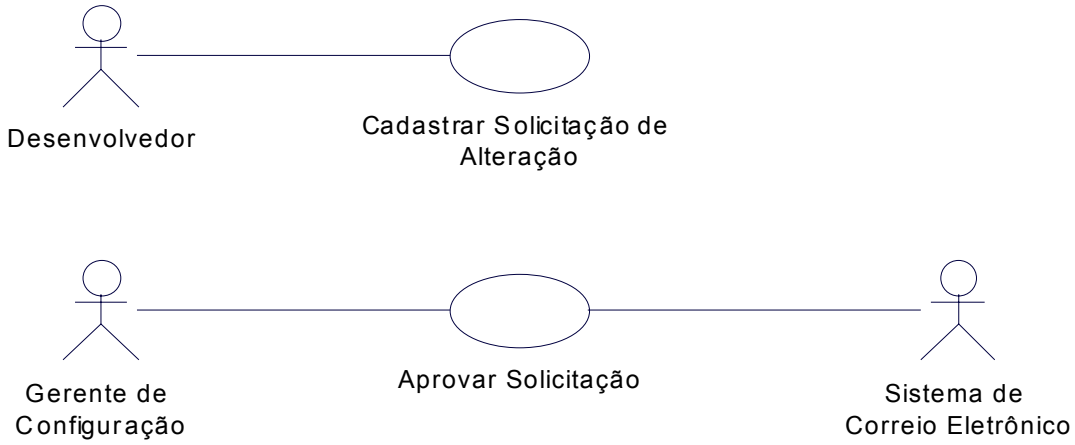


Figura 5.2 – Diagrama de Caso de Uso Controlar Solicitação de Alteração.

- **Caso de Uso *Cadastrar Solicitação de Alteração* –** Por meio desse caso de uso, o desenvolvedor pode fazer uma solicitação de alteração de variações de artefatos que estejam sob gerência de configuração. Pode-se, ainda cancelar, alterar, excluir ou consultar uma solicitação de alteração.

- **Caso de Uso *Aprovar Solicitação* –** Permite ao Gerente de Configuração aprovar uma solicitação de alteração feita por um desenvolvedor, designando os responsáveis pela sua realização. Pode, ainda, incluir na solicitação sub-variações ou variações dependentes, caso julgue necessário.

5.1.2 Diagrama de Caso de Uso *Controlar Alteração*

Este diagrama consiste dos casos de uso Retirar para Alteração e Registrar Alteração, realizados pelo Desenvolvedor; o caso de uso Auditoria de Alteração, que pode ser realizado por um Gerente de Configuração ou por um Gerente de Projeto, e o caso de uso Excluir Alteração, que compete a um Gerente de Configuração. Tais casos de uso são mostrados na figura 5.3 e descritos a seguir.

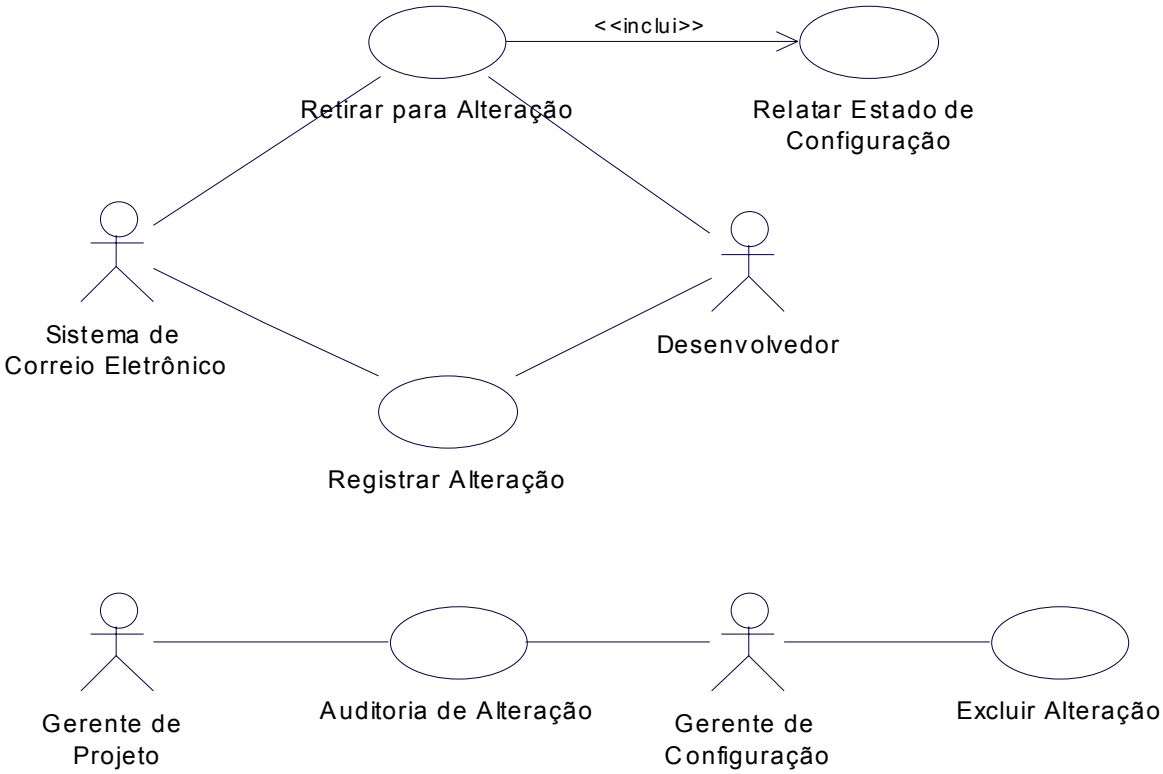


Figura 5.3 – Diagrama de Caso de Uso Controlar Alteração.

- **Caso de Uso *Retirar para Alteração (checkout)* –** Permite ao Desenvolvedor retirar variações de itens de configuração para alteração, após terem sido solicitadas e aprovadas. Caso as variações solicitadas e suas super-variações não estejam em alteração, o *checkout* é realizado e as variações são bloqueadas e disponibilizadas, a fim de que não seja possível haver alterações de desenvolvedores diferentes ao mesmo tempo. Se a variação que está sendo bloqueada é parte da composição de uma outra variação, essa segunda variação também deve ser bloqueada, porém não será disponibilizada aos desenvolvedores, a não ser que tenha sido explicitamente solicitada.

- **Caso de Uso *Registrar Alteração* –** Responsável por registrar as alterações feitas pelo Desenvolvedor. O desenvolvedor deve informar as variações que foram alteradas, suas sub-variações e suas dependências. As novas variações são criadas, assim como os respectivos arquivos XML. Também são criadas novas variações para as super-variações das variações alteradas. Todas as variações envolvidas (alteradas ou não) são desbloqueadas, inclusive suas super-variações.

- **Caso de Uso *Excluir Alteração* –** Permite ao Gerente de Configuração excluir o registro de uma alteração e suas correlações.

- **Caso de Uso *Auditoria Alteração* –** O Gerente de Projeto realiza uma avaliação das alterações realizadas, para que sirva de fonte de consulta.

5.2 Estrutura Interna da Gerência de Configuração em ODE

Neste trabalho, a Gerência de Configuração se dá no contexto de um projeto que está sendo desenvolvido e controlado por um ADS (ODE) e que possui seus artefatos e recursos humanos próprios. Assim, para que seja implementada, é necessário que o processo de software seja previamente definido. Essa etapa de definição do processo é apoiada pelo ambiente, que mantém informações sobre um projeto, o processo para ele definido, suas atividades, recursos etc. Esse conjunto de informações está descrito no pacote *Controle*, (BERTOLLO et al., 2002), apresentado parcialmente na sub-seção 5.2.2.

Além disso, conforme discutido no capítulo 2, muitas das funcionalidades de ODE, incluindo algumas da GCS, dependem de meta-dados descritos no pacote *Conhecimento*. Portanto, esse pacote é também parcialmente apresentado (sub-seção 5.2.1).

A sub-seção 5.2.3 apresenta o pacote *Documentação* (SILVA, 2004), que possui uma grande relevância para este trabalho, em especial pelo fato da estrutura de artefatos seguir a ontologia de artefato definida e devido ao fato das funcionalidades disponibilizadas por esse pacote serem utilizadas pelas demais ferramentas de ODE, dentre elas XMLDoc e a ferramenta de GCS.

Finalmente, na sub-seção 5.2.4, é apresentado o pacote de *Gerência de Configuração* propriamente dito. Vale lembrar que o pacote *Gerência de Conhecimento*, também relevante para este trabalho, foi apresentado no capítulo 2 (figura 2.5) e, uma vez que não sofreu alteração, dada a abordagem escolhida, não será apresentado neste capítulo. A figura 5.4 exibe um diagrama de pacotes mostrando as relações de dependência entre os pacotes citados.

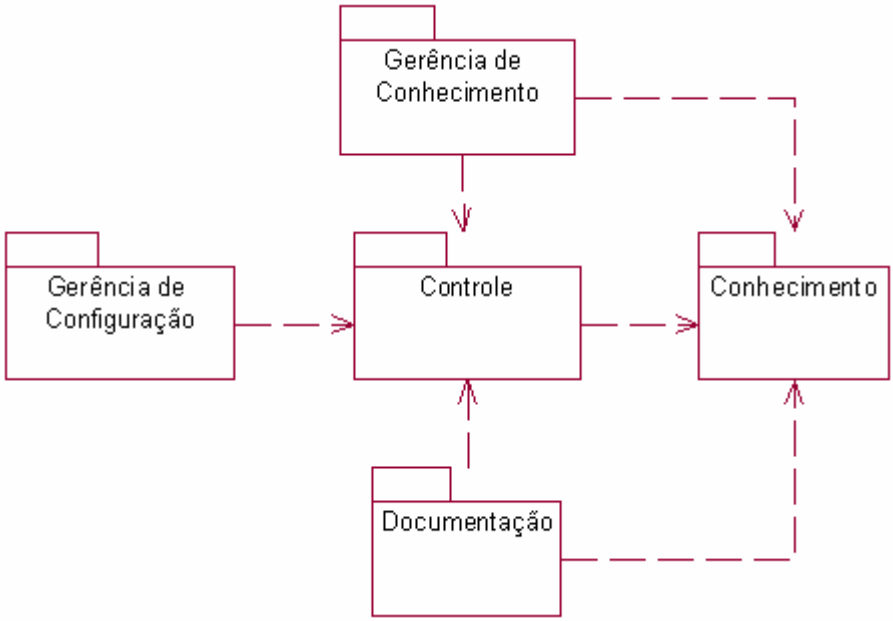


Figura 5.4 – Diagrama de Pacotes.

5.2.1 Pacote *Conhecimento*

O pacote *Conhecimento* define o conhecimento sobre os objetos de certo domínio: suas características, como interage com outros objetos e também as relações existentes com os outros objetos do domínio modelado. Desta forma, provê mecanismos para o armazenamento, acesso e utilização de conhecimento em ODE. As classes desse pacote são diretamente derivadas de ontologias, sendo seus objetos itens da instanciação dessas ontologias. Isto é, o conhecimento modelado diz respeito às ontologias utilizadas em ODE (NATALI, 2003).

A figura 5.5 apresenta um diagrama parcial do pacote *Conhecimento*, com as classes relevantes para este trabalho. Por motivos de padronização, em ODE as classes de conhecimento são precedidas pela letra "K". As classes *KArtefato* e *KFerramentaSoftware*, por exemplo, têm grande importância para este trabalho, pois descrevem no meta-nível o conhecimento relativo às classes *Artefato* e *FerramentaSoftware* (figura 5.6) que são os objetos que poderão ser colocados sob GCS.

Como mostra a figura 5.5, artefatos são os elos de comunicação entre as diversas ferramentas de ODE, uma vez que podem ser insumos ou produtos de várias atividades. Uma atividade, por sua vez, pode adotar procedimentos para sua realização, dentre os quais se incluem os roteiros para confecção de artefatos. Um modelo de documento é um tipo de roteiro estruturado, organizado em modelos de seção que, por sua vez, podem ser compostos por artefatos ou outros modelos de seção. Os conceitos de modelo de documento e modelo de

seção são muito relevantes, pois foram criados a partir da sub-ontologia de Documento apresentada no capítulo 3 e utilizada em (SILVA, 2004) para a adequação de XMLDoc.

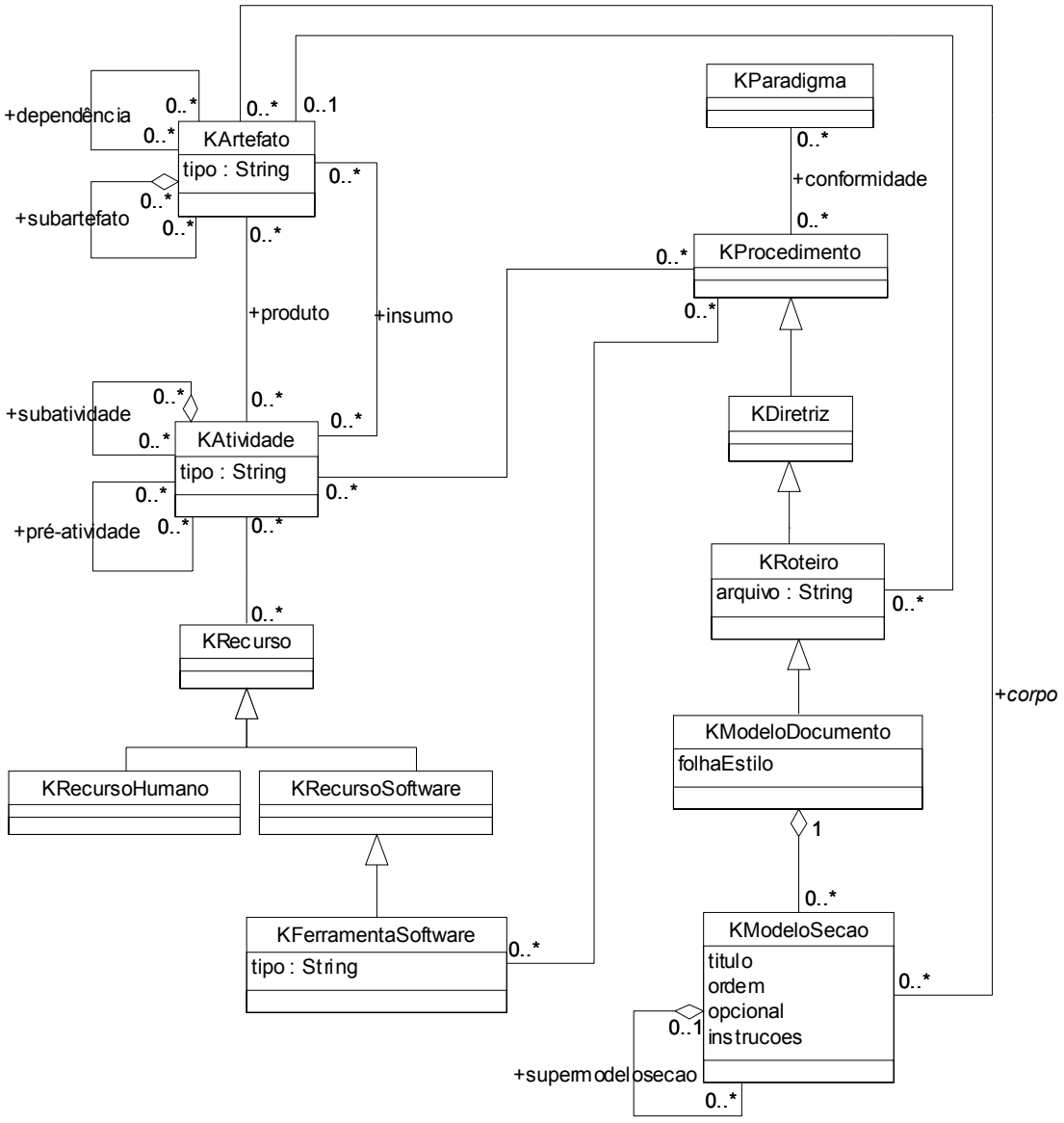


Figura 5.5 - Diagrama de Classes Parcial do Pacote *Conhecimento*.

5.2.2 Pacote *Controle*

As classes do pacote *Controle*, como o próprio nome já diz, são responsáveis pelo controle do processo de software, sendo, portanto, comuns a todas as ferramentas integradas a ODE (BERTOLLO et al., 2002).

No desenvolvimento de software, alguns recursos são necessários para que as atividades de um projeto sejam executadas, tais como recursos humanos e ferramentas de software. Além disso, artefatos e ferramentas de software são peças fundamentais por poderem ser colocados sob gerência de configuração.

No contexto da gerência de configuração, o pacote *Controle* fornece, por exemplo, os artefatos e ferramentas de software de um determinado projeto, os recursos humanos envolvidos em tal projeto e quais deles possuem o papel de gerente de projeto, gerente de configuração ou desenvolvedor (instâncias da classe *KRecursoHumano* do pacote *Conhecimento*).

A figura 5.6 apresenta apenas as classes do pacote *Controle* que são relevantes para este trabalho, a saber: *Projeto*, *Artefato*, *Atividade*, *Recurso* (*FerramentaSoftware* e *RecursoHumano*) e *Equipe*.

Artefatos são produzidos por determinada atividade no contexto de um projeto de software, tornando-se, assim, insumo para outras atividades. Tal projeto possui um recurso humano designado como gerente de projeto e uma equipe composta por diversos recursos humanos, dentre os quais se encontram os responsáveis pela aprovação dos artefatos de tal projeto. Tanto artefatos como ferramentas de software podem ser colocados sob GCS.

Vale ressaltar que algumas classes possuem o atributo *tipo*, estabelecendo uma relação direta com as respectivas classes do pacote *Conhecimento*. Assim, o tipo de um artefato é uma instância da classe *KArtefato*, da mesma forma que o tipo de uma atividade é uma instância da classe *KAtividade* e o tipo de um recurso é uma instância de *KRecurso*. Essa estrutura serve como um esquema de anotação com meta-dados. Desta forma, instâncias de *Atividade* representam atividades concretas realizadas no contexto de um projeto específico, como, por exemplo, *Planejamento Inicial do Projeto X*, *Especificação de Requisitos Preliminar do Projeto X* etc, enquanto instâncias de *KAtividade* representam tipos de atividades no desenvolvimento de software, como, por exemplo, *Planejamento*, *Especificação de Requisitos* etc. Desta forma, o nível de conhecimento pode ser usado para guiar a realização de atividades do nível base.

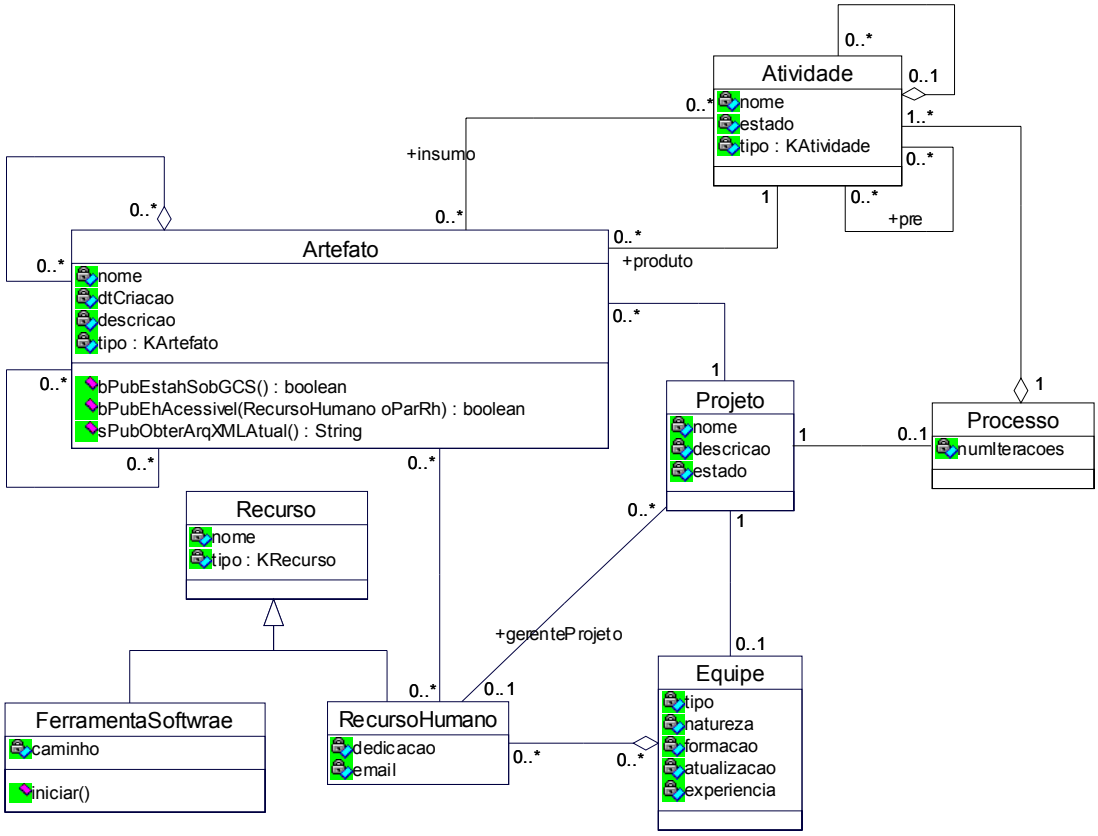


Figura 5.6 – Diagrama de Classes Parcial do Pacote Controle.

5.2.3 Pacote *Documentação*

O pacote *Documentação* foi originalmente desenvolvido em (SOARES, 2002) e posteriormente adaptado em (SILVA, 2004) para adequação à ontologia de artefato proposta neste trabalho. O diagrama de classes do pacote *Documentação* é apresentado na figura 5.7.

Um documento é um tipo de artefato que pode aplicar um modelo de documento. A partir dos modelos de seção do correspondente modelo de documento, são definidas as seções do documento, ou seja, para cada modelo de seção definida pelo modelo de documento, o documento deve ter uma seção associada. Além disso, as seções podem ter como corpo outros artefatos, que devem ser do mesmo tipo de artefato definido como corpo do modelo de seção ao qual a seção está associada. Assim como os modelos de seção, uma seção também pode possuir várias sub-seções.

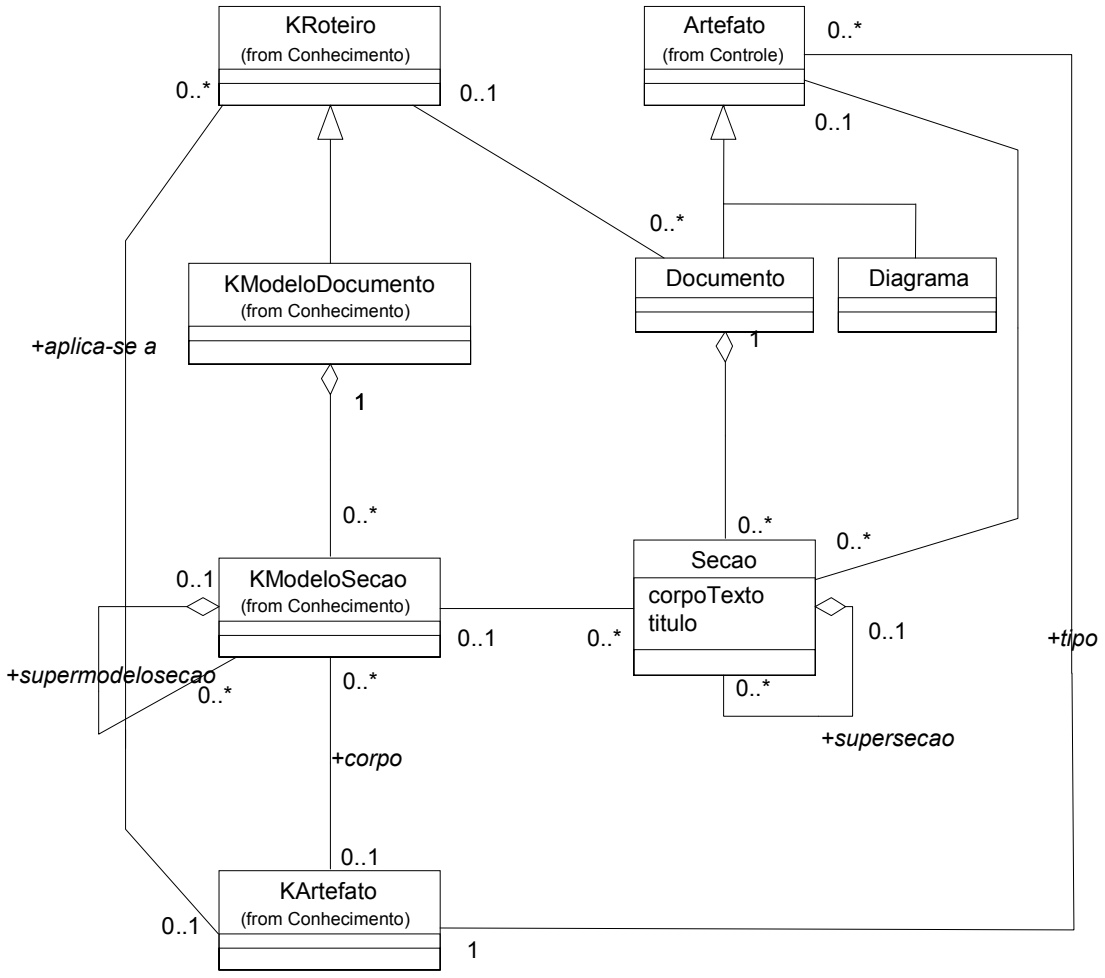


Figura 5.7 - Diagrama de Classes do Pacote Documentação (SILVA, 2004).

