UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
DEPARTAMENTO DE INFORMÁTICA

Aline Freitas Martins

APOIO À ENGENHARIA DE REQUISITOS EM ODE

Vitória
2007

Aline Freitas Martins

APOIO À ENGENHARIA DE REQUISITOS EM ODE

Monografia apresentada ao Departamento de Informática da Universidade Federal do Espírito Santo, como parte do requisito para obtenção do Grau de Bacharel em Ciência da Computação.

Orientador: Profº. Ricardo de Almeida Falbo

Vitória

2007

Aline Freitas Martins

APOIO À ENGENHARIA DE REQUISITOS EM ODE

Aprovada em 08 de Fevereiro de 2007.

COMISSÃO EXAMINADORA

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

Prof. Davidson Cury, D.Sc.

Prof. Giancarlo Guizzardi, PhD.

Agradecimentos

Agradeço a Deus, pela vida, pela saúde... por tudo. Obrigada também, por colocar em meu caminho pessoas maravilhosas que também ajudaram nesta vitória.

Agradeço a meus pais, Lindalva e Vanderley, por me darem essa oportunidade e por me apoiarem em todos os momentos. Obrigada pelo amor, carinho e atenção. Vocês foram o exemplo que segui para chegar até aqui.

À minha irmã, Cecília, companheira e amiga, que sempre estava do meu lado e me ajudou muito. Obrigada.

Agradeço ao meu namorado, Felipe, pelo amor, compreensão e pela força nas horas de desânimo.

Obrigada, Falbo, que foi mais do que um orientador, foi para mim um amigo e conselheiro. Obrigada pela orientação no trabalho e na vida.

Agradeço à galera animada do LabES, que me ajudou, me ensinou e me fez rir muito durante este trabalho. Obrigada Júlio, Rodrigo, Lucas, Silvano, Vítor, Bruno, Thiago, Alexandre e os demais.

Aos grandes amigos que surgiram durante o curso, como Veruska, Evelin, Viviane, Nuno, Felhberg... a todos agradeço pela companhia e pela ajuda.

Agradeço também aos mestres que me passaram o conhecimento que levarei comigo durante toda a minha vida profissional.

Aos amigos distantes que guardo no meu coração e que me deram estrutura pra chegar até aqui, Ana Paula e Fabrício. Saudades.

Às meninas da República, que foram amigas em horas importantes e me fizeram rir nos dias mais difíceis. Obrigada Marília, Rachel, Lívia e Marina.

Agradeço também a todos os outros que de forma direta ou indireta contribuíram para que eu chegasse até aqui. Com certeza sem o carinho, atenção, paciência, trabalho e apoio de todos eu nada teria conseguido.

Resumo

Requisitos são serviços ou funções que um sistema deverá prover, bem como restrições e propriedades importantes. A Engenharia de Requisitos (ER) é o processo que envolve as atividades relativas ao tratamento desses requisitos, incluindo levantamento, modelagem, negociação, documentação, validação e gerência de requisitos. Esse processo envolve diversas informações a respeito de um requisito e de seus relacionamentos com outros elementos do processo de software. A ER é um processo complexo, que envolve grande quantidade de informações, e é de grande importância, pois garante que o sistema realmente atenderá às necessidades do usuário.

ReqODE é a ferramenta de apoio à Engenharia de Requisitos de ODE, desenvolvida neste trabalho. Ela provê o cadastro e controle de requisitos durante o desenvolvimento de software. Com foco na rastreabilidade, ReqODE possibilita gerenciar relacionamentos entre requisitos, como dependência, conflitos e composição, e de requisitos com outros elementos do desenvolvimento, tais como casos de uso, classes, módulos do projeto, artefatos, recursos humanos e contexto de criação. Por meio de mecanismos de busca e geração de documentos, é possível rastrear requisitos bidirecionalmente, para que, em casos de mudanças, seja possível identificar elementos envolvidos e avaliar o impacto da alteração. Isso possibilita manter consistência entre requisitos e os demais elementos do desenvolvimento e evita ambigüidades.

Palavras-Chaves: Engenharia de Requisitos, Gerência de Requisitos, Rastreabilidade.

Lista de Figuras

Sumário

CAPÍTULO 1INTRODUÇÃO

O processo de desenvolvimento de software tem início pela identificação dos requisitos, ou seja, os serviços e funções que o sistema deverá prover, bem como suas restrições e propriedades importantes. A medida de sucesso de um software está fortemente relacionada ao quanto ele atendeu a esses requisitos iniciais. Para conseguir esse sucesso, é importante não só um bom levantamento dos requisitos, mas também acompanhar as atividades relacionadas a estes durante todo o ciclo de desenvolvimento do software.

O processo de Engenharia de Requisitos (ER) envolve as atividades relativas ao tratamento dos requisitos de um sistema, dentre elas: levantamento, modelagem, negociação, documentação, validação e gerência de requisitos. A ER é fundamental, pois possibilita estimativas de custo e tempo mais precisas e permite que mudanças em requisitos levantados inicialmente sejam melhor gerenciadas. Dentre os problemas de um processo de ER ineficiente, podem-se citar (KOTONYA; SOMMERVILLE, 1998): (i) requisitos inconsistentes, (ii) produto final com custo maior do que o esperado, (iii) software instável e com altos custos de manutenção e (iv) clientes insatisfeitos.

Por ser um processo complexo e que envolve uma grande quantidade de informações, é importante prover ferramentas automatizadas para apoiar a ER. Além disso, para melhor gerenciar os requisitos de um projeto, é importante que uma ferramenta de apoio à ER esteja integrada às demais ferramentas utilizadas no desenvolvimento de software. Assim, será possível manter as ligações necessárias entre os requisitos e outros elementos envolvidos no processo de desenvolvimento. Uma forma de se conseguir isso é por meio de sua integração a um Ambiente de Desenvolvimento de Software (ADS). Este trabalho segue essa idéia e apresenta ReqODE, uma ferramenta de apoio à ER integrada ao ambiente de desenvolvimento ODE (*Ontology-based Development Environment*) (FALBO et al., 2003).

1.1CONTEXTO E OBJETIVOS

Uma vez capturados, os requisitos de software devem ser modelados, documentados, validados e acompanhados. Nesse processo, as propriedades de um requisito (tais como sentença, descrição, data de criação, prioridade etc.) e os relacionamentos com outros elementos do processo de software são definidos e alterados. Neste contexto, torna-se fundamental a rastreabilidade, isto é, a habilidade de se acompanhar a vida de um requisito em ambas as direções do processo de software e durante todo o ciclo de vida.

A rastreabilidade de requisitos só é possível se houver ligações explícitas entre requisitos e outros elementos do processo de software. Dessa forma, a identificação da composição de requisitos, das dependências entre requisitos, de requisitos conflitantes, da origem dos requisitos e de seus interessados, além da identificação de em qual artefato (documento, diagrama etc.) produzido durante o processo de software um requisito é tratado, é de fundamental importância para que a rastreabilidade seja implementada (KOTONYA; SOMMERVILLE, 1998).

Diante das necessidades citadas, foram estabelecidas as funcionalidades de ReqODE, sendo o objetivo principal deste trabalho aprimorar a rastreabilidade, para apoiar de forma mais eficiente a gerência dos requisitos. Assim, com ReqODE é possível registrar e classificar os requisitos, manter as ligações com outros elementos do processo de desenvolvimento, gerar relatórios de rastreabilidade e realizar buscas que viabilizem uma rastreabilidade bidirecional. A saber, é possível relacionar em ReqODE um requisito:

- a outros requisitos, registrando relações de dependência, conflitos e composição;
- a interessados e a seus responsáveis, recursos humanos cadastrados no ambiente por meio da ferramenta de Gerência de Recursos Humanos de ODE (COELHO, 2007) ao projeto em questão;
- a casos de uso e classes, registrados em ODE por meio da ferramenta de modelagem de UML de ODE, OODE (SILVA, 2003), ou pelo cadastro de elementos de modelo desenvolvido também neste trabalho;
- a um contexto de criação, que mantém informação das atividades, artefatos e recursos humanos envolvidos na criação do requisito;
- a módulos do projeto ao qual ele pertence; e

- a artefatos internos, produzidos pelas demais ferramentas de ODE, ou externos a ODE, cadastrados no ambiente por meio da ferramenta de apoio ao planejamento da documentação (SEGRINI, 2007).

Com essas ligações definidas, se alguma mudança for necessária em um requisito, é possível identificar os elementos relacionados para verificar o impacto da mudança e alterá-los, evitando que algum desses se torne inconsistente com os demais. O mesmo se dá para o caso contrário. É possível verificar quais requisitos devem ser alterados para manter a consistência com os artefatos, modelos ou classes que foram alterados.

1.2METODOLOGIA

O desenvolvimento de ReqODE iniciou-se com um projeto de iniciação científica que tinha como propósito criar uma ferramenta que permitisse o registro e rastreabilidade dos requisitos de um sistema.

Foi necessário, inicialmente, criar um cadastro de conhecimento sobre tipos de requisitos, integrado a ODE como parte do Cadastro de Conhecimento do mesmo, o que tornou possível armazenar informações a respeito dos tipos de requisitos. Isto foi necessário, pois, normalmente, os requisitos são classificados segundo alguma categorização definida de acordo com as práticas de cada organização. Com isso, a organização que utiliza a ferramenta tem como manter uma classificação mais adequada a suas próprias características.

Partiu-se para a modelagem e implementação do cadastro de Requisitos, que permitia inicialmente registrar e classificar os requisitos. A seguir, desenvolveu-se uma forma de registrar os conflitos e dependências entre requisitos e foram implementados, também, meios de manter a rastreabilidade entre requisitos e outros elementos do processo de desenvolvimento. Consideraram-se, inicialmente, apenas casos de uso e artefatos.

Ao final do projeto de pesquisa, tinha-se uma ferramenta que permitia cadastrar e classificar requisitos do sistema, associá-los a seus dependentes, conflitos, requisitos que os compõem, casos de uso, artefatos e interessados (*stakeholders*), além de gerar um relatório de rastreabilidade com essas informações.

Para o desenvolvimento deste trabalho, foram verificadas novamente as atividades do processo de ER com objetivo de identificar novas funcionalidades que seriam interessantes de se acrescentar à ferramenta. Para tal, nova revisão da literatura foi feita e, como foco, escolheu-se a gerência de requisitos, com ênfase na rastreabilidade.

O objetivo foi tornar possível o registro de informações como prioridade e contexto de origem de um requisito, associá-lo a módulos do projeto que o tratam, associá-lo a classes que o implementem e buscar novas formas de rastrear essas ligações, a fim de tornar bidirecional e mais abrangente a rastreabilidade.

1.3ORGANIZAÇÃO DA MONOGRAFIA

Esta monografia contém, além da presente Introdução, mais cinco capítulos e três anexos.

O Capítulo 2 – Engenharia de Requisitos de Software – apresenta uma revisão bibliográfica de conceitos que foram fundamentais para o desenvolvimento deste trabalho, bem como foi justificada a necessidade do apoio automatizado à atividade de Engenharia de Requisitos e foram descritos os objetivos da criação de ReqODE para o apoio à gerência de requisitos.

O Capítulo 3 – Processo de Desenvolvimento Adotado – descreve as atividades realizadas durante o processo de desenvolvimento da ferramenta e funcionalidades associadas.

O Capítulo 4 – Especificação e Análise de Requisitos – apresenta os principais resultados da especificação de requisitos e da sua análise para o desenvolvimento de ReqODE, bem como da parte da ferramenta de cadastro de conhecimento de ODE responsável pelo cadastro de conhecimento sobre tipos de requisitos e da ferramenta de apoio ao cadastro de casos de uso, atores e classes.

O Capítulo 5 – Projeto, Implementação e Testes - apresenta a arquitetura do sistema organizada em componentes e seus respectivos diagramas de classes, levando em consideração requisitos tecnológicos. Finalmente, são tecidas algumas considerações sobre as fases de implementação e testes e é apresentada a ferramenta.

O Capítulo 6 – Considerações Finais – apresenta conclusões obtidas ao longo do desenvolvimento do trabalho e lista algumas linhas de pesquisa que poderão ser seguidas no mesmo contexto.

Em anexo, há três conjuntos de documentos, a saber:

- O Anexo A – Documentação da Ferramenta de Cadastro de Conhecimento sobre Tipos de Requisitos – inclui os documentos de especificação de requisitos, análise e projeto da parte da ferramenta de cadastro de conhecimento de ODE responsável pelo cadastro de conhecimento sobre tipos de requisitos.

- O Anexo B – Documentação da Ferramenta de Cadastro de Elementos de Modelo – inclui os documentos de especificação de requisitos, análise e projeto da ferramenta de apoio ao cadastro de casos de uso, atores e classes.
- O Anexo C – Documentação da Ferramenta ReqODE – inclui os documentos de especificação de requisitos, análise e projeto da ferramenta ReqODE.

CAPÍTULO 2ENGENHARIA DE REQUISITOS DE SOFTWARE

Requisitos definem o que o software realmente deverá fazer, quais são suas restrições e características. Eles devem refletir as necessidades e expectativas do usuário. Esse fato liga os requisitos à qualidade do produto final. O IEEE (*The Institute of Electrical and Electronics Engineers*) (IEEE,1998) define qualidade de software como: o grau com que um sistema, componente ou processo atende (1) aos requisitos especificados e (2) às expectativas ou necessidades de clientes ou usuários. Portanto, é importante o bom gerenciamento do processo de tratamento dos requisitos para garantir que, ao final do processo de desenvolvimento, os requisitos levantados tenham sido atendidos.

Engenharia de Requisitos é o nome dado a esse processo, que envolve as atividades de identificação, documentação e manutenção dos requisitos do sistema. Neste capítulo são apresentados conceitos importantes no contexto da ER que justificam a criação da ferramenta ReqODE.

1.4REQUISITOS

Identificar bem os requisitos é um passo essencial para se conseguir a satisfação do cliente por entregar um produto sólido e de qualidade (WIEGERS, 2003). Os requisitos guiam várias etapas do desenvolvimento e, por isso, é essencial um bom entendimento sobre eles, sua classificação e documentação, pois isto determinará o bom andamento de todo o processo de desenvolvimento de software.

Existem diversas definições para o termo requisito. Robertson e Robertson (ROBERTSON; ROBERTSON, 1999) descrevem requisitos como coisas que você deve descobrir antes de começar a construir seu produto.

Kotonya e Sommerville (KOTONYA; SOMMERVILLE, 1998) definem requisitos como a descrição de uma facilidade no nível do usuário, uma propriedade geral do sistema, uma restrição específica sobre o sistema, a especificação de um algoritmo particular que deva ser empregado em cálculos particulares ou uma restrição aplicável ao processo de desenvolvimento de software.

A norma IEEE Std 1233 – IEEE *Guide for Developing System Requirements Specifications* (IEEE, 1998) – em sua seção de definições, define requisito como: (i) uma

condição ou capacidade necessária para o usuário resolver um problema ou atingir um objetivo; (ii) uma condição ou capacidade que precisa ser atendida ou estar presente em um sistema ou componente para satisfazer um contrato, uma norma, uma especificação ou outro documento imposto formalmente; (iii) uma representação documentada de uma condição ou capacidade, tal como definido em (i) ou (ii).

Requisitos guiam várias etapas do desenvolvimento de um projeto, dentre elas planejamento, projeto (*design*), codificação e testes. Abaixo são citadas algumas situações, em diversas etapas, nas quais os requisitos são importantes (WIEGERS, 2003):

- No Planejamento de Projeto, requisitos são úteis para dimensionar o projeto, sendo utilizados para estimar o tamanho do produto. Planos são atualizados com mudanças em requisitos e requisitos prioritários são usados para direcionar interações.
- No Projeto e na Codificação, desenvolvedores revisam os requisitos, atributos de qualidade (requisitos não funcionais) são usados para direcionar o projeto arquitetural, requisitos são alocados a componentes e requisitos devem ser rastreados para o projeto e o código gerado.
- No que se refere aos Testes, requisitos permitem testes no início do projeto, além de guiar testes de sistema e de aceitação.

2.1.1 Tipos de Requisitos

Existem na literatura diversas classificações para requisitos. Porém, a classificação deve seguir uma forma de categorização que pode ser definida de acordo com as práticas de cada organização.

Em (PFLEEGER, 1998), são apresentados os seguintes tipos:

- *Funcional* (ações principais): Descreve as atividades do sistema, o que ele deve fazer.
- *Comportamental* (atividades de controle): Descreve a hierarquia das funções e atividades do sistema. Faz o controle do sistema.
- *Não-Comportamental* (atributos): Descreve as regras de funcionamento e condições necessárias para a execução de algumas funções. Trata também da garantia da qualidade.

Segundo (SOMMERVILLE, 2003) e (KOTONYA; SOMMERVILLE, 1998), como mostra a Figura 1, os requisitos podem ser:

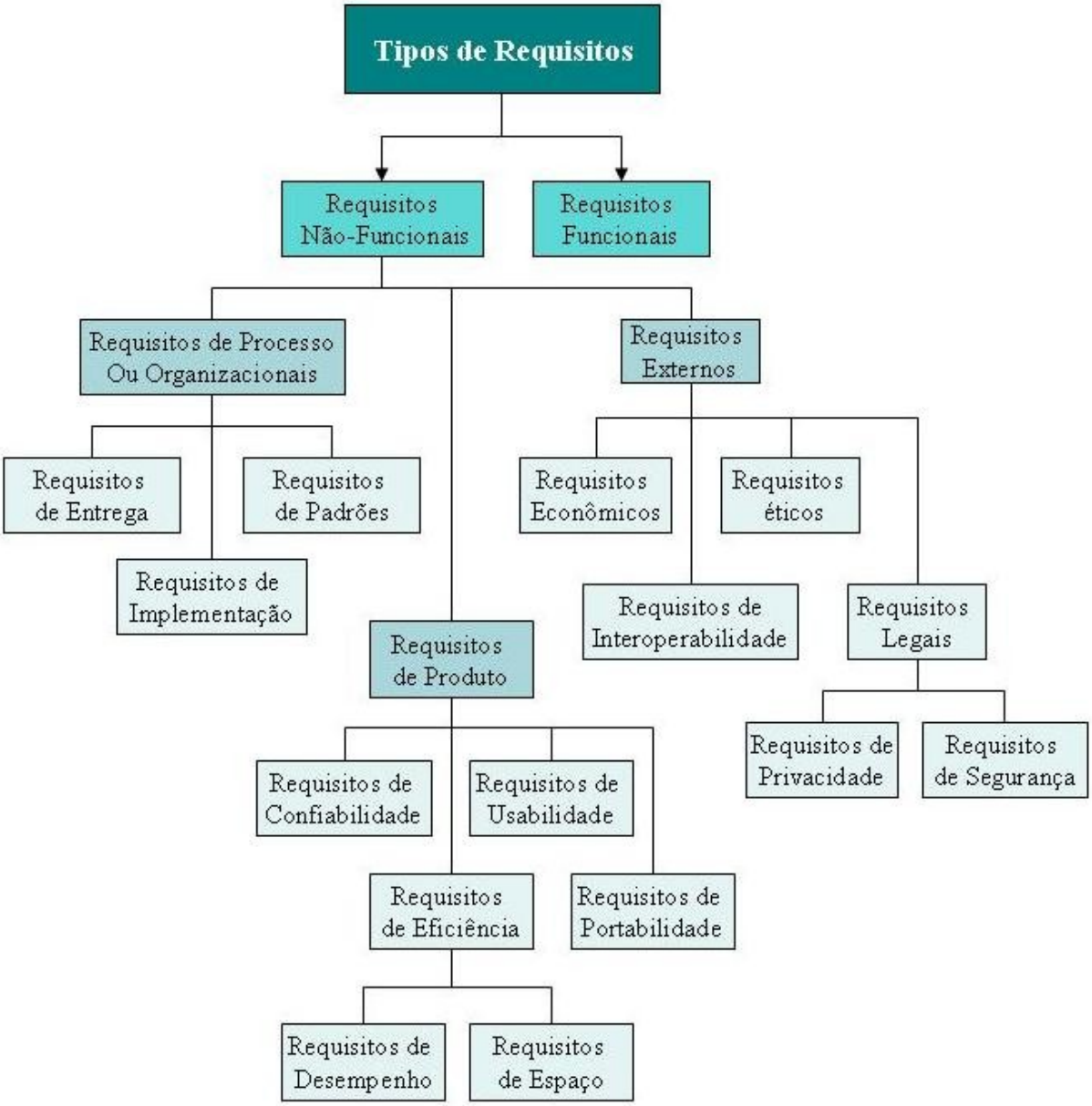


Figura 1 - Tipos de Requisitos

- *Funcionais:* descrevem a funcionalidade ou os serviços que se espera que o sistema forneça. Dependem do tipo de software, dos usuários e do tipo de sistema em desenvolvimento.
- *Não funcionais:* restrições sobre os serviços ou funções oferecidos pelo sistema. Podem estar relacionados a propriedades como confiabilidade, eficiência, usabilidade etc. Surgem das necessidades dos usuários em razão de restrições de orçamento, políticas organizacionais, necessidade de interoperação com outros sistemas, hardware, legislação, fatores externos etc. Alguns tipos de requisitos não funcionais incluem:

- Requisitos de Produto: especificam o comportamento do produto (sistema). Podem ser divididos em:
  - Requisitos de Confiabilidade: estabelecem a taxa aceitável de falhas e recuperabilidade.
  - Requisitos de Usabilidade: determinam a facilidade de uso e de aprendizagem do software.
  - Requisitos de Portabilidade: especificam em quais ambientes de hardware e / ou software o sistema deve operar.
  - Requisitos de Eficiência: que incluem:
    - Requisitos de Desempenho: especificam a rapidez em que o sistema deve operar (processamento e tempo de resposta).
    - Requisitos de Espaço: especificam quanta memória o sistema requer.
- Requisitos Organizacionais ou de Processo: procedentes de políticas e procedimentos das organizações. Dividem-se em:
  - Requisitos de Entrega (ou fornecimento): especificam quando o produto e seus documentos devem ser entregues.
  - Requisitos de Padrões: definem os padrões de processo que devem ser utilizados.
  - Requisitos de Implementação: definem a linguagem de programação ou método de projeto utilizado.
- Requisitos Externos: procedem de fatores externos e do seu processo de desenvolvimento. Dentre estes destacam-se:
  - Requisitos Econômicos: definem restrições de custo.
  - Requisitos Éticos: garantem que o sistema será aceitável para seus usuários e o público em geral.
  - Requisitos de Interoperabilidade: definem como o sistema deve interagir com outros sistemas, potencialmente em outras organizações.
  - Requisitos Legais: buscam assegurar que o sistema vai opera de acordo com a lei. Incluem-se neste caso requisitos de

Privacidade e de Segurança, que regem as normas de acesso de usuários.