5.2.4 Pacote *Gerência de Configuração*

Mudanças em ferramentas de software ou em artefatos ocorrem durante todo o processo de desenvolvimento de software. Desta forma, uma ferramenta de Gerência de Configuração deve ser capaz de controlar tais mudanças, garantir que uma mudança esteja sendo adequadamente implementada e relatar esta mudança a outras pessoas que possam ter interesse. Assim, a Gerência de Configuração pode ser vista como uma atividade que é aplicada durante todas as fases do processo de Engenharia de Software (PRESSMAN, 2002).

Para que haja um controle de alteração, as ferramentas de software ou artefatos devem primeiramente se tornar itens de configuração. Porém, além de selecionar os itens que serão gerenciados, deve-se indicar a composição de cada item e descrever como eles se relacionam.

Isso é muito importante para as futuras manutenções, pois permite identificar de maneira eficaz os itens afetados em decorrência de uma alteração (SANCHES, 2001b).

De posse dos itens de configuração, de suas composições e suas relações de dependência, é possível estabelecer uma linha-base. A partir daí, qualquer alteração a ser realizada em um artefato deve passar por um rigoroso controle, que inclui a solicitação da alteração, sua aprovação, a indicação dos responsáveis pela alteração, a saída para alteração (*checkout*), a alteração em si e o registro da alteração (*checkin*). No caso das ferramentas de software, não é possível realizar alterações, porém a mudança de versão de uma ferramenta para outra também deve ser controlada.

Um desenvolvedor só conseguirá alterar um artefato se tiver acesso a tal artefato e for designado como responsável pela execução. Nenhum desenvolvedor deve conseguir alterar esse mesmo artefato e nem mesmo enxergar as alterações que estão sendo realizadas. Ou seja, o desenvolvedor responsável pela alteração em uma variação de um artefato poderá realizar alterações nela. Aos demais desenvolvedores restará apenas visualizar tais artefatos na sua forma mais atual antes de terem saído para alteração.

Cada alteração ou mudança realizada em um item de configuração indica uma nova versão ou variante deste item, ou seja, uma nova variação.

Porém, mesmo com os mecanismos de controle mais bem sucedidos, não é possível garantir que as modificações foram corretamente implementadas. Assim, auditorias de alteração são necessárias. A auditoria de alteração é o registro de uma atividade humana que avalia se uma mudança foi adequadamente implementada.

Quando a exclusão de uma solicitação de alteração não é desejada ou não é permitida, o desenvolvedor pode, em alguns casos, optar por cancelar tal solicitação. Este fato fica registrado em um cancelamento de alteração.

A figura 5.8 apresenta o diagrama de classes, construído com base na Ontologia de Gerência de Configuração de Software, apresentado no capítulo 3, com o intuito de modelar os conceitos apresentados. É através dessas classes que o controle de mudanças é implementado pela ferramenta de Gerência de Configuração.

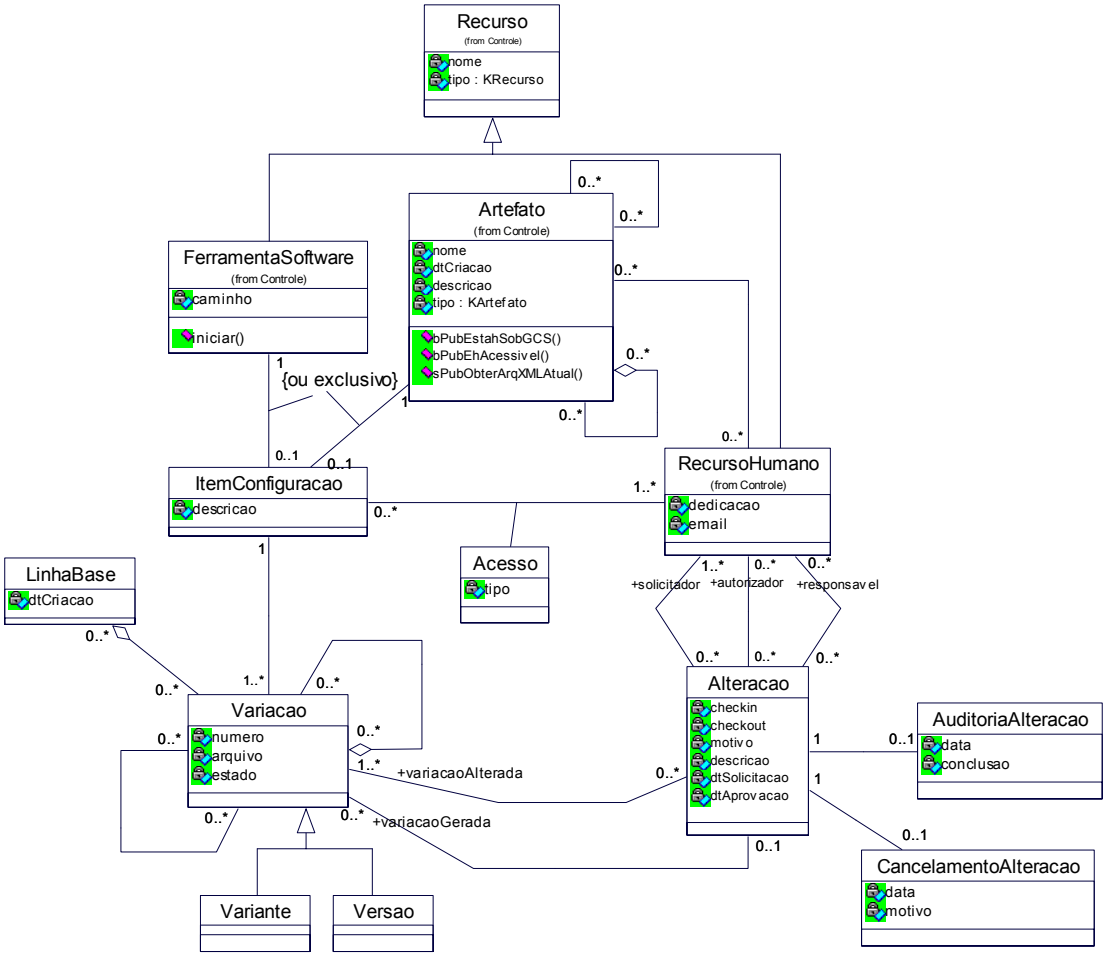


Figura 5.8 – Diagrama de Classes do Subsistema de Gerência de Configuração.

5.3 A Ferramenta de Gerência de Configuração de ODE

Nessa seção são apresentadas as funcionalidades implementadas para a ferramenta de apoio à Gerência de Configuração em ODE, que são baseadas nos casos de uso definidos na seção 5.1 e na estrutura interna da seção 5.2. A maioria dessas funcionalidades é interna à ferramenta, mas algumas estão disponíveis no ambiente ODE, para permitir que a Gerência de Configuração possa ser utilizada pelas suas ferramentas. O uso das funcionalidades da GCS por outras ferramentas de ODE é discutido nas seções seguintes.

É importante lembrar que para integrar a ODE uma nova ferramenta que gere artefatos, ela deve ser capaz de gerar seus artefatos de acordo com a ontologia de artefato, proposta no capítulo 3. Essa é uma premissa do ambiente ODE para estabelecimento de comunicação entre suas ferramentas e entre seus desenvolvedores. Por fim, deve-se

estabelecer as relações de composição e dependência entre os artefatos criados, o que pode ser feito através da ferramenta de Controle de Documentação.

A ferramenta de Controle de Documentação exibida na figura 5.9 tem a finalidade de ser utilizada apenas para informar a composição e as dependências dos artefatos. Assim, a criação de um artefato deve ficar a cargo da sua ferramenta responsável. Por exemplo, um artefato do tipo *Plano de Riscos* deve ser criado através da ferramenta de apoio à Gerência de Riscos, GeRis (FALBO et al., 2004c). Contudo, a indicação de que o plano de riscos é parte de um plano de projeto é feita na ferramenta de Controle de Documentação.

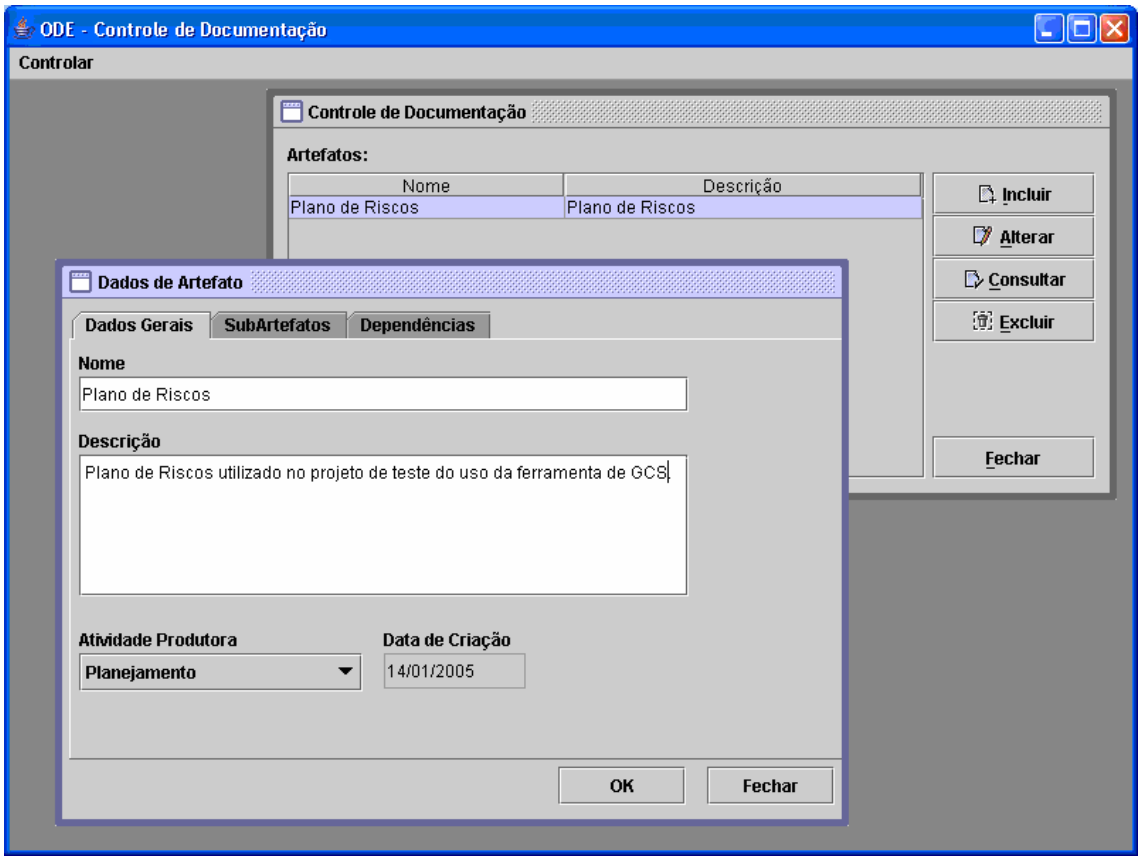


Figura 5.9 – Ferramenta de Controle de Documentação.

Ao se clicar na aba *SubArtefatos*, como o próprio nome diz, é possível selecionar os sub-artefatos de tal artefato, como mostra a figura 5.10. O mesmo ocorre para se efetuar a seleção de dependências.

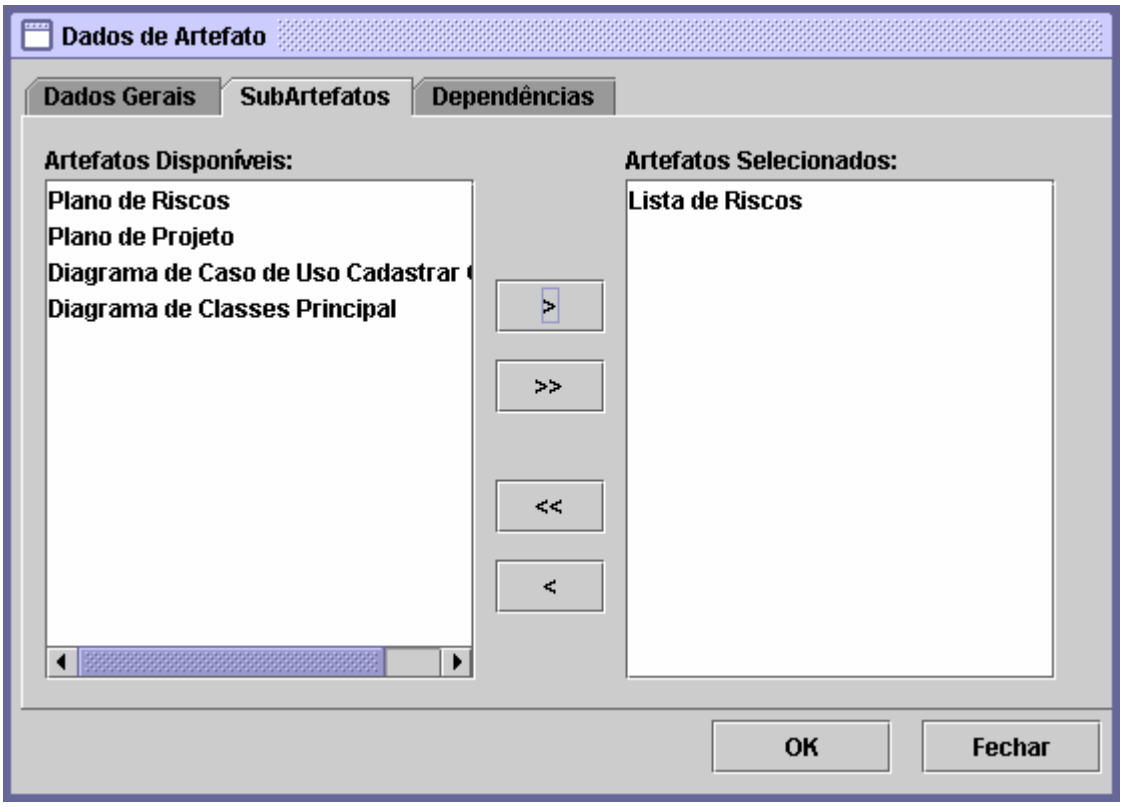


Figura 5.10 – Aba SubArtefatos da ferramenta de Controle de Documentação.

A partir do momento que os artefatos são cadastrados no Controle de Documentação de ODE, eles podem ser submetidos à Gerência de Configuração. Vale destacar, no entanto, que, para que a Gerência de Configuração de ODE seja efetiva, as ferramentas que manipulam os artefatos têm de ser adaptadas para incorporar os mecanismos de controle de acesso aos artefatos, conforme discutido na seção 4.2 do capítulo 4 e tratado com maiores detalhes na seção 5.4 deste capítulo.

A figura 5.11 mostra a janela principal da ferramenta de Gerência de Configuração de Software de ODE, cujo menu principal possui a estrutura apresentada na figura 5.12. Vale destacar que as funcionalidades efetivamente implementadas neste trabalho são aquelas que aparecem em destaque (negrito) na figura 5.12.

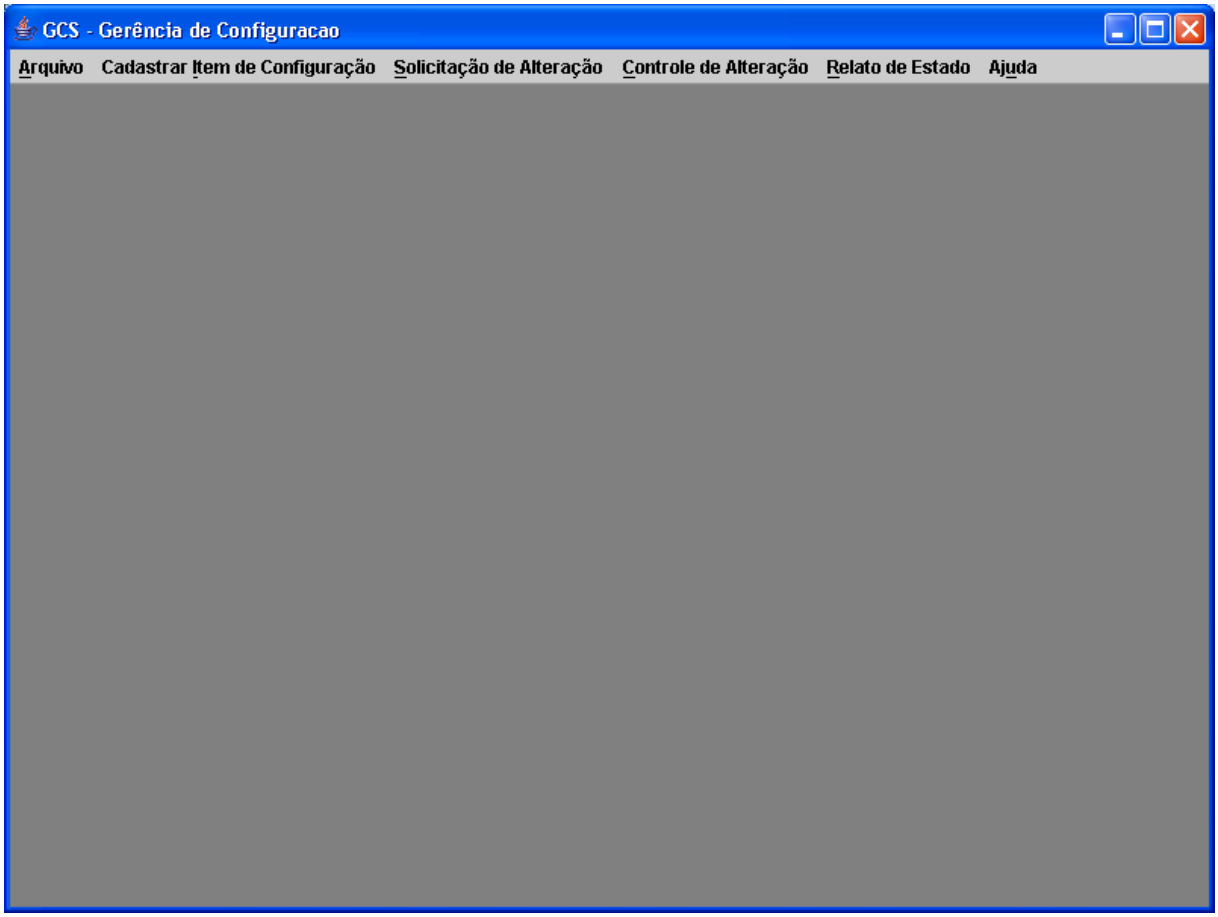


Figura 5.11 – Ferramenta de Gerência de Configuração de ODE.

Figura 5.12 – Estrutura do Menu da Ferramenta de Gerência de Configuração de ODE.

Porém, só serão exibidas no menu principal as funcionalidades a que o usuário tiver acesso. Assim, para o Gerente de Projeto serão exibidas apenas: *Cadastrar Item de Configuração*, *Auditoria de Alteração* e *Relato de Estado*. Para o Gerente de Configuração serão exibidas: *Cadastrar Item de Configuração*, *Aprovar Solicitação de Alteração*, *Excluir Alteração* e *Auditoria de Alteração*. Enfim, para os desenvolvedores serão exibidas: *Cadastrar Solicitação de Alteração*, *Retirar para Alteração*, *Registrar Alteração* e *Relato de Estado*.

Vale destacar que a definição de papéis em ODE é realizada na ferramenta de cadastro de recursos humanos, presente no módulo de cadastro de recursos.

A seguir, cada uma das funcionalidades implementadas é brevemente apresentada.

Cadastrar Item de Configuração

O primeiro passo na utilização da ferramenta é definir os artefatos que serão colocados sob Gerência de Configuração, ou seja, cadastrar os itens de configuração do projeto. A figura 5.13 mostra a janela a partir da qual é possível *incluir*, *alterar*, *consultar* ou *excluir* um item de configuração. Nessa janela são exibidos os artefatos que estão sob GCS (seus nomes e as descrições dos respectivos itens de configuração).

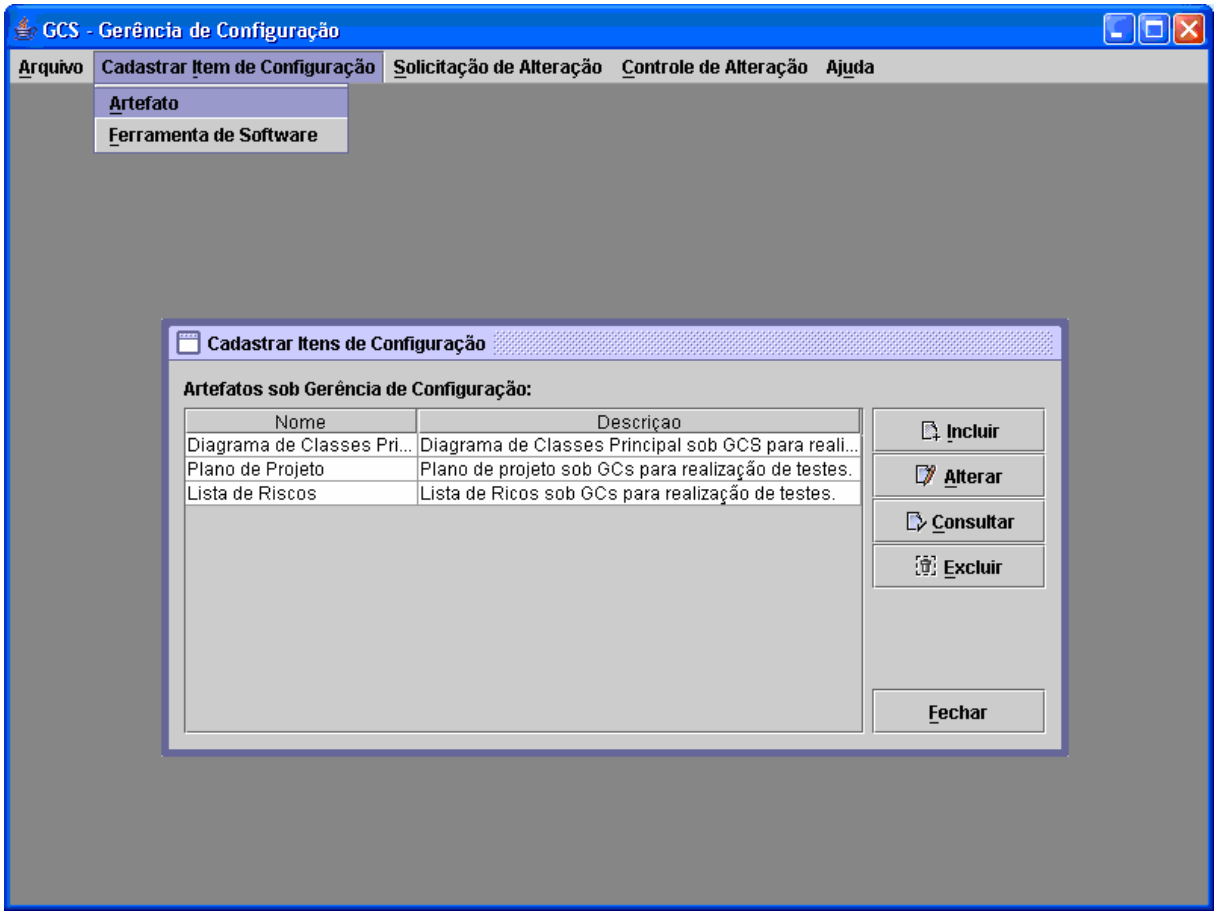


Figura 5.13 – Cadastrar Item de Configuração – Artefato.

Para se incluir um novo item de configuração, deve-se clicar no botão incluir. A janela apresentada na figura 5.14 é, então, exibida ao usuário, apresentando a aba *Dados Gerais*, com as seguintes informações:

- Artefato – o usuário deve selecionar o artefato que deseja colocar sob GCS. Neste campo são listados apenas os artefatos do projeto atual que ainda não estão sob GCS.
- Variação – indica a variação em que o artefato se encontra. Este campo vem automaticamente preenchido com o valor 1.0.0, porém, ele pode ser alterado, se desejado.
- Data de Criação – é a data de criação do item de configuração, ou seja, a data em que o artefato foi colocado sob GCS. Este campo vem automaticamente preenchido com a data atual e não pode ser alterado.
- Descrição – neste campo o usuário deve informar uma descrição para o item de configuração.

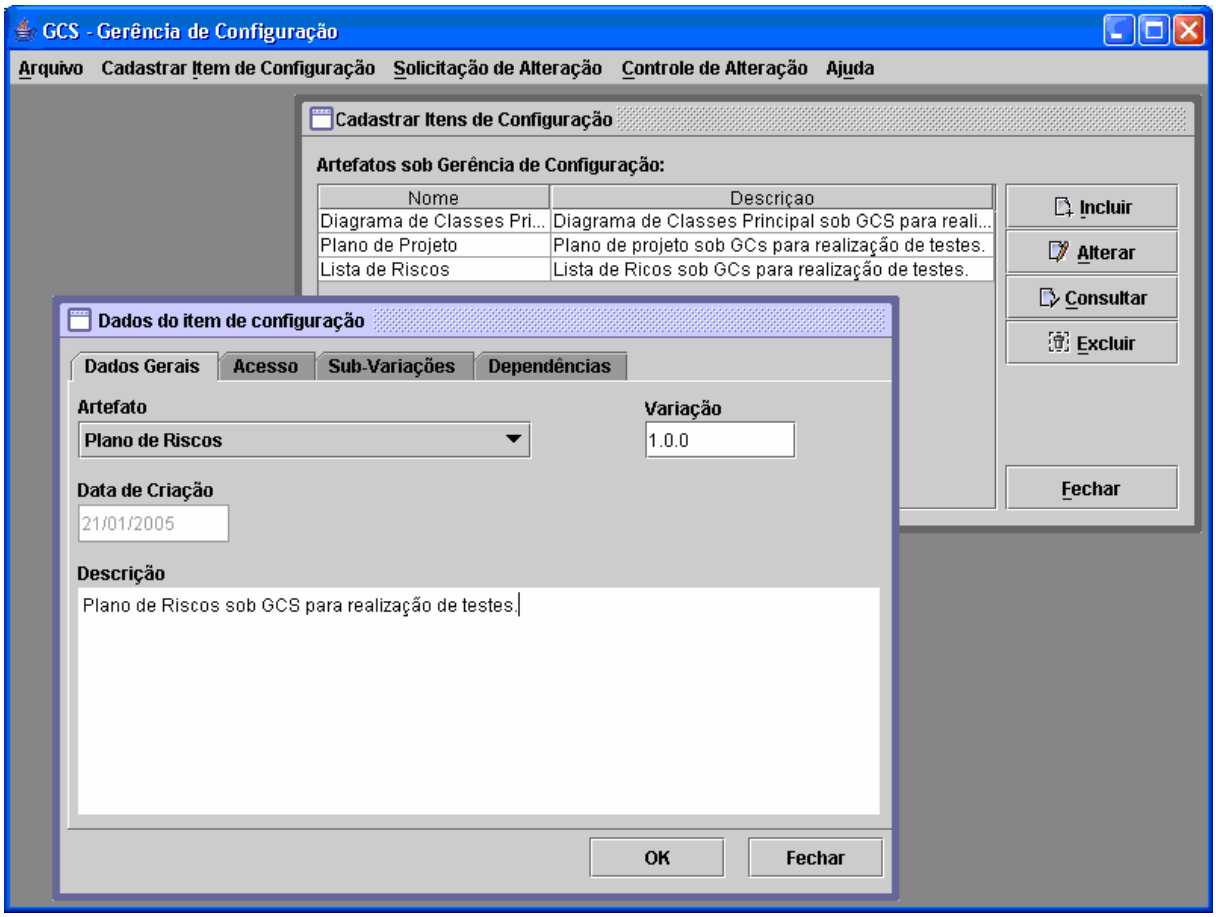


Figura 5.14 – Cadastrar Item de Configuração – Tela de Inclusão – Dados Gerais.

Após fornecer os dados gerais, deve-se informar, ainda, seus acessos, sub-variações e dependências, preenchendo os respectivos formulários nas demais abas. A figura 5.15 mostra

a aba *Acessos*, em que se deve selecionar os recursos humanos que terão acesso a tal item de configuração, com os respectivos tipos de acesso. Para se definir o acesso dado a um recurso humano, basta clicar em tal recurso humano e, em seguida, clicar no tipo de acesso desejado, sendo: *Leitura* (o usuário pode apenas visualizar o item de configuração), *Alteração* (o usuário pode ler e alterar o item de configuração) e *Exclusão* (o usuário pode ler, alterar e excluir um item de configuração).

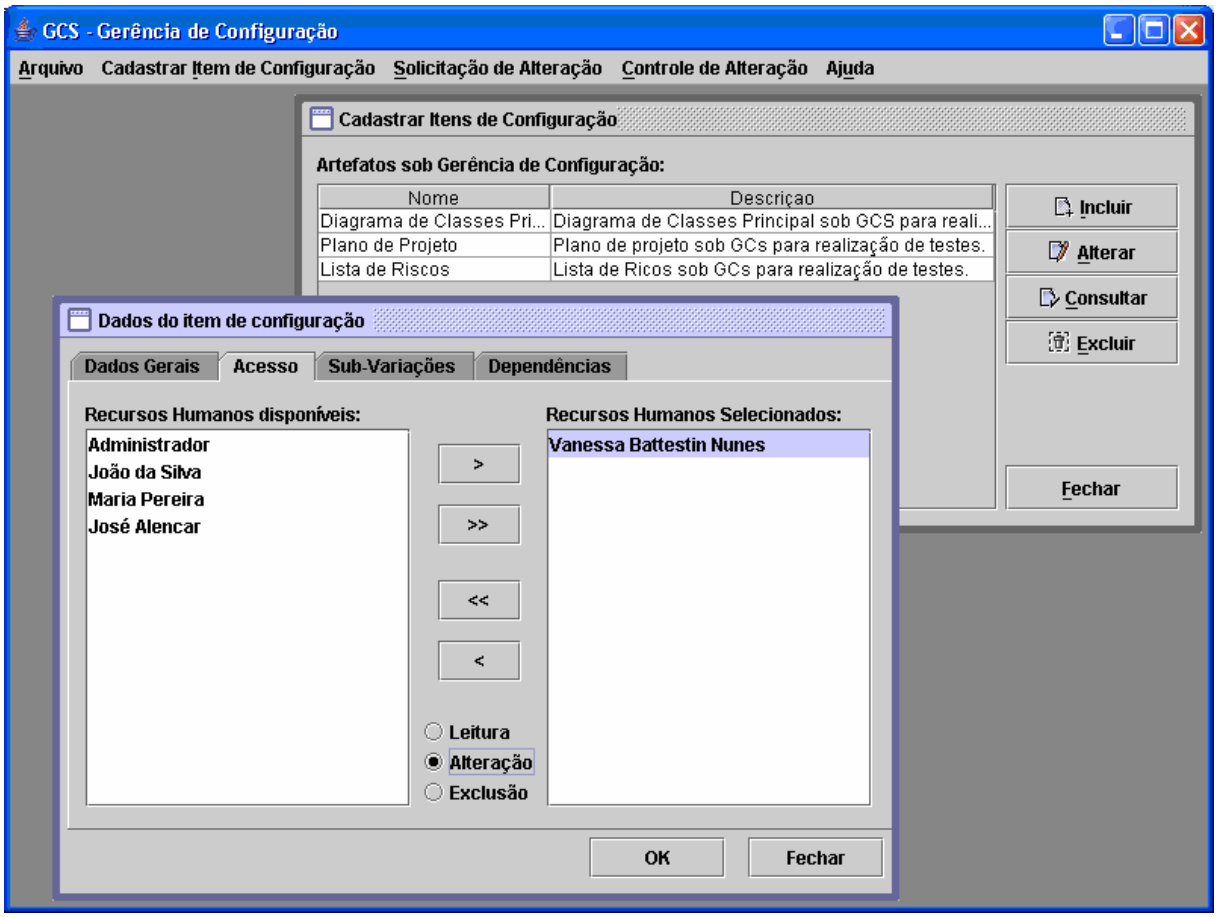


Figura 5.15 – Cadastrar Item de Configuração – Tela de Inclusão – Acesso.

As abas de *Sub-Variações* e *Dependências* são semelhantes às do controle de documentação, mostrado na figura 5.10. Na aba Sub-variações, são exibidas apenas as sub-variações dos artefatos que são sub-artefatos do artefato selecionado na aba *Dados Gerais*, ou seja, os artefatos que não forem sub-artefatos do artefato em questão, não terão suas variações exibidas aqui. Da mesma forma, na aba *Dependências* são exibidas apenas as variações das quais o artefato em questão depende. As duas abas fazem referência a *Variações*, logo, pressupõe-se que todos os artefatos listados tenham variações, ou seja, estejam sob GCS. Desta forma, artefatos que não estiverem sob GCS, não serão citados nesta lista. Porém, caso, futuramente, um de seus artefatos venha a se tornar um item de configuração, poder-se-á incluí-lo na lista, através da alteração do item de configuração.

Após clicar no botão *OK*, o item de configuração será criado, assim como sua variação e seu arquivo XML correspondente. A princípio, a variação estará *desbloqueada*, ou seja, passível de ser retirada para alteração. Porém, caso alguma de suas sub-variações esteja bloqueada, ela também será *bloqueada*.

Alterações e consultas em um determinado item de configuração serão realizadas nas mesmas janelas citadas (figuras 5.14 e 5.15), porém, o conteúdo do artefato em si não poderá ser alterado. Nos casos em que o item de configuração não puder ser alterado, ele será exibido na forma de consulta.

Quando for solicitado excluir um item de configuração, será pedida uma confirmação. Em caso afirmativo, o item de configuração, suas variações e seus arquivos XML serão excluídos.

Solicitação de Alteração

Após itens de configuração terem sido cadastrados, os desenvolvedores podem iniciar as alterações em seus artefatos. Porém, antes disso é necessário fazer uma solicitação de alteração.

Na janela mostrada na figura 5.16 é possível *incluir*, *alterar*, *consultar, excluir* ou *cancelar* uma solicitação de alteração. Nessa janela são exibidos: o identificador da solicitação de alteração, o motivo pelo qual o desenvolvedor deseja realizar a alteração e a situação da alteração.

Deve-se observar que apenas os itens de menu referentes a desenvolvedores estão exibidos nessa figura.

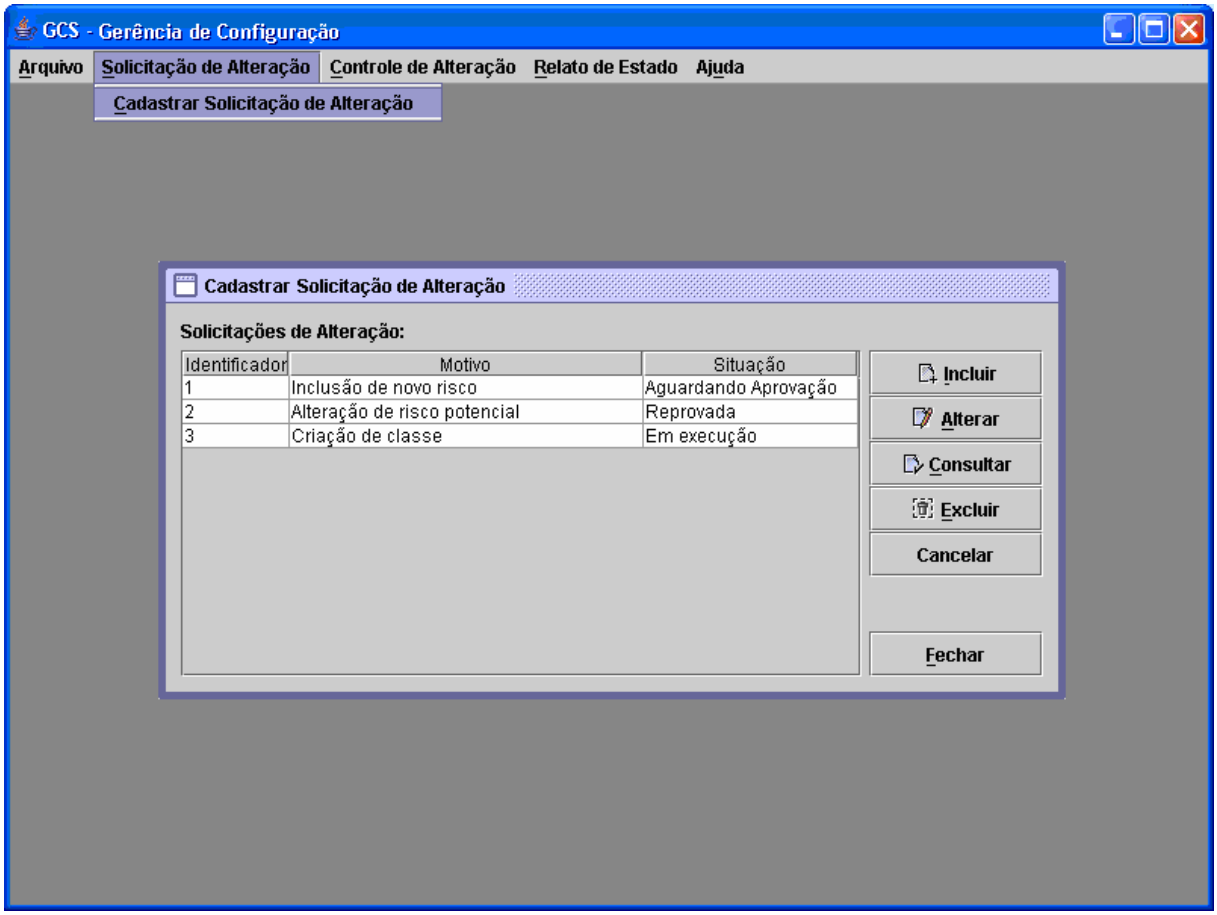


Figura 5.16 – Cadastrar Solicitação de Alteração.

Para se incluir uma nova solicitação de alteração, deve-se clicar no botão incluir. A janela apresentada na figura 5.17 será, então, exibida ao usuário, apresentando a aba *Dados Gerais*, com as seguintes informações:

- Identificador – contém um valor numérico de identificação da solicitação de alteração. Esse valor é gerado automaticamente, incrementando o último identificador existente, e não pode ser alterado.
- Situação – informa o andamento de uma alteração. Os valores possíveis para este campo são: *Aguardando Aprovação* (foi criada uma solicitação de alteração, mas essa ainda não foi aprovada), *Aprovada* (a solicitação de alteração foi aprovada), *Cancelada* (a solicitação de alteração foi cancelada), *Reprovada* (a solicitação de alteração não foi aceita), *Em Execução* (a alteração está em andamento) ou *Finalizada* (a alteração foi concluída). O valor desse campo é preenchido automaticamente e não pode ser alterado.
- Solicitada por – usado para se informar o recurso humano que está efetuando a solicitação. O valor desse campo também é gerado automaticamente, uma vez que o ambiente tem o controle de que usuário o está utilizando, e não pode ser alterado.

- Data de Solicitação – é a data de criação da solicitação de alteração. Esse campo vem automaticamente preenchido com a data atual e não pode ser alterado.
- Motivo – o desenvolvedor deve escrever nesse campo uma justificativa da necessidade de alteração.

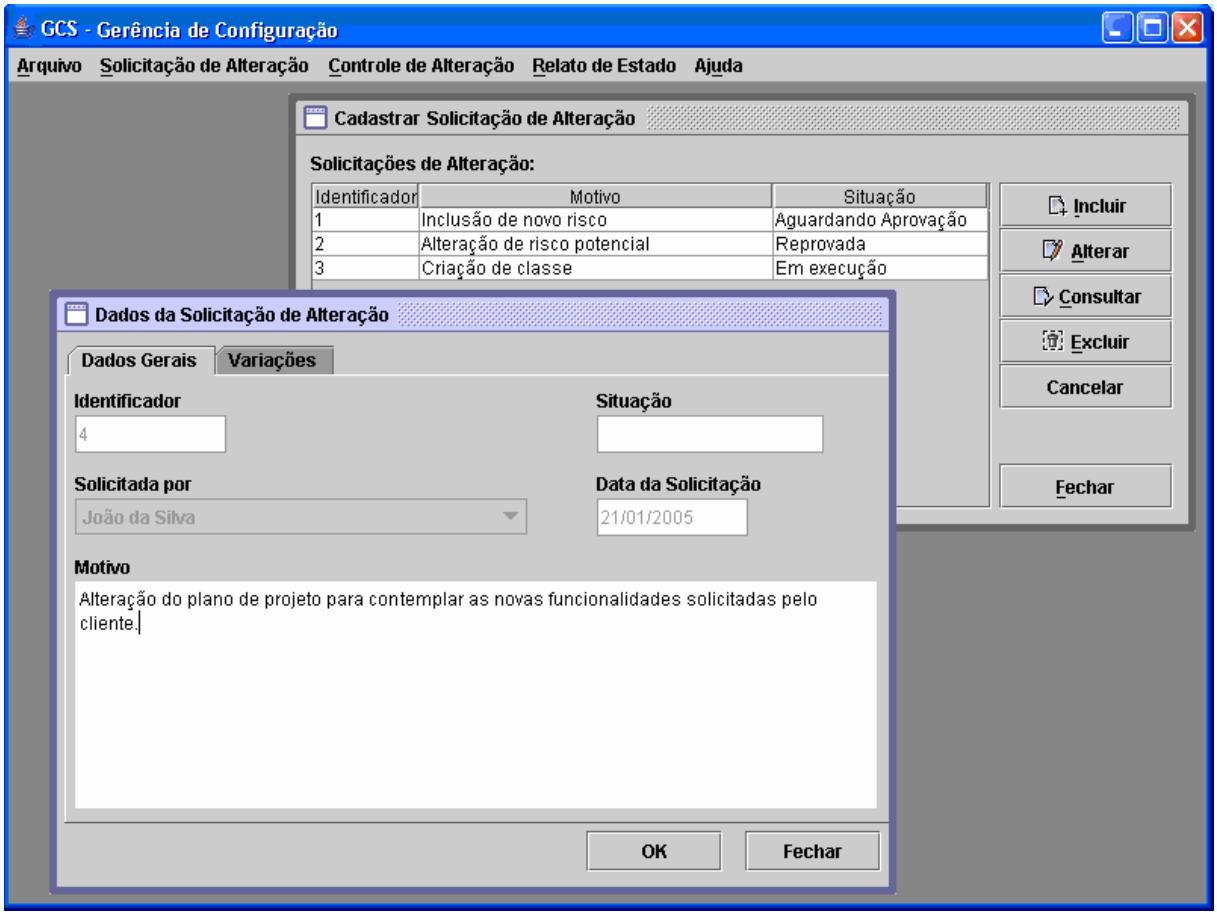


Figura 5.17 – Cadastrar Solicitação de Alteração – Dados Gerais.

Após os dados gerais terem sido informados, o desenvolvedor deve informar quais as variações que deseja alterar. Isso deve ser feito na aba *Variações,* em que são listadas as variações atuais de todos os artefatos do projeto em questão. Essa aba é semelhante à aba de sub-variações do Controle de Documentação, mostrado na figura 5.10.

Após clicar no botão *OK*, a solicitação de alteração será criada, com a situação *Aguardando Aprovação*.

Alterações e consultas em uma determinada solicitação de alteração serão realizadas nas mesmas janelas citadas (figuras 5.16 e 5.17). Porém, após tal solicitação ser aprovada, ela não poderá mais ser alterada nem excluída e será exibida apenas para consulta. Caso se tenha desistido de uma solicitação de alteração, mas ela se encontra em um estágio que não pode ser excluída, existe a opção de cancelá-la.

Aprovar Solicitação de Alteração

Periodicamente, o Gerente de Configuração deverá verificar se há solicitações de alteração aguardando aprovação, acessando o item de menu *Aprovar Solicitação de Alteração*, mostrado na figura 5.18. Diferente da janela de *Solicitação de Alteração*, em que são exibidas todas as solicitações existentes, com suas respectivas situações, aqui são exibidas apenas as solicitações estão aguardando aprovação.

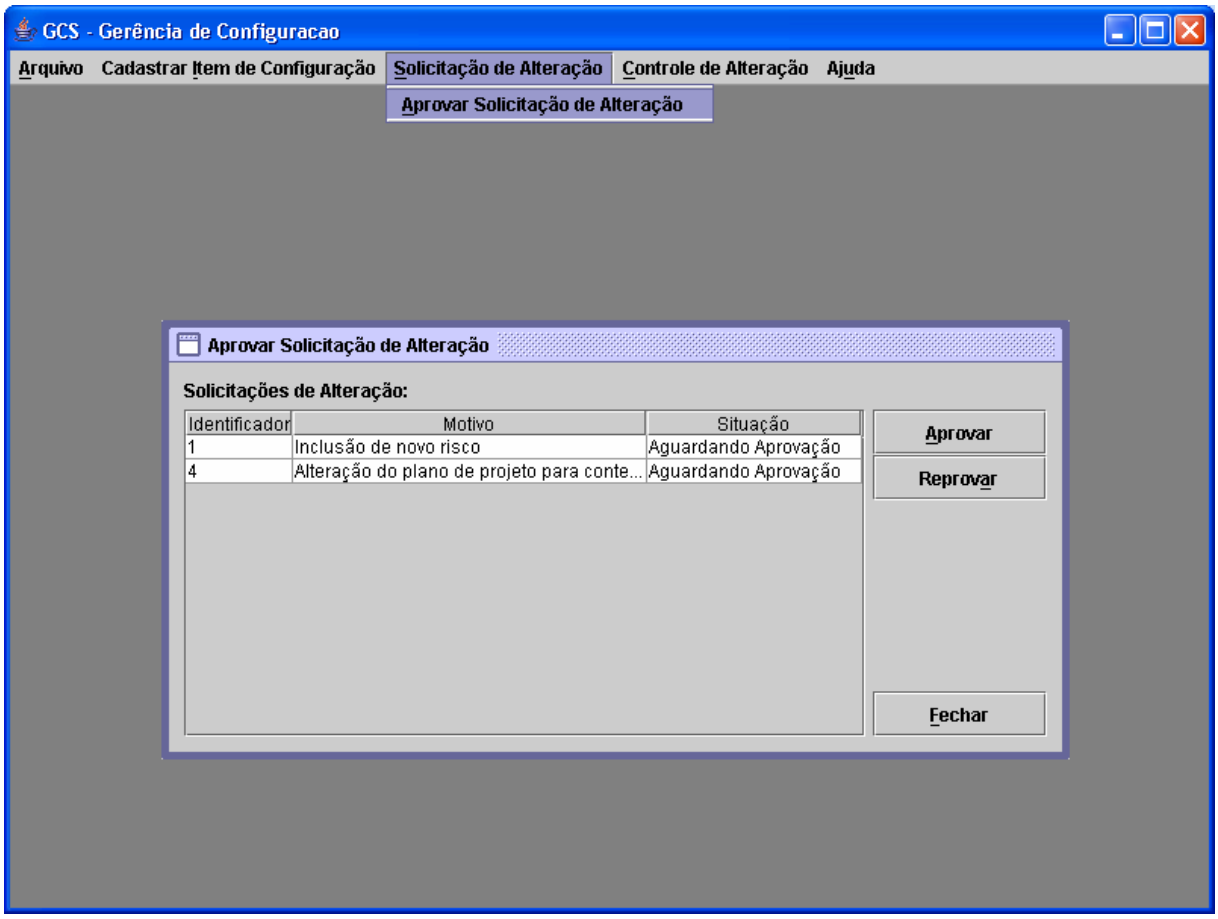


Figura 5.18 – Aprovação de Solicitação de Alteração.