Requisitos não funcionais, muitas vezes, são associados a características de qualidade. Em (ROBERTSON; ROBERTSON, 1999), definem-se requisitos não-funcionais como "propriedades ou qualidades que o produto deve ter". Logo um requisito não-funcional pode ser abordado como um item de qualidade, pois integra aspectos de qualidade na definição do próprio software (ROCHA et al., 2001). Assim a tarefa de gerenciar requisitos passa a cuidar não só de aspectos de funcionalidade, mas também de qualidade. Note que os tipos de requisitos não-funcionais de produto são semelhantes às características de qualidade indicadas pela ISO 9126-1 (ISO, 1999), a saber: confiabilidade, usabilidade, eficiência, portabilidade e manutenibilidade. Isso porque requisitos não funcionais são características e qualidades que tornam o produto atrativo, usável, confiável ou rápido, e são usados pelo cliente e usuários para julgar o produto (ROBERTSON; ROBERTSON, 1999).

2.1.2 Características de um requisito

Num mundo ideal, os requisitos, independentemente do tipo, deveriam apresentar as seguintes características (WIEGERS, 2003) (PFLEEGER, 1998):

- Completo: o requisito deve descrever completamente a funcionalidade a ser entregue. Ele deve conter todas as informações necessárias para o desenvolvedor projetar e implementar essa funcionalidade.
- Correto: cada requisito deve descrever exatamente a funcionalidade a ser construída.
- Consistente: o requisito não deve ser ambíguo ou conflitar com outro requisito.
- Possível (realista): deve ser possível implementar o requisito com a capacidade e com as limitações do sistema e do ambiente de desenvolvimento.
- Necessário: Os requisitos devem descrever algo que o cliente realmente precise ou que é requerido por algum fator externo ou padrão da organização.
- Passível de ser priorizado: os requisitos devem ter ordem de prioridade para facilitar o gerenciamento durante o desenvolvimento do sistema.
- Verificável e passível de confirmação: deve ser possível desenvolver testes para verificar se o requisito foi realmente implementado.

- Rastreável: deve ser possível identificar qual requisito gerou determinado produto do desenvolvimento, bem como identificar que produtos foram originados a partir de um requisito.

1.5ENGENHARIA DE REQUISITOS

A Engenharia de Requisitos é o ramo da Engenharia de Software que envolve todas as atividades relativas a desenvolver, documentar e dar manutenção ao conjunto de requisitos de um sistema (KOTONYA; SOMMERVILLE, 1998). Pode ainda ser descrita como um processo, ou seja, um conjunto organizado de atividades, métodos, técnicas, práticas e transformações que deve derivar, validar e manter os requisitos gerados.

O processo de engenharia de requisitos envolve criatividade, interação de diferentes pessoas, conhecimento e experiência para transformar informações diversas (sobre a organização, sobre leis, sobre o sistema a ser construído etc.) em modelos e documentos que direcionem o desenvolvimento de software (Figura 2) (KOTONYA; SOMMERVILLE, 1998).

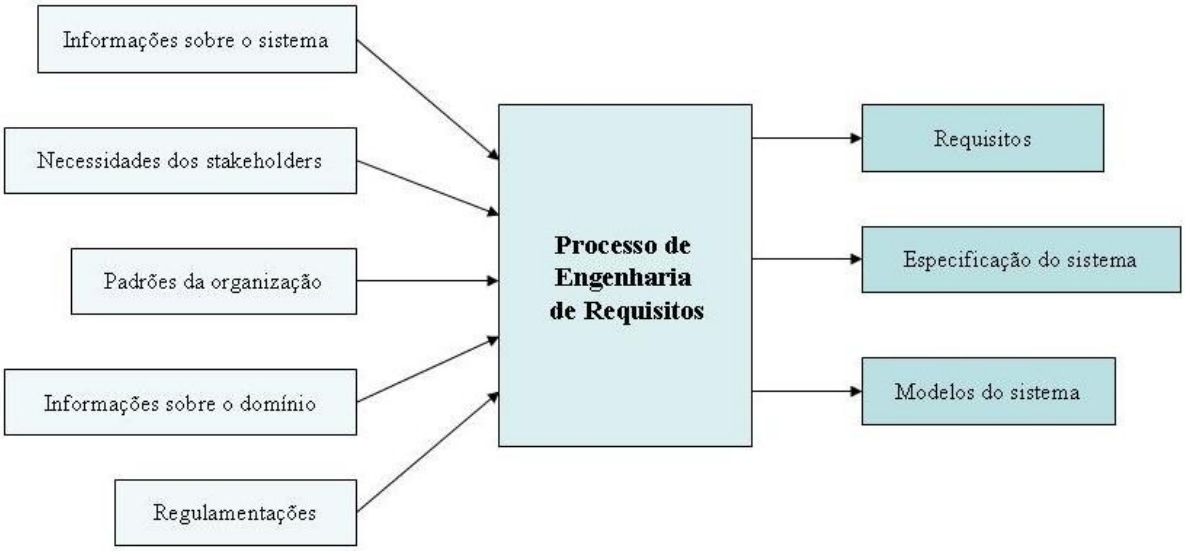


Figura 2 - Entradas e saídas do processo de engenharia de requisitos (KOTONYA; SOMMERVILLE, 1998)

KOTONYA e SOMMERVILLE (1998) definem Engenharia de Requisitos como a forma como escolhemos denominar as atividades desenvolvidas, no contexto do ciclo de vida de software, relacionadas com a definição dos requisitos de um sistema. Eles propõem que nesse processo sejam realizadas as atividades mostradas na Figura 3, a saber: levantamento (ou elicitação), análise e negociação, documentação, verificação e validação, e gerência de requisitos.

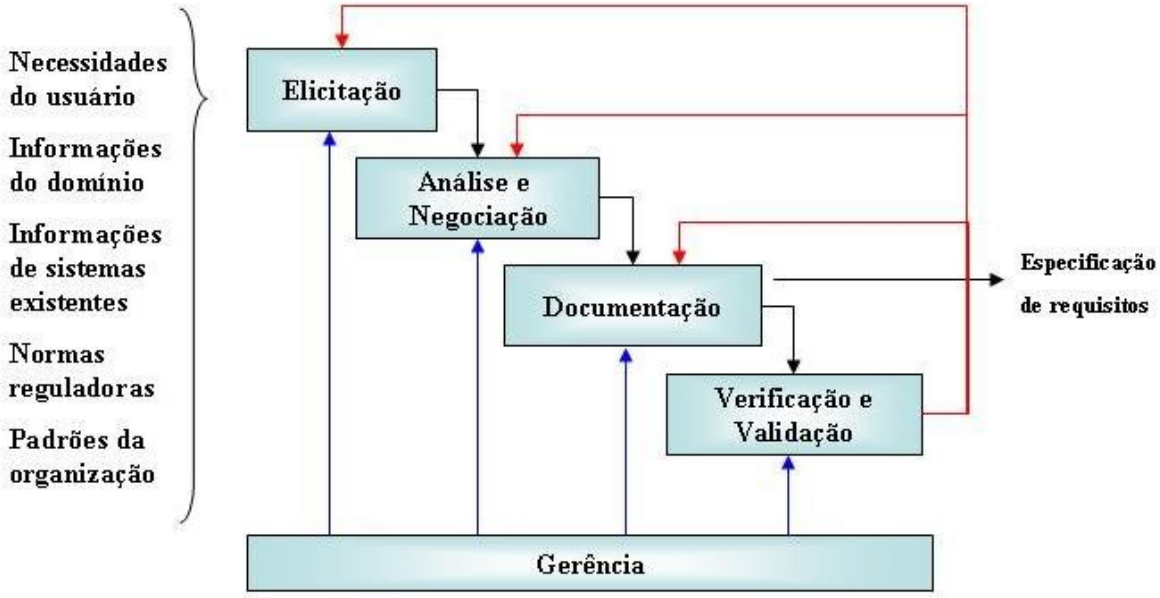


Figura 3 - Atividades do Processo de Engenharia de Requisitos.

2.1.3 Levantamento de Requisitos

O levantamento de requisitos corresponde à fase inicial do processo de Engenharia de Requisitos e envolve as atividades de descoberta dos requisitos. Costuma ser chamado também de processo de aquisição de requisitos ou de descoberta de requisitos.

Para levantar quais são os requisitos de um sistema, devem-se obter informações dos interessados (*stakeholders*), consultar documentos, obter conhecimentos do domínio e estudar o negócio da organização. Assim, são consideradas quatro dimensões para o levantamento de requisitos, como mostra a Figura 4 (KOTONYA; SOMMERVILLE, 1998):

- Entender o domínio da aplicação: entendimento geral da área na qual o sistema será aplicado;
- Entender o problema: entendimento dos detalhes do problema específico a ser resolvido com o auxílio do sistema a ser desenvolvido;
- Entender como o sistema irá afetar a organização e como contribuirá para os objetivos do negócio.
- Entender as necessidades e restrições do sistema: entendimento detalhado das necessidades de apoio a serem providas pelo sistema à realização do trabalho e aos interesses de cada um dos *stakeholders*, e dos processos de trabalho a serem apoiados

pelo sistema e do papel de eventuais sistemas existentes na execução e condução dos processos de trabalho.

Figura 4 - Dimensões do levantamento de requisitos

A atividade de levantamento de requisitos é dominada por fatores humanos, sociais e organizacionais e envolve pessoas de diferentes conhecimentos e objetivos, o que a torna complexa. CHRISTEL e KANG (CHRISTEL; KANG,1992 apud PRESSMAN, 2002) citaram alguns problemas que tornam o levantamento de requisitos uma tarefa difícil:

- Problemas de escopo: as fronteiras do sistema são mal definidas ou os clientes/usuários especificam detalhes técnicos desnecessários que podem confundir, em vez de esclarecer, os objetivos globais do sistema.
- Problemas de entendimento: Os clientes/usuários não estão completamente certos do que é necessário, têm pouca compreensão das capacidades e limitações de seu ambiente computacional, não têm pleno entendimento do domínio do problema, têm dificuldade de comunicar as necessidades ao engenheiro de sistemas, omitem informação que acreditam ser "óbvia", especificam requisitos que conflitam com as necessidades de outros clientes/usuários ou especificam requisitos que são ambíguos ou impossíveis de testar.
- Problemas de volatilidade: Os requisitos mudam ao longo do tempo.

Outras dificuldades citadas por KOTONYA e SOMMERVILLE (1998) complementam e reforçam os problemas acima citados. Eles descrevem quatro pontos principais:

- Compreender e coletar informações quando existem muitos termos desconhecidos, manuais técnicos etc.

- Pessoas que entendem o problema a ser resolvido podem ser muito ocupadas e não ter muito tempo para, juntamente como analista, levantar os requisitos e entender o sistema.
- Políticas organizacionais podem influenciar nos requisitos de um sistema.
- Os interessados (*stakeholders*) não sabem muito o que querem do sistema e não conhecem muitos termos.

Diversas técnicas podem ser utilizadas para a descoberta dos requisitos, cada uma delas possuindo um objeto diferente de observação ou investigação, e ainda o foco em tipos diferentes de requisitos. Logo, pela integração dessas técnicas podemos ter um levantamento mais eficaz. Algumas técnicas encontradas em (KENDAL; KENDAL, 1992, apud FALBO, 2000) e (KOTONYA; SOMMERVILLE, 1998) incluem:

- Amostragem: Selecionam-se elementos representativos de uma população. Estes são analisados e assume-se que a análise revela informações acerca da população como um todo. É um processo que diminui custos e tempo, é eficiente e não tendencioso.
- Investigação ou análise de documentos: captam informações e detalhes difíceis de conseguir por entrevista e observação. Revelam um histórico da organização e sua direção.
- Entrevistas: tipo mais comum que consiste em conversas direcionadas com um propósito específico e com formato "pergunta-resposta". Seus objetivos são descobrir problemas a serem tratados, levantar procedimentos importantes e saber a opinião e expectativas do entrevistado sobre o sistema. Seus passos principais consistem no planejamento, condução da entrevista, e a elaboração de um relatório. Existem técnicas para apoio e estruturas bem definidas para essa atividade.
- Questionários: permitem atingir mais pessoas que serão afetas pelo sistema. Conseguem-se obter informações como postura, crenças, comportamentos e características dessas pessoas.
- Observação: consiste em observar o comportamento e o ambiente dos indivíduos de vários níveis organizacionais. Utilizando-se essa técnica, é possível capturar o que realmente é feito e qual o tipo de suporte computacional é realmente necessário. Ajuda na confirmação de informações obtidas com outras técnicas e ajuda a identificar tarefas que podem ser automatizadas e não foram identificadas pelos interessados.
- Cenários: criam-se exemplos reais com uma abstrata descrição das funções providas pelo sistema. São importantes para identificar o estado do sistema antes do cenário, o

fluxo normal dos eventos do cenário, exceções no fluxo normal, informações sobre as atividades que ocorrem em paralelo e descrição do estado após o cenário.

- Prototipação: usada para capturar informações específicas sobre requisitos de informação do usuário, reações iniciais e sugestões do usuário, inovações e informações para revisão de planos para estabelecer prioridades e redirecionar planos.

2.1.4 Análise e Negociação de Requisitos

Análise e negociação são atividades que envolvem descobrir problemas com os requisitos do sistema e alcançar a concordância em mudanças para satisfazer os interessados (*stakeholders*). São atividades caras, porque pessoas experientes despendem tempo lendo documentos cuidadosamente e pensando sobre a implicação das afirmações desses documentos (KOTONYA; SOMMERVILLE, 1998).

A Análise de Requisitos consiste no estudo dos produtos do levantamento de requisitos para identificar os problemas e conflitos na definição dos requisitos (KOTONYA; SOMMERVILLE, 1998), com o objetivo de estabelecer um conjunto de requisitos completo e consistente.

Resumidamente pode-se dizer que a análise atende a dois propósitos: (i) providenciar um caminho para clientes e desenvolvedores concordarem com o que o sistema irá fazer e (ii) a especificação provê um guia para o projetista do sistema (PFLEEGER, 1998).

Algumas atividades a serem realizadas são (KOTONYA; SOMMERVILLE, 1998):

- Analisar a necessidade dos requisitos,
- Verificar a consistência e completude,
- Verificar se é possível que o sistema atinja esses requisitos.

Segundo (PRESSMAN, 2002), devem-se categorizar os requisitos e os organizar em subconjuntos relacionados, explorar cada um deles em relação aos demais, examiná-los quanto à consistência, omissões e ambigüidades, e ordená-los com base nas necessidades dos clientes/usuários.

Algumas das boas práticas citadas em (WIEGERS, 2003) complementam essas atividades citadas por (KOTONYA; SOMMERVILLE, 1998), a saber:

- Desenvolver um diagrama de contexto: trata-se de criar um simples modelo de análise que mostre como o novo sistema se adequa ao ambiente. Definem-se interfaces entre o sistema a ser produzido e entidades externas (usuários, dispositivos de hardware, outros sistemas etc.).

- Criar interfaces de usuário aplicando técnicas de prototipação: usa-se esta técnica quando usuários e desenvolvedores não têm muita certeza sobre os requisitos. Assim os clientes/usuários ajudam os desenvolvedores no entendimento de como o problema deverá ser resolvido.
- Modelar os requisitos: envolve a criação de um modelo de análise descrevendo requisitos num nível de abstração mais alto. Modelos podem indicar requisitos incorretos, inconsistentes, supérfluos e ausentes.
- Criar um dicionário de dados: deve-se confeccionar um dicionário de dados contendo a descrição de todos os dados e estruturas associados.
- Alocar os requisitos aos subsistemas: requisitos para sistemas complexos, que possuem vários módulos, devem ser separados por subsistema a que pertencem.

Há ainda na literatura algumas dicas de que artefatos podem ser utilizados para auxiliar na etapa de análise dos requisitos. KOTONYA e SOMMERVILLE (1998) incentivam o uso de *checklists*, ou seja, uma lista de questões que o analista deve usar para avaliar cada requisito, e matrizes de interação que são utilizadas para descobrir interações dos requisitos, como conflitos e sobreposição.

Descobertos os conflitos e definidas as propriedades do conjunto de requisitos do sistema, é necessário discutir essas conclusões com o cliente. Essa atividade é chamada de negociação e envolve a discussão dos conflitos e prioridades entre requisitos com os interessados.

Neste momento conflitos são inevitáveis, visto que cada interessado tem um ponto de vista diferente sobre o sistema. Alguns passos podem ser tomados (KOTONYA; SOMMERVILLE, 1998):

- Discussão: requisitos em que problemas foram encontrados são discutidos e os interessados presentes opinam sobre eles.
- Priorização: requisitos são priorizados para identificar requisitos críticos e ajudar nas decisões e planejamento.
- Concordância: soluções para os problemas são identificadas, mudanças são feitas e um compromisso sobre um conjunto de requisitos é acertado.

Usa-se uma abordagem iterativa, em que requisitos são eliminados, combinados e/ou modificados, de modo que cada parte alcance algum grau de satisfação (PRESSMAN, 2002).

2.1.5 Documentação

Os requisitos capturados nas etapas anteriores são descritos e modelados em documentos. Documentação é, portanto, uma atividade de registro e oficialização dos resultados da engenharia de requisitos, que tem como produto principal o Documento de Requisitos de Software ou Especificação de Requisitos de Software (ERS) (SOMMERVILLE, 2003).

A ERS é uma declaração oficial do que é exigido dos desenvolvedores de sistema e deve incluir os requisitos de usuário e uma especificação detalhada dos requisitos de sistema (SOMMERVILLE, 2003).

Numa ERS é possível representar os requisitos de software de diversas formas (WIEGERS, 2003), dentre elas: (i) descrições em linguagem natural, (ii) modelos gráficos e (iii) especificações formais que definem requisitos usando linguagens lógicas.

A ERS abrange um conjunto diversificado de interessados (SOMMERVILLE, 2003). KOTONYA e SOMMERVILLE (1998) descrevem qual a função e interesse de cada nível:

- Clientes: especificam os requisitos e os lêem para verificar se eles atendem a suas necessidades. Especificam mudanças nos requisitos.
- Gerentes: Utilizam o documento de requisitos para planejar um pedido de proposta para o sistema e para planejar o processo de desenvolvimento de software.
- Engenheiros de sistema: Utilizam os requisitos para compreender o sistema a ser desenvolvido.
- Engenheiros de teste de sistema: Utilizam os requisitos para desenvolver testes de validação para o sistema.
- Engenheiros de manutenção de sistema: Utilizam os requisitos para ajudar a compreender o sistema e as relações entre suas partes.