Nota-se que nessa janela há duas funções possíveis: *Aprovar* ou *Reprovar* uma solicitação de alteração.

Ao clicar em *Aprovar,* é exibida a janela mostrada na figura 5.19, apresentando a aba *Dados Gerais*. As informações exibidas são as mesmas da janela de Solicitação de Alteração, com exceção de:

- Autorizador - informa o recurso humano que está autorizando a solicitação de alteração. O valor desse campo é preenchido automaticamente, uma vez que o ambiente tem o controle de qual gerente de configuração o está utilizando no momento, e não pode ser alterado.

- Data de Aprovação – é a data em que a solicitação de alteração está sendo aprovada. Este campo é automaticamente preenchido com a data atual e não pode ser alterado.

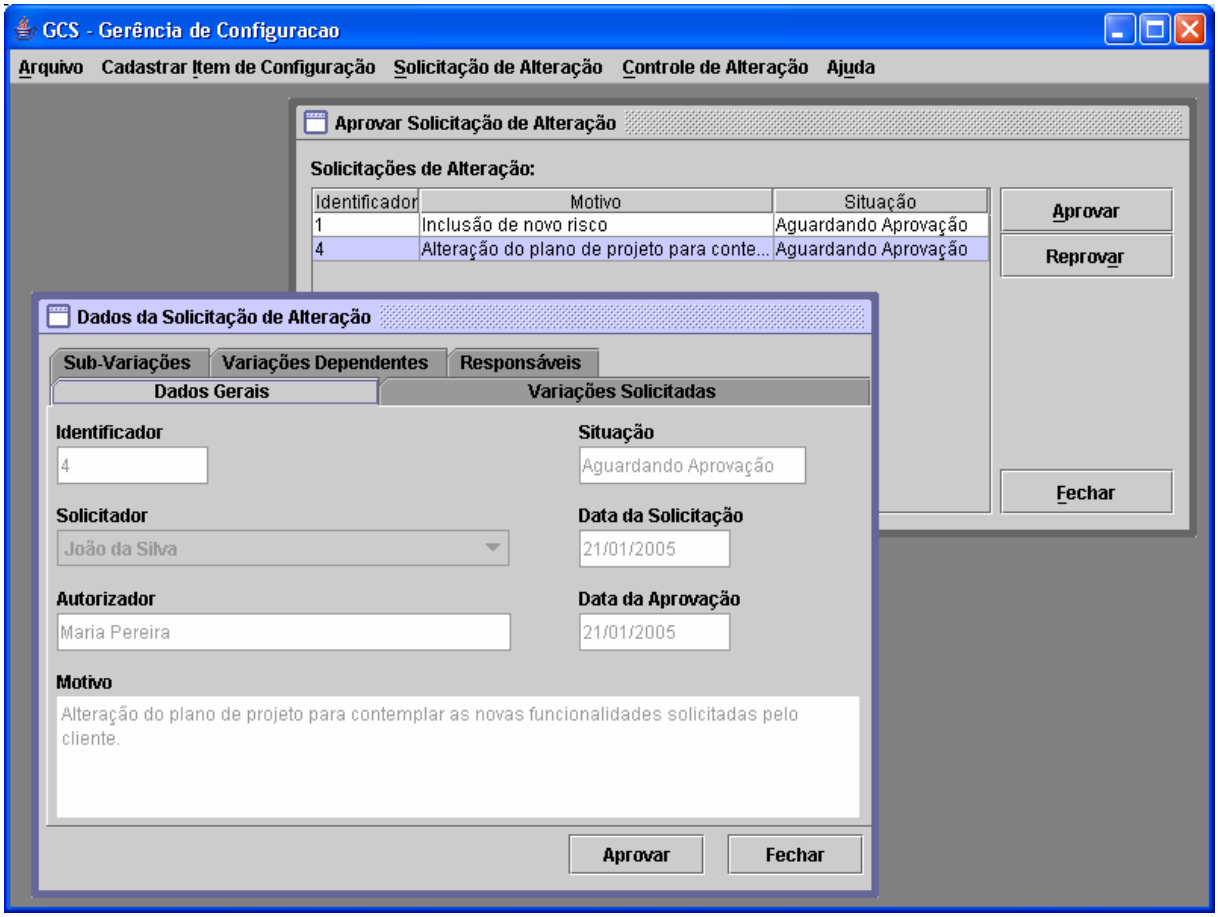


Figura 5.19 – Aprovar Solicitação de Alteração – Dados Gerais.

Antes de aprovar a solicitação de alteração, o Gerente de Configuração deve passar por todas as outras abas. Na aba *Variações Solicitadas*, poderá visualizar as variações que o desenvolvedor deseja alterar. É uma aba apenas de consulta e não poderá ser alterada.

Na aba *Sub-Variações*, são exibidas as sub-variações das variações selecionadas pelo desenvolvedor, para que o Gerente de Configuração avalie a necessidade de alteração também nessas variações.

A aba *Variações Dependentes* tem finalidade semelhante, ou seja, informar ao Gerente de Configuração quais são as variações que dependem das variações que estão sendo solicitadas, para que ele avalie a possibilidade de alteração nessas outras. Vale à pena observar a diferença entre a aba *Dependências* da janela de *Cadastro de Itens de Configuração* e a aba *Variações Dependentes* a que se está referindo aqui. No primeiro caso, trata-se das variações das quais a variação do item de configuração depende. No segundo caso é o contrário, trata-se das variações que dependem das variações selecionadas.

Enfim, é por meio da aba *Responsáveis* que o Gerente de Configuração vai designar os responsáveis pela alteração, que não são necessariamente os mesmos que fizeram a solicitação. Só serão exibidos nessa aba os recursos humanos que possuírem acesso de alteração ou exclusão em todas as variações selecionadas nas abas *Variações*, *Sub-variações* e *Variações Dependentes*.

Após clicar no botão *Aprovar*, a solicitação de alteração passará a ter a situação *Aprovada* e não será mais exibida na lista de solicitações aguardando aprovação, mostrada na figura 5.18.

Por outro lado, o Gerente de Configuração pode não ter achado uma boa alternativa a realização das alterações solicitadas e, portanto, pode desejar reprovar a solicitação. Neste caso, basta clicar no botão *Reprovar* da janela exibida na figura 5.18. A situação da solicitação de alteração passará para *Reprovada*.

Retirar para Alteração (*CheckOut*)

Após uma solicitação de alteração ter sido aprovada, os desenvolvedores designados para efetuar a alteração poderão retirar as variações dos artefatos solicitados para alteração, ou seja, realizar o *checkout*. Isso é feito na janela mostrada na figura 5.20, cujas informações apresentadas são:

- Identificador da Solicitação – exibe as solicitações de alteração aprovadas pelas quais o desenvolvedor logado é um dos responsáveis, para que ele selecione a que será retirada para alteração.
- Dados da Solicitação – acionando esse botão, o usuário poderá acessar a janela com as informações da solicitação de alteração selecionada (figura 5.14), para que ele possa consultar a solicitação de alteração, já que o número da solicitação, muitas vezes, não é sugestivo o bastante.
- Data do *Checkout* – é a data em que a solicitação de alteração está sendo retirada para alteração. Este campo vem automaticamente preenchido com a data atual e não pode ser alterado.

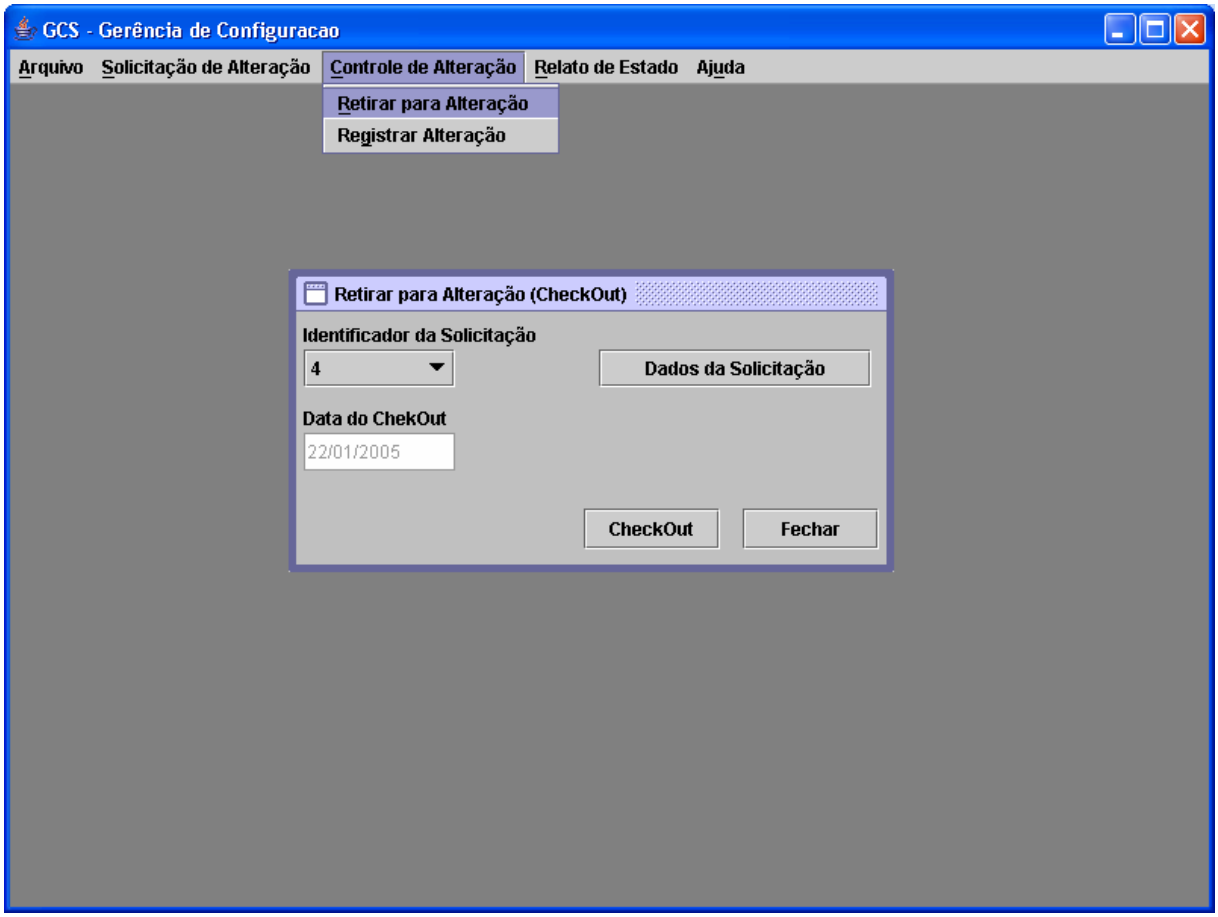


Figura 5.20 – Retirar para Alteração (*Checkout*).

Após clicar no botão *CheckOut*, as variações em questão são bloqueadas, a fim de que não seja possível haver alterações de desenvolvedores diferentes ao mesmo tempo. Se uma certa variação que está sendo bloqueada é parte da composição de uma variação de um outro artefato, essa segunda variação também será bloqueada, porém não será disponibilizada para alteração. Desta forma, as variações selecionadas, com as respectivas hierarquias de super-variações serão bloqueadas. Vale destacar, ainda, que o *checkout* não poderá ser realizado caso alguma variação ou alguma de suas super-variações esteja em alteração.

Feito o *checkout*, sempre que os desenvolvedores responsáveis abrirem o artefato, serão exibidas as informações da base de dados, enquanto para os demais desenvolvedores serão exibidos os respectivos arquivos XML.

Registrar Alteração (*CheckIn*)

Após o desenvolvedor ter concluído suas alterações, ele deve realizar o *checkin* para que os demais desenvolvedores tenham acesso às alterações realizadas e para que as variações envolvidas sejam liberadas para futuras alterações. A figura 5.21 mostra a janela de *checkin*, com as seguintes informações:

- Identificador da Solicitação – exibe as solicitações de alteração em execução pelas quais o desenvolvedor logado é um dos responsáveis, para que ele selecione a que terá suas alterações registradas.
- Dados da Solicitação – da mesma forma como no *checkout*, acionando esse botão, o usuário poderá acessar a janela com as informações da solicitação de alteração selecionada (figura 5.17), para que ele possa consultar a solicitação de alteração, já que o número da solicitação pode não ser suficientemente sugestivo para identificar a solicitação.
- Data do *Checkin* – é a data em que a alteração está sendo registrada. Esse campo é automaticamente preenchido com a data atual e não pode ser alterado.
- Descrição das alterações – neste campo o desenvolvedor deve fazer um relato do que foi alterado.
- Variações Alteradas – apresenta uma tabela com cinco colunas: a primeira serve para o desenvolvedor marcar as variações que foram alteradas; a coluna *Artefato* contém o nome do artefato retirado para alteração; a coluna *Variação Atual* contém a variação do artefato retirado para alteração; a coluna *Nova Variação* é utilizada para o desenvolvedor informar a nova variação do artefato, caso ela tenha sido alterada; e a coluna *Versão/Variante* é para o desenvolvedor informar se a nova variação é uma versão ou uma variante da anterior.

Ao abrir a tela de *checkin*, as colunas *Artefato* e *Variação Atual* vêm automaticamente preenchidas, porém as demais colunas estarão vazias. Neste momento, as colunas *Nova Variação* e *Versão/Variante* estão bloqueadas, ou seja, não podem ser editadas.

No momento em que o desenvolvedor informa que determinada variação foi alterada, ou seja, assim que ele clica na primeira coluna de certa linha, as colunas *Nova Variação* e *Versão/Variante* são desbloqueadas e, automaticamente, são gerados valores padrão para cada uma delas. Para a coluna *Nova Variação*, o valor padrão é o número da variação anterior incrementado de "1" na última posição, como, por exemplo, 1.0.1, se a variação anterior era 1.0.0. Para a coluna *Versão/Variante*, o valor padrão é *Versão*. Apesar disso, o desenvolvedor poderá alterar o valor dessas colunas, caso necessário. Caso o usuário clique em uma linha marcada, os valores dessas duas colunas serão apagados e elas serão novamente bloqueadas.

Caso o desenvolvedor não tenha alterado certa variação ou tenha alterado, mas tenha desistido da alteração realizada, tal variação não deverá ser marcada, para que não seja criada uma nova variação para o referido artefato.

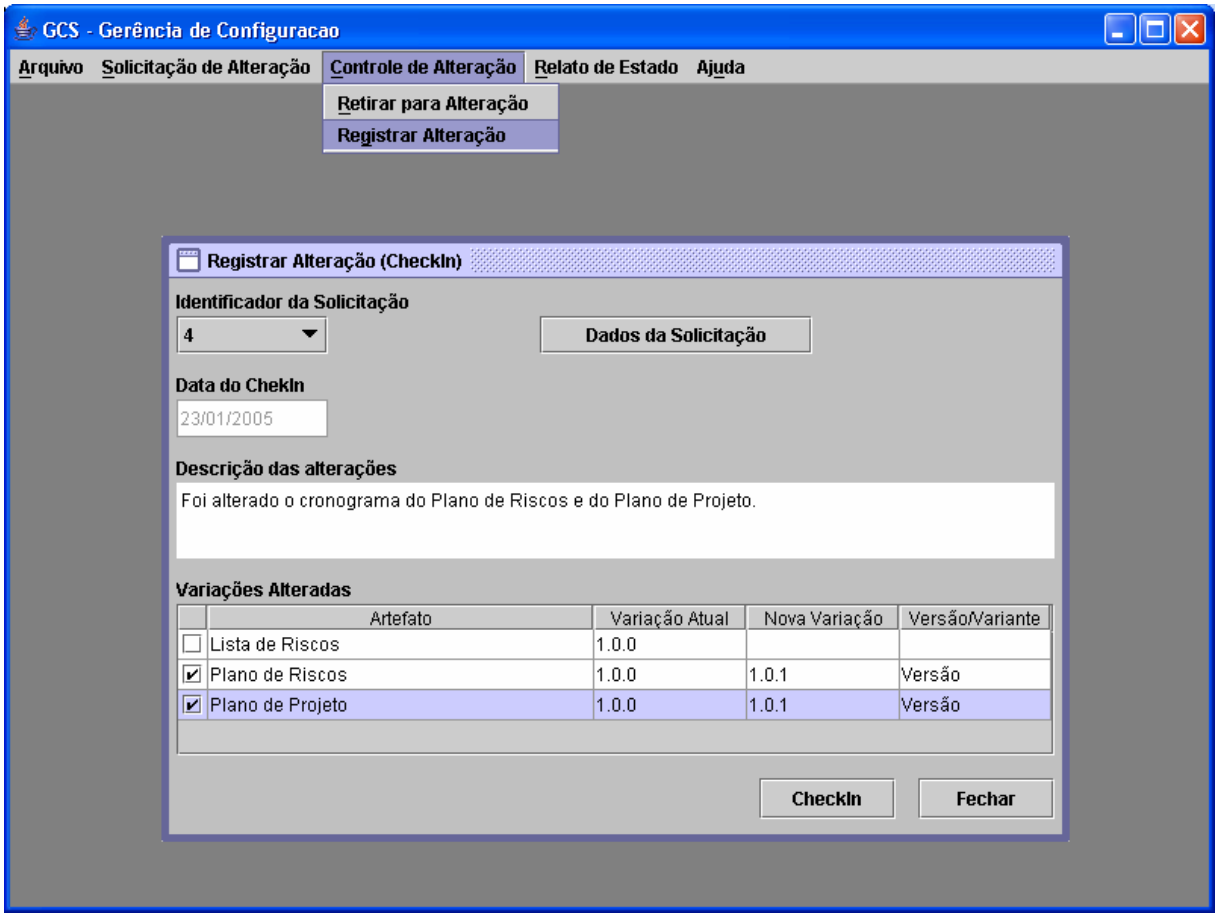


Figura 5.21 – Registrar Alteração (*Checkin*).

Após clicar no botão *CheckIn*, é criada uma nova variação referente a cada linha marcada. Além disso, também são criadas novas variações para toda a hierarquia de super-variações, ou seja, para as super-variações das variações marcadas, para as super-variações de tais super-variações e assim sucessivamente. Todas variações e super-variações são desbloqueadas e são criados arquivos XML correspondentes para cada uma delas.

Caso o desenvolvedor desista da alteração de alguma variação, não a marcando como alterada na tabela da figura 5.21, deverá ser feita uma cópia do arquivo XML atual do referido artefato para a base de dados. Esse procedimento é necessário, pois como a base de dados é única e o desenvolvedor pode ter alterado a variação, ao desistir da alteração é necessário restaurar a variação antiga, que está armazenada na cópia do arquivo XML correspondente.

Porém, diferente das demais telas do sistema, que são fechadas após concluir determinada operação, esta tela continuará aberta após clicar no botão de *Checkin* e será exibida a mensagem da figura 5.22.

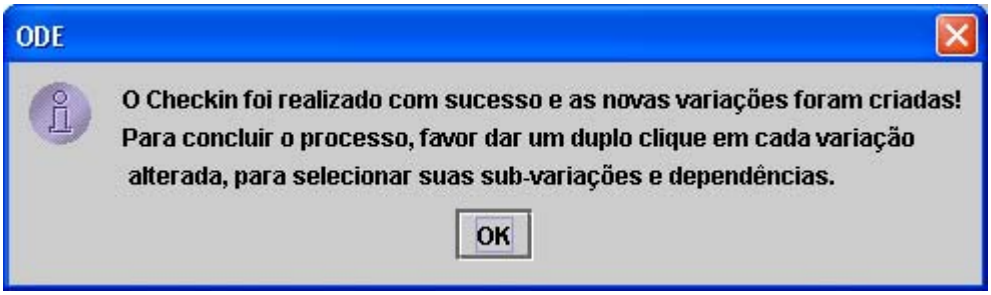


Figura 5.22 – Mensagem de Registro de Alteração bem sucedido.

Como se pode verificar na mensagem acima, o processo de *checkin* foi realizado, porém, para que seja finalizado, o desenvolvedor deverá dar um duplo clique em cada variação alterada para selecionar suas sub-variações e dependências. Todos os campos da tela 5.21 são bloqueados e não poderão mais ser alterados. Além disso, no lugar do botão *Checkin* aparece o botão OK.

Ao dar o duplo clique em uma variação, será exibida a janela da figura 5.23, em que o usuário poderá selecionar as sub-variações da variação selecionada e as variações das quais ela depende, por meio das abas *Sub-Variações* e *Variações Dependentes*, respectivamente. Serão exibidas apenas as variações atuais de cada artefato do projeto. Por default, as sub-variações e as variações dependentes da antiga variação serão exibidas como pré-selecionadas na lista da direita, porém, as variações exibidas serão as mais atuais. Como exemplo, imagine-se a variação 1.0.1 do artefato Plano de Projeto. Na variação antiga, 1.0.0, ele possuía uma sub-variação: 1.0.0 do artefato Plano de Riscos. Assim, na lista da direita será exibida a variação 1.0.1 (mais atual) do artefato Plano de Riscos, como mostrado na figura 5.23.

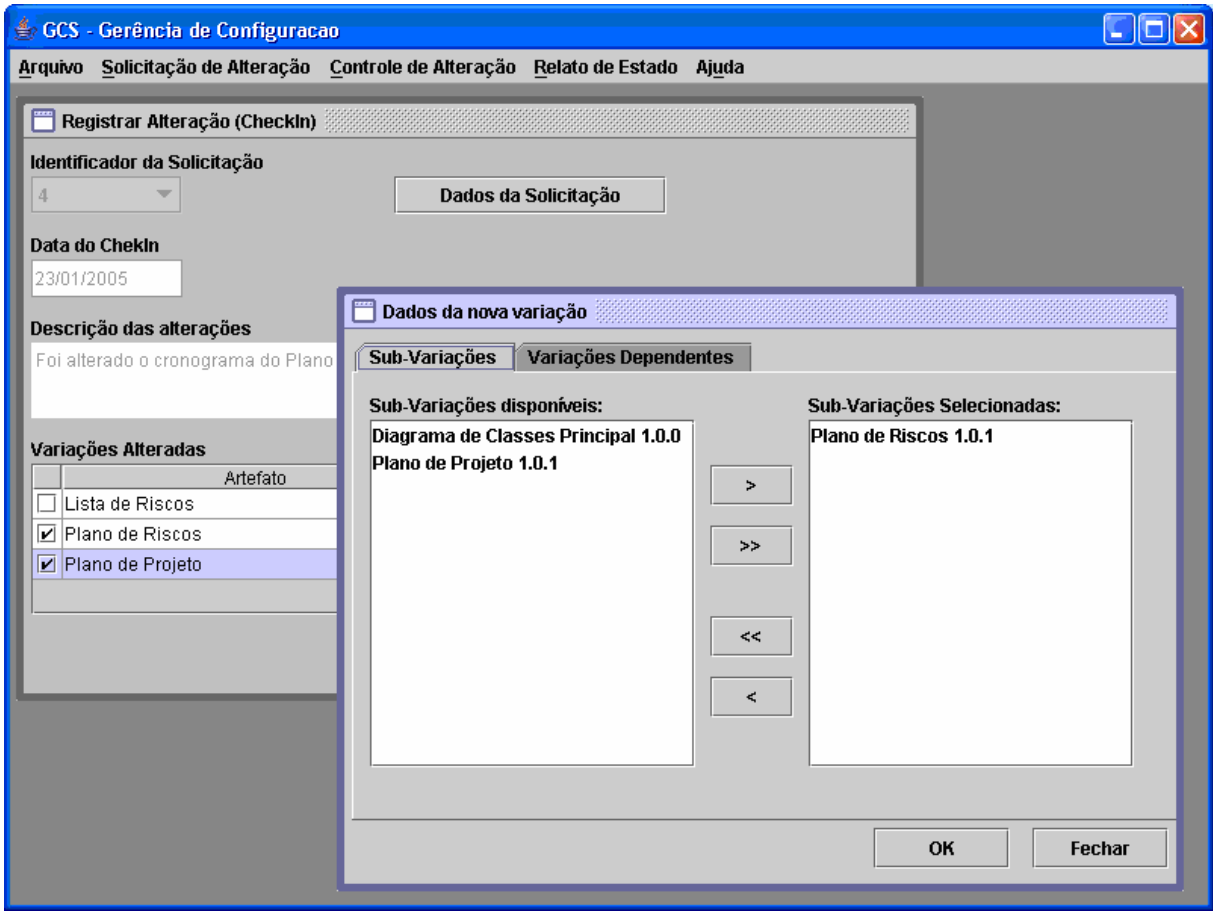


Figura 5.23 – Registrar Alteração (*Checkin*) – Seleção de Sub-Variações e Variações Dependentes.