Uma série de organizações de grande porte definiu padrões para os documentos de requisitos, mas o padrão mais conhecido é o padrão IEEE/ANSI 830-1993 (IEEE, 1993). Embora esse padrão não seja ainda ideal, ele pode ser adaptado para definir um padrão dirigido às necessidades de uma organização e pode depender também do tipo de software que está sendo desenvolvido e da abordagem de desenvolvimento utilizada (SOMMERVILLE, 2003).

Algumas atividades são importantes de serem consideradas em uma organização para a construção de uma documentação de requisitos (WIEGERS, 2003):

- Adotar um padrão de estrutura para o documento.

- Identificar as fontes dos requisitos, ou seja, manter dados da origem do requisito. Assim, se alguma mudança for solicitada, será possível descobrir com quem deve ser validada essa mudança.
- Criar um identificador único para o requisito. Isso facilitará a rastreabilidade e controle das mudanças.
- Documentar também as regras do negócio e definir ligações entre os requisitos e as regras correspondentes.
- Especificar atributos de qualidade no Documento de Especificação de Requisitos, pois eles são importantes para o produto satisfazer as necessidades dos clientes.

2.1.6 Validação e Verificação

Nas atividades de Verificação e Validação de requisitos examina-se a especificação para assegurar que: (i) todos os requisitos do sistema tenham sido declarados de modo não-ambíguo, (ii) as inconsistências, omissões e erros tenham sido detectados e corrigidos, (iii) os requisitos estão em conformidade com as características de qualidade e (iv) realmente satisfazem a necessidades dos usuários (PRESSMAN, 2002) (KOTONYA; SOMMERVILLE, 1998) (WIEGERS, 2003).

Essa atividade envolve interessados (*stakeholders*), engenheiros de requisitos e desenvolvedores, que verificam os documentos de requisitos para garantir consistência, completude, exatidão, conformidade com padrões de qualidade, inexistência de erros de modelagem, ausência de ambigüidades e identificação de todos os conflitos (KOTONYA; SOMMERVILLE, 1998). Desta forma alguns problemas podem ser resolvidos, tais como falta de informações, conflitos, requisitos irreais, redundância, inconsistência, desorganização, fuga aos padrões da organização, requisitos incompletos e requisitos não claros.

Algumas atividades são essenciais para uma validação eficiente (WIEGERS, 2003):

- Inspecionar o documento de especificação de requisitos: envolve um grupo de pessoas que lêem e analisam os requisitos, procurando por problemas e discutindo soluções para os problemas encontrados. Os principais estágios são: (i) planejamento da revisão, que envolve a seleção de uma equipe e lugar para as reuniões, (ii) distribuição dos documentos entre os membros da equipe, (iii) preparação para revisão, quando cada membro lê os documentos para identificar conflitos, omissões, inconsistências, desvios dos padrões e outros problemas, (iv) acompanhamento das ações e (v) revisão (reformulação) do documento.

- Testar os requisitos: casos de teste são usados para verificar se modelos de análise e protótipos estão corretos. Vale ressaltar que o objetivo é validar os requisitos, não o sistema.
- Definir critérios de aceitação: é importante pedir para os usuários descreverem como eles vão determinar se o produto atende a suas necessidades e se é adequado para uso.

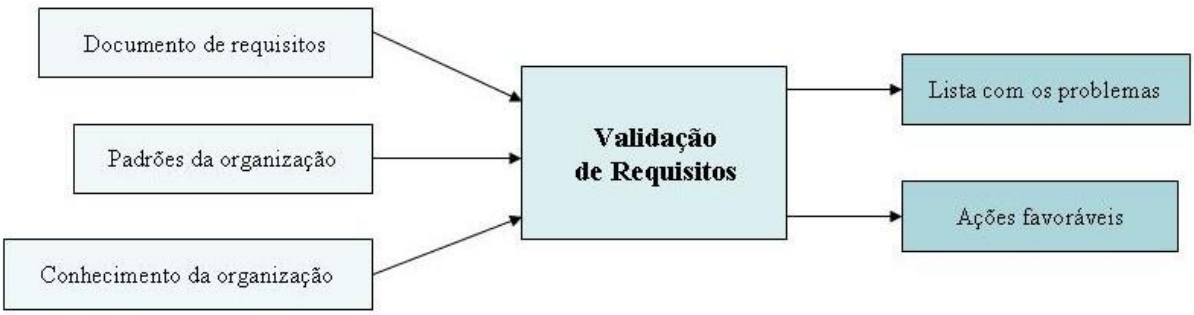


Figura 5 - Entradas e Saídas da atividade de Verificação e Validação de Requisitos

Resumidamente podemos dizer que a validação corresponde à certificação de que se está produzindo o produto correto. Mas é necessário, ainda, checar se o produto está sendo construído corretamente. Verificação se refere a esse processo de determinar se os produtos gerados estão sendo construídos de forma correta, de acordo com padrões previamente definidos (KOTONYA; SOMMERVILLE, 1998).

2.1.7 Gerência de Requisitos

A Gerência de Requisitos é definida como o conjunto de atividades que ajudam a equipe de projeto a identificar, controlar e rastrear requisitos e gerenciar mudanças de requisitos em qualquer época, à medida que o projeto prossegue (KOTONYA; SOMMERVILLE, 1998) (PRESSMAN, 2002). Envolve o gerenciamento de relacionamentos entre requisitos e de dependências entre documentos de requisitos e outros documentos do processo de desenvolvimento (KOTONYA; SOMMERVILLE, 1998).

A Gerência de Requisitos de Software é uma área de processo do nível 2 do CMMI, tendo como propósito gerenciar os requisitos dos produtos, do projeto e dos componentes do produto e identificar as inconsistências entre os requisitos e os planos do projeto e produtos de trabalho. Segundo o modelo, as principais atividades da gerência de requisitos são documentar as mudanças de requisitos e manter a rastreabilidade bidirecional entre requisitos

fonte e todos os requisitos do produto e dos componentes do produto (CHRISSIS *et al*., 2003).

O processo de Gerência de Requisitos, segundo (WIEGERS, 2003), inclui todas as atividades que mantêm a integridade e a exatidão dos requisitos levantados. Wiegers propõe as atividades mostradas na Figura 6, a saber: controle de mudanças, controle de versão, acompanhamento dos estados dos requisitos e rastreabilidade de requisitos.

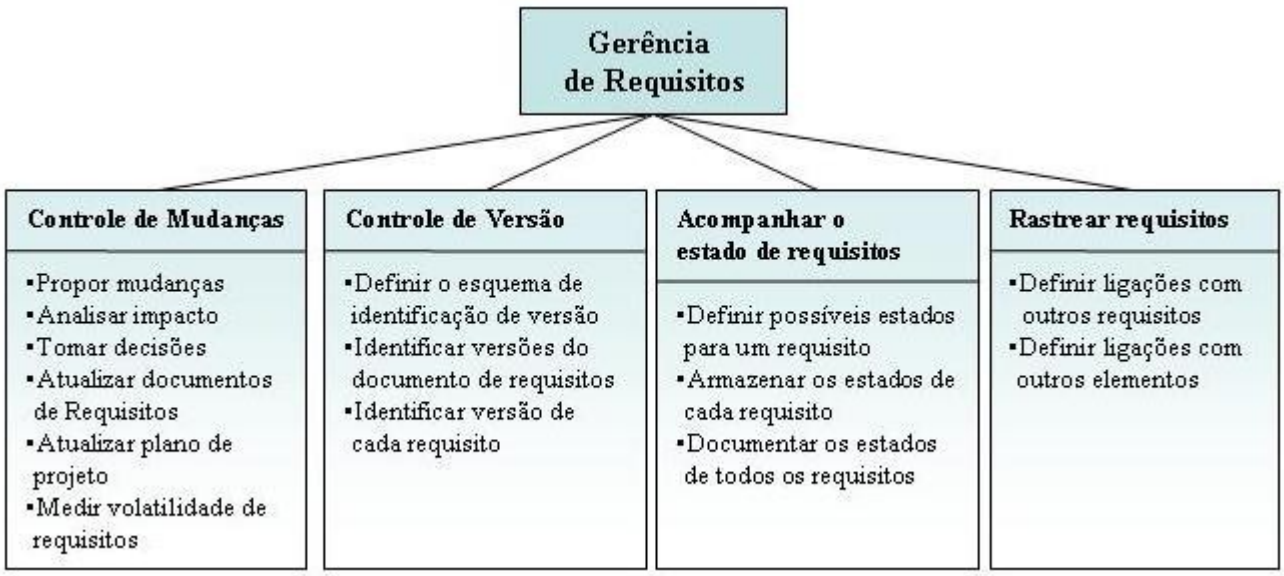


Figura 6 - Atividades da Gerência de Requisitos (WIEGERS, 2003)

No contexto de mudanças, requisitos podem ser classificados em dois tipos: estáveis, que são os responsáveis pela essência do sistema e do domínio da aplicação, e voláteis, que são específicos do sistema em um ambiente e para um cliente (KOTONYA; SOMMERVILLE, 1998). Os requisitos voláteis ainda podem ser divididos em quatro tipos diferentes: (i) requisitos mutáveis, que mudam por causa do ambiente em que o sistema opera, (ii) requisitos emergentes, que não podem ser completos durante a especificação e são completados durante projeto ou implementação, (iii) requisitos resultantes, que surgem durante a utilização do sistema, e (iv) requisitos de compatibilidade, que dependem de um equipamento.

Mudanças podem ser necessárias em diferentes momentos e por diferentes razões. Algumas causas citadas por (KOTONYA; SOMMERVILLE, 1998) são: (i) erros e mal entendimento dos requisitos, (ii) mudanças em legislações, (iii) novas informações sobre o

sistema e (iv) mudança nas prioridades do negócio. O gerenciamento dessas mudanças envolve as seguintes atividades (KOTONYA; SOMMERVILLE, 1998) (WIEGERS, 2003):

- Verificar se uma mudança é válida.
- Descobrir os requisitos afetados pela mudança.
- Rastrear informações.
- Propor mudanças nos requisitos.
- Estimar o impacto e o custo das mudanças.
- Desenvolver matrizes de rastreabilidade.
- Negociar com clientes.

No centro da atividade de gerência de requisitos está a rastreabilidade (LOPES, 2002), isto é, a habilidade de se acompanhar a vida de um requisito em ambas as direções do processo de software e durante todo o ciclo de vida. A rastreabilidade de requisitos só é possível se houver ligações explícitas entre requisitos e outros elementos do processo de software. Dessa forma, a identificação da composição de requisitos, das dependências entre requisitos, de requisitos conflitantes, da origem dos requisitos e de seus interessados, além da identificação de em qual artefato (documento, diagrama etc.) produzido durante o processo de software um requisito é tratado, é de fundamental importância para que a rastreabilidade seja implementada (KOTONYA; SOMMERVILLE, 1998).

Matrizes de rastreabilidade são os principais artefatos produzidos na fase de gerência de requisitos. Elas relacionam os requisitos identificados a um ou mais aspectos do sistema ou do seu ambiente, de modo que elas possam ser procuradas rapidamente para entender como uma modificação em um requisito vai afetar diferentes aspectos do sistema construído. Entre as muitas possíveis matrizes de rastreabilidade estão as seguintes (PRESSMAN, 2002):

- Matriz de rastreabilidade de características: mostra como os requisitos se relacionam a características importantes do sistema observáveis pelo cliente.
- Matriz de rastreabilidade de fontes: indica a fonte de cada requisito.
- Matriz de rastreabilidade de dependência: indica como os requisitos estão relacionados uns com os outros.
- Matriz de rastreabilidade de subsistemas: caracteriza os requisitos pelo subsistema que eles governam.
- Matriz de rastreabilidade de interface: mostra como os requisitos se relacionam com as interfaces internas e externas do sistema.

Algumas boas práticas são citadas por (WIEGERS, 2003) e resumem e completam as atividades acima citadas. São elas: (i) definir um processo por meio do qual as mudanças nos requisitos são propostas, analisadas e resolvidas; (ii) estabelecer um grupo de controle de mudanças (*Change Control Board*), que corresponde a um pequeno grupo de interessados que receberá as propostas de mudanças, avaliará e decidirá quais serão aceitas e rejeitadas e quais terão prioridade; (iii) realizar análise de impacto e custo das mudanças; (iv) estabelecer uma linha base (*baseline*) e controlar versões de documentos de requisitos; (v) manter um histórico das mudanças, armazenando sua data, quais alterações foram realizadas, quem as realizou, estado, entre outros; (vi) acompanhar o estado dos requisitos; (vii) medir a volatilidade dos requisitos; (viii) usar ferramentas de apoio à gerência de requisitos; e (ix) criar matrizes de rastreabilidade.

1.6APOIO AUTOMATIZADO À ENGENHARIA DE REQUISITOS

Pela breve exposição anterior, é possível notar o quanto é complexo o processo de Engenharia de Requisitos (ER). Dada essa complexidade, construir ferramentas de apoio às atividades desse processo é fundamental para a efetiva execução do mesmo. Além disso, alguns aspectos da ER, tal como a gerência de requisitos, envolve um grande número de informações o que torna o trabalho praticamente inviável sem o apoio de ferramentas (LOPES, 2002).

WIEGERS (2003) considera uma boa prática a utilização de uma ferramenta de apoio. KOTONYA e SOMMERVILLE (1998) também destacam essa importância e descrevem as principais funcionalidades que uma ferramenta deve ter para apoiar o processo de ER: (i) armazenamento dos requisitos, (ii) facilidade para criar documentos, (iii) gerência de mudanças e (iv) rastreabilidade de requisitos.

Para manter as ligações de rastreabilidade com outros elementos do desenvolvimento, é importante que a ferramenta esteja integrada às demais utilizadas no desenvolvimento de software. Isso pode ser feito pela integração da ferramenta a um Ambiente de Desenvolvimento de Software (ADS). Um ADS pode ser definido como um conjunto de ferramentas integradas, de modo que o desenvolvedor de software possa ser apoiado ao longo de todo o processo de software, ou pelo menos em porções significativas dele (HARRISON et al., 2000).

Neste trabalho apresentamos ReqODE, uma ferramenta de apoio à Engenharia de Requisitos, integrada a ODE (*Ontology-based software Development Environment*) (FALBO *et al*., 2003), um Ambiente de Desenvolvimento de Software Centrado em Processo.

1.7ReqODE

ReqODE é a ferramenta de apoio ao processo ER em ODE que foi construída tomando por base uma ontologia de requisitos e, portanto, grande parte de suas tarefas é realizada utilizando e respeitando os modelos e restrições impostos por essa ontologia (NARDI ; FALBO, 2006). Ela permite tratar requisitos desde o levantamento, quando são capturados os requisitos, até a gerência dos mesmos.

O fato de ReqODE estar integrada a ODE permite que diversos elementos já cadastrados no ambiente possam ser associados a um requisito, a saber:

- É possível relacionar requisitos a interessados e a seus responsáveis, recursos humanos cadastrados no ambiente por meio da ferramenta de Gerência de Recursos Humanos e Alocação de Recursos ao projeto em questão (COELHO, 2007).
- Requisitos podem ser associados a casos de uso e classes, registrados em ODE por meio de OODE, a ferramenta de modelagem UML de ODE (SILVA, 2003), e por meio do cadastro de elementos de modelo desenvolvido neste trabalho.
- Requisitos podem ser associados a artefatos internos, produzidos pelas demais ferramentas de ODE, ou externos a ODE, cadastrados no ambiente por meio da ferramenta de apoio ao planejamento da documentação (SEGRINI, 2007). Esse é um aspecto chave para se tratar a rastreabilidade.
- Requisitos podem ser associados aos módulos do projeto cadastrados por meio da ferramenta de decomposição do produto (ARANTES, 2006).
- Podem ser armazenadas as origens de um requisito, ou seja, informações como atividades, artefatos e recursos humanos que originaram o requisito. Essa funcionalidade de cadastro de contexto de criação de requisitos foi desenvolvida também neste trabalho.

São diversas as informações acerca de um requisito. Dentre elas, serão armazenadas em ReqODE:

- Identificador: gerado automaticamente para controle e identificação de um requisito.

- Sentença: descreve em poucas palavras do que trata o requisito.
- Descrição: descrição detalhada do requisito.
- Data da criação: data em que o requisito foi registrado.
- Prioridade: indica o nível de importância do requisito.
- Estado: informa em qual estado encontra-se o desenvolvimento do requisito.
- Razões da Criação: justificativas e importância do requisito.
- Comentários Gerais: qualquer outra informação acerca do requisito que seja relevante para o processo de desenvolvimento.
- Composição: requisitos que compõem determinado requisito.
- Dependentes: requisitos que dependem do requisito em questão.
- Conflitos: Requisitos conflitantes, tendo que ser escolhidos quais devem ser priorizados.
- Tipo Primitivo: identifica se o requisito é funcional ou não funcional.
- Tipo do Requisito: uma classificação organizacional mais detalhada.
- Projeto: projeto ao qual pertence o requisito.
- Módulo: módulo do projeto ao qual pertence o requisito.
- Recursos Humanos: incluindo os interessados (*stakeholders*) no requisito e os responsáveis pelo mesmo.
- Contexto de origem: define a fonte, ou seja, o contexto em que o requisito foi capturado, armazenando artefatos analisados, atividade e pessoas envolvidas.
- Casos de Uso: casos de uso que foram derivados do requisito.
- Classes: classes que implementam o requisito.
- Artefatos: artefatos que foram gerados durante o desenvolvimento tendo como base o requisito.

Pelo uso de ReqODE será possível criar um Documento de Rastreabilidade. Assim, se alguma alteração em um requisito for necessária, permite-se identificar que elementos provavelmente também sofrerão mudanças, possibilitando a análise do impacto que essa alteração vai causar. Esse documento pode ser gerado de duas formas: (i) completo, contendo todas as informações armazenadas acerca de um requisito, ou (2) customizado, apresentando apenas informações filtradas de acordo com a necessidade do usuário.

O relatório acima citado toma como foco o requisito, descrevendo, para cada requisito, quais objetos estão relacionados a ele. Dessa forma, uma busca de sentido contrário seria

exaustiva e, dependendo do número de requisitos cadastrados, poderia ser até inviável. Para evitar esse tipo de situação, foi adicionada a ReqODE a funcionalidade de buscar por requisitos que estejam relacionados a um elemento específico, de modo a garantir a rastreabilidade bidirecional e a identificação de pontos de mudança durante o processo de software. Essas buscas podem ser feitas para os seguintes elementos: casos de uso, classes, contextos de criação, artefatos de origem, artefatos gerados, responsáveis, interessados e módulos do projeto.

No que se refere à implementação, ReqODE foi desenvolvida utilizando-se a linguagem Java, priorizando a portabilidade, com interfaces Swing. A persistência dos objetos é feita no banco de dados relacional PostgreSQL, utilizando-se o *framework* de mapeamento objeto-relacional Hibernate. Foi uma premissa do desenvolvimento que as tecnologias utilizadas fossem livres, pois o grupo de desenvolvimento de ODE tem a intenção de disponibilizar o ambiente como software livre em um futuro próximo.

CAPÍTULO 3PROCESSO DE DESENVOLVIMENTO ADOTADO

Um processo é um conjunto de atividades organizadas de forma a produzir um produto que satisfaça um conjunto de objetivos e padrões. Ele é importante, pois mantém a consistência e estrutura entre as atividades a serem realizadas no desenvolvimento de um software (PFLEEGER, 1998). Ainda, segundo (PRESSMAN, 2002), ele é um conjunto de tarefas necessárias para construir um software de alta qualidade.

O processo de desenvolvimento de software, para ser eficiente, deve ser adequado ao domínio da aplicação e ao projeto específico. Processos devem ser definidos caso a caso considerando-se as especificidades da aplicação, a tecnologia a ser adotada, padrões da organização e o grupo de desenvolvedores (FALBO, 2002).