Após selecionar as sub-variações e as variações dependentes de todas as variações alteradas, o usuário deve clicar no botão OK para o processo ser finalizado.

Apesar desse ser o lugar apropriado para se informar as sub-variações e dependências de cada variação, caso o usuário tenha esquecido de alguma ou tenha optado por informá-las mais tarde, isso poderá ser feito a partir do *Cadastro de Item de Configuração* (Figura 5.13), que exibe sempre as variações atuais dos artefatos.

5.4 Utilização da GCS pelas Ferramentas de ODE

Na seção 4.2 do capítulo 4, foi discutido conceitualmente o que deve ser feito para se integrar as ferramentas de ODE à GCS. De posse de tais informações e da estrutura interna da ferramenta de GCS, definida na seção 5.2, é possível tratar dessa integração com maior riqueza de detalhes.

A maioria das funcionalidades da Gerência de Configuração de Software de ODE é interna à ferramenta de GCS. Porém, algumas delas devem ser disponibilizadas para todas as ferramentas do ambiente. Como a relação entre a GCS e as demais ferramentas de ODE está

centrada nos artefatos, nada mais natural que tais funcionalidades estejam disponíveis na classe *Artefato*, conforme mostrado na figura 5.6. Assim, foram adicionados a essa classe os seguintes métodos:

- *bPubEstahSobGCS(): boolean* – Verifica se um artefato está sob Gerência de Configuração.
- *bPubEhAcessivel(RecursoHumano oParRh): boolean* – Verifica se um artefato está acessível a um recurso humano, ou seja, se: (i) o artefato não está sob GCS ou (ii) está sob GCS, está em alteração e o recurso humano informado é um dos responsáveis pela alteração.
- *sPubObterArqXMLAtual(): String* – Obtém o arquivo XML da versão mais atual do artefato.

Cada ferramenta fica livre para utilizar tais métodos da forma que julgar mais adequada, podendo, por exemplo, bloquear a exibição de um artefato que não esteja acessível ou bloquear a própria ferramenta, conforme discutido no capítulo 4. Genericamente, a utilização das funcionalidades providas pela GCS pode ser resumida segundo o seguinte esquema:

1. Identificar os tipos de artefatos (*KArtefato*) manipulados pela ferramenta, tais como planos de riscos, código fonte, diagramas de casos de uso etc. Para cada tipo de artefato, verificar se existe um artefato desse tipo para o projeto em questão.
    1.1. Caso não exista um artefato do tipo para o projeto, disponibilizar a ferramenta para que se crie um novo artefato.
    1.2. Caso exista um artefato, verificar se ele está sob GCS, usando o método *bPubEstahSobGCS(): boolean*, da classe *Artefato*.
        2.1. Caso o artefato não esteja sob GCS, permitir fazer alterações livremente.
        2.2. Caso o artefato esteja sob GCS, verificar se ele está acessível ao recurso humano (*RecursoHumano*) logado no ambiente, executando o método *bPubEhAcessivel(RecursoHumano oParRh): boolean*.
            3.1. Caso esteja, disponibilizar a ferramenta com suas funcionalidades ao recurso humano, permitindo alterar o artefato.
            3.2. Caso não esteja acessível ao recurso humano:
                4.1. Obter o arquivo XML atual do artefato, através do método *sPubObterArqXMLAtual(): String*, e chamar a ferramenta

XMLDoc, que irá exibir tal arquivo. Além disso, deve-se exibir uma mensagem informando que alterações não poderão ser realizadas, pois o artefato está sob GCS e, caso desejado, deve-se fazer uma solicitação formal de alteração, na ferramenta de GCS.

4.2. Verificar se é necessário bloquear a ferramenta.

4.3. Caso seja necessário bloquear a ferramenta, verificar se é possível criar outro artefato para o mesmo tipo de artefato (*KArtefato*) no projeto. Por exemplo, verificar se é possível criar um outro plano de riscos quando já se tem um criado para aquele projeto.

5.1. Caso seja possível criar um novo artefato, abrir a ferramenta apenas com as funcionalidade de criação de um novo artefato disponíveis.

5.2. Caso não seja possível criar outros artefatos, bloquear a ferramenta.

O esquema acima é apenas para facilitar a utilização dos métodos disponibilizados pela GCS para os desenvolvedores das ferramentas de ODE, podendo ser adaptado de acordo com as características de cada uma delas.

Vale ressaltar, contudo, que a integração com XMLDoc vai um pouco além do exposto acima. Conforme mencionado no capítulo 2, SILVA (2004) isolou a persistência de artefatos em XML, que anteriormente era feita internamente à XMLDoc, e a disponibilizou a todo ambiente, para que todas as ferramentas pudessem gerar seus arquivos XML a partir dos dados dos artefatos na base de dados e vice-versa. Novamente, a classe *Artefato* foi escolhida para disponibilizar essas funcionalidades, por meio dos seguintes métodos:

- *pubSalvarXML(String sParTitulo, String sParFolhaEstilo)* – Grava as informações do artefato em um arquivo XML.
- *pubCarregarXML()* – Obtém informações do artefato a partir de um arquivo XML.

Essas informações incluem todas as associações e os atributos referentes à classe Artefato e às suas super-classes na hierarquia. Dessa forma, futuras subclasses de Artefato só precisam se preocupar com as suas próprias informações, deixando por conta das superclasses as informações delas. Uma forma padronizada de se fazer isso é sobrescrevendo operações

específicas de obtenção – *pubMontarObjeto(Element oParEArtefato)* – e geração – *oPubMontarXML():Element* – de forma que cada subclasse chame a operação sobrescrita da superclasse (SILVA, 2004).

Para salvar um artefato, é necessário, ainda, um método auxiliar – *oPubGerarXML():Element* – que utiliza o método *oPubMontarXML():Element.* Da mesma forma, para o carregamento, necessita-se de um método auxiliar – *pubObterDeXML(Element oParElement)* – que utiliza o método *pubMontarObjeto(Element oParEArtefato).*

Assim, para que uma ferramenta gere seus artefatos em XML e gere artefatos na base de dados a partir de arquivos XML, ela deve implementar os métodos *pubObterDeXML(Element oParElement)* e *oPubGerarXML():Element* para tais artefatos.

A Gerência de Configuração, porém, apesar de lidar diretamente com artefatos, controlando seu processo de alteração, não faz a manipulação dos mesmos, não precisando, assim, de se preocupar em reimplementar métodos, como ocorre com as ferramentas que geram artefatos. A única preocupação é chamar os métodos apropriados, nos momentos adequados.

Assim, a ferramenta de GCS de ODE chama esses métodos disponíveis na classe *Artefato*, nas seguintes situações:

- Ao colocar um artefato sob GCS: o método *oPubMontarXML():Element* é chamado para criar um arquivo XML do artefato. Os arquivos XML são armazenados no diretório **Ode\GerenciaConfiguracao\ArtefatosXML\<nome_projeto>**. Os nomes dos arquivos XML criados seguem a seguinte padronização: <nome do artefato> + "_" + <ido do artefato> + "_" + "v" + <número da variação>. Ex.: Plano de Riscos_ 14.7_v1.0.xml.
- No *Checkin*, para variações solicitadas e não alteradas: busca-se a variação mais atual em XML do artefato em questão e faz-se uma copia para a base de dados, executando o método *pubMontarObjeto(Element oParEArtefato).*
- No *Checkin,* para novas variações criadas: cria-se o arquivo XML correspondente à nova variação do artefato, chamando o método *oPubMontarXML():Element.*

Esses métodos são chamados diretamente na classe *Artefato*, que seguindo sua hierarquia, determina a que classe concreta pertence o objeto artefato em questão e chama o método correspondente dessa classe. A única exigência é que tais métodos estejam implementados nas classes-folha da hierarquia de artefatos, ou seja, caso se esteja colocando um artefato sob GCS, tal artefato deve ter os métodos acima implementados. Assim, por

exemplo, se a ferramenta de GCS chamar tais métodos na classe *Artefato* e ela verificar que a classe em questão é um documento, ela irá chamar os métodos da classe *Documento*. Após verificar que o documento em questão é um Plano de Riscos, chamará os respectivos métodos da classe *PlanoRiscos*, ficando todo esse processo transparente para a GCS.

Além da utilização dos métodos acima, a GCS de ODE ainda se apóia na forma como XMLDoc apresenta os artefatos usando seus arquivos XML. Apesar desse meio ser utilizado pelas ferramentas de ODE e não diretamente pela GCS, pode-se dizer que ela o utiliza indiretamente, dada a segurança que se é conseguida ao se abrir um artefato em XML ao invés de diretamente da base de dados. Essa segurança é necessária, pois garante que o acesso a artefatos não disponíveis seja apenas para leitura, assegurando a consistência dos mesmos. Assim, só terão acesso aos artefatos armazenados na base de dados, os desenvolvedores autorizados, responsáveis por alterações em andamento, com exceção, é claro, dos artefatos que não estiverem sob GCS, que podem ser acessados diretamente na base de dados.

5.4.1 Utilização da GCS na Ferramenta de Gerência de Riscos de ODE

Definidas as características necessárias para a utilização da Gerência de Configuração pelas ferramentas de ODE, tem-se agora o intuito de exemplificar sua utilização por uma das ferramentas de ODE, a ferramenta *GeRis* (FALBO et al., 2004c).

GeRis é uma ferramenta de Gerência de Riscos, que oferece suporte baseado em conhecimento às principais atividades do processo de gerência de riscos, usando a infra-estrutura de Gerência de Conhecimento de ODE. Foi construída com base na ontologia de Gerência de Riscos definida em (FALBO et al., 2004c). O planejamento de riscos em GeRis envolve os seguintes passos(FALBO et al., 2004c):

- Identificação de riscos potenciais para o projeto;
- Avaliação dos riscos, definindo a probabilidade e o impacto da sua ocorrência;
- Definição dos riscos que serão gerenciados e suas prioridades;
- Definição de ações de mitigação e contingência para os riscos de alta prioridade.

GeRis cria automaticamente, para cada projeto de ODE, o artefato Plano de Riscos, que é constituído das informações obtidas dos passos acima. A figura 5.24 mostra a estrutura de um plano de riscos.

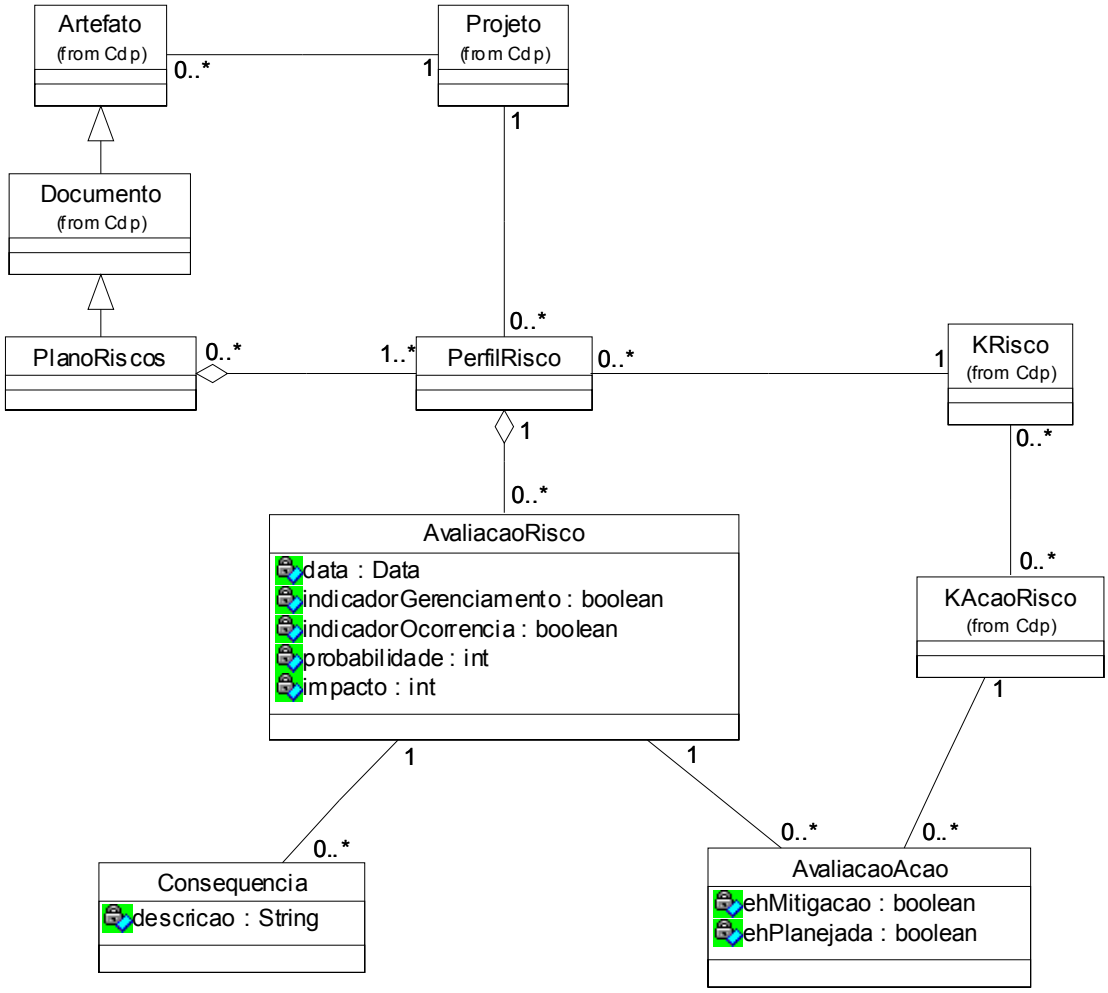


Figura 5.24 – Estrutura de planos de riscos.

Pode-se perceber por essa figura que, sempre que qualquer informação é alterada em GeRis, como por exemplo, a probabilidade de um risco, seu plano de riscos também é alterado. Portanto, controlar alterações nesse artefato significa impedir ou liberar a utilização da ferramenta. Em outras palavras, para usufruir da GCS, optou-se por impedir a utilização de GeRis nos casos em que o artefato não estiver acessível. Assim, caso esse artefato esteja sob GCS e não possa ser alterado em certo momento, a ferramenta como um todo será bloqueada. Caso contrário, a ferramenta é disponibilizada ao usuário.

Para utilizar as funcionalidades que a GCS disponibilizou na classe *Artefato*, GeRis seguiu o seguinte conjunto de passos:

1. Verifica-se se há um plano de riscos para o projeto.
   1.1. Se não houver um plano de riscos para o projeto, disponibiliza-se a ferramenta para que se crie um novo.

1.2. Se houver um plano de riscos para o projeto, verifica-se se o plano de riscos está sob GCS, através do método *bPubEstahSobGCS(): boolean*, da classe *Artefato*.

2.1. Se o plano de riscos não estiver sob GCS, disponibiliza-se GeRis, com todas as suas funcionalidades, permitindo fazer alterações livremente.

2.2. Caso o plano de riscos esteja sob GCS, verifica-se se o plano de riscos está acessível ao recurso humano correntemente logado no ambiente, executando o método *bPubEhAcessivel(RecursoHumano oParRh): boolean*.

2.2.1. Caso esteja, permite-se que o recurso humano utilize livremente GeRis, inclusive alterando o plano de riscos.

2.2.2. Caso não esteja acessível ao recurso humano, obtém-se o arquivo XML atual do artefato, através do método *sPubObterArqXMLAtual(): String*, e chama-se a ferramenta XMLDoc, que irá exibir tal arquivo. É exiba, ainda, uma mensagem informando que alterações não poderão ser realizadas, pois o artefato está sob GCS e, caso desejado, deve-se fazer uma solicitação formal de alteração, na ferramenta de GCS. A ferramenta GeRis é bloqueada.

Além de adaptar a funcionalidade de acesso à ferramenta GeRis, foram implementados os métodos de gravação das informações do artefato Plano de Riscos em arquivos XML e de obtenção das informações do artefato a partir de um arquivo XML, conforme definido em (SILVA, 2004) e brevemente apresentado na seção anterior.

O passo seguinte foi colocar o artefato Plano de Riscos sob Gerência de Configuração. Isso foi feito através da funcionalidade de Cadastro de Itens de Configuração, exibida nas figuras 5.13 e 5.14. A partir deste momento, qualquer usuário de GeRis apenas poderá utilizá-la nos casos previstos pela GCS. Assim, o usuário deverá solicitar a alteração de uma variação do Plano de Riscos, sua solicitação deverá ser aprovada, sendo ele designado um dos responsáveis pela alteração, e deve retirar o artefato para alteração (*checkout*).

Quando um usuário tenta utilizar GeRis em um projeto cujo Plano de Riscos está sob GCS e ele não tem acesso a esse plano para alteração, a utilização de GeRis é impedida. A ferramenta XMLDoc é, então, aberta, para exibir o arquivo XML da variação mais atual do artefato Plano de Riscos.

Nos casos em que o artefato não estiver sob GCS ou estiver acessível, a ferramenta é liberada.

5.5 Integração da GCS com a Gerência de Conhecimento de ODE

Na seção 4.3 do capítulo 4 foi discutido como deve ocorrer a adaptação da Gerência de Conhecimento de ODE a GCS, com relação a itens de conhecimento do tipo artefato. Como citado, a memória organizacional passa a ser composta apenas dos artefatos sob GCS, em sua versão mais atual, sem as alterações que, por ventura, estejam sendo realizadas. Além disso, devido à infra-estrutura de GCS definida, os artefatos são exibidos em formato XML, através de XMLDoc. Em suma, a adaptação se restringiu ao seguinte esquema:

- Verifica-se se um artefato está ou não sob GCS, através do método *bPubEstahSobGCS()*.
    - Caso o artefato esteja sob GCS, busca-se o arquivo XML da variação mais atual do artefato, através do método *sPubObterArqXMLAtual()* e o mesmo é apresentado usando XMLDoc.
    - Caso o artefato não esteja sob GCS, não será considerado pela Gerência de Conhecimento.

Essa adaptação é necessária nas funcionalidades da Gerência de Conhecimento que lidam com recuperação de artefatos, a saber, busca e disseminação do conhecimento.

Na busca de conhecimento, é possível buscar quaisquer tipos de conhecimento armazenados na memória organizacional de ODE. O usuário deve, portanto, definir o tipo de conhecimento que deseja buscar (artefatos, no caso) e fornecer dados que caracterizem os itens de conhecimento que deseja recuperar. No caso de artefatos, esses dados incluem projeto, atividade e tipo do artefato, como mostra a figura 5.25. Com base nesses critérios, uma lista de artefatos é apresentada e o usuário pode selecionar o artefato que deseja visualizar. Após o usuário selecionar um dos artefatos, o mesmo é apresentado usando XMLDoc.

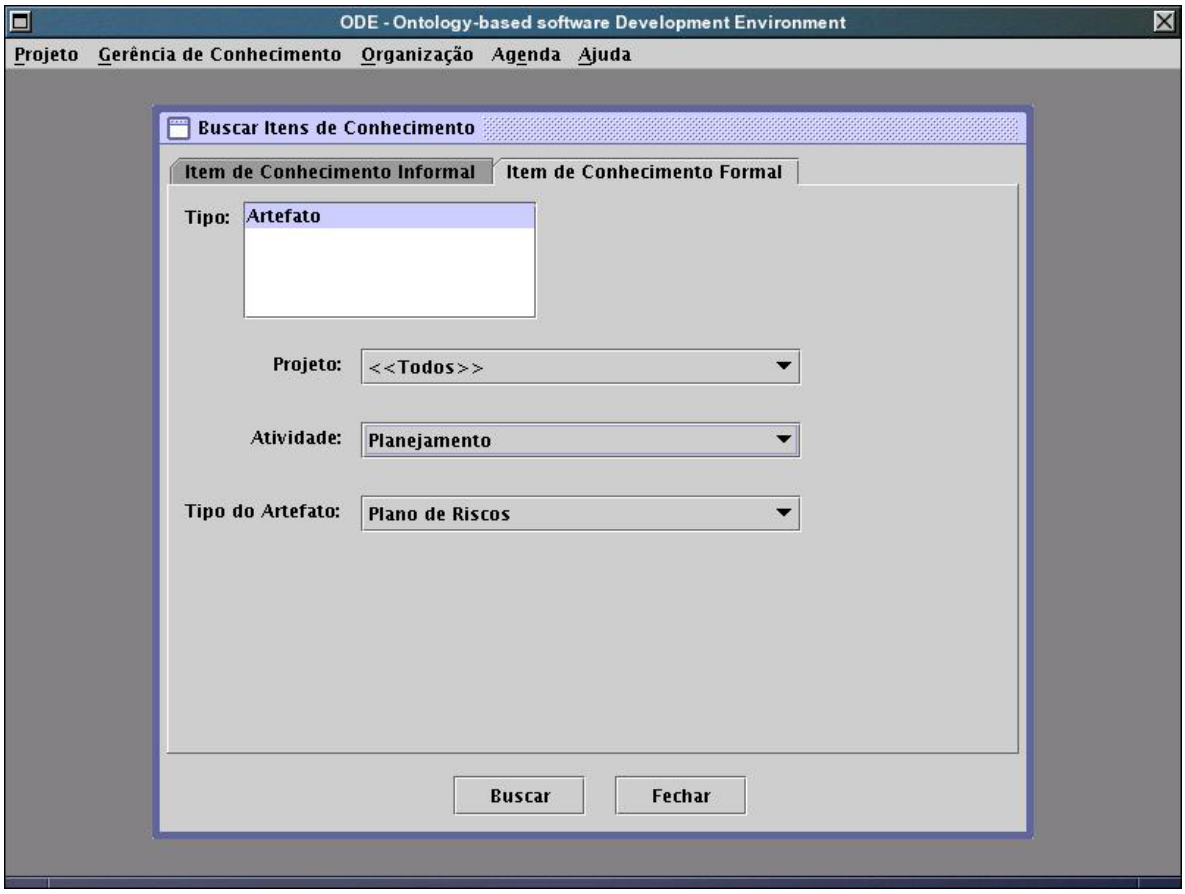


Figura 5.25 – Busca de Itens de Conhecimento na Gerência de Conhecimento.

Na *disseminação de conhecimento*, a Gerência de Conhecimento desempenha um papel ativo, através de agentes de software que monitoram as ações dos usuários, enquanto eles utilizam ODE e suas ferramentas. Da mesma forma que na busca de conhecimento, apenas são utilizados artefatos atuais que estejam sob GCS e a exibição também se dá através de XMLDoc.

5.6 Conclusões do Capítulo

No desenvolvimento de sistemas, as diversas ferramentas do ambiente ODE, assim como de qualquer ADS, geram artefatos constantemente. Porém, para que tais artefatos não sejam alterados sem controle algum, torna-se necessária a existência de uma ferramenta de apoio à Gerência de Configuração de Software.

No caso de ODE, a ferramenta construída foi baseada em um protótipo desenvolvido em (REZENDE, 2001), na ontologia de artefatos definida no capítulo 3 e na infra-estrutura de

GCS proposta no capítulo 4, levando-se em conta, sobretudo, aspectos de integração com outras ferramentas, em especial com XMLDoc e com a Gerência de Conhecimento de ODE.

Tal ferramenta busca fornecer as principais funcionalidades que uma ferramenta de GCS deve prover e sua estrutura interna trabalha, principalmente, sobre dois pacotes de ODE: o pacote Controle e o pacote Gerência de Configuração. O primeiro se refere às classes gerais relacionadas ao controle do processo de software, sendo, portanto, comuns a todas as ferramentas integradas a ODE. O segundo contém classes específicas da GCS.

Porém, apesar de passível de expansão, o sistema de GCS de ODE até o momento contempla apenas a Gerência de Configuração de artefatos. Além disso, apenas as suas funcionalidades essenciais foram implementadas, tratando apenas as variações mais atuais dos artefatos.

Finalmente, foi exemplificada a utilização da GCS por um ferramenta de ODE, a ferramenta de Gerência de Riscos, GeRis (FALBO et al., 2004c) e como a GCS foi integrada à Gerência de Conhecimento de ODE.

Capítulo 6

Conclusões e Perspectivas Futuras

Neste capítulo são apresentadas as considerações finais a respeito do trabalho desenvolvido. A seção 6.1 apresenta as principais conclusões enquanto na seção 6.2 são enfocadas as perspectivas futuras para continuidade deste trabalho.

6.1 Conclusões

Um dos grandes desafios da Engenharia de Software é fazer com que os desenvolvedores possam trabalhar em um ambiente único, com todas as ferramentas necessárias integradas, compartilhando o conhecimento adquirido e cometendo o mínimo possível de erros. Desta forma, ter-se-ia um processo de desenvolvimento e seus produtos gerados com maior grau de qualidade.

A questão de se ter um ambiente único remete aos Ambientes de Desenvolvimento de Software (ADSs), que provêem suporte para o desenvolvimento e a manutenção de produtos de software e para o gerenciamento dessas atividades (LIMA, 2004). ADSs têm sido cada vez mais utilizados por auxiliar o desenvolvimento de software seguindo um processo bem definido, facilitando a transferência de informação entre ferramentas, provendo coordenação entre os recursos humanos e aumentando o controle do projeto, através de melhor planejamento, monitoramento e comunicação (PRESSMAN, 2001).

Entretanto a integração de ferramentas tem sido, ao longo dos anos, um grande obstáculo, uma vez que são disponibilizadas no mercado diversas ferramentas, de diversos fabricantes, porém com conceituações e padrões diferentes, que muitas vezes conflitam entre si.

Para que um ADS tenha suas ferramentas integradas, deve-se estabelecer uma infra-estrutura capaz de acomodar tais ferramentas, permitindo o compartilhamento de informações entre elas, de forma segura. A Gerência de Configuração de Software (GCS) aparece neste contexto com um papel principal: deve ser o centro de um ADS, controlando os artefatos produzidos e compartilhados pelas diversas ferramentas, estabelecendo e mantendo a

integridade dos mesmos e garantindo que sejam apenas alterados pelos recursos humanos autorizados. A GCS envolve, dentre outras coisas, a identificação da configuração de software, ou seja, dos itens que devem ser controlados; o controle de mudanças e versão na configuração e os registros e relatos de estado do processo de alteração (FIORINI et al., 1998).

Porém, além da atuação da GCS, é necessário que os artefatos de um ADS possuam um vocabulário comum para que, assim, possam servir de fonte de comunicação entre as ferramentas. Ontologias merecem bastante destaque neste sentido, pois se referem a um entendimento compartilhado de algum domínio de interesse, podendo ser usadas para resolver problemas como comunicação precária, dificuldade na identificação de requisitos, interoperabilidade, reuso e compartilhamento de informações, dentre outros (USCHOLD et al., 1996).

Com o uso de ontologias, um ADS pode passar a concentrar atenções na gerência do conhecimento criado durante o desenvolvimento software. Conforme apontado por STAAB et al. (2001), ontologias são a "cola" que mantém unidas as atividades da gerência de conhecimento. Assim, abre-se espaço para que o conhecimento possa ser usado para agilizar a execução das atividades do processo de software, mantendo e até aumentando a qualidade dos produtos gerados. A gerência de conhecimento apresenta-se como uma potencial solução, fornecendo mecanismos de disseminação de conhecimento, para que os desenvolvedores realizem suas atividades melhor e mais rapidamente, mostrando que as falhas e as boas práticas encontradas por outros podem auxiliar a tomada de decisão (NATALI, 2003).

Com o acúmulo de conhecimento em uma organização, o foco recai sobre a necessidade de adicionar semântica às informações armazenadas, auxiliando na busca por informações relevantes ao contexto em mãos. O objetivo passa a ser, então, de evoluir um ADS tradicional para um ADS semântico (FALBO et al., 2004a).

O ambiente ODE (*Ontology based software Development Environment*) (FALBO et al., 2003) (FALBO et al., 2004a) é um exemplo ADS que tem buscado se tornar um ADS semântico, tipicamente através da utilização de ontologias, dentre as quais merece destaque a ontologia de processo de software definida em (FALBO, 1998).

Neste processo de contínua evolução de ODE, uma questão crucial tinha de ser tratada: a necessidade de se estabelecer facilidades em ODE para tratar a Gerência de Configuração de Software, incluindo uma efetiva integração das ferramentas que geram e utilizam artefatos. Seguindo a tendência de ODE em direção a um ADS Semântico, era

necessária também a definição de ontologias para tal. Para solucionar tais problemas, o presente trabalho trouxe, então, as seguintes contribuições:

Construção de uma ontologia de artefatos que, além de abrigar aspectos comuns a estes, foi subdividida para tratar informações específicas para os artefatos de código, documentos e diagramas. Além disso, houve uma posterior extensão dessa ontologia para contemplar aspectos da Gerência de Configuração, através de uma ontologia de GCS.

- Análise de ferramentas de GCS existentes e de abordagens de infra-estrutura de GCS, para que pudesse ser criada a infra-estrutura que melhor atendesse às necessidades de ODE. A infra-estrutura definida gerou poucas alterações em ODE, facilitou a integração com as demais ferramentas do ambiente, não teve grande dificuldade técnica e ao mesmo tempo forneceu as funcionalidades necessárias à GCS. Um grande diferencial em relação a outras ferramentas de GCS é o fato de tratar os artefatos como sendo compostos de partes, ao invés de enxergá-los como "caixas pretas".
- Definição de uma abordagem para integração da Gerência de Configuração de Software com a ferramenta de documentação de ODE, XMLDoc (SILVA, 2004), e com a infra-estrutura de Gerência de Conhecimento do ambiente (NATALI, 2003).
- Adaptação da Gerência de Conhecimento de ODE à GCS, com definição da relação existente entre a memória organizacional, o repositório de ODE e o repositório GCS.
- Criação de uma ferramenta de apoio à GCS para o ambiente ODE, tomando por base a ontologia de artefato desenvolvida e a infra-estrutura de Gerência de Configuração e a abordagem de integração definidas.
- Definição de um procedimento padrão para a integração da GCS de ODE às demais ferramentas. Esse procedimento foi adotado na ferramenta de apoio à gerência de riscos de ODE, com o objetivo de servir de base para que trabalhos futuros tenham um exemplo a seguir.

Em suma, a infra-estrutura de Gerência de Configuração de Software proposta neste trabalho facilita a comunicação com as demais ferramentas de ODE, além de tratar os artefatos como peças estruturadas, dotadas de composições e dependências.

Além disso, pelo fato da ferramenta de apoio à GCS ser integrada ao ambiente ODE, várias informações necessárias ao seu funcionamento, ao invés de serem solicitadas ao

usuário, são adquiridas diretamente através da interação com o ambiente. Essa integração possibilitou, ainda, que a ferramenta fosse construída nos mesmos moldes que as demais ferramentas de ODE, utilizando os mesmos padrões de interface, nomenclatura e codificação, os mesmos mecanismos de acesso a dados, a mesma base conceitual etc. Tornou-se, portanto, uma ferramenta amigável, de fácil utilização e de fácil adaptação ou extensão para os desenvolvedores de ODE.

6.2 Perspectivas Futuras

Ainda que a infra-estrutura de GCS de ODE apresente alguns diferenciais em relação às demais ferramentas de GCS disponíveis no mercado (que têm seus artefatos armazenados na forma de arquivos, não enxergam a estrutura interna destes e dificultam a integração com as demais ferramentas de um ambiente), tais ferramentas possuem um grau mais elevado de completeza, por disponibilizarem uma gama maior de funcionalidades que a ferramenta de apoio à GCS proposta neste trabalho, que se ateve somente às funcionalidades básicas. Desta forma, uma primeira melhoria seria no sentido de implementar as funcionalidades especificadas e não-implementadas e criar funcionalidades adicionais que atendam a outros anseios dos recursos humanos envolvidos.

Outra vantagem visível em algumas ferramentas de mercado é o apoio ao desenvolvimento concorrente, por permitirem trabalhos paralelos, ao contrário da ferramenta de GCS de ODE, que faz uso do bloqueio de artefatos. Desta forma, uma possível melhoria na infra-estrutura e na ferramenta propostas inclui adaptações visando o desenvolvimento colaborativo. Deve-se lembrar que a infra-estrutura atual possui apenas uma base de dados e para a GCS atuar de forma colaborativa será necessário criar uma estrutura de forma que vários desenvolvedores possam alterar um mesmo artefato simultaneamente. Uma alternativa é a utilização de bases de dados locais. Neste contexto, a funcionalidade de *checkout* deve ser alterada para permitir retiradas de um mesmo artefato e a de *checkin* deve ser alterada para realizar fusões (*merges*) das alterações.

Além destas, outras limitações foram impostas na ferramenta de GCS de ODE. Apesar de extensível para outros tipos de itens de configuração, atualmente a ferramenta só contempla a gerência de configuração de artefatos. Assim, é interessante que ela evolua para passar a considerar também ferramentas de software e outros itens que possam ser controlados, tais como atividades, processos, métodos, requisitos, dentre outros.

Outra limitação que merece destaque se refere à seleção de variações para alteração. Na sua forma atual, a ferramenta apenas permite fazer solicitações de alteração para as variações atuais dos artefatos, ou seja, não contempla alterações de variações anteriores. Apesar disso, a modelagem e o projeto da ferramenta foram desenvolvidos visando permitir futuras adaptações e extensões, inclusive para tratar variações antigas. Para tal adaptação, deve-se, dentre outras coisas, decidir se variações diferentes de um mesmo artefato podem ser alteradas simultaneamente e como serão estabelecidas as relações de composição e dependência entre as diversas variações.

Outras melhorias passíveis de serem implementadas dizem respeito à automatização de algumas funcionalidades da ferramenta de GCS de ODE. Como exemplo, quando um artefato passa a ter um outro artefato como sub-artefato ou dependência, sua variação atual não é atualizada com tais informações, ou seja, suas sub-variações e dependências não são identificadas automaticamente. Da mesma forma, a ferramenta não consegue discernir quais artefatos retirados para alteração foram efetivamente alterados, ficando a cargo do usuário fornecer tais informações.

Uma característica marcante da ferramenta de apoio à GCS deste trabalho é o fato de ter sido construída para um ambiente específico – o ADS ODE. Apesar de trazer os vários benefícios já citados, essa abordagem leva a uma outra desvantagem em relação às demais ferramentas, que é o fato de não poder ser utilizada em ambientes com características diferentes das de ODE. Entretanto, como a ferramenta foi construída com base na ontologia de artefato, sua estrutura permite a extensão para uma ferramenta de GCS genérica, que, conseqüentemente, pode ser utilizada em qualquer ambiente.

Voltando-se para o contexto de ontologias, apesar da ontologia de documentação ter sido criada com maior grau de aprofundamento e da ontologia de diagrama estar baseada em um sub-conjunto do meta-modelo da UML, ainda há muito o que desenvolver em termos de ontologia de artefatos. A ontologia de artefato de código, por exemplo, encontra-se ainda em um estágio bem inicial, enquanto nem sequer foram criadas ontologias para os demais tipos de artefatos. Tais ontologias serão extremamente necessárias no momento em que ferramentas que gerem artefatos destes tipos forem integradas a ODE. Também terão papel fundamental quando tais ferramentas passarem a utilizar a Gerência de Configuração.

Finalmente, outro trabalho futuro de caráter mais imediato seria adaptar as demais ferramentas de ODE que geram artefatos, tanto à ontologia de artefato quanto aos procedimentos necessários para integração com a GCS. Pode-se sugerir, ainda, a adaptação

das funcionalidades de *busca* e *disseminação* da Gerência de Conhecimento, que pode, agora, passar a considerar a estrutura interna dos artefatos, tomando por base a ontologia de artefato.

Referências Bibliográficas

AHERN, D. M., CLOUSE, A., TURNER, R. *CMMI Distilled.* SEI Series in Software Engineering. Addison Wesley, 2004.

ARBAOUI, S., DERNIAME, J. C., OQUENDO, F., VÉRJUS, H. *A Comparative Review of Process-Centered Software Engineering Environments.* Annals of Software Engineering 14, 311–340, 2002.

BARRETO, A.S. *Uma Ferramenta CASE Integrada para Construção de Modelos Orientados a Objetos.* Projeto Final de Graduação, Ciência da Computação – UFES, Outubro de 2002.

BECHHOFER, S., HARMELEN, F. van, HENDLER, J., HORROCKS, I., MCGUINNESS, D., PATEL-SCHNEIDER, P., STEIN, L.A. *OWL Web Ontology Language Reference.* W3C Recommendation, February, 10th 2004. Disponível em: <http://www.w3.org/TR/owl-ref/>. Acesso em: 13.02.2005.

BERTOLLO, G., RUY, F.B., MIAN, P.G., PEZZIN, J., SCHWAMBACH, M., NATALI, A.C.C., FALBO, R.A. *ODE – Um Ambiente de Desenvolvimento de Software Baseado em Ontologias. Anais do XVI Simpósio Brasileiro de Engenharia de Software - Caderno de Ferramentas.* Gramado, Outubro de 2002.

BOOCH, G., RUMBAUGH, J., JACOBSON, I. *UML, Guia do Usuário*, Editora Campus: 2000.

CAETANO, C. *CVS: Controle de Versões e Desenvolvimento Colaborativo de Software.* Editora Novatec, 2004.

CRANEFIELD, S., PURVIS, M. "UML as an Ontology Modelling Language", *Proceedings of the IJCAI-99, Workshop on Intelligent Information*, 16th International Joint Conference on AI, Stockholm, Sweden, July 1999.

DART, S. *Concepts in Configuration Management Systems.* Proceedings of the 3rd International Workshop on Software Configuration Management. Trondheim, Norway, 1991.

DUARTE, K. C., FALBO, R. A. *Uma Ontologia de Qualidade de Software*. In: Anais do VII Workshop de Qualidade de Software. João Pessoa, Paraíba, Brasil, Outubro de 2000.

ESTUBLIER, J. *Software Configuration Management: A Roadmap.* In *Proc.* of The Future of Software Engineering, ICSE'2000, Ireland, 2000.

FALBO, R. A., TRAVASSOS, G. H. *Ambientes de Desenvolvimento de Sistemas Baseados em Conhecimento,*XXI conferência latino-americana de informática – CLEI'95. Canela, RS, 1995.

FALBO, R. A. *Integração de Conhecimento em um Ambiente de Desenvolvimento de Software*. Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, Dezembro de 1998.

FALBO R.A., MENEZES, C.S., ROCHA, A.R.C. *A Systematic Approach for Building Ontologies*. Proceedings of the 6th Ibero-American Conference on Artificial Intelligence. Lisbon, Portugal, Lecture Notes in Computer Science, vol. 1484, 1998.

FALBO, R. A., NATALI, A. C. C., MIAN, P. G., BERTOLO, G., RUY, F. B. *ODE: Ontology-based software Development Environment*. IX Congreso Argentino de Ciencias de la Computación, p. 1124-1135. La Plata, Argentina, Outubro de 2003.

FALBO, R.A., RUY, F. B., PEZZIN, J., MORO, R. D. *Ontologias e Ambientes de Desenvolvimento de Software Semânticos.* Actas de las IV Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento, JIISIC'2004, Volumen I, pp. 277-292. Madrid, España, Noviembre 2004a.

FALBO, R.A., ARANTES D. O., NATALI, A. C. C. *Integrating Knowledge Management and Groupware in a Software Development Environment.* Proceedings of the 5th International Conference on Practical Aspects of Knowledge Management – PAKM'2004,
Karagiannis, D., Reimer, U. (Eds.): LNAI 3336, pp. 94-105, Springer-Verlag Berlin Heidelberg, Vienna, Austria, December 2004b.

FALBO, R.A., RUY, F. B., BERTOLLO, G., TOGNERI, D. *Learning How to Manage Risks Using Organizational Knowledge.* Advances in Learning Software Organizations (Proceedings of the 6th International Workshop on Learning Software Organizations – LSO'2004), Melnik G. and Holz, H. (Eds.): LNCS 3096, pp. 7-18. Springer-Verlag Berlin Heidelberg, Banff, Canada, June 2004c.

FALBO, R. A. *Experiences in Using a Method for Building Domain Ontologies*. Proceedings of the Sixteenth International Conference on Software Engineering and Knowledge Engineering, SEKE'2004, International Workshop on Ontology In Action, OIA'2004. Banff, Alberta, Canada, June 2004.

FENSEL, D., HORROCKS, I., VAN HARMELEN, F., DECKER, S., ERDMANN, M., KLEIN, M. *OIL in a Nutshell*. In Proc. of EKAW- 2000. LNAI, 2000.

FIORINI, S. T., STAA, A. V., BAPTISTA, R. M. *Engenharia de Software com CMM*. Brasport, Rio de Janeiro, 1998.

FISCHER, G., OSTWALD, J. *Knowledge Management: Problems, Promises, and Challenges***.** IEEE Intelligent Systems*,* v. 16, n. 1, pp. 60-72, January/February 2001.

FUGGETTA, A. *Software Process: A Roadmap.* In *Proc.* of The Future of Software Engineering, ICSE'2000, Ireland, 2000.

GUARINO, N. *Understanding, building and using ontologies.* International Journal Human-Computer Studies, v. 45, n. 2/3 (Feb/Mar) 1997.

GUARINO, N. *Formal Ontology and Information Systems*. In: Proceedings of the First International Conference on Formal Ontology in Information Systems (FOIS'98), Trento, Italy, June 1998.

GRUBER, T.R. *Ontolingua: A mechanism to support portable ontologies.* version 3.0. Technical Report, Knowledge Systems Laboratory, Stanford University, California, 1992.

GRUBER, T.R., "Towards principles for the design of ontologies used for knowledge sharing". International Journal of Human-Computer Studies, v. 43, n. 5/6, 1995.

GRUHN, V. "Process-Centered Software Engineering Environments - A Brief History and Future Challenges". Annals of Software Engineering 14, 363–382, 2002.

HARRISON, W., OSSHER, H., TARR, P. *Software Engineering Tools and Environments: A Roadmap.* In *Proc.* of The Future of Software Engineering, ICSE'2000, Ireland, 2000.

HOLZ, H., KÖNNECKER, A., MAURER, F.*,* "Task-Specific Knowledge Management in a Process-Centred SEE". In: *Advances in Learning Software Organizations*, v. 2176, *Lecture Notes in Computer Science*, Springer, pp. 163-177, 2001.

ISO 9001:2000. *Quality management systems*. Requirements, 2000.

ISO/IEC TR 15504, *Parts 1-9: Information technology – software process assessment*, 1998.

JASPER, R., USCHOLD, M. *A Framework for Understanding and Classifying Ontology Applications.* Proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management – KAW'99, Alberta, Canada, 1999.

LASSILA, O., SWICK, R.R.. *Resource description framework (RDF) model and syntax specification*. Technical report, W3C. W3C Recommendation. http://www.w3.org/TR/REC-rdf-syntax. 1999.

LIMA, K.V.C., *Definição e Construção de Ambientes de Desenvolvimento de Software Orientados a Organização.* Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, Brasil, 2004.

MARTIN, D., BIRBECK, M., KAY, M., LOESGEN, B., PINNOCK, J., LIVINGSTONE, S., STARK, P., WILLIANS, K., ANDERSON, R., MOHR, S., BALILES, D., PEAT, B., OZU, N. *Professional XML.* Editora Ciência Moderna, 2001.

MIAN, P.G., NATALI, A.C.C., CARVALHO, A.L., FALBO, R.A. *Orientação a Domínio em ODE.* Proceedings of the XXVIII Latin-American Conference on Informatics. CLEI'2002, Montevideo, Uruguay, November 2002.

MIAN, P. G. *ODEd: Uma Ferramenta de Apoio ao Desenvolvimento de Ontologias em um Ambiente de Desenvolvimento de Software.* Tese de Mestrado. UFES, Vitória, Brasil, Abril de 2003.

MIAN, P.G., FALBO, R.A. *Supporting Ontology Development with ODEd.* Journal of the Brazilian Computer Science, vol. 9, no. 2, pp 57-76, November 2003.

MONTONI, M., MIRANDA, R., ROCHA, A. R., TRAVASSOS, G. H. *Knowledge Acquisition and Communities of Practice: An Approach to Convert Individual Knowledge into Multi-Organizational Knowledge*. In: Workshop Learning Software Organization, Banff, 2004.

MORO, R. D., NARDI, J. C., FLABO, R. A. *Uma Ferramenta de Acompanhamento de Projetos Integrada a um Ambiente de Desenvolvimento de Software*. Caderno de Ferramentas do SBES, Uberlândia, 2005.

NATALI, A. C. C., FALBO, R. A. *Gerência de Conhecimento em ODE.* Anais do XVII Simpósio Brasileiro de Engenharia de Software - SBES'2003, pp. 270-285. Manaus, Amazonas, Brasil, Outubro de 2003.

NATALI, A. C. C. *Uma infra-estrutura para gerência de conhecimento em um ambiente de desenvolvimento de software.* Tese de Mestrado. UFES, Vitória, ES, Abril de 2003.

NBR ISO/IEC 12207 – *Tecnologia da Informação – Processos de Ciclo de Vida de Software,* 1998.

NGUYEN, T. N., MUNSON, E. V., BOYLAND, J. T. *Object-Oriented, Structural Software Configuration Management.* OOPSLA'04. Vancouver, British Columbia, Canadá, October 2004.

NUNES, V. B., SOARES, A. O., FALBO, R. A. *Apoio à Documentação em um Ambiente de Desenvolvimento de Software.* Memorias de VII Workshop Iberoamericano de Ingeniería de Requisitos y Desarrollo de Ambientes de Software - IDEAS'2004, pp. 50-55. Arequipa, Perú, Maio 2004.

O'LEARY, D. *Enterprise Knowledge Management.* IEEE Computer, v. 31, n. 3, pp. 54-61. March 1998a.

O'LEARY, D. *Using AI in Knowledge Management: Knowledge Bases and Ontologies*. *IEEE* Computer*,* v. 13. n. 3, pp. 34-39, May/June 1998b.

O'LEARY, D.E., STUDER, R. *Knowledge Management: An Interdisciplinary Approach.* IEEE Intelligent Systems, Vol. 16, No. 1. January/February 2001.

OLIVEIRA, K. *Modelo para Construção de Ambientes de Desenvolvimento Orientados a Domínio.* Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, Brasil, Outubro de 1999.

OLIVEIRA, K.M., ZLOT, F., ROCHA, A.R., TRAVASSOS, G.H., GALOTTA, C., MENEZES, C.S. *Domain-oriented software development environment.* The Journal of Systems and Software 72 145 .161, 2004.

OMG – Object Management Group. *Unified Modeling Language Specification*, versão 1.4, Setembro de 2001.

PFLEEGER, S.L. *Software Engineering: Theory and Practice.* 2nd edition, New Jersey: Prentice Hall, 2001.

PRESSMAN, R.S. *Software Engineering: A Practitioner's Approach.* 5th Edition, New York: McGraw-Hill, 2001.

REZENDE, C. F. *Ferramenta de Apoio à Gerência de Configuração de Software*. Projeto de Graduação. UFES, Vitória, ES, Abril de 2001.

REZENDE, S. O. *Sistemas Inteligentes – Fundamentos e Aplicações*. RECOPE-IA (Rede Cooperativa de Pesquisa em Inteligência Artificial), Editora Manole, 2003.

RICHTER, M. M., MAURER, F., *MILOS and MASE: Past & Present*, Technical Report, University of Calgary, 2003. Disponível em: http://ebe.cpsc.ucalgary.ca/ebe/Wiki.jsp?page=Root.Publications

RUY, F.B., *Infra-estruturas de Apoio à Integração de Dados e Conhecimento em ODE* Projeto de Graduação. UFES, Vitória, Brasil, de 2003.

RUS, I., LINDVALL, M., *Knowledge Management in Software Engineering.* IEEE Software, pp. 26 – 38, May/June 2002.

SANCHES R. *Documentação de Software.* In: Qualidade de Software: Teoria e Prática, Eds. A.R.C. Rocha, J.C. Maldonado, K. Weber, Prentice Hall, 2001a.

SANCHES R. *Gerência de Configuração.* In: Qualidade de Software: Teoria e Prática, Eds. A.R.C. Rocha, J.C. Maldonado, K. Weber, Prentice Hall, 2001b.

SARMA, A., NOROOZI, Z., HOEK, A. V. D. *Palantir: Raising Awareness among Configuration Management Workspaces.* Department of Information and Computer Science, University of California, Irvine, USA, IEEE, 2003.

SILVA, P. B. *Adequação da Ferramenta de Documentação de ODE a uma Ontologia de Artefatos.* Projeto de Graduação. UFES, Vitória, Brasil, Dezembro de 2004.

SOARES, A.O. *Ferramenta de Apoio à Documentação*. Projeto de Graduação. Curso Ciência da Computação, UFES, 2002.

STAAB, S., STUDER, R., SCHNURR, H., SURE, Y. *Knowledge Process and Ontologies*. IEEE Intelligent Systems, v.16, n.1, pp. 26-34, January/February 2001.