O passo inicial para a definição de um processo é a escolha de um modelo de ciclo de vida, pois ele organiza as macro-atividades básicas estabelecendo precedências e dependência entre as mesmas. De maneira geral, as seguintes fases estão presentes em um modelo de ciclo de vida (FALBO, 2002):

- **Planejamento de Projeto**: estabelecido o escopo do software, deve-se elaborar um plano de projeto, planejar as atividades a serem realizadas, escolher um modelo de ciclo de vida, estimar os recursos necessários e o cronograma a ser seguido para o desenvolvimento do produto de software.
- **Especificação de Requisitos**: atividade responsável pelo refinamento do escopo e captura dos requisitos do produto a ser desenvolvido;
- **Análise de Requisitos**: responsável por modelar, avaliar e documentar os requisitos funcionais do sistema;
- **Projeto**: envolve duas etapas principais: projeto da arquitetura do sistema e projeto detalhado. Na primeira define-se a arquitetura geral do software, tendo por base o modelo construído na fase de análise de requisitos. Na segunda detalha-se o projeto de software para cada componente identificado na etapa anterior.
- **Implementação**: nesta fase, cada unidade de software projetada é implementada.
- **Testes**: esta atividade deve incluir vários níveis de testes: teste de unidade, teste de integração e teste de qualificação. Inicialmente, cada unidade de software implementada é testada e seus resultados são documentados. O próximo passo é

integrar os diversos componentes de software até se obter o sistema. A seguir, todo o sistema é testado;

- **Implantação:** uma vez testado, o software deve ser colocado em produção;
- **Operação:** nesta fase, o software é utilizado pelos usuários no ambiente de produção;
- **Manutenção:** após ter sido entregue para o usuário, alterações ocorrerão porque erros foram encontrados, porque o software precisa ser adaptado para acomodar mudanças em seu ambiente externo ou porque o cliente necessita de funcionalidade adicional ou aumento de desempenho.

Nas seções que seguem é discutida a escolha do modelo de ciclo de vida e descrito o processo de desenvolvimento adotado para a realização deste trabalho, detalhando suas principais atividades.

1.8MODELO DE CICLO DE VIDA

O modelo de ciclo de vida adotado neste trabalho foi o modelo espiral (PRESSMAN, 2002), uma vez que esse permite a criação de versões, que podem ser incrementadas até a obtenção do produto completo (FALBO, 2002). Tal modelo abrange as melhores características do ciclo de vida clássico quanto da prototipação. Além disso, ele prevê desenvolvimento evolutivo e cíclico (PRESSMAN, 2002).

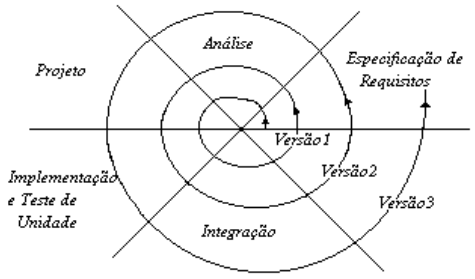


Figura 7 - Ciclo de vida espiral

Este trabalho foi desenvolvido em três ciclos principais. O primeiro ciclo foi realizado durante o período da iniciação científica e produziu uma versão que provia classificação e

cadastro de requisitos e definição de algumas ligações de rastreabilidade. Definiu-se a arquitetura de pacotes e foram derivadas as classes necessárias a partir da ontologia de requisitos definida em (NARDI; FALBO, 2006). Criou-se, portanto, um cadastro de tipos de requisitos e um cadastro de requisitos, já associando um requisito a alguns elementos do desenvolvimento, a saber, casos de uso, recursos humano e artefatos.

O segundo ciclo gerou um protótipo disponibilizado para avaliação por uma organização de software real e que foi apresentado na Sessão de Ferramentas do XX Simpósio Brasileiro de Engenharia de Software (SBES´2006) (MARTINS et al., 2006). Nesse ciclo, fizeram-se ajustes objetivando ampliar a rastreabilidade, passando-se a manter a rastreabilidade entre requisitos e módulos de projeto e entre requisitos e contexto de origem. Além disso, um relatório com as ligações de rastreabilidade registradas na ferramenta foi provido.

No terceiro ciclo, definiu-se um escopo que ampliava e ajustava a rastreabilidade para melhor apoiar a gerência dos requisitos no ambiente. Passou-se a possibilitar rastrear um requisito desde a sua origem, o que inclui artefatos, atividades e recursos humanos envolvidos, até o nível de classes, com seus métodos e atributos. Foram criadas novas funcionalidades de relatórios customizados e buscas de rastreabilidade bidirecional. Para a associação com classes e casos de uso, foi definida uma reformulação na forma como esses elementos eram criados. Anteriormente a este trabalho, só era possível cadastrá-los por meio da ferramenta de modelagem UML de ODE, OODE (SILVA, 2003). Criou-se, então, uma ferramenta de cadastro de classes e casos de uso, que permite criá-los sem usar OODE.

A seguir, as principais atividades do processo adotado são brevemente descritas.

1.9ESPECIFICAÇÃO DE REQUISITOS

Na fase de especificação de requisitos, foram estudadas algumas ferramentas CASE disponíveis no mercado, objetivando observar as funcionalidades providas a seus usuários. Além disso, foram utilizadas diversas fontes bibliográficas, tais como (PFLEEGER, 1998), (KOTONYA; SOMMERVILLE, 1998), (PRESSMAN, 1992), (ROBERTSON; ROBERTSON, 1999), (SOMMERVILLE, 2003), (WIEGERS, 2003), para identificar funcionalidades necessárias para o bom apoio ao processo de Engenharia de Requisitos.

Grande parte dos requisitos veio de uma parceria selada com uma organização de desenvolvimento de software que buscava a certificação CMMI nível 2 e, para isso,

necessitava de uma ferramenta de gerência de requisitos. Essa empresa forneceu informações muito importantes sobre necessidades reais de uma organização de software no tratamento dos requisitos de um sistema.

O resultado desse estudo permitiu definir o escopo deste trabalho bem como modelar os requisitos funcionais das ferramentas propostas por ele. Essa fase gerou modelos de casos de uso, além de um documento de especificação de requisitos apresentado parcialmente no capítulo 4 e integralmente nos anexos A, B e C.

1.10ANÁLISE DE REQUISITOS

Identificados os requisitos e tomando por base a ontologia de requisitos proposta em (NARDI; FALBO, 2006), foram elaborados os diagramas de classes para as ferramentas necessárias para o efetivo gerenciamento dos requisitos durante o desenvolvimento de software. Os principais modelos produzidos nesta atividade são apresentados parcialmente no capítulo 5 e integralmente nos anexos A, B e C.

1.11PROJETO

Nessa fase incorpora-se a requisitos tecnológicos aos requisitos essenciais do usuário, projetando o que será construído na implementação. De modo geral, dois grandes passos podem ser identificados (FALBO, 2003):

- Projeto de Arquitetura do Sistema: descreve cada um dos componentes do sistema e as comunicações entre eles.
- Projeto de Objetos: descreve aspectos de implementação de cada uma das classes do sistema, incluindo o projeto procedural de cada operação, a definição de classes internas e o projeto de estruturas de dados para os atributos das classes.

O ponto de partida para a definição da arquitetura é a organização das classes em pacotes, agrupando as classes de acordo com sua função no sistema, ou seja, seu estereótipo. As ferramentas deste trabalho tiveram seus pacotes organizados nos seguintes componentes básicos, derivados a partir do modelo MVC (Modelo-Visão-Controlador) (SUN, 2006):

- **Componente de Domínio do Problema:** responsável por implementar diretamente as classes do domínio do problema;
- **Componente de Interface com o Usuário:** implementa as interfaces com o usuário;

- **Componente de Gerência de Tarefas:** responsável por controlar as tarefas do sistema, implementando os casos de uso identificados no levantamento de requisitos;
- **Componente de Controle de Interação:** controla as interações, fazendo a comunicação entre as interfaces e a gerência de tarefas.
- **Componente de Gerência de Dados:** responsável por armazenar e recuperar objetos. Foi construída baseada no padrão DAO (*Data Access Object*) (SUN, 2007), usando o *framework* de persistência objeto-relacional Hibernate (BAUER; KING, 2005).

Os resultados dessa fase são discutidos em mais detalhes no capítulo 5. Assim como os demais documentos, a documentação completa das especificações de projeto produzidas neste trabalho estão nos anexos A, B e C.

1.12IMPLEMENTAÇÃO E TESTE DE UNIDADE

A implementação concretiza as decisões tomadas na atividade de projeto utilizando uma linguagem de programação. Para a implementação deste trabalho, utilizou-se a linguagem Java, que segue o paradigma orientado a objetos. Vale ressaltar que procurou-se reutilizar código e seguiu-se os padrões de interface propostos no contexto do Projeto ODE.

Conforme citado anteriormente, o mapeamento para o banco de dados de ODE, implementado em PostgreSQL (POSTGRESQL, 2007), foi feito utilizando-se o *framework* Hibernate (BAUER; KING, 2005), que permite a distribuição do ambiente e o tratamento de versão para controle de acesso concorrente.

Para criação de relatórios foi utilizado o *framework* JasperReports (JASPERREPORTS, 2007) e a ferramenta iReport (IREPORT, 2007), que apóiam a criação de documentos em ambientes Java.

Por fim, durante o processo de implementação, à medida que os módulos do sistema iam sendo implementados, realizaram-se testes de unidade.

1.13TESTES DE INTEGRAÇÃO E VALIDAÇÃO

Além dos testes de unidade realizados durante a fase de implementação, realizaram-se também testes de integração com o objetivo de verificar se os componentes integrados funcionavam de acordo com o desejado.

Uma vez testado o sistema como um todo, realizou-se a validação, com o propósito de conferir se os requisitos de usuário foram satisfeitos pelo sistema.

CAPÍTULO 4ESPECIFICAÇÃO E ANÁLISE DE REQUISITOS

O processo de desenvolvimento de software inicia-se pelo entendimento do que o sistema deverá fazer e de como o sistema será usado. Devem-se desenvolver modelos simples e passíveis de entendimento por desenvolvedores, clientes e usuários. Os modelos iniciais devem descrever o sistema, seu ambiente e como eles estão relacionados. Utiliza-se comumente a modelagem de casos de uso, pois permite estruturar uma visão mais externa do sistema. Uma vez que isto esteja bem definido, a modelagem pode então ser iniciada (FALBO, 2002).

Resumidamente as atividades de análise e especificação de requisitos compõem um processo de descoberta, refinamento, modelagem e especificação dos requisitos levantados inicialmente (FALBO, 2002).

Para iniciar o desenvolvimento deste trabalho, foram levantados os requisitos das ferramentas a serem desenvolvidas. O escopo foi refinado e os requisitos identificados e analisados. As funcionalidades foram identificadas e documentadas, usando modelos de casos de uso.

Identificados os requisitos, estes foram modelados, avaliados e documentados. Utilizou-se orientação a objetos para definição do sistema e notações da UML para modelagem de classes e estados.

Este capítulo apresenta parcialmente nas seções 4.1, 4.2 e 4.3 a especificação de requisitos e os modelos de análise elaborados para as ferramentas de Cadastro de Conhecimento sobre Tipos de Requisitos, Cadastro de Elementos de Modelo e de Apoio à Engenharia de Requisitos (ReqODE), respectivamente. Os documentos completos são encontrados, respectivamente, nos Anexos A, B e C.. A seção 4.4 trata dos requisitos não funcionais considerados para todas as ferramentas.

1.14CONHECIMENTO SOBRE TIPOS DE REQUISITOS

É importante manter padronizada a classificação dos requisitos de uma organização. Para isso, é essencial saber quais os tipos de requisitos por ela considerados. Assim é necessário desenvolver uma ferramenta que forneça ao usuário apoio direcionado ao conhecimento organizacional sobre requisitos. A ferramenta proposta é integrada a ODE

como parte do Cadastro de Conhecimento do mesmo, o que torna possível armazenar informações a respeito dos tipos de requisitos, podendo associá-los a outros elementos cadastrados no ambiente.

Os elementos que definem a ferramenta de Cadastro de Conhecimento sobre Tipos de Requisitos são tratados no pacote Conhecimento.requisito, um sub-pacote do Conhecimento, como mostra o diagrama de pacotes da Figura 8.

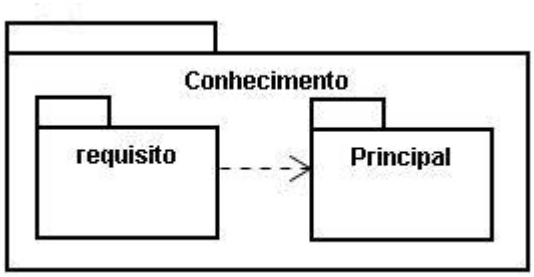


Figura 8 - Diagrama de pacotes da Ferramenta de Cadastro de Conhecimento sobre Tipos de Requisitos

A Figura 9 apresenta o diagrama de casos de uso desse pacote, que no momento considera apenas o seguinte caso de uso:

- **Cadastrar Tipo de Requisito:** incluindo funcionalidades para a criação, alteração, consulta e exclusão de tipos de requisitos, que permitirão a devida classificação dos requisitos tratados pelos sistemas de uma organização.

Figura 9 - Diagrama de Casos de Uso do pacote *Conhecimento.requisito*

Dentre os pacotes acima mostrados na Figura 8, apenas o pacote *Conhecimento.requisito* foi criado para atender aos objetivos deste trabalho. O restante já existia no ambiente ODE e é utilizado com o objetivo de facilitar a integração e manter a consistência do ambiente.

A Figura 10 mostra o diagrama de classes do pacote *Conhecimento.requisito*. Para representar os tipos de requisitos no ambiente, foi criada a classe *KTipoRequisito*. Como um tipo de requisito pode ser subtipo de outro, foi adicionado o auto-relacionamento representado no modelo. É importante ressaltar que essa relação é transitiva (isto é, se um tipo de requisito *tr1* é subtipo de outro *tr2*, que por sua vez é um subtipo de *tr3*, então *tr1* é também um subtipo de *tr3*) e irreflexiva (ou seja, um tipo de requisito não pode ser subtipo dele mesmo). Note que *KTipoRequisito* herda da classe *Conhecimento*; isso ocorre em ODE com todas as classes que representam algum conhecimento organizacional.

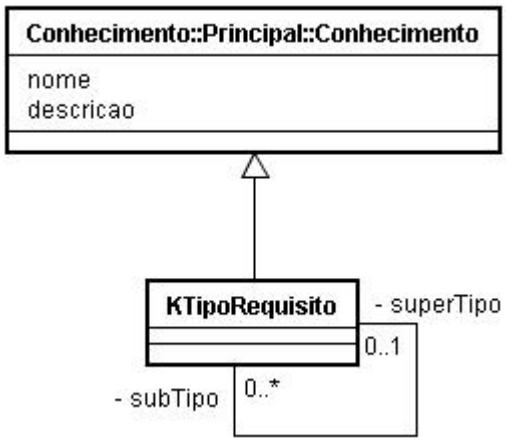


Figura 10 - Diagrama de classes do pacote *Conhecimento.requisito*

1.15CADASTRO DE ELEMENTOS DE MODELO

Durante o processo de desenvolvimento de software, vários artefatos são produzidos para que informações importantes não sejam perdidas e para facilitar a visualização de problemas e soluções. Dentre esses artefatos, podemos citar diagramas de classes, diagramas de casos de uso, diagramas de pacotes e diagramas de componentes, dentre outros. Para a construção desses diagramas, devem ser definidos os elementos básicos que os compõem, denominados elementos de modelo, segundo a UML.

No ambiente ODE existe a ferramenta de modelagem UML OODE (SILVA, 2003), usada normalmente para definir Modelos de Objeto e Elementos de Modelo. Porém, muitas vezes, é necessário criar alguns desses elementos em outro momento do desenvolvimento, como por exemplo, nas estimativas por pontos de casos de uso, na qual podem ser informados casos de uso, atores e classes. Logo, para dar maior flexibilidade e agilidade, é útil ter um

cadastro mais simples, no qual esses elementos não são modelados, e sim, cadastrados no sistema.

Para esse cadastro apenas algumas informações são requeridas: das classes, deve-se informar o nome e a descrição, podendo ser informados, ainda, atributos e operações; dos casos de uso, precisa-se saber o nome, a descrição, os atores que o realizam, eventos que fazem parte do mesmo e as classes que estão relacionadas ao caso de uso; de atores, é necessário saber nome e descrição.

O cadastro de elementos de modelo é tratado nos pacotes *UML.ElementosComportamentais.CasosUso* e *UML.Fundacao.Nucleo*, mostrados no diagrama de pacotes da Figura 11.

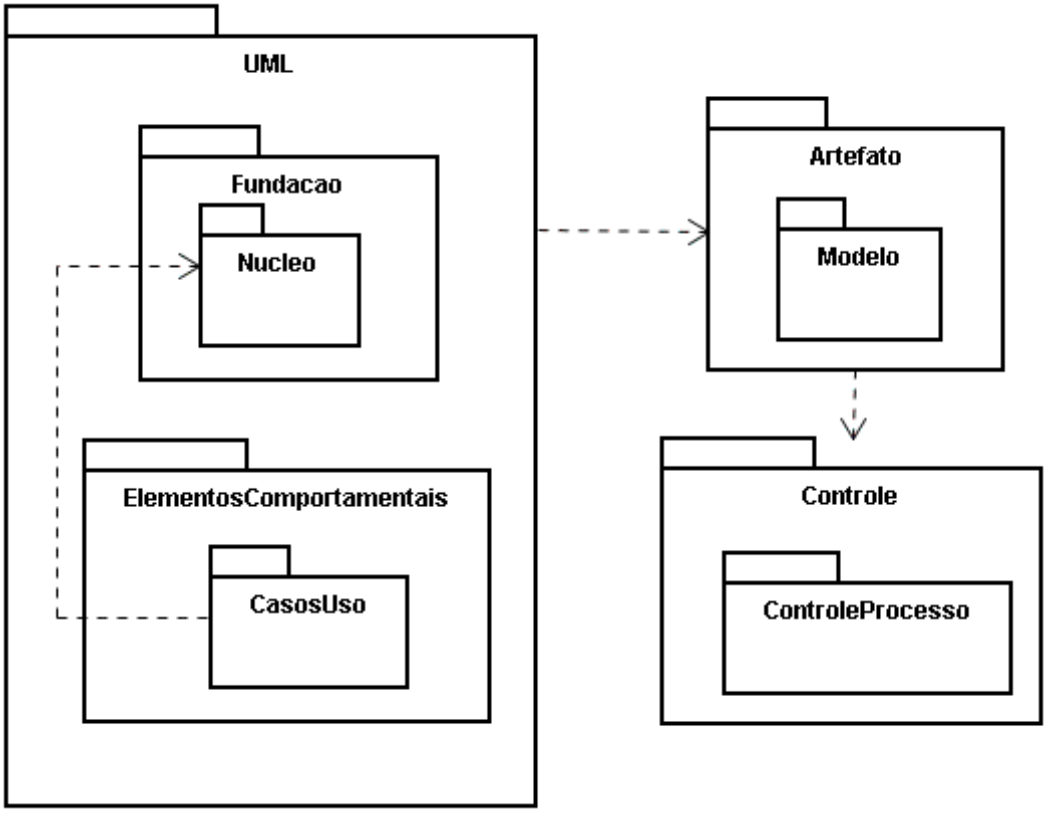


Figura 11 - Diagrama de Pacotes da Ferramenta de Cadastro de Elementos de Modelo

Conforme citado anteriormente, dentro do pacote UML estão todas as classes relativas ao cadastro de elementos de modelo. De interesse direto, têm-se apenas algumas classes dos pacotes Nucleo e CasoUso. Nos pacotes que restam, a saber, *Controle.Processo* e *Artefato.Modelo*, existem as classes responsáveis por armazenar informações sobre modelos

de objetos, que armazenam os elementos de modelo de interesse e os relacionam ao projeto no contexto do qual foram criados.

A Figura 12 mostra o diagrama de casos de uso referente ao sub-pacote *UML.ElementosComportamentais.CasosUso*, cujas descrições são apresentadas a seguir.

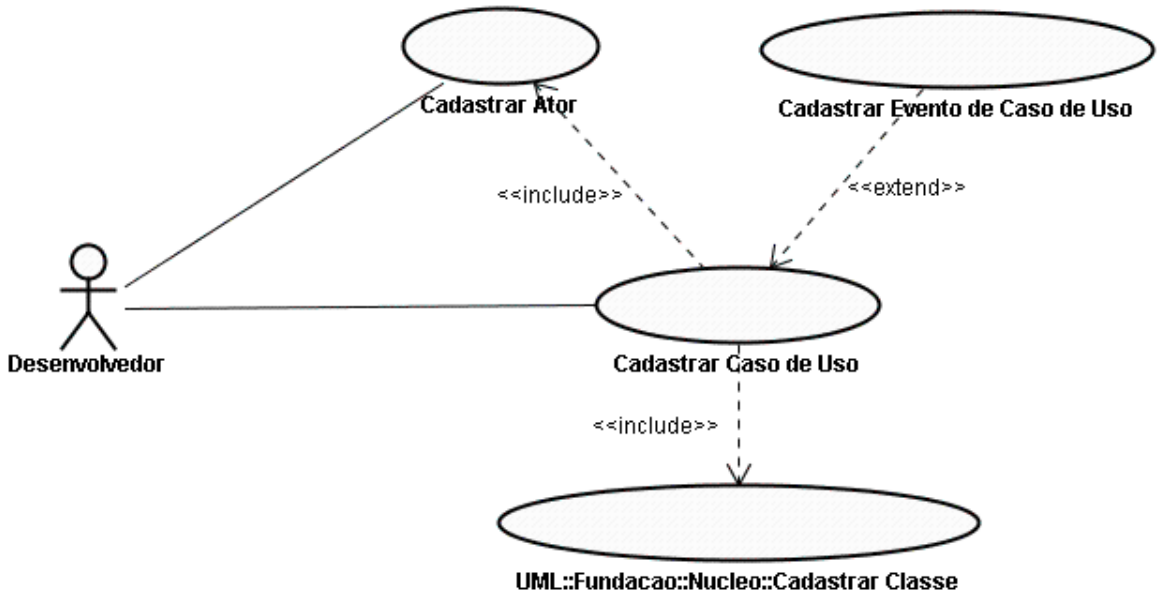


Figura 12 - Diagrama de casos de uso do pacote *UML.ElementosComportamentais.CasosUso*

- **Cadastrar Caso de Uso**: permite criar um novo caso de uso ou alterar ou excluir um caso de uso existente no projeto. Durante sua realização, é possível cadastrar classes, por meio do caso de uso *Cadastrar Classe*.
- **Cadastrar Ator**: permite criar um novo ator ou alterar ou excluir um ator existente no projeto.
- **Cadastrar Evento de Caso de Uso**: permite criar um novo evento de caso de uso para um caso de uso, alterar ou excluir um evento de caso de uso existente no caso de uso.

A Figura 13 mostra o diagrama de casos de uso referente ao sub-pacote *UML.Fundacao.Nucleo*, cujas descrições sucintas são:

- **Cadastrar Classe:** permite criar uma nova classe ou alterar ou excluir uma classe existente no projeto. Durante sua realização, é possível cadastrar atributos e operações,

por meio dos casos de uso *Cadastrar Atributo* e *Cadastrar Operação*, respectivamente.

- **Cadastrar Atributo:** permite criar um novo atributo de uma classe ou alterar ou excluir um atributo existente na classe.
- **Cadastrar Operação:** permite criar uma nova operação de uma classe ou alterar ou excluir uma operação existente na classe.

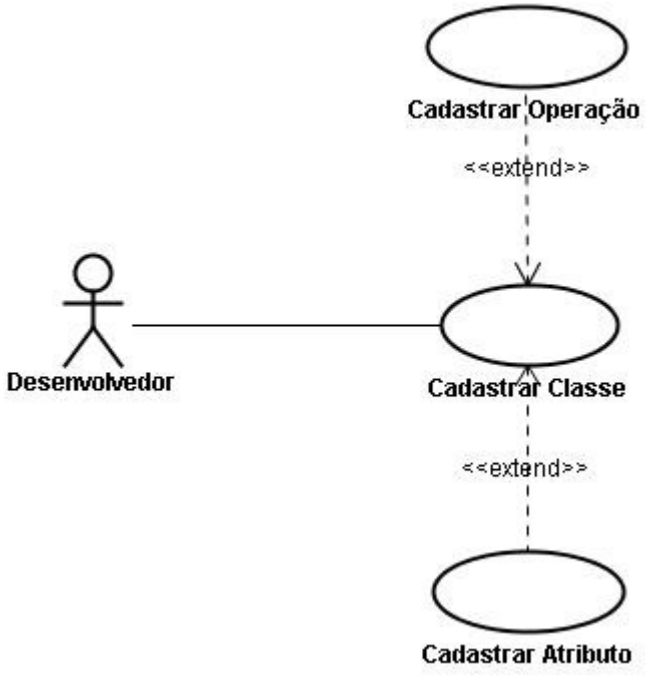


Figura 13 - Diagrama de casos de uso do pacote *UML.Fundacao.Nucleo*

A Figura 14 apresenta o diagrama de classes de parte do pacote *UML.Fundacao.Nucleo*. Esse pacote foi desenvolvido em (SILVA, 2003), estando em conformidade com o meta-modelo da UML.

A hierarquia apresentada tem como raiz a classe *Elemento* que nada mais é que um elemento básico produzido em um processo de software. Um elemento específico de modelagem de sistemas é denominado Elemento de Modelo, que, por sua vez, é classificado em Parâmetro, Propriedade, Espaço de Nome, Elemento Generalizável e Relacionamento. Neste trabalho consideramos apenas as classes *Propriedade* e *ElementoGeneralizavel*, pois em sua descendência estão as classes de nosso interesse.

Um Elemento Generalizável, como o próprio nome diz, é aquele passível de ser generalizado. Um classificador é um elemento generalizável que descreve propriedades.

Classes, atores e casos de uso são subtipos de classificadores, enquanto operações e atributos são herdeiros de *Propriedade*, sendo operação uma Propriedade Comportamental e atributo um tipo de Propriedade Estrutural.

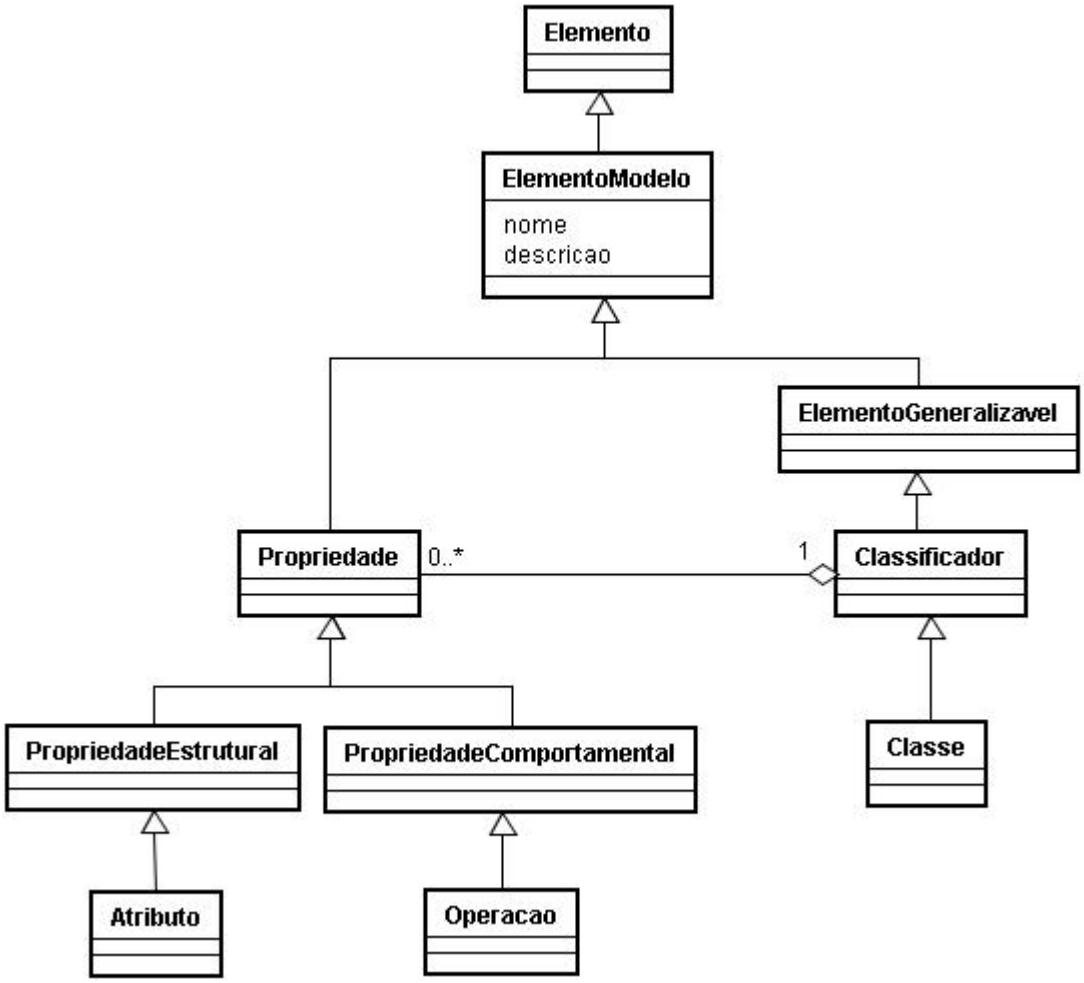


Figura 14 - Diagrama de Classes parcial do pacote *Nucleo*

O pacote *UML.ElementosComportamentais* contém a infra-estrutura da linguagem que permite especificar o comportamento dinâmico nos modelos. Dentro dele está o pacote Casos de Uso, que especifica o comportamento usando atores e casos de uso, parcialmente mostrado na Figura 15. Conforme citado anteriormente, assim como uma classe, casos de uso e atores são classificadores. Já eventos de casos de uso são tipos de elemento de modelo, herdando diretamente de *ElementoModelo*.

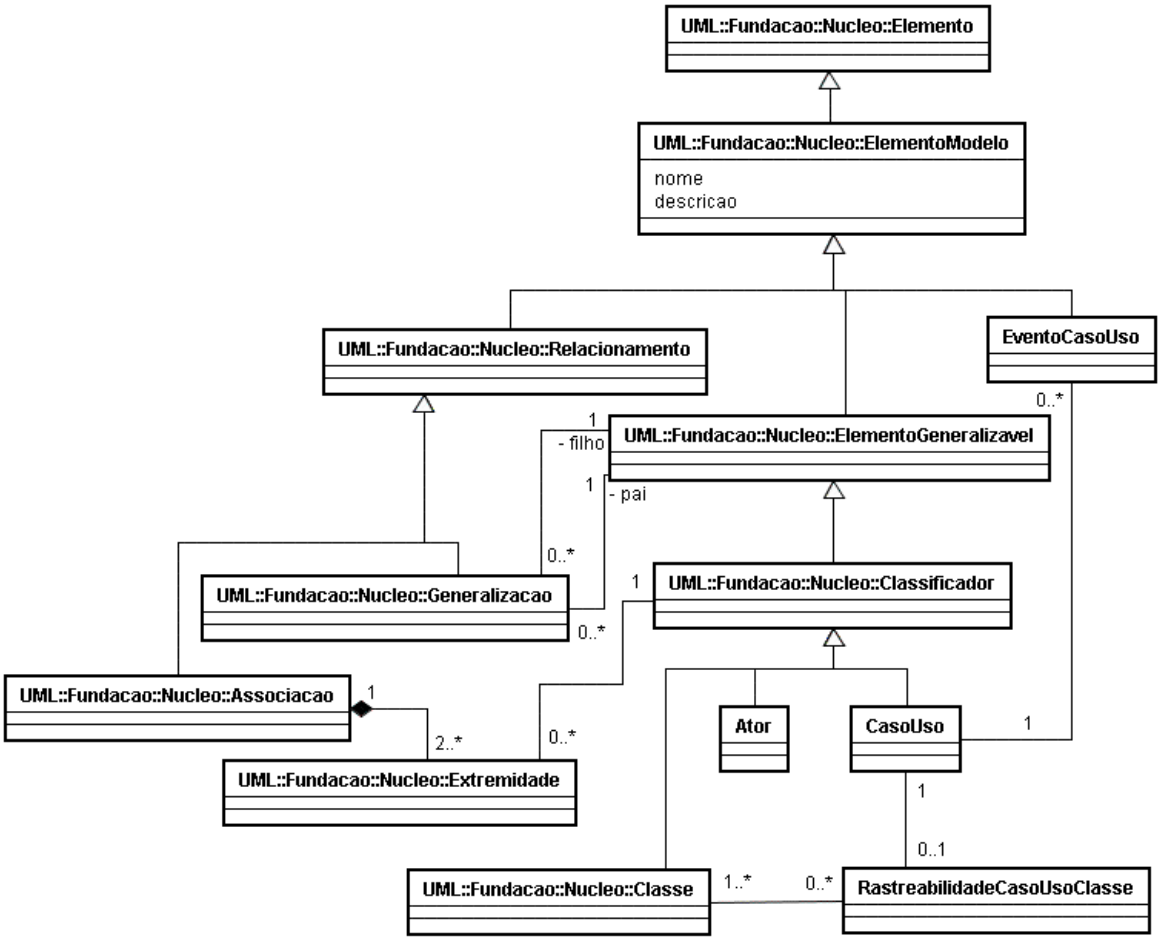


Figura 15 - Diagrama de Classes do pacote *CasoUso*

Ao longo de um processo de software diversos artefatos são gerados no contexto de um projeto. A classe *Modelo* retrata um tipo de artefato que pode agregar vários elementos de modelo. Essa classe representa quaisquer modelos que um desenvolvedor pode construir, como, por exemplo, modelos estruturados e modelos orientados a objetos. Além disso, a classe *Modelo* possui a especialização *ModeloObjetos*, que representa um modelo direcionado, unicamente, à modelagem orientada a objetos, como mostra a Figura 16.

A associação existente entre ElementoModelo e Modelo é de grande importância no contexto da ferramenta de Cadastro de Elementos de Modelo, pois é ela que permite descobrir a qual projeto um caso de uso, classe ou ator pertence, bem como é possível obter esses elementos a partir de um projeto.

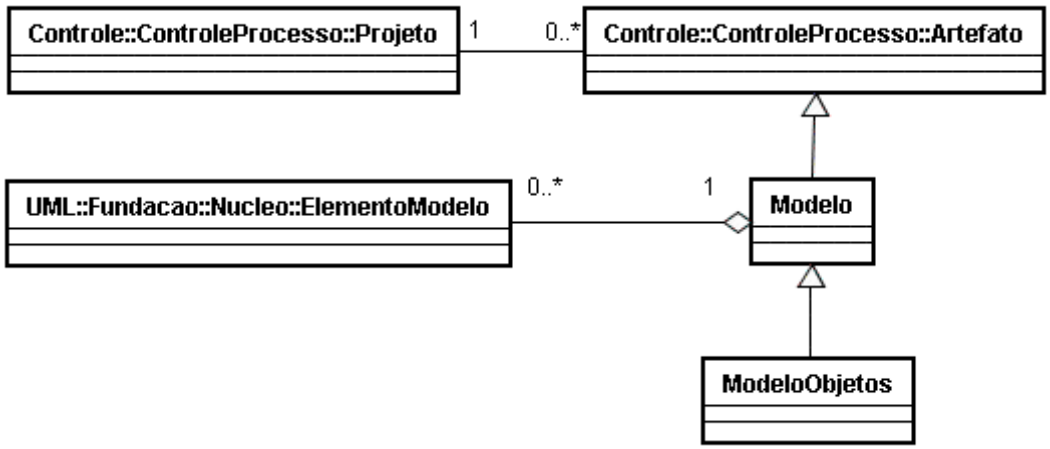


Figura 16 - Diagrama de Classes do pacote *Modelo*

1.16FERRAMENTA DE APOIO À ENGENHARIA DE REQUISITOS

A Engenharia de Requisitos é o ramo da Engenharia de Software que envolve todas as atividades relativas a desenvolver, documentar e dar manutenção ao conjunto de requisitos de um sistema. O processo de Engenharia de Requisitos possui como principais atividades (KOTONYA e SOMMERVILLE, 1998): (i) levantamento de requisitos, (ii) análise de requisitos, (iii) documentação de requisitos, (iv) verificação e validação de requisitos e (v) gerência de requisitos.

Conforme discutido no Capítulo 2, muitas informações acerca de um requisito devem ser mantidas para que esse processo seja eficiente. Dentre elas: identificador, sentença, descrição, data da criação, prioridade, estado, razões da criação, comentários gerais, dependentes, tipo primitivo, conflitos, projeto, tipo, recursos humanos (interessados e o responsável, contexto de origem, casos de uso, classes, artefatos e módulos do projeto.

Percebe-se que a Engenharia de Requisitos é um processo complexo que envolve grande quantidade de informação e, por isso, é importante prover uma ferramenta para apoiá-la. Foi, então, desenvolvida a ferramenta ReqODE que tem como objetivo apoiar a parte do processo da Engenharia de Requisitos no ambiente ODE. O diagrama de pacotes da Figura 17 mostra as dependências existentes entre os pacotes relevantes para essa ferramenta.

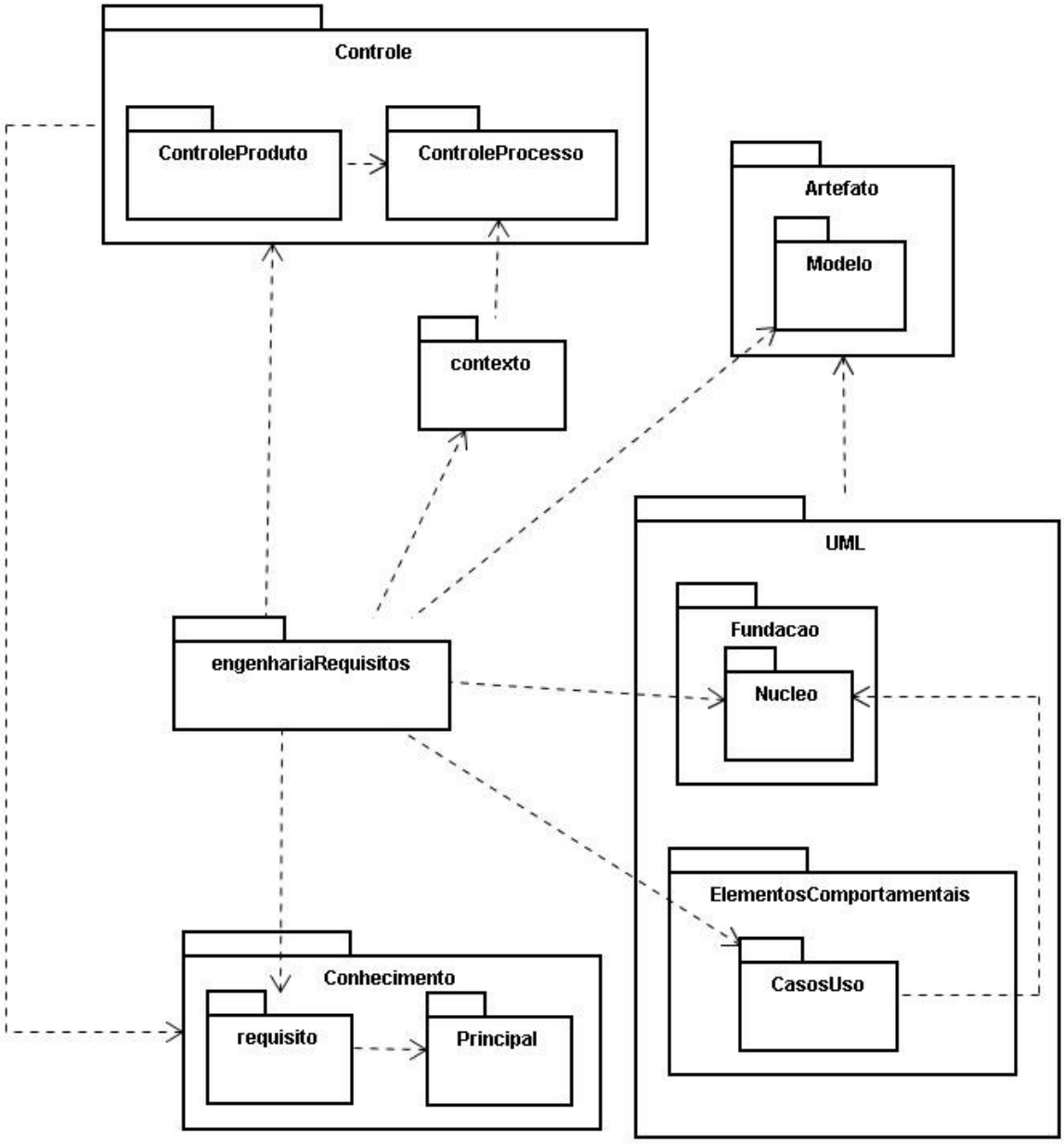


Figura 17 - Diagrama de pacotes de ReqODE

O pacote principal da ferramenta é o pacote *engenhariaRequisitos*, que contém as classes que, de fato, guardam as informações dos requisitos. Outro pacote desenvolvido para esta ferramenta é o pacote *contexto*, utilizado para registrar o contexto em que um requisito foi criado. O restante dos pacotes já existia em ODE e os pacotes *engenhariaRequisitos* e *contexto* apresentam as dependências mostradas, mantendo a integração com o ambiente e permitindo manter associações com os demais elementos já tratados no mesmo.