SWEBOK, *Guide to the Software Engineering Body of Knowledge,* IEEE Computer Society, May 2001.

THOMAS I., NEJMEH, B.A. "Definitions of Tool Integration for Environments", IEEE Software, 29-35, March 1992.

TRAVASSOS, G. H. *O Modelo de Integração de Ferramentas da Estação TABA.* Tese de Doutorado. COPPE/UFRJ, Rio de Janeiro, Brasil, 1994.

USCHOLD, M., GRUNINGER, M. *Ontologies: Principles, Methods and Applications.* Knowledge Engineering Review, Vol. 11, No. 2, June 1996.

VALENTE, A. Legal Knowledge Engineering - A Modelling Approach. IOS Press, 1995.

WAHLI, U., BROWN, J., TEINONEN, M., TRULSSON, L. *Software Configuration Management - A Clear Case for IBM Rational ClearCase and ClearQuest UCM.* International Technical Support Organization, ibm.com/redbooks, December 2004.

Anexo A

Funcionalidades da Gerência de Configuração de Software de ODE

Na seção 5.1 do capítulo 5, foram apresentados os casos de uso necessários para modelar as funcionalidades que a Gerência de Configuração de ODE deve oferecer, tomando-se por base o modelo proposto em (REZENDE, 2001). Tanto as funcionalidades internas à ferramenta de gerência de configuração de software quanto as oferecidas no próprio ambiente ODE foram citadas de maneira sucinta. Neste anexo, porém, é apresentada uma especificação detalhada dos casos de uso do sistema, de modo a servir de base para futuras consultas ou manutenções pelos desenvolvedores de ODE.

O diagrama de casos de uso principal é apresentado na seção A.1. Este diagrama possui seis casos de uso: Cadastrar Item de Configuração, Controlar Solicitação de Alteração, Controlar Alteração, Relatar Estado de Configuração, Abrir Artefato e Salvar Artefato Como.

Na seção A.2 é apresentado o diagrama de casos de uso Controlar Solicitação de Alteração, que é subdividido nos casos de usos Cadastro da Solicitação de Alteração e Aprovação da Solicitação.

Por fim, na seção A.3 é apresentado o diagrama de casos de uso Controlar Alteração, que é subdividido nos casos de usos Retirar para Alteração, Relatar Estado de Configuração, Registrar Alteração, Auditoria de Alteração e Excluir Alteração.

A.1 Diagrama de casos de uso *Principal*

O diagrama de casos de uso principal, mostrado na figura A.1, apresenta as funcionalidades que a Gerência de Configuração de ODE deve fornecer.

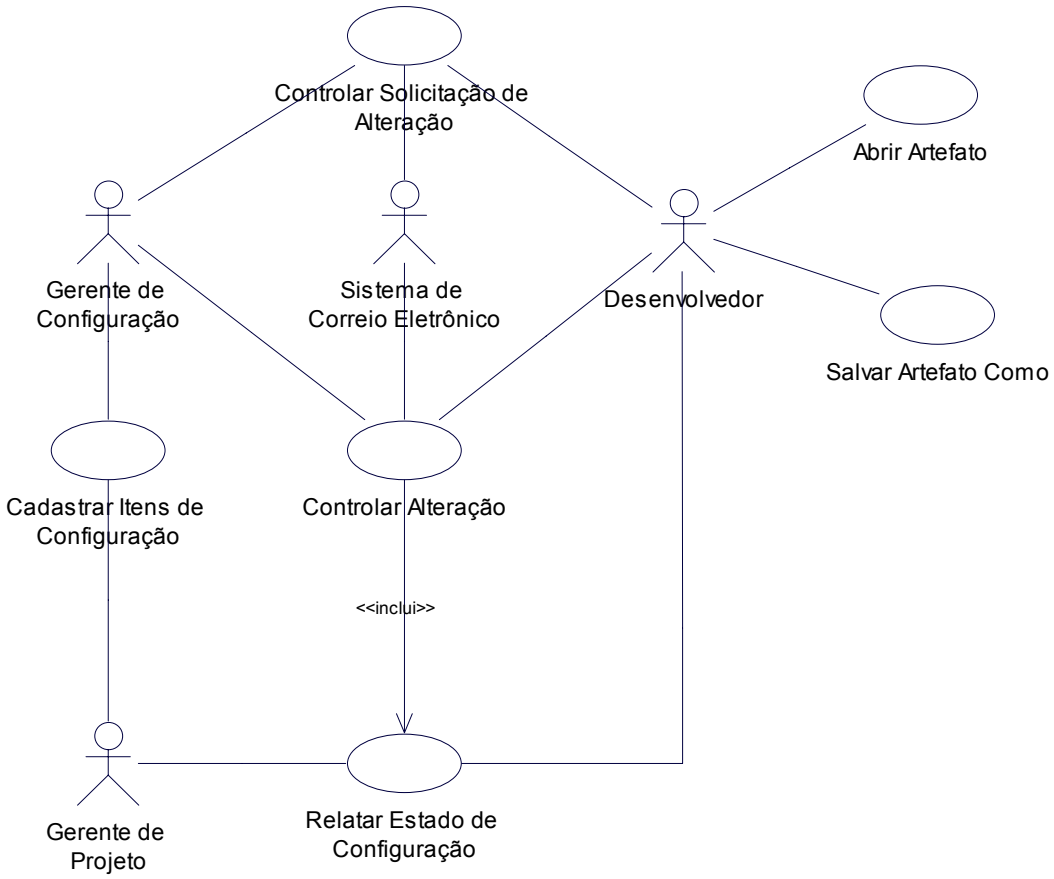


Figura A.1 – Diagrama de caso de uso principal.

Como citado no capítulo 5, os atores envolvidos são o Gerente de Configuração, o Desenvolvedor e o Gerente de Projeto. O Gerente de Configuração é responsável por controlar as alterações sobre os itens que estão sob gerência de configuração, o Desenvolvedor é o responsável pelas alterações em um item de configuração e o Gerente de Projeto acompanha o estado dos itens de configuração dos projetos que gerencia. O ator Sistema de Correio Eletrônico é um sistema externo que está sendo integrado ao ambiente ODE e que possui a funcionalidade de enviar e receber mensagens eletrônicas (*e-mails*) (FALBO et al., 2004b).

Optou-se por este tipo de comunicação entre as pessoas que participam do processo de gerência de configuração, mas outra forma de comunicação pode ser adotada.

Os casos de uso Cadastrar Item de Configuração, Relatar Estado de Configuração, Abrir Artefato, Controlar Artefato e Salvar Artefato Como são descritos a seguir. Como os casos de uso Controlar Solicitação de Alteração e Controlar Alteração possuem diagramas próprios, eles são descritos nas seções posteriores.

A.1.1 Caso de Uso *Cadastrar Itens de Configuração*

Descrição:

Este caso de uso descreve as atividades realizadas pelo Gerente de Configuração ou Gerente de Projeto para controlar as ferramentas e os artefatos de ODE. Através deste caso de uso eles podem criar um novo item de configuração, ou seja, colocar um item (artefato ou ferramenta de software) sob gerência de configuração, alterar os dados de um item de configuração, consultar os dados de um item de configuração e excluir um item de configuração.

Curso Normal:

Incluir Novo Item de Configuração

Se for desejado colocar um artefato sob Gerência de Configuração, deve-se selecionar o artefato e informar a variação, uma descrição, os Recursos Humanos que terão acesso ao novo item de configuração e o tipo de acesso de cada Recurso Humano (Leitura, Alteração ou Exclusão). Caso tal artefato seja composto por outros artefatos, as variações (versões ou variantes) dos artefatos que o compõem são apresentadas, para que sejam selecionadas. As variações dos artefatos das quais o artefato em questão depende também serão apresentadas para serem selecionadas.

Ao se criar um novo item de configuração, é criada uma variação para ele e a data de criação é considerada a data corrente. Caso todas as suas sub-variações estejam desbloqueadas essa variação também será desbloqueada. Um arquivo XML correspondente à nova variação também é criado.

Alterar Item de Configuração

O usuário seleciona o item de configuração que deseja alterar. Pode-se alterar a descrição, os acessos, as sub-variações e as dependências de tal item de configuração.

Consultar Item de Configuração

O usuário informa o item de configuração que deseja consultar. Os dados do item de configuração são apresentados.

Excluir Item de Configuração

O usuário seleciona o item de configuração que deseja excluir. Os dados do item de configuração são apresentados e é solicitada uma confirmação. Se a exclusão for confirmada, o item de configuração é excluído, assim como todas suas variações.

A exclusão de um item de configuração só pode ser feita pelos Gerentes de Configuração ou Gerentes de Projeto que tiverem acesso de Exclusão a tal item de configuração.

Cursos Alternativos:

Incluir Novo Item de Configuração

- Caso o artefato a ser colocado sob Gerência de Configuração possua sub-artefatos ou dependa de artefatos que não estão sob Gerência de Configuração, tais artefatos não serão exibidos, ou seja, não poderão ser selecionados.
- Caso alguma de suas sub-variações esteja bloqueada (em alteração), ela também será bloqueada, uma vez que alterada a parte, altera-se o todo.

Alterar Item de Configuração

- Um item de configuração não poderá ser alterado se ele estiver em alteração ou quando tiver uma solicitação de alteração aprovada.

Excluir Item de Configuração

- Caso um usuário que não seja Gerente de Configuração ou Gerente de Projeto, ou que não tenha acesso de Exclusão a tal item de configuração, tente excluí-lo, a exclusão não será realizada. Além disso, será exibida uma mensagem de que o usuário não tem acesso de exclusão nesse item de configuração.
- Um item de configuração não pode ser excluído nas seguintes situações:
    - Se o usuário não possui acesso de exclusão neste item de configuração.
    - Se uma das suas variações estiver com estado *check-out*;
    - Se uma das suas variações faz parte da composição de outra variação;
    - Se existir uma solicitação aprovada para uma de suas variações.
- Se houver uma solicitação de alteração não aprovada para uma das variações do item de configuração, este poderá ser excluído. Neste caso, uma mensagem é enviada ao

solicitador da alteração para informar que o artefato foi excluído e, portanto, que a solicitação também foi excluída.

Classes: Artefato, ItemConfiguração, Projeto, RecursoHumano, Variação, Versão, Variante, Acesso, Alteração.

Restrições de Integridade:

- Existe apenas uma variação atual para cada Item de Configuração.
- Cada sub-variação atual de uma variação deve corresponder a um sub-artefato do artefato de tal variação.
- Cada dependência atual de uma variação deve corresponder a uma dependência no artefato de tal variação.

A.1.2 Caso de Uso *Relatar Estado de Configuração*

Descrição:

Este caso de uso permite ao Desenvolvedor ou ao Gerente de Projeto verificar o estado de um artefato que esteja sob gerência de configuração, bem como ao Gerente de Projeto verificar o estado de um projeto como um todo.

Curso Normal:

Para obter o relatório de estado de um determinado artefato, o Desenvolvedor ou Gerente de Projeto deve selecionar a variação desejada. Caso seja o desenvolvedor, ele deve ter acesso ao artefato selecionado. Este relatório também é obtido quando o Desenvolvedor realiza o cenário "Retirar para Alteração" do caso de uso "Controlar Alteração".

As seguintes informações são disponibilizadas neste relatório: quem solicitou a alteração, quem autorizou, quem é(são) a(s) pessoa(s) responsável(is) pela alteração, o motivo e a descrição da alteração, as variações solicitadas, as variações alteradas, as variações geradas e as datas de solicitação, aprovação, check-out e check-in. Para as variações compostas por outras variações ou que tenham dependências, sua composição e dependência serão exibidas.

Para obter o relatório de estado de configuração de um projeto, o Gerente de Projeto deve ter autorização em tal projeto. Neste caso, um relatório é gerado, informando o estado e o histórico dos artefatos sob gerência de configuração de tal projeto. As informações

disponibilizadas são: quais são artefatos sob gerência de configuração, o número de alterações sofridas por cada um deles e quais são as variações mais recentes.

Cursos Alternativos:

- Se o usuário for um desenvolvedor que não tenha acesso ao artefato selecionado, o relatório não será exibido. Além disso, será exibida uma mensagem informando tal fato.

Classes: Artefato, ItemConfiguração, Projeto, RecursoHumano, Variação, Versão, Variante, Acesso, Alteração.

A.1.3 Caso de Uso *Abrir Artefato*

Descrição:

Este caso de uso ocorre quando o desenvolvedor abre um artefato em alguma ferramenta do ambiente ODE.

Curso Normal:

Inicialmente, é verificado se o artefato é um item de configuração, ou seja, se está sob gerência de configuração. Caso seja, é verificado se o artefato está disponível ao desenvolvedor, ou seja, se o desenvolvedor possui alguma alteração em execução que utilize uma variação deste artefato. Em caso afirmativo, a variação, inclusive com as últimas alterações realizadas, é exibida ao desenvolvedor, que poderá realizar novas alterações.

Cursos Alternativos:

- Caso o artefato não esteja disponível, é verificado se o desenvolvedor possui acesso de leitura a tal artefato. Caso possua, o artefato mais recente é disponibilizado ao desenvolvedor, porém sem as alterações que por ventura estejam sendo realizadas. Além disso, o artefato é disponibilizado apenas para leitura e não para alteração. Caso o desenvolvedor não possua acesso, é exibida uma mensagem de erro, informando que ele não está autorizado a visualizar este artefato e o artefato não é exibido.
- Caso o artefato não seja um item de configuração, alterações podem ser realizadas livremente. Assim, o artefato é exibido e é permitido ao desenvolvedor realizar novas alterações sobre ele.

Classes: Artefato, ItemConfiguração, Projeto, RecursoHumano, Variação, Versão, Variante, Acesso, Alteração.

A.1.4 Caso de Uso *Salvar Artefato Como*

Descrição:

Devido à contínua necessidade de reutilização, ao abrir um artefato em alguma ferramenta do ambiente ODE, um desenvolvedor pode constatar que tal artefato possui elementos que gostaria de reutilizar. Desta forma, pode optar pela realização deste caso de uso, ou seja, salvar o artefato como outro artefato.

Curso Normal:

O usuário seleciona o artefato que deseja copiar e o salva como outro artefato.

Mesmo que o artefato aberto esteja sob gerência de configuração, o novo artefato não estará, uma vez que, para se tornar um item de configuração, deverá passar pelo processo formal em que o Gerente de Projeto ou Gerente de Configuração realize o caso de uso Cadastrar Item de Configuração.

O desenvolvedor pode, ainda, reutilizar a decomposição e as dependências do artefato anterior ou alterar conforme desejado.

Classes: Artefato, ItemConfiguração, Projeto, Variação, Acesso.

A.2 Diagrama de Casos de Uso *Controlar Solicitação de Alteração*

Este caso de uso permite acompanhar todo o processo de solicitação de alteração de um item de configuração do tipo artefato, que inclui o cadastro da solicitação de alteração pelo Desenvolvedor e a aprovação da solicitação pelo Gerente de Configuração, como mostra a figura A.2.

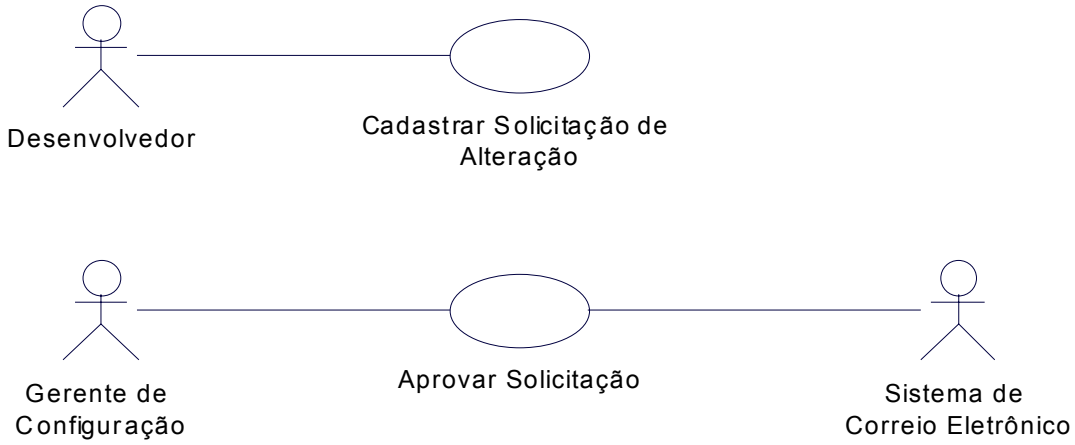


Figura A.2 – Diagrama de Caso de Uso Controlar Solicitação de Alteração.

A.2.1 Caso de Uso Cadastrar Solicitação de Alteração

Descrição:

Este caso de uso é responsável por fazer a solicitação de alteração de variações de artefatos que estejam sob gerência de configuração, uma vez que, antes de se iniciar uma alteração, deve haver uma solicitação de alteração ao Gerente de Configuração. Além de solicitar uma alteração, o desenvolvedor pode, ainda: cancelar uma solicitação de alteração, desde que a alteração não tenha sido processada; alterar uma solicitação, desde que esta não tenha sido aprovada ou processada, excluir uma solicitação, desde que esta não tenha sido aprovada ou consultar uma solicitação.

Curso Normal:

Incluir Solicitação de Alteração

Além das variações que farão parte da solicitação, deve-se informar o motivo da solicitação. Um identificador será automaticamente gerado para tal solicitação e após a conclusão, uma mensagem é enviada ao Gerente de Configuração. A data de solicitação é considerada a data corrente, o solicitador é o usuário que está utilizando a transação e a situação é registrada como “Aguardando Aprovação”.

Excluir Solicitação de Alteração

O usuário seleciona a solicitação de alteração que deseja excluir. Os dados da solicitação são apresentados e é pedida uma confirmação. Se a exclusão for confirmada, a solicitação de alteração é excluída.

Alterar Solicitação de Alteração

O usuário seleciona a solicitação de alteração que deseja alterar. Pode-se alterar as variações que compõem a solicitação e o motivo da alteração.

Consultar Solicitação de Alteração

O usuário informa a solicitação de alteração que deseja consultar. Os dados da solicitação de alteração são apresentados.

Cancelar Solicitação de Alteração

O usuário informa a solicitação de alteração que deseja cancelar. Os dados da solicitação são apresentados e é pedida uma confirmação. Se o cancelamento for confirmado, a solicitação de alteração é cancelada e sua situação passa a ser "Cancelada".

Cursos Alternativos:

Incluir Solicitação de Alteração

- Caso o artefato não possua acesso de alteração ou exclusão em determinado item de configuração, este artefato não será listado, assim não poderá ser solicitado para alteração.

Excluir Solicitação de Alteração

- Caso a solicitação tenha uma situação diferente de "Aguardando Aprovação" ela não poderá ser excluída e sim cancelada, através do cenário "Cancelar Solicitação de Alteração" deste mesmo caso de uso. Uma mensagem é exibida ao usuário informando o ocorrido.

Alterar Solicitação de Alteração

- Caso a solicitação de alteração tenha uma situação diferente de "Aguardando Aprovação" ela não poderá ser alterada.

Classe: Artefato, ItemConfiguração, Projeto, RecursoHumano, Variação, Versão, Variante, Acesso, Alteração, CancelamentoAlteração.

Restrições de Integridade:

- Tanto ferramentas quanto artefatos podem ser colocados sob gerência de configuração, porém, apenas variações de itens de configuração derivados de artefatos podem ser submetidos a alteração

A.2.2 Caso de Uso *Aprovar Solicitação*

Descrição:

Este caso de uso permite ao Gerente de Configuração registrar a aprovação ou reprovação de uma solicitação de alteração feita por um desenvolvedor, cuja situação seja "Aguardando Aprovação".

Curso Normal:

Aprovar Solicitação de Alteração

O aprovador seleciona a solicitação de alteração que deseja analisar. São exibidos todos os dados da solicitação. O aprovador é considerado o usuário que está acessando a transação e a data de aprovação é a data corrente.

Caso o usuário decida por aprovar a solicitação, ele deve informar as pessoas que serão responsáveis pela sua realização. Porém, os recursos humanos designados responsáveis para realização da alteração devem possuir acesso de alteração ou exclusão aos artefatos envolvidos.

Além disso, as sub-variações das variações em questão e as que dependem delas são exibidas ao Gerente de Configuração, para que ele avalie a necessidade de alteração nestas outras variações, podendo assim, incluí-las como itens a serem alterados.

Após a solicitação de alteração ser aprovada, uma mensagem é enviada ao solicitador e aos responsáveis por ela, para que, assim, possam proceder a alteração. A solicitação de alteração passará a ter a situação "Aprovada".

Reprovar Solicitação de Alteração

O aprovador seleciona a solicitação de alteração que deseja reprovar. É solicitada uma confirmação. Se a reprovação for confirmada, a solicitação de alteração é reprovada e não poderá ser retirada para alteração.

O aprovador é considerado o usuário que está acessando a transação e a data de aprovação (neste caso, reprovação) é a data corrente. A solicitação de alteração passará a ter a situação “Reprovada”.

Curso Alternativo:

Aprovar Solicitação de Alteração

- Caso o recurso humano que se deseja colocar como responsável pela alteração não tenha acesso de alteração ou exclusão a tal artefato, tal recurso humano não poderá ser designado como um dos responsáveis. Para que isto ocorra, deve-se realizar primeiro o cenário “Alterar Item de Configuração” do caso de uso “Cadastrar Item de Configuração”, dando acesso de alteração ou exclusão ao referido recurso humano.

Classe: Artefato, ItemConfiguração, Projeto, RecursoHumano, Variação, Versão, Variante, Acesso, Alteração.

A.3 Diagrama de Caso de Uso *Controlar Alteração*

Este diagrama de caso de uso descreve as atividades realizadas pelo Desenvolvedor para que este execute uma alteração sobre um artefato; a atividade de auditoria de alteração, que pode ser realizada por um Gerente de Configuração ou por um Gerente de Projeto e a atividade de exclusão de alteração, que pode ser realizada por um Gerente de Configuração. Os casos de uso são mostrados na figura A.3.

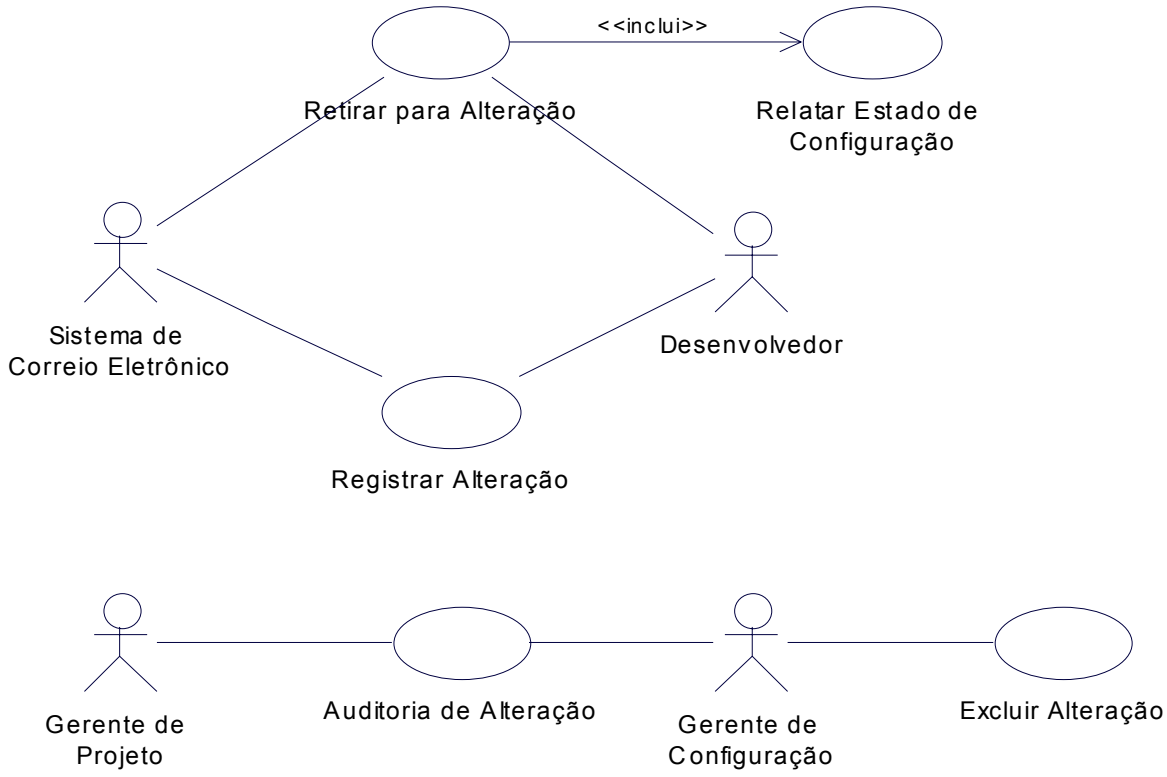


Figura A.3 – Diagrama de Caso de Uso Controlar Alteração.