A Figura 18 mostra o diagrama de casos de uso referente aos pacotes *engenhariaRequisitos* e *contexto*, cujas descrições sucintas são apresentadas a seguir.

- **Cadastrar Requisitos:** permite que o analista crie um requisito, altere dados gerais de um requisito e exclua um requisito de um projeto.
- **Definir Relações de Rastreabilidade:** permite definir as ligações utilizadas para manter a rastreabilidade entre requisitos e entre requisitos e artefatos gerados ao longo do processo de software, dentre elas dependências e conflitos entre requisitos, contexto de origem, alocação a artefatos, módulos e casos de uso, e ligação com classes que tratam o requisito.
- **Gerar Relatórios de Rastreabilidade:** trata da geração de relatórios de rastreabilidade, permitindo a customização pelo usuário.
- **Rastrear Requisitos:** provê diversos tipos de busca a requisitos, tais como busca de requisitos por caso de uso, classe, módulo, contexto de origem, artefatos de origem e gerados, responsáveis e interessados de modo a garantir a rastreabilidade bidirecional e a identificação de pontos de mudança durante o processo de software.
- **Cadastrar Contexto:** permite criar, alterar, consultar e excluir contextos.

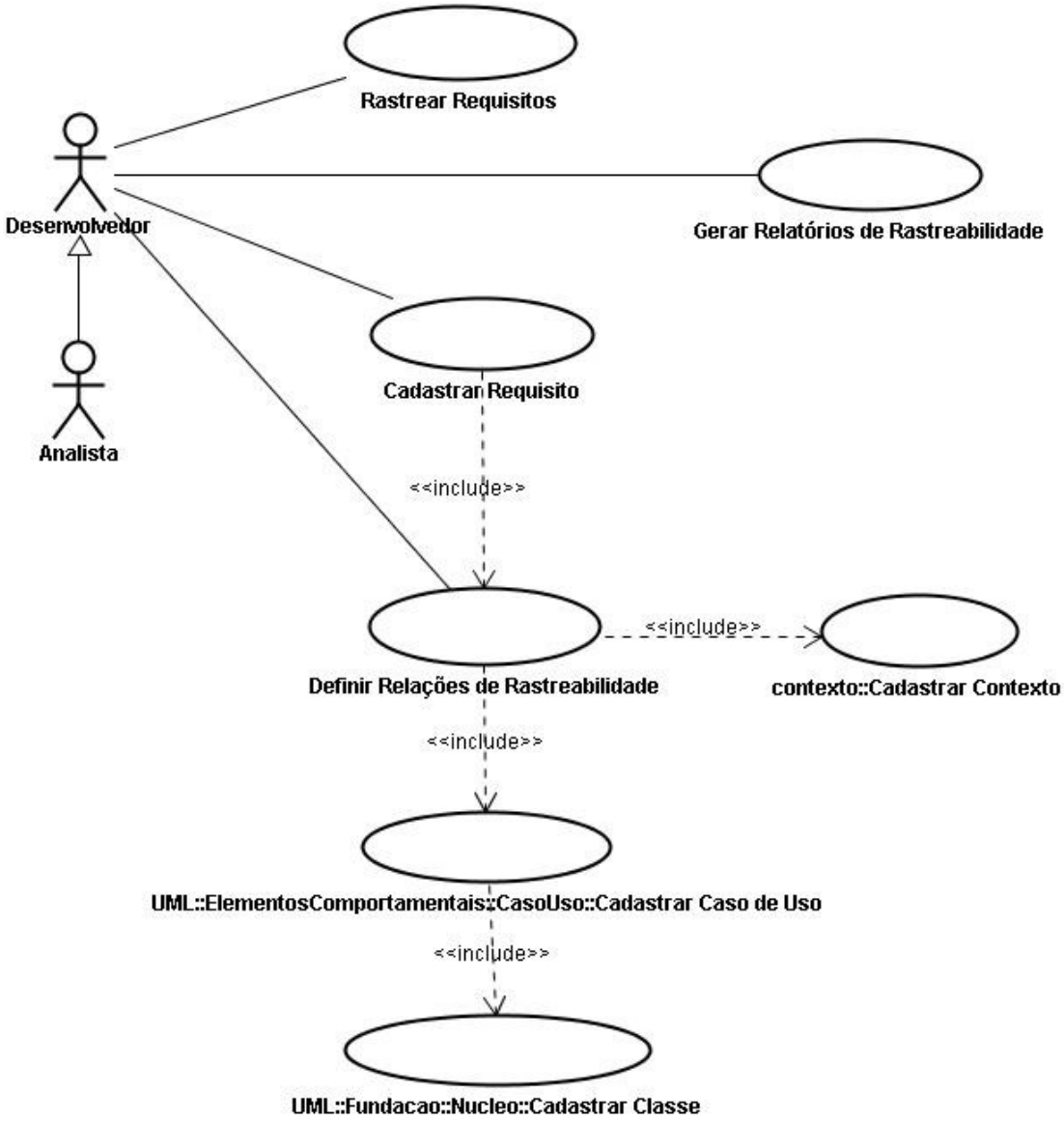


Figura 18 - Diagrama de Casos de Uso de ReqODE

No processo de engenharia de requisitos encontram-se como principais atores o analista e o desenvolvedor, como mostra a Figura 18. O desenvolvedor pode realizar os eventos dos casos de uso mais gerais, que envolvem consulta de informações sobre requisitos, geração de documentos e cadastro de contexto de origem, sendo que o analista, por herdar de desenvolvedor, também pode realizar esses eventos. Porém, além dessas atividades, o analista tem a permissão de cadastrar requisitos, associá-lo a outros elementos do desenvolvimento e pode, ainda, criar alguns desses elementos.

A Figura 19 mostra o diagrama de classe do pacote *engenhariaRequisito*. Esse diagrama aponta que requisitos podem ser decompostos em outros requisitos, bem como podem ser dependentes ou estar em conflito com outros requisitos. Visando à rastreabilidade, requisitos podem ser associados a casos de uso e artefatos produzidos no em um projeto. Além disso, o contexto no qual o mesmo foi levantado (o que inclui a atividade do processo de software, documentos analisados e participantes que atuaram em sua identificação) pode ser registrado. Por fim, para facilitar o gerenciamento do projeto, requisitos são alocados a módulos que definem o escopo do projeto e recursos humanos são indicados como responsáveis e interessados.

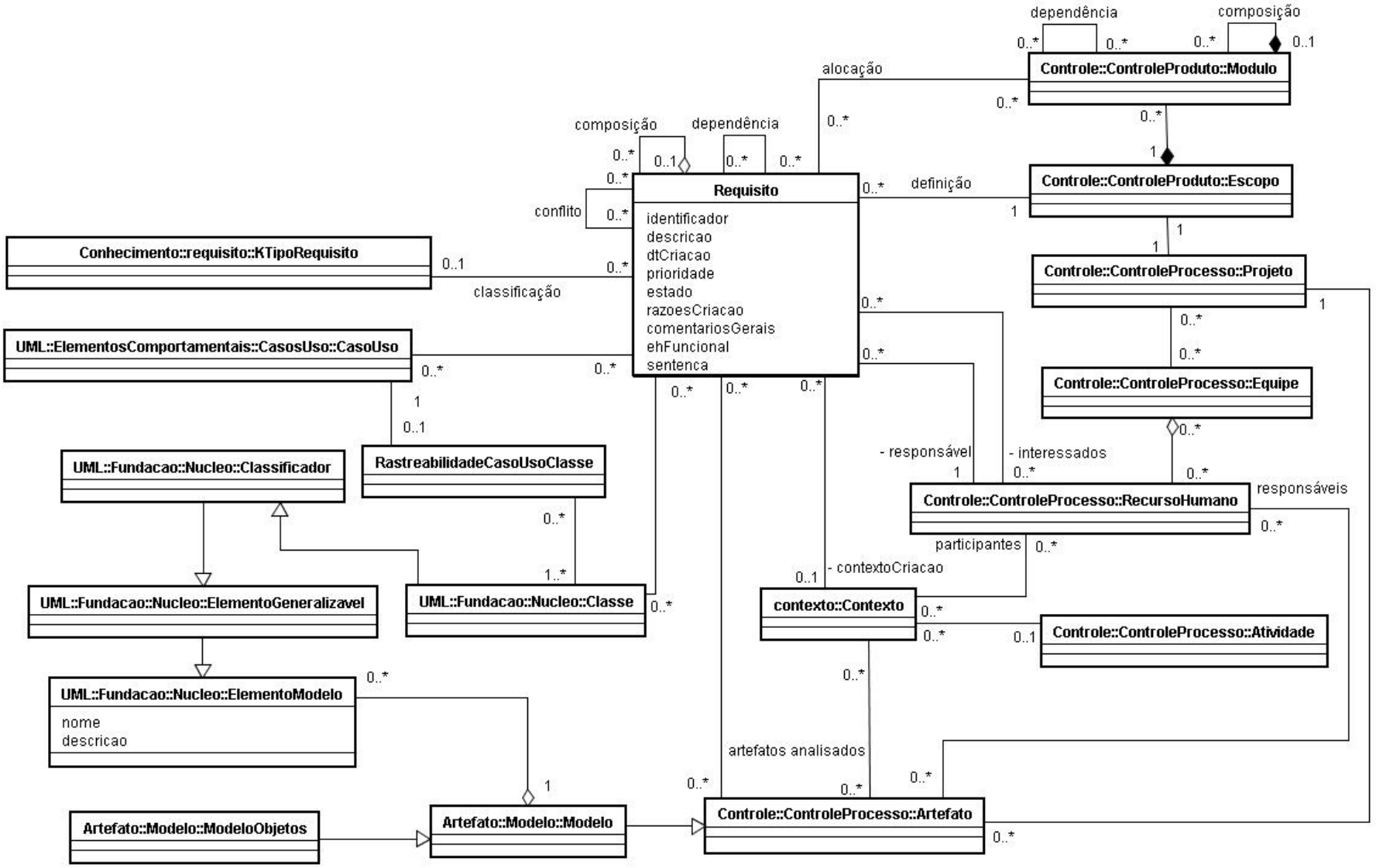


Figura 19 - Diagrama de classes do pacote *engenhariaRequisitos*

Um requisito, depois de identificado durante o levantamento ou outra etapa do processo de software, pode sofrer alterações ou pode mesmo ser rejeitado durante outras atividades, como verificação e validação, por exemplo. A Figura 20 mostra o diagrama de estados da classe Requisito.

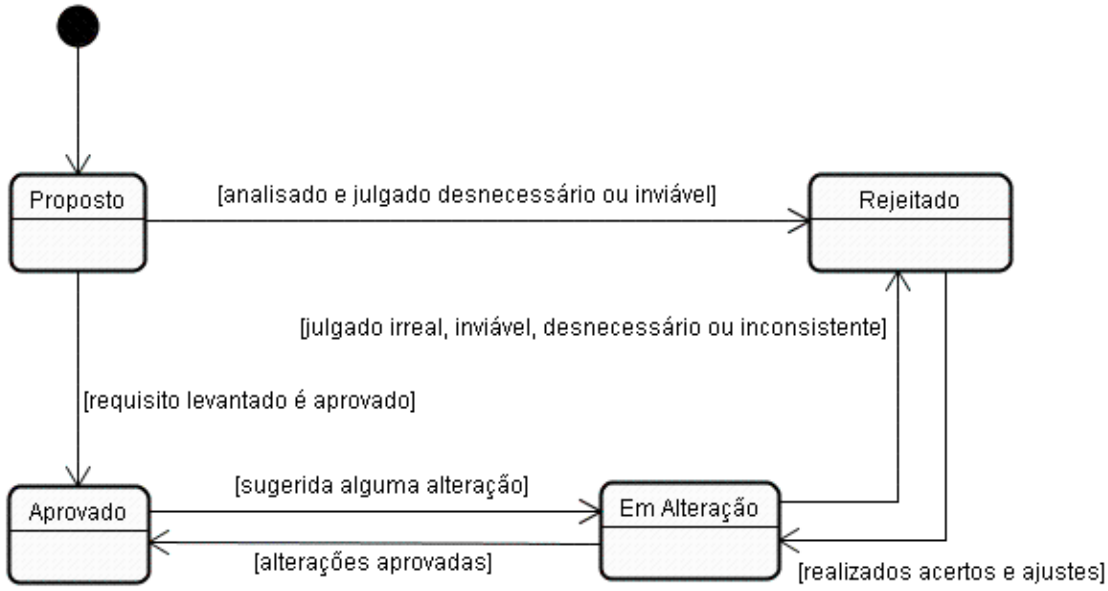


Figura 20 - Diagrama de Estados de objetos da classe *Requisito*

Em vários momentos do processo de desenvolvimento de um software, requisitos podem ser criados, alterados ou mesmo rejeitados. Inicialmente, ao ser criado, um requisito é tido como *Proposto*. Durante a etapa de análise ele pode ser *Aprovado*, ou mesmo julgado desnecessário ou inviável, quando seu estado muda para *Rejeitado*. Um requisito *Aprovado* ainda pode sofrer alterações, passando para o estado *Em Alteração*, e estas podem ser rejeitadas num processo de análise, negociação, verificação ou validação, tornando o requisito *Rejeitado*. No estado *Rejeitado* um requisito pode, ainda, ser revisto e se pode propor alterações para que ele seja avaliado novamente passando ao estado *Em Alteração*. Todas essas transições de estados são provocadas pela realização do evento de caso de uso "Alterar Dados Gerais de Requisito" do caso de uso "Cadastrar Requisitos". As condições de guarda mostradas não são, de fato, tratadas pelo sistema, cabendo ao analista analisá-las.

1.17ESPECIFICAÇÃO DE REQUISITOS NÃO FUNCIONAIS

Nas seções anteriores tratou-se dos requisitos funcionais, ou seja, aqueles que tratam das funcionalidades do sistema, não levando em consideração a tecnologia a ser utilizada na implementação. Agora, são tratados os requisitos não funcionais requeridos pelo software. Esses levam em consideração aspectos tecnológicos, juntamente com suas imperfeições, e podem determinar o sucesso ou o fracasso de um sistema, sendo o diferencial que torna um produto atrativo, usável, confiável ou rápido, sendo usados pelo cliente e usuários para julgar o produto (ROBERTSON; ROBERTSON, 1999).

Dentre os requisitos não funcionais, podemos citar (ISO, 1999):

- **Confiabilidade:** diz respeito à capacidade do software manter seu nível de desempenho, sob condições estabelecidas, por um período de tempo. Tem como sub-características: maturidade, tolerância a falhas, recuperabilidade e conformidade;
- **Usabilidade:** refere-se ao esforço necessário para se utilizar um produto de software, bem como o julgamento individual de tal uso por um conjunto de usuários. Tem como sub-características: inteligibilidade, apreensibilidade, operacionalidade, atratividade e conformidade;
- **Eficiência:** diz respeito ao relacionamento entre o nível de desempenho do software e a quantidade de recursos utilizados sob condições estabelecidas. Tem como sub-características: comportamento em relação ao tempo, comportamento em relação aos recursos e conformidade;
- **Manutenibilidade:** concerne ao esforço necessário para se fazer modificações no software. Tem como sub-características: analisabilidade, modificabilidade, estabilidade, testabilidade e conformidade;
- **Portabilidade:** refere-se à capacidade do software ser transferido de um ambiente para outro. Tem como sub-características: adaptabilidade, capacidade para ser instalado, coexistência, capacidade para substituir e conformidade.

Dentre as propriedades descritas acima, as de maior relevância para este trabalho são: Manutenibilidade, Usabilidade e Portabilidade.

4.1.1 Manutenibilidade

A manutenabilidade do sistema será dada por uma arquitetura em camadas. A estrutura em camadas provê uma estratégia de como distribuir a funcionalidade da aplicação entre classes. Além disso, arquiteturas em camadas provêem uma indicação sobre com que outros tipos de classes uma certa classe pode interagir e como esta interação acontecerá (AMBLER, 1998).

O padrão DAO (*Data Access Object*) (SUN, 2006) também possibilita a separação dos códigos que tratam as operações envolvendo bancos de dados. Desta forma, qualquer mudança na persistência não implicará em mudanças no restante do código, e sim, apenas na camada de persistência.

Além disso, utilizaram-se os modelos de nomenclatura e codificação existentes no Projeto ODE.

4.1.2Usabilidade

A interface com o usuário é o mecanismo por meio do qual se estabelece um diálogo entre o software e o ser humano. Se os fatores humanos (percepção visual, memória, raciocínio, nível de habilidade, perfil e comportamento do usuário) tiverem sido levados em consideração, o resultado obtido será um diálogo harmonioso (PRESSMAN, 2002).

No ambiente ODE muitas janelas possuem uma superclasse comum com características padronizadas. Isso mantém uma uniformidade das interfaces, tornando mais simples a utilização das ferramentas desenvolvidas.

4.1.3Portabilidade

Por ser desenvolvido em Java, o ambiente ODE apresenta a característica de ser portável a qualquer sistema operacional, mantendo as mesmas características e funcionalidades. Possui, ainda, a característica de ser multiusuário, possibilitada pelo uso do *Hibernate*, podendo ter seu banco de dados sendo acessado por máquinas de diferentes sistemas.

CAPÍTULO 5PROJETO, IMPLEMENTAÇÃO E TESTES

Após a análise é necessário projetar o sistema, adicionando as informações dos requisitos tecnológicos e não funcionais. Em seguida, deve-se construí-lo e testá-lo. Três atividades técnicas são, então, necessárias: projeto, implementação e testes (PRESSMAN, 2002).

Na fase de projeto a tecnologia é incorporada aos requisitos essenciais do usuário, projetando o que será construído na implementação (FALBO, 2003). Na implementação o código é gerado e, a partir deste, o executável que deverá ser testado, verificando-se os casos de usos, restrições de integridade e a integração com o ambiente, entre outros.

Este capítulo discute o projeto das ferramentas propostas, trata sucintamente de sua implementação e testes, e apresenta o sistema resultante dessas atividades.

1.18PROJETO DA ARQUITETURA DO SISTEMA

O Projeto da Arquitetura do Sistema consiste em mapear os requisitos essenciais em uma arquitetura técnica, introduzindo aspectos de implementação e definindo como o sistema funcionará em um ambiente operacional.

O ponto de partida para a definição da arquitetura é a organização das classes em pacotes. Desta forma, são estabelecidos níveis de abstração, que são organizados em camadas e tratados separadamente durante a fase de projeto (FALBO, 2003).

No contexto do Projeto ODE, duas formas complementares de organização são adotadas. Com a finalidade de estabelecer níveis de abstração para o sistema, primeiro as classes do projeto são organizadas em pacotes de acordo com o domínio do problema, tomando por base os modelos de análise. Complementando a organização das classes, há um outro nível de divisão em pacotes, segundo a função que as classes exercem no sistema, ou seja, segundo os seus estereótipos, mostrado na Figura 21.

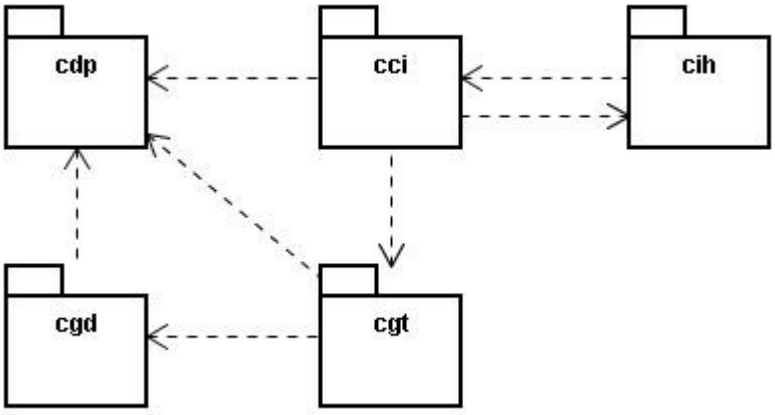


Figura 21 - Componentes de cada pacote da ferramenta

O pacote *cdp* corresponde ao Componente de Domínio do Problema, cujas classes correspondem a abstrações do domínio do problema e, portanto, são derivadas do modelo de análise.

O pacote *cgd* corresponde ao Componente de Gerência de Dados e envolve as classes relativas à persistência de dados.

O pacote *cgt* corresponde ao Componente de Gerência de Tarefas e contém as aplicações que implementam os casos de usos definidos na especificação de requisitos.

O pacote *cih* é o Componente de Interação Humana e possui as classes relativas às interfaces gráficas da ferramenta.

O pacote *cci* é o Componente de Controle de Interação cujas classes têm o papel principal de manter a comunicação entre as classes de interface (*cih*) e de aplicação (*cgt*). Suas classes fazem o devido tratamento dos retornos da aplicação para a exibição de resultados na interface, bem como informam às classes de aplicação o que o usuário solicitou por meio de uma interface.

Na próxima seção, é apresentado o projeto da ferramenta ReqODE, adicionando-se as classes essenciais das ferramentas de Cadastro de Conhecimento sobre Tipos de Requisitos e de Cadastro de Elementos de Modelo. Todos os pacotes das ferramentas apresentadas foram decompostos nesta arquitetura, porém o enfoque dado neste capítulo recai sobre o pacote principal da ferramenta ReqODE (*engenhariaRequisitos*). Os demais seguem o mesmo padrão e estão detalhados nos Anexos A (ferramenta de Cadastro de Conhecimentos sobre Requisitos) e B (ferramenta de Cadastro de Elementos de Modelo). No Anexo C encontra-se a documentação completa de projeto de ReqODE.

1.19PROJETO DE ReqODE

A Figura 22 apresenta o diagrama de pacotes de nível mais alto de ReqODE. Em relação ao modelo de análise foi introduzido o pacote *Utilitario* que contém classes que são utilizadas por diversas ferramentas do ambiente.

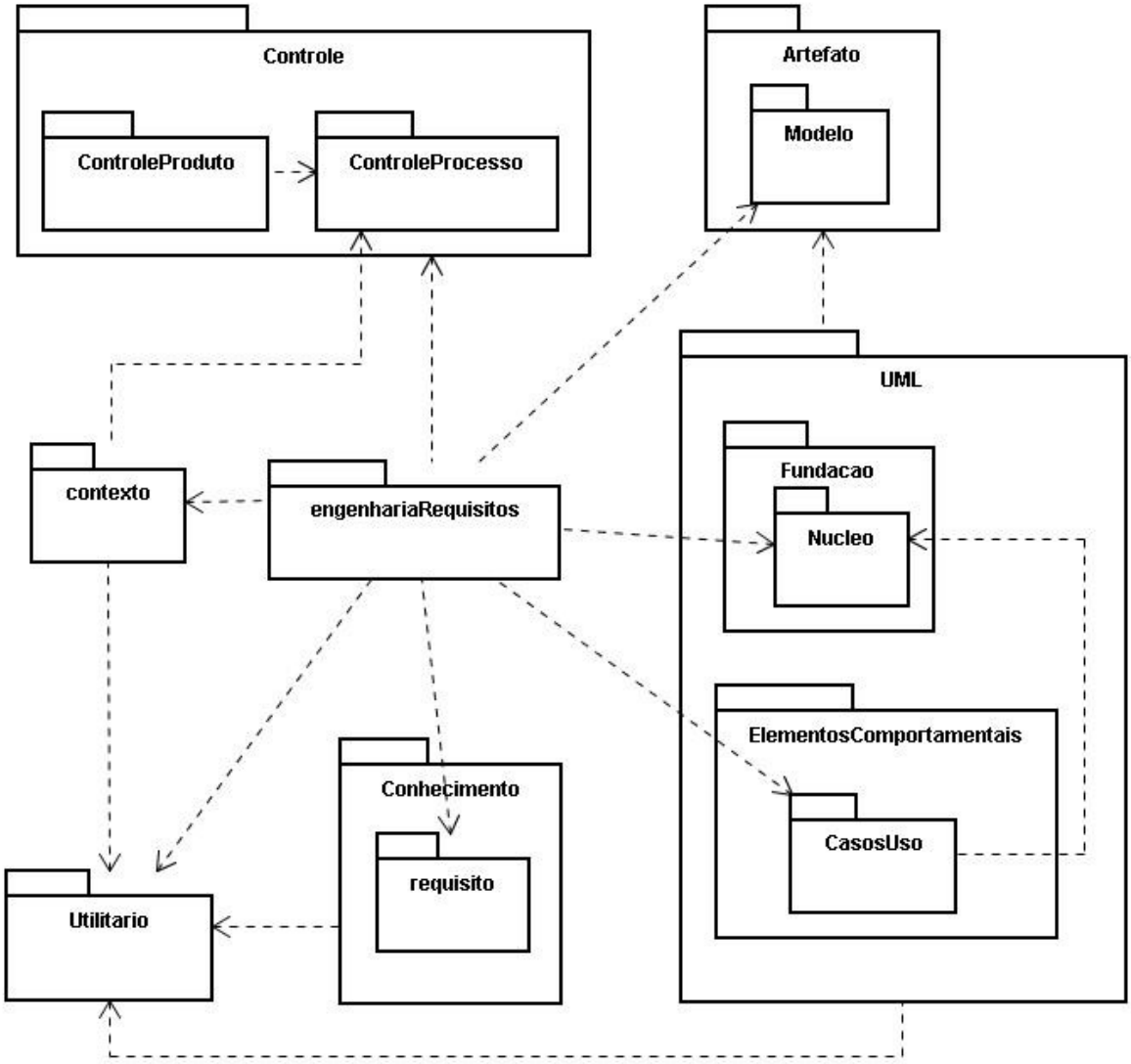


Figura 22 - Diagrama de Pacotes de ReqODE

5.1.1 Componente de Domínio do Problema (cdp)

A Figura 23 apresenta o diagrama de classes do Componente do Domínio do Problema do pacote *engenhariaRequisisto*s. Como se pode notar, comparando com o correspondente modelo da fase de análise, não houve mudanças significativas, sendo tratados apenas tipos de dados, navegabilidades e operações.

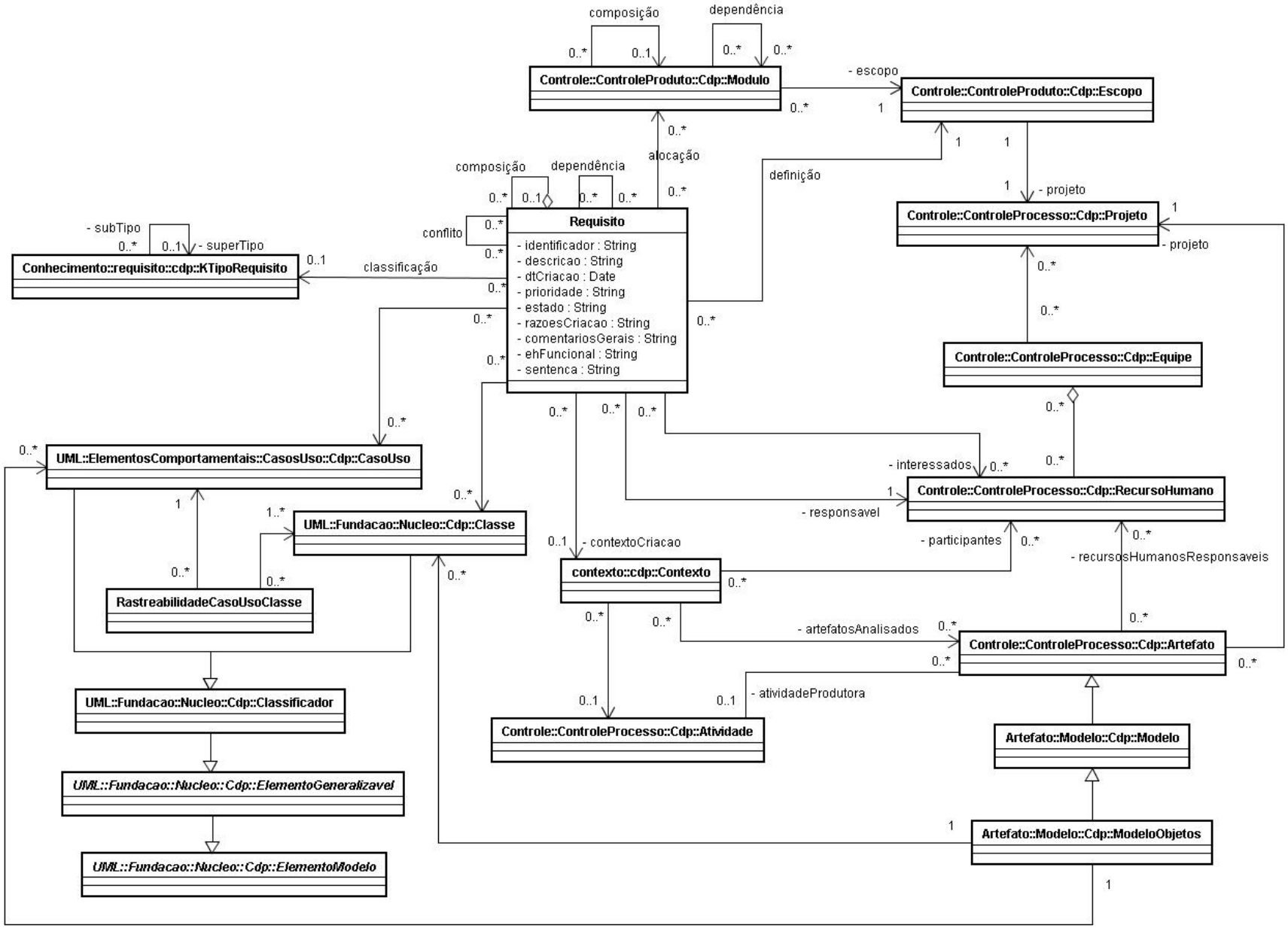


Figura 23 - Diagrama de Classes do pacote *cdp*

5.1.2Componente de Gerência de Dados (cgd)

O ambiente ODE possui uma camada de persistência de objetos, construída usando *framework* Hibernate (BAUER; KING, 2005). Nessa infra-estrutura, as classes de domínio (*cdp*) que necessitam ter informações persistidas têm no topo de sua herança a classe *ObjetoPersistente*, que se encontra no pacote *Utilitario.Persistencia.hibernate3*. Isto é importante para manter as informações necessárias para a sua persistência em bancos de dados e traz as vantagens da reutilização, como a facilidade de implementação e abstração de informações de persistência, ou seja, geração de um domínio livre de informações sobre o banco de dados. Como a infra-estrutura do ambiente é baseada no padrão DAO (*Data Access Object*) (SUN, 2006), para cada classe do domínio a ser persistida é criada uma interface DAO e uma classe de persistência que utiliza o *framework* Hibernate, como mostra a Figura 24.

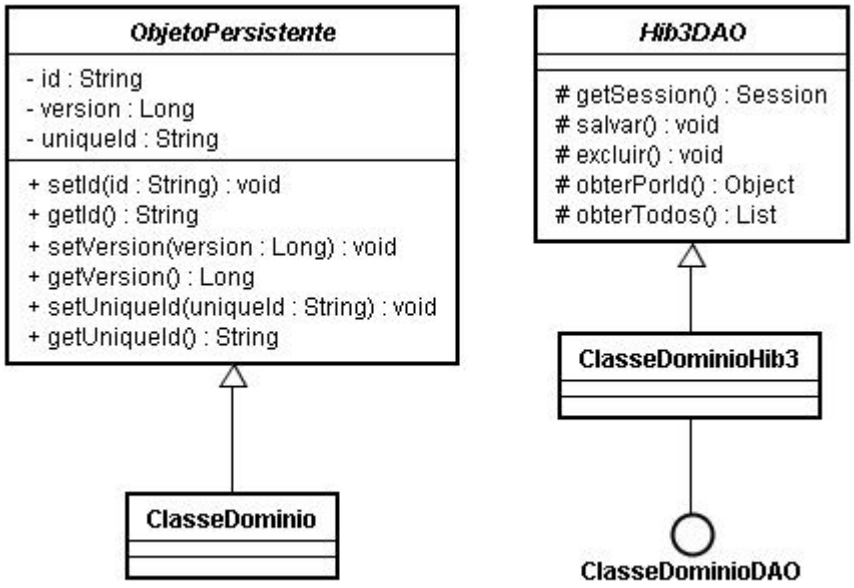


Figura 24 - Infra-estrutura de Persistência de ODE

Portanto, tomando por base a infra-estrutura de Persistência de ODE e o modelo de domínio apresentado na Seção 5.2.1, foram criadas as classes apresentadas nas Figuras 25 e 26.

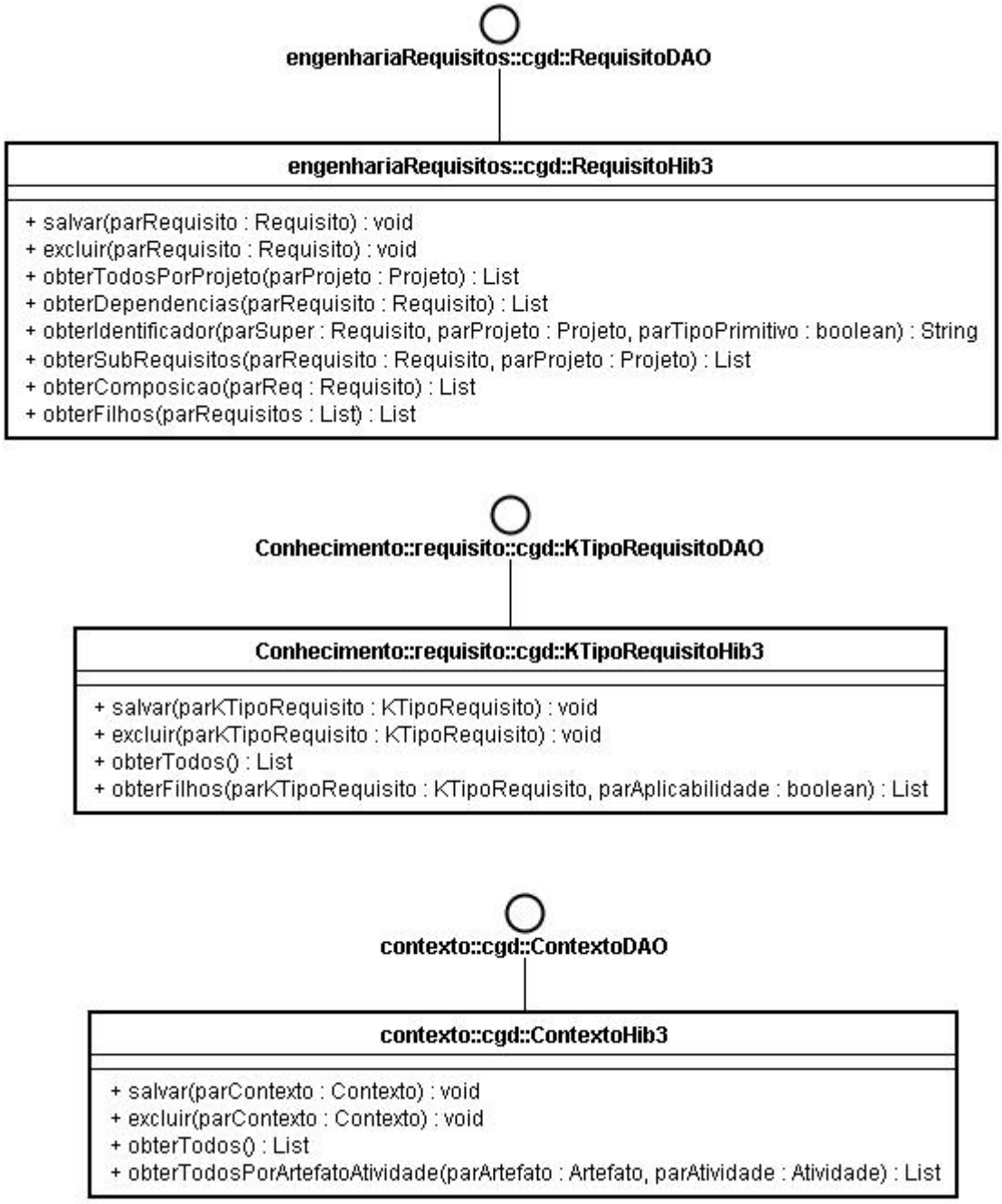


Figura 25 - Diagrama de Classes do pacote *cgd*

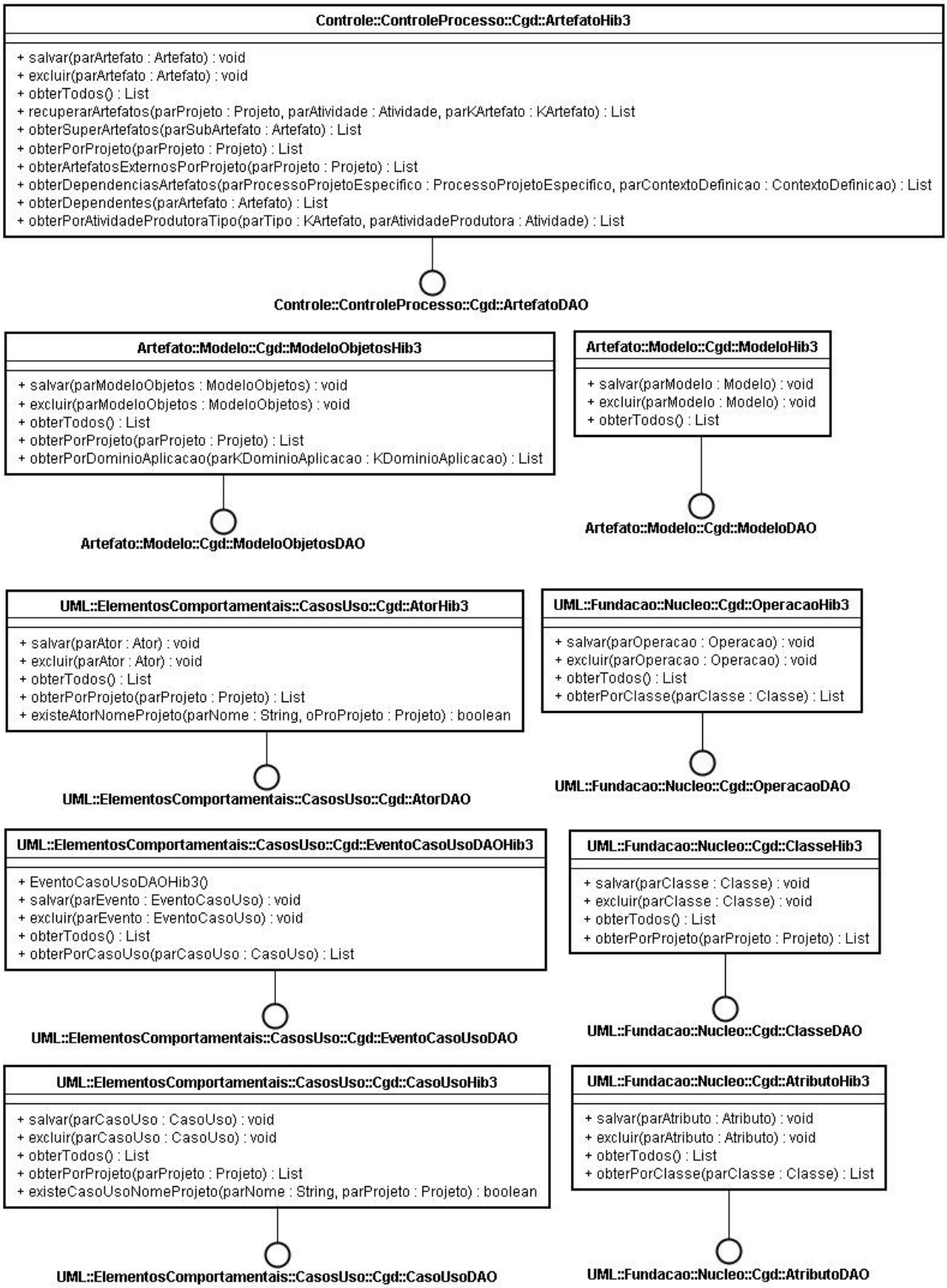


Figura 26 - Diagrama de Classes do pacote *cgd*

5.1.3Componente de Gerência de Tarefas (cgt)

Em ReqODE foram criadas duas classes de aplicação: *AplCadastrarRequisito* e *AplRastrearRequisitos*. A primeira trata os casos de uso relativos ao cadastro de informações sobre requisitos, a saber "Cadastrar Requisito" e "Definir Relações de Rastreabilidade", discutidos na seção 4.3. Já a segunda aplicação trata os casos de uso que permitem rastrear requisitos, seja por buscas ou por geração de documentos, que são os casos de uso "Gerar Relatórios de Rastreabilidade" e "Rastrear Requisitos". Os eventos de caso de uso do caso de uso "Rastrear requisitos" geraram apenas uma operação nessa classe de aplicação, a saber, "rastrearRequisitos", uma vez que é possível fazer um método genérico que abranja todas as situações levantadas na especificação de requisitos.

Como o pacote *contexto* apresenta apenas um caso de uso, foi criada apenas a classe de aplicação *AplCadastrarContexto* para tratar os eventos do caso de uso "Cadastrar Contexto".

Analogamente às aplicações de ReqODE, mostradas na Figura 27, foram criadas, ainda, as seguintes aplicações: (i) *AplCadastrarKTipoRequisito,* para tratar os eventos relativos ao caso de uso "Cadastro Tipo de Requisito", da ferramenta de Cadastro de Conhecimento sobre Tipos de Requisitos, (ii) *AplCadastrarClasse, AplCadastrarAtributos e AplCadastrarOperacoes* para tratar, respectivamente, os casos de uso "Cadastrar Classe", "Cadastrar Atributo" e "Cadastrar Operação", da ferramenta de Cadastro de Elementos de Modelo e (iii) *AplCadastrarCasoUso, AplCadastrarEventoCasoUso e AplCadastrarAtor* para tratar, respectivamente, os casos de uso "Cadastrar Caso de Uso", "Cadastrar Evento de Caso de Uso" e "Cadastrar Ator", da ferramenta de Cadastro de Elementos de Modelo. O item (i) pode ser verificado no anexo A e os demais no anexo B.

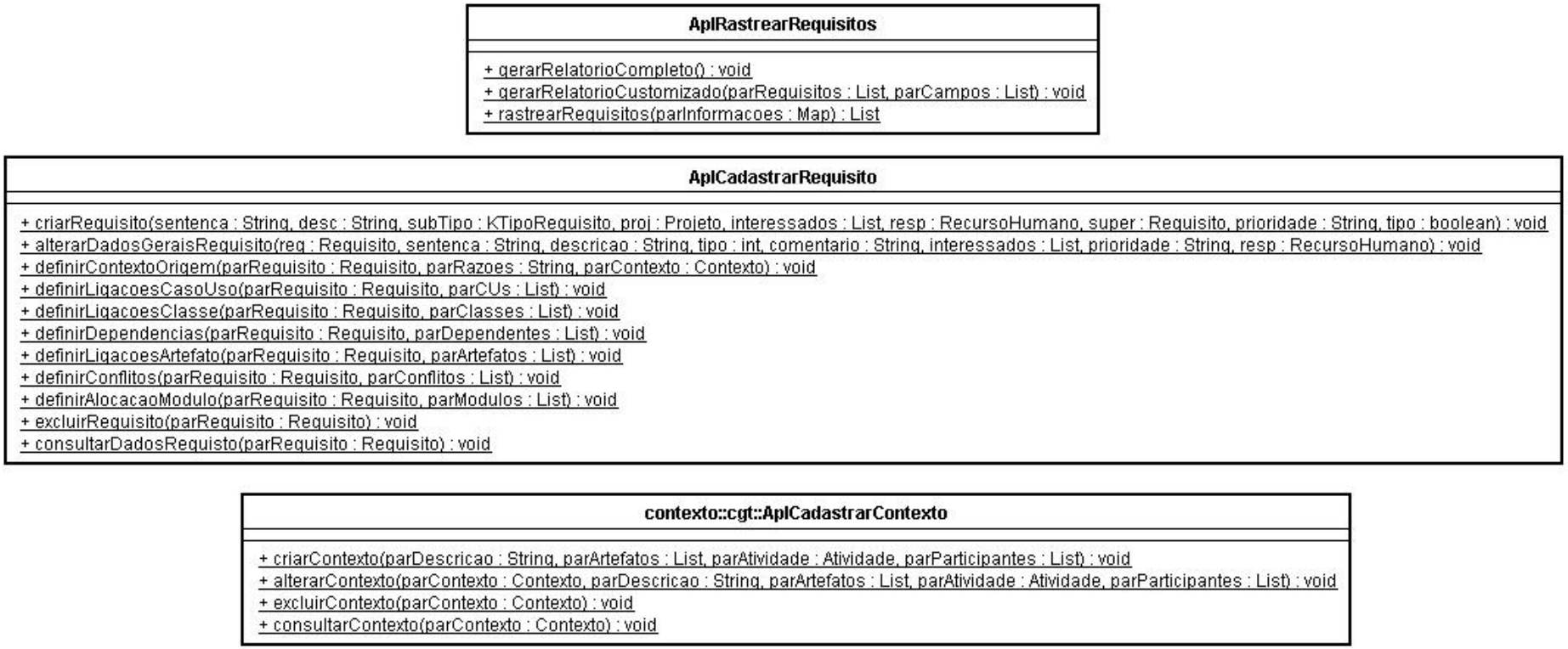


Figura 27 - Diagrama de Classes do pacote *cgt*

5.1.4Componente de Interação Humana (cih)

A Figura 28 apresenta o diagrama de classes do Componente Interação Humana de ReqODE.

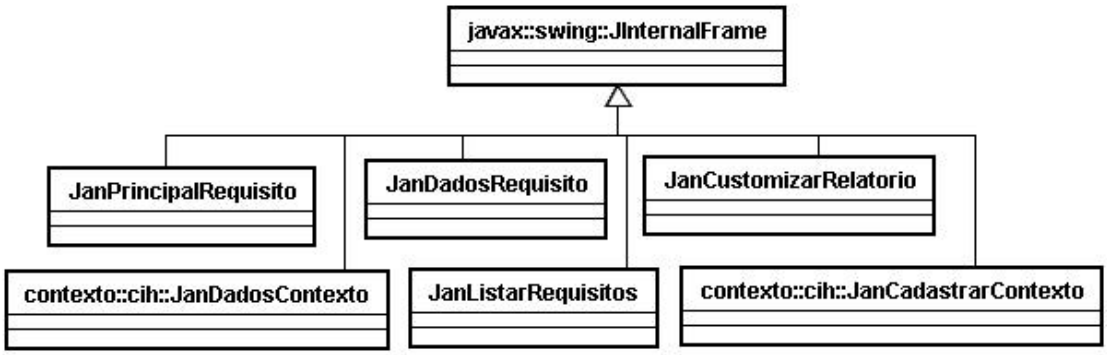


Figura 28 - Diagrama de Classes do pacote *cih*

A janela de principal da ferramenta (*JanPrincipalRequisito*) exibe em seu painel principal os requisitos cadastrados no sistema e disponibiliza as funcionalidades de criar um novo requisito, alterar, definir ligações de rastreabilidade, visualizar ou excluir um requisito previamente selecionado. Na criação de um requisito são informados alguns dados básicos, por meio da janela *JanDadosRequisito,* ilustrada na Figura 29.

Na barra de menu da janela principal, assim como no painel principal, é possível acessar a Ferramenta de Cadastro de Elementos de Modelo e a Ferramenta de Modelagem OODE, para cadastrar ou modelar casos de uso, classe e atores. Pelo menu é possível, ainda, gerar relatórios de rastreabilidade completos e visualizar a janela de criação de relatórios customizados (*JanCustomizarRelatório*), onde é possível escolher os itens a serem apresentados no documento. Existe, ainda, um menu para acessar a janela de buscas de requisitos (*JanRastrearRequisitos*) que possibilita rastrear os requisitos de acordo com filtros estabelecidos pelo usuário. Os resultados dessas buscas são exibidos na janela *JanListarRequisitos*.

A janela de cadastro de contexto (*JanCadastroContexto*) exibe contextos cadastrados no sistema e disponibiliza as funcionalidades de criar um novo contexto, alterar, visualizar ou excluir um contexto previamente selecionado.

Para visualizar ou alterar os dados de um contexto, ou mesmo inserir informações de um novo contexto, a janela de dados *JanDadosContexto* é utilizada. Ela é exibida sempre que

as opções de cadastro forem solicitadas na janela de cadastro de contexto (*JanCadastroContexto*).

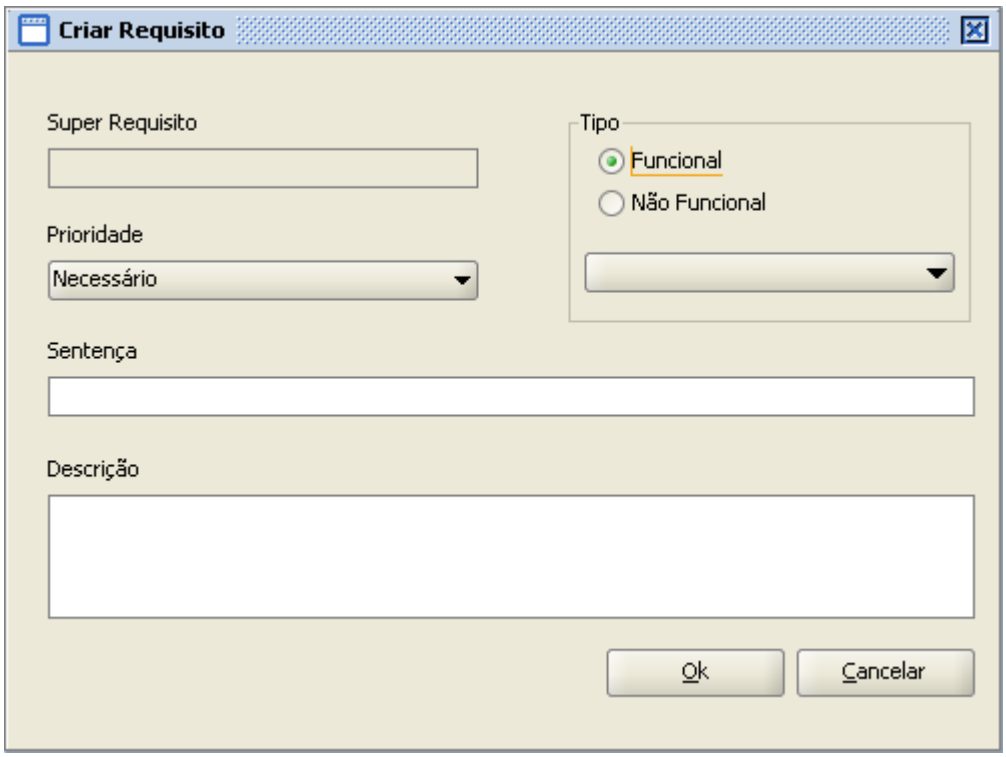


Figura 29 - Layout de *JanDadosRequisito*

Analogamente às classes de interface de ReqODE, mostradas na Figura 28, foram criadas as classes de interface para os pacotes das demais ferramentas, *Conhecimento.requisito* e *UML*, cujos diagramas de classes são apresentados nos anexos A e B, respectivamente.

5.1.5Componente de Controle de Interação (cci)

A Figura 30 apresenta o diagrama de classes do Componente de Controle de Interação da ferramenta ReqODE.

A classe *CtrlCadastrarRequisitos* tem o papel principal de fazer a comunicação entre os objetos das camadas de aplicação (*cgt*) e de interface (*cih*) desta ferramenta, isolando as interfaces. Isso facilita alterações, já que as interfaces não têm acesso direto a métodos da aplicação.

A classe *CtrlCadastrarRequisitos* tem seus relacionamentos com navegabilidade dupla com as janelas do *cih*, a saber *JanDadosRequisito*, *JanPrincipalRequisito*, *JanRastrearRequisitos*, *JanListarRequisitos* e *JanCustomizarRelatorio*. Desta forma, esse

controlador pode exibir ou atualizar as janelas nos momentos devidos e as janelas podem invocar métodos do controlador para chamar outra janela ou para solicitar a realização de operações implementadas nas aplicações *AplCadastrarRequisito* e *AplRastrearRequisitos*.

Note que a janela principal de requisitos (*JanPrincipalRequisito*) tem relacionamento com os controladores *CtrlCadastrarClasse* e *CtrlCadastrarCasoUso*, ambos do pacote *UML*, *e CtrlCadastrarContexto*, do pacote *contexto*. Isso é vantajoso, pois não é necessário adicionar código de cadastro de classes, casos de uso e contextos no controlador de cadastro de requisitos (*CtrlCadastrarRequisito*). Assim, caso seja necessário cadastrar quaisquer um desses elementos, a janela principal passa o controle para o controlador correspondente, que exibe as janelas necessárias e executa os devidos métodos da aplicação correspondente.

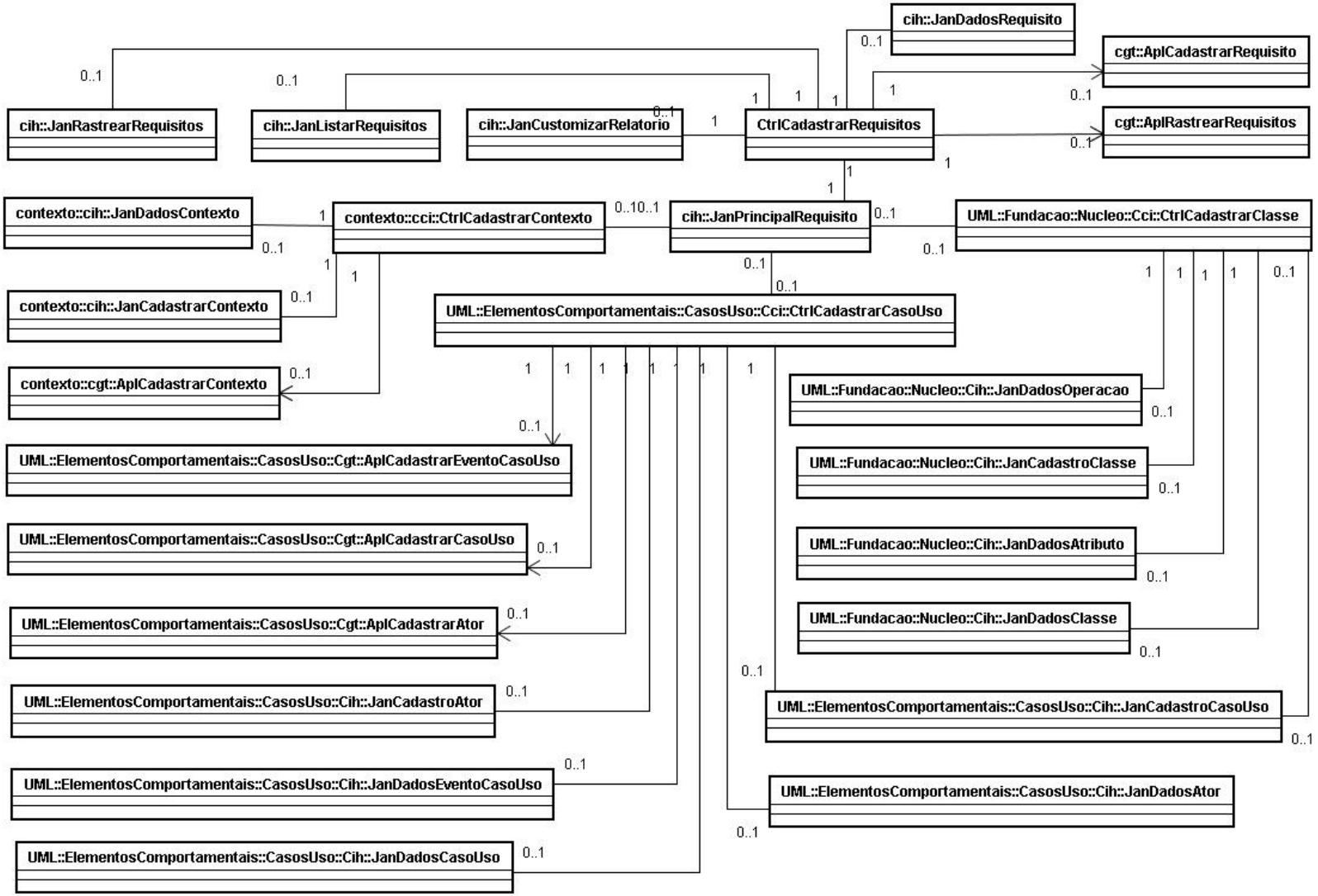


Figura 30 - Diagrama de Classes do pacote cih

1.20IMPLEMENTAÇÃO

Tomando como base o modelo de projeto, as classes precisam ser implementadas, para então produzir os arquivos fontes que resultarão em um executável. Pode-se dizer, portanto, que a fase de implementação corresponde à codificação dos módulos projetados.

Este trabalho segue os mesmos padrões de implementação utilizados no projeto ODE. Neste, utiliza-se a linguagem de programação Java, o *framework* de mapeamento objeto-relacional Hibernate (BAUER, 2005) e a ferramenta XDoclet (XDOCLET, 2007) para agilizar e facilitar o mapeamento do modelo em tabelas. Ainda, para a criação de relatórios, são utilizadas as ferramentas IReport (IREPORT, 2007) e JasperReports (JASPERREPORTS, 2007).

Iniciou-se a implementação das ferramentas pela Ferramenta de Cadastro de Conhecimento sobre Tipos de Requisitos. Após isso, partiu-se para a parte inicial de desenvolvimento de ReqODE, que consistiu em prover a funcionalidade de cadastro de um requisito, que inicialmente permitia classificação e associação com recursos humanos interessados e responsáveis.

A primeira ligação de rastreabilidade implementada foi entre os próprios requisitos, ou seja, as ligações de dependência e conflito. Tendo essa funcionalidade definida, partiu-se para a rastreabilidade em relação a outros elementos do desenvolvimento. Iniciou-se com casos de uso e artefatos, visto que já existiam neste momento ferramentas de cadastro para os mesmos no ambiente ODE, a saber OODE e Ferramenta de Planejamento de Documentação, respectivamente. Porém, a respeito de casos de uso, foi necessário criar um cadastro que não incluísse a modelagem do mesmo, Assim iniciou-se a construção da ferramenta de Cadastro de Elementos de Modelo, centrada, inicialmente, no cadastro de casos de uso e atores.

Tendo já relações de rastreabilidade importantes, implementou-se a geração de relatórios. O foco passou a ser o aperfeiçoamento da rastreabilidade vertical, criando-se o cadastro de contexto e viabilizando uma forma de associar requisitos a ele. Além disso, acrescentou-se a possibilidade de associar requisitos a módulos de um projeto. Faltava, ainda, associar requisitos a classes. De modo similar a casos de uso e atores, apesar de já existir no ambiente um cadastro de classe em OODE, para não obrigar a modelagem dessas, criou-se o cadastro de classes, criando também uma forma de associá-las a requisitos.

Neste ponto, com as possíveis ligações de rastreabilidade no ambiente estabelecidas, buscou-se desenvolver as consultas que permitissem identificar os requisitos associados a

quaisquer elementos e completou-se o relatório com as novas informações, bem como foi disponibilizada a funcionalidade de criar relatórios customizados.

1.21TESTES

Para assegurar a qualidade do produto de software sendo gerado, faz-se necessária a atividade de testes. Seu objetivo é identificar erros gerados durante as diversas fases do processo de desenvolvimento, através de um número de casos de teste que avaliem diferentes aspectos de cada módulo sendo testado.

Para tanto, os testes efetuados foram basicamente orientados a caso de uso, ou seja, à medida que se implementava um caso de uso, os testes iam sendo efetuados para avaliar elementos do domínio do problema, elementos de interface e a execução dos eventos dos casos de uso.

Paralelamente, foram realizados testes unitários e testes para verificação de persistência e acesso concorrente, isso por que a infra-estrutura de persistência ainda era nova e apresentava problemas durante o desenvolvimento deste trabalho.

Pelo fato de grande parte das funcionalidades de ReqODE terem como foco a associação de um Requisito a outro elemento do desenvolvimento cadastrado no ambiente, foi necessário testar a integração de ReqODE com as demais ferramentas do ambiente ODE, verificando se estavam sendo respeitadas as restrições impostas pelas ferramentas envolvidas. Assim, foi necessário verificar se estavam sendo tratadas as restrições de integridade do banco de dados ao alterar ou excluir um elemento, quando associado a um requisito. Da mesma forma, foi necessário verificar se os elementos disponibilizados para associação com requisitos realmente pertenciam ou estavam alocados ao projeto em questão. Com respeito aos elementos de modelo, foi necessário verificar a integração com a ferramenta OODE, para que os elementos criados na ferramenta pudessem ser importados para os diagramas durante a modelagem. Esses testes de integração foram sendo realizados à medida que as funcionalidades iam sendo implementadas, seguindo