A.3.1 Caso de Uso *Retirar para Alteração (CheckOut)*

Descrição:

Este caso de uso permite ao Desenvolvedor retirar variações de itens de configuração para alteração (*checkout*), após terem sido solicitadas e aprovadas, ou seja, quando a solicitação de alteração tiver sua situação como "Aprovada".

Curso Normal:

O Desenvolvedor designado como uma das pessoas responsáveis pela alteração seleciona a solicitação de alteração que deseja retirar para alterar.

Caso nenhuma das variações solicitadas, e nenhuma das suas super-variações, esteja em alteração, o *checkout* é realizado. A data do *checkout* é, então, registrada como sendo a data corrente e as variações dos artefatos em questão são bloqueadas, a fim de que não seja possível haver alterações de desenvolvedores diferentes ao mesmo tempo. As super–variações

das variações solicitadas também são bloqueadas, pois está sendo seguida a filosofia de que quando se altera a parte, o todo é automaticamente alterado. Porém, apesar das super-variações ficarem bloqueadas, elas não são disponibilizadas para os desenvolvedores, com exceção das que foram explicitamente solicitadas, seja no processo de solicitação de alteração ou no processo de aprovação de alteração. A situação da solicitação de alteração passa a ser "Em Andamento".

A seguir, o cenário "Relatar Estado de Configuração de um Item de Software" do caso de uso "Relatar Estado de Configuração" é realizado.

Uma mensagem é enviada a todos os desenvolvedores que possuem acesso ao artefato cuja variação será alterada, a fim de informar que essa variação está indisponível, bem como porque será alterada.

Curso Alternativo:

- Caso alguma das variações solicitadas ou alguma das suas super-variações esteja em alteração, o *checkout* não é realizado. Além disso, uma mensagem é exibida informando o ocorrido.

Classe: Artefato, ItemConfiguração, Projeto, RecursoHumano, Variação, Versão, Variante, Acesso, Alteração.

Restrições de Integridade:

- Se uma variação estiver bloqueada, todas suas super-variações estarão bloqueadas.

A.3.2 Caso de Uso *Registrar Alteração (CheckIn)*

Descrição:

Este caso de uso é responsável por registrar as alterações feitas pelo Desenvolvedor (*Checkin*) após a solicitação de alteração ter sido retirada para alteração, ou seja, com a situação "Em andamento".

Curso Normal:

O desenvolvedor registra as alterações realizadas, informando uma descrição do que foi modificado e as variações alteradas (variantes ou versões). Caso as variações sejam

compostas, devem ser informadas as variações que as compõem. Da mesma forma, caso as variações dependam de outras variações, estas devem ser informadas. A data em que o *check-in* foi feito é registrada como sendo a data corrente e um novo número de variação é gerado para cada variação alterada, incrementando o número da variação anterior. Este número, contudo, pode ser alterado pelo Desenvolvedor quando necessário.

As novas variações são criadas, assim como os respectivos arquivos XML, que passam a ser as variações mais atuais dos respectivos artefatos. Também são criadas novas variações para as super-variações das variações alteradas. Todas as variações envolvidas (alteradas ou não) são desbloqueadas, inclusive suas super-variações. A situação da solicitação de alteração é considerada "Finalizada".

Uma mensagem é enviada aos Desenvolvedores que possuem acesso ao artefato cuja variação foi alterada, para informar que tal variação está disponível, bem como a descrição do que foi feito.

Curso Alternativo:

- Caso um determinado artefato não tenha sido alterado, não será criada uma nova variação para ele, a menos que ele seja super-artefato de algum outro artefato alterado. Neste caso, será criada uma nova variação para o artefato, mas seu conteúdo permanecerá o mesmo. É importante destacar que o usuário pode ter realizado alterações no artefato, mas ter desistido de tais alterações. Para que tais alterações não sejam consideradas, é feita uma cópia do arquivo XML mais atual do artefato para a base de dados.

Classe: Artefato, ItemConfiguração, Projeto, RecursoHumano, Variação, Versão, Variante, Acesso, Alteração.

Restrições de Integridade:

- O *check-in* de uma alteração deve sempre ser posterior ao *check-out*.

A.3.3 Caso de Uso *Excluir Alteração*

Descrição:

Este caso de uso permite ao Gerente de Configuração excluir o registro de uma alteração e suas correlações.

Curso Normal:

O Gerente de Configuração informa a alteração que deseja excluir. Após a confirmação de exclusão, as variações criadas nesta alteração e o registro de alteração são excluídos. Porém, as variações que fazem parte da composição de outras variações não podem ser excluídas. As solicitações de alteração associadas a tais variações também são excluídas, a menos que a alteração já esteja em execução.

Curso Alternativo:

Caso a alteração esteja em alteração, nem a solicitação de alteração nem as variações em questão podem ser excluídas, ou seja, a alteração não poder ser excluída.

Classe: Artefato, ItemConfiguração, Projeto, RecursoHumano, Variação, Versão, Variante, Acesso, Alteração.

A.3.4 Caso de Uso *Auditoria Alteração*

Descrição:

Este caso de uso permite ao Gerente de Projeto fazer auditorias das alterações realizadas.

Curso Normal:

O Gerente de Projeto realiza uma avaliação das alterações realizadas, informando a data da auditoria e a conclusão da mesma. Posteriormente, o registro da auditoria poderá ser consultado sempre que necessário.

Classe: Artefato, ItemConfiguração, Projeto, RecursoHumano, Variação, Versão, Variante, Acesso, Alteração, AuditoriaAlteração.

Anexo B

Glossário

Nesta seção são apresentados os principais termos utilizados neste trabalho, em especial os termos referentes às ontologias criadas – Artefato, Documento, Diagrama, Artefato de Código e Gerência de Configuração de Software.

B.1 Termos

Acesso – Relação entre item de configuração e recurso humano, indicando qual tipo de acesso (leitura, alteração, exclusão, etc.) determinado recurso humano tem sobre determinado item de configuração.

Adoção – Relação entre artefato de código e linguagem de programação, indicando em quais linguagens um determinado artefato de código foi escrito.

ADSs (Ambientes de Desenvolvimento de Software) – Coleções de ferramentas integradas que facilitam as atividades da engenharia de software, durante todo o ciclo de vida do processo ou pelo menos em porções significativas dele (HARRISON et al., 2000).

ADSCPs (Ambientes de Desenvolvimento de Software Centrados em Processo) –ADSs que integram ferramentas para apoiar o desenvolvimento do software e para apoiar a modelagem e a execução do processo de software que desenvolve este produto (HARRISON et al., 2000; FUGGETTA, 2000).

ADSODs (Ambientes de Desenvolvimento de Software Orientados a Domínio) –ADSs que além de conter um repositório e um conjunto de serviços e ferramentas, incorporam conhecimento sobre o domínio de aplicação e viabilizam o uso deste conhecimento ao longo do processo de software (OLIVEIRA, 1999).

ADSOrg (Ambientes de Desenvolvimento de Software Orientados a Organização) – ADSs que têm o objetivo de fornecer aos desenvolvedores conhecimento organizacional

necessário à realização das atividades do processo de desenvolvimento e manutenção de software, tais como relativo às diretrizes e melhores práticas organizacionais, às lições aprendidas e aos métodos e técnicas de software (LIMA et al., 2004).

Alteração – Representa uma solicitação de alteração efetuada por um recurso humano, indicando as variações de itens de configuração que poderão ser alteradas. Esta alteração, sendo aprovada, deverá ser executada por recursos humanos, dando origem a novas variações. Para um controle formal, deverão ser executados procedimentos de *check-out* e *check-in*.

Aplicação – Relação entre documento e roteiro, indicando quais roteiros podem ser utilizados na elaboração de um determinado documento.

Aprovação – Relação entre recurso humano e artefato, indicando quais recursos humanos podem aprovar um determinado artefato.

Artefato – Um insumo para, ou um produto de, uma atividade, no sentido de ser um objeto de transformação da atividade. Em função de sua natureza, artefatos podem ser classificados em: documentos, diagramas, artefatos de código, componentes de software ou produtos de software.

Artefato de Código – Porção de código-fonte, passível de execução, gerada no próprio desenvolvimento. Ex: programas, sub-programas, classes, rotinas, funções, etc.

Associação – Define um relacionamento semântico entre dois ou mais classificadores. O conjunto de instâncias de uma associação é o conjunto de tuplas relacionando instâncias dos classificadores envolvidos na associação.

Atividade – Ação que transforma artefatos de entrada (insumos) em artefatos de saída (produtos). Ex.: atividade de especificação de requisitos, atividades de testes, etc.

Autorização – Relação entre recurso humano e alteração, indicando quais recursos humanos são os responsáveis por autorizar uma determinada alteração.

CASE (*Computer Aided Software Engineering*) – Ferramentas de software que provêem a habilidade de automatizar atividades manuais e aumentar a percepção do processo de desenvolvimento (PRESSMAN, 2001).

Check-in – Propriedade de alteração que indica quando as variações submetidas para uma alteração foram devolvidas ao repositório central, estando disponíveis para novas alterações. Contém a data/hora em que as variações tornaram-se disponíveis novamente.

Check-out – Propriedade de alteração que indica quando cópias das variações submetidas para alteração foram retiradas do repositório central e disponibilizadas na área de trabalho do desenvolvedor, para que sejam manipuladas. Contém a data/hora em que as variações foram retiradas do repositório central. A partir deste momento, nenhum outro recurso humano poderá alterar estas variações até que se tenha realizado um procedimento de *check-in*.

Classificador – Elemento generalizável que descreve propriedades. Ex.: classe, interface, tipo de dado, etc.

ClearCase – Ferramenta de GCS da IBM – Rational, que oferece uma família de produtos cuja utilização varia desde pequenos grupos de trabalhos até empresas globalmente distribuídas, possibilitando o desenvolvimento paralelo. Seu processo permite que se inicie a utilização da ferramenta e que este seja adaptado à medida que o projeto cresce (WAHLI et al., 2004).

Componente de Software – Artefato de software que provê um conjunto bem definido de serviços, desenvolvido de forma independente e manipulado (usado, vendido, mantido em uma biblioteca, etc.) como uma unidade coerente, que pode ser combinado para formar artefatos maiores, oferecendo uma interface bem definida e explícita. Ex.: classes, frameworks, padrões gerativos, etc.

Conformidade – Relação entre linguagem de programação e paradigma, indicando quais os paradigmas suportados por uma linguagem de programação.

Correspondência – Relação entre seção e modelo de seção, indicando o modelo de seção que corresponde a uma determinada seção.

CVS (Concurrent *Version System*) – Ferramenta de código aberto (*open source*) e multiplataforma que implementa as principais funções da gerência de configuração de software e é amplamente utilizada por diversos projetos em todo o mundo (CAETANO, 2004).

Dependência entre artefatos – Relação entre artefatos, indicando quais artefatos dependem de um determinado artefato. Assim, pode-se identificar quais artefatos podem ser impactados quando da ocorrência de mudanças em um artefato específico.

Dependência entre Variações – Relação entre variações indicando as variações que são dependentes de uma determinada variação. Assim, é possível identificar todas as variações que podem ser afetadas em uma determinada alteração e não apenas as solicitadas.

Derivação – Relação entre item de configuração e artefato (ou entre item de configuração e ferramenta de software), indicando que um determinado artefato (ou ferramenta de software) está sob gerência de configuração, ou seja, torna-se um item de configuração.

Diagrama – Artefato gráfico, não passível de execução, que consiste em uma apresentação gráfica de um conjunto de elementos de modelo, em geral representada como um gráfico conectado de vértices (elementos de modelo) e arcos (relacionamentos). Ex.: diagrama de casos de uso, diagrama de classes, diagrama de entidades e relacionamentos, diagrama relacional, etc.

Diretriz – Procedimento que visa estabelecer um padrão para a realização de atividades. Diretrizes podem ser divididas em roteiros e normas.

Documento – Artefato de software não passível de execução, constituído tipicamente de declarações textuais, normalmente associado a padrões organizacionais (roteiros) que definem a forma em que eles devem ser produzidos. Ex.: documento de especificação de requisitos, plano de projeto, plano de qualidade, relatório de avaliação da qualidade, etc.

Elemento – Constituinte atômico de um modelo.

Elemento de Modelo – Elemento que é uma abstração em forma de um desenho para facilitar a compreensão humana.

Elemento Generalizável – Elemento de modelo que pode participar em de um relacionamento de generalização.

Estado – Relação entre item de configuração e variação que indica quais as variações de um certo item de configuração.

Execução – Relação entre recurso humano e alteração, indicando quais recursos humanos são responsáveis pela execução de uma determinada alteração, ou seja, quais recursos humanos estão (ou estiveram) de posse de determinadas variações para, possivelmente, realizar alterações.

Extremidade – Relação entre classificador e associação indicando os classificadores que se conectam através de uma associação. A multiplicidade de uma extremidade indica com quantos elementos do classificador a associação pode estar relacionada. O papel de uma extremidade indica a regra de associação com o classificador. Ex.: No contexto de uma locadora de vídeo, extremidade da associação "alugar" com a classe (classificador) "cliente", que possui multiplicidade "0..n" e papel "locatário".

Ferramenta de Software – Recurso de software utilizado para (semi-)automatizar um procedimento adotado na realização de uma atividade. Ex.: Editor de textos, Planilha eletrônica, ferramentas CASE etc.

Filho – Papel da relação entre elemento generalizável e generalização, designando que o elemento generalizável é uma especialização de outro elemento generalizável.

GCS (Gerência de Configuração de Software) – Disciplina responsável pelo controle da evolução dos sistemas de software (ESTUBLIER, 2000; DART, 1991). Deve identificar e documentar os produtos de trabalho que podem ser modificados, através de características físicas e funcionais, estabelecer as relações entre eles e os mecanismos para administrar suas diferentes versões ou variantes, controlar as modificações, além de fazer auditoria e preparar relatórios sobre tais modificações (PRESSMAN, 2001; SWEBOK, 2001).

Generalização – É um relacionamento taxonômico entre um elemento de modelo mais geral e um mais específico. O elemento de modelo mais específico é completamente consistente com o elemento de modelo mais geral (ele possui todas suas propriedades, membros e relacionamentos) e pode conter informações adicionais.

Gerência de Conhecimento – Gerenciamento formal que facilita a criação, o acesso e o reuso do conhecimento, tipicamente usando tecnologia avançada (O'LEARY, 1998b).

Insumo – Relação entre um artefato e uma atividade, indicando que o artefato é utilizado como "matéria-prima" pela atividade, sendo de alguma forma incorporado a outro artefato, o produto da atividade.

Item de Configuração – Item que está sob gerência de configuração e, assim, só pode ser alterado segundo um procedimento de controle de alteração formalmente estabelecido e documentado. Pode possuir várias variações. Pode ser uma ferramenta de software ou um artefato, como, por exemplo, um determinado plano de projeto ou um certo artefato de código.

Linguagem de programação – Linguagem de programação que pode ser utilizada na construção de um artefato de código. Ex.: Java, C, SQL, Script Unix, etc.

Linha Base – Conjunto de itens de configuração em determinadas variações, que serve de base para o desenvolvimento ulterior. Ex.: uma linha base formada pela porção de código X (variação 2.0), pelo documento de plano de projeto Z (variação 1.1.1), pelo diagrama de casos de uso (variação 1.0) etc.

Modelo Documento – Tipo de roteiro que estabelece a estrutura de um documento e instruções para sua elaboração. A estrutura é definida na forma de modelos de seção. Ex.: modelo de documento de plano de projeto, modelo de relatório de avaliação da qualidade, etc.

Modelo Seção – Parte da estrutura de um modelo de documento, que estabelece um padrão para construção de seções desse modelo de documento. Define instruções para a elaboração de uma seção de um documento aderente a ele. Um modelo de seção pode ser opcional, quando não necessitar de uma seção correspondente nos documentos que aplicam o modelo de documento, ou obrigatório, quando há necessidade de uma seção correspondente para ele. Ex.: Modelo de Introdução de um modelo de documento para elaboração de plano de projeto.

Modelo Sub-Seção – Papel da relação de composição entre dois modelos de seção m_1 e m_2. Se m_2 é parte de m_1 então m_2 é dito um modelo-sub-seção de m_1.

Modelo Super-Seção – Papel da relação de composição entre dois modelos de seção m_1 e m_2. Se m_1 é decomposto em outros modelos de seção, dentre eles m_2, então m_1 é dito um modelo-super-seção de m_2.

ODE (*Ontology-based software Development Environment*) – Ambiente de Desenvolvimento de Software desenvolvido no Laboratório de Engenharia de Software da Universidade Federal do Espírito Santo (LabES/UFES), tendo por base ontologias e buscando tratar características de um ADS Semântico (FALBO et al., 2003; FALBO et al., 2004a).

Ontologia – É uma especificação de uma conceituação, isto é, uma descrição de conceitos e relações e suas definições, propriedades e restrições. Tenta resolver problemas como falta de comunicação dentro das organizações, evitando dificuldades na identificação dos requisitos de um sistema. Não descrevem apenas conhecimento imediato, isto é, conhecimento factual que pode ser obtido diretamente a partir da observação do domínio, mas também conhecimento derivado, ou seja, aquele obtido através de inferência sobre o conhecimento imediato disponível (FALBO, 1998).

Ontologias de Aplicação – Ontologias que descrevem conceitos dependentes do domínio e da tarefa particulares. Esses conceitos freqüentemente correspondem a papéis desempenhados por entidades do domínio quando da realização de certa atividade.

Ontologias de Domínio – Ontologias que expressam conceituações de domínios particulares, descrevendo o vocabulário relacionado a certo domínio, tal como Medicina, Direito, Engenharia de Software etc.

Ontologias de Tarefas – Ontologias que expressam conceituações sobre a resolução de problemas, independentemente do domínio em que ocorram, isto é, descrevem o vocabulário relacionado a uma atividade ou tarefa genérica, tal como, diagnose ou vendas.

Ontologias de Representação – Ontologias que explicam as conceituações que fundamentam os formalismos de representação de conhecimento.

Ontologias Genéricas – Ontologias que descrevem conceitos gerais, tais como, espaço, tempo, matéria, objeto, evento, ação, etc., que são independentes de um problema ou domínio particular.

Pai – Papel da relação entre elemento generalizável e generalização, designando que o elemento generalizável é uma generalização de outro elemento generalizável.

Paradigma – Filosofia adotada na construção do software, abrangendo um conjunto de princípios e conceitos que norteiam o desenvolvimento. Ex.: paradigma estruturado, paradigma orientado a objetos, etc (FALBO, 1998).

Procedimento – Conduta bem estabelecida e ordenada para a realização de uma atividade. Quanto à sua natureza, procedimentos podem ser classificados em métodos, técnicas e diretrizes.

Produto – Relação entre um artefato e uma atividade, indicando que o artefato é produzido pela atividade.

Produto de Software – Artefato resultante de um processo de desenvolvimento de software concluído e entregue ao usuário, ou seja, colocado em operação. Inclui o conjunto de todos artefatos gerados no desenvolvimento, além das ferramentas e softwares necessários para colocar o produto em operação. Em um ciclo de vida seqüencial, qualquer alteração necessária em um produto de software se dará em uma atividade de manutenção e não mais em uma atividade de desenvolvimento do software. Em um ciclo de vida iterativo, porém, alterações podem significar a criação de variações de um produto de software.

Propriedade – É uma característica de um classificador.

Propriedade Comportamental – Refere-se a uma propriedade dinâmica de um elemento de modelo. Ex.: operação de uma classe.

Propriedade Estrutural – Refere-se a uma propriedade estática de um elemento de modelo. Ex.: atributo de uma classe.

Recurso Humano – Agente humano necessário para a realização de uma atividade. Ex: engenheiro de software, programador, especialista de domínio, etc.

Relacionamento – É uma conexão semântica entre elementos de modelo.

Resultado – Relação entre variação e alteração, indicando quais variações foram produzidas em uma determinada alteração, ou seja, quais as variações resultantes de uma determinada alteração.

Roteiro – Diretriz para a elaboração de documentos. Ex.: roteiro de plano de projeto.

SABiO (*Systematic Approach for Building Ontologies*) – método para a construção da ontologias (FALBO, 2004) (FALBO, 1998).

Seção – Parte da estrutura de um documento, constituída tipicamente de declarações textuais, de outras seções ou de diagramas. Ex.: seção de introdução, seção de referências bibliográficas, etc.

Sistema de Gerência de Configuração – Sistema para apoiar todas as atividades do processo de GCS ou, pelo menos, prover alguma forma de controle de versão, identificação da

configuração e ter o intuito de prover gerência de configuração (em algum grau) (DART, 1991).

Sistema de Gerência de Conhecimento – Sistema de suporte à gerência do conhecimento em uma organização, o que implica uma infra-estrutura formada por uma dimensão técnica e outra organizacional. É uma solução híbrida que envolve pessoas e tecnologia e propõe a memória organizacional como o núcleo da dimensão técnica e, ao redor desta memória, os vários serviços para apoiar as atividades de gerência de conhecimento (LIMA, 2004).

Solicitação – Relação entre recurso humano e alteração, indicando quais recursos humanos fizeram a solicitação de uma determinada alteração.

SQA (*Software Quality Assurance* ou Garantia da Qualidade de Software) – Disciplina cujo objetivo principal é monitorar o software e seu processo de desenvolvimento, assegurando a utilização de padrões e procedimentos (SWEBOK, 2001).

Submissão – Relação entre variação e alteração, indicando quais variações foram submetidas para modificação em uma determinada alteração.

Sub-Artefato – Papel da relação de agregação entre dois artefatos s_1 e s_2. Se s_2 é parte de s_1, então s_2 é dito um sub-artefato de s_1.

Sub-Variação – Papel da relação de agregação entre duas **variações** v_1 e v_2. Se v_2 é parte de v_1, então v_2 é dita uma sub-variação de v_1

Super-Artefato – Papel da relação de agregação entre dois artefatos s_1 e s_2. Se s_1 é decomposto em outros artefatos, dentre eles s_2, então s_1 é dito um super-artefato de s_2.

Super-Variação – Papel da relação de agregação entre duas variações v_1 e v_2. Se v_1 é decomposta em outras variações, dentre eles v_2, então v_1 é dita um super-variação de v_2.

Sub-modelo-documento – Papel da relação de composição entre dois modelos de documento d_1 e d_2. Se d_2 é parte de d_1, então d_2 é dito um sub-modelo-documento de d_1.

Sub-seção – Papel da relação de composição entre duas seções s_1 e s_2. Se s_2 é parte de s_1, então s_2 é dita uma sub-seção de s_1.

Super-modelo-documento – Papel da relação de composição entre dois modelos de documento d_1 e d_2. Se d_1 é decomposto em outros modelos de documento, dentre eles d_2, então d_1 é dito um super-modelos-documento de d_2.

Super-seção – Papel da relação de composição entre duas seções s_1 e s_2. Se s_1 é decomposta em outras seções, dentre eles s_2, então s_1 é dita uma super-seção de s_2.

Tipo – Papel da relação entre classificador e propriedade estrutural que designa o classificador de quem instâncias são valores das propriedades estruturais.

Variação – Indica qual a versão ou variante de um determinado item de configuração. É caracterizada por um número único para cada variação de um mesmo item de configuração.

Variante – Estado em que um item de configuração existe simultaneamente em duas ou mais formas diferentes que atendam a requisitos similares. Ex.: Um documento estava na versão 1.0 e após alterações passou para a versão 1.1. Porém, foi criada a variante 2.0 que, apesar de atender aos mesmos requisitos que a versão 1.1, foi construída de forma diferente.

Versão – Estado em que se encontra um item de configuração. Cada alteração realizada em um item de configuração gera uma nova versão deste item. Ex.: Um documento que estava na versão 1.0 e após algumas alterações passou para versão 1.1.

XML (*eXtensible Markup Language* - Linguagem de Marcação Extensível) –linguagem poderosa e elegante para se pensar em dados, trocá-los e apresentá-los de forma independente de plataforma. Permite representar metadados, isto é, informações sobre um conjunto de dados (MARTIN et al., 2001).

XMLDoc – Ferramenta de apoio à documentação de ODE, baseada na tecnologia XML (SOARES, 2002; SILVA, 2004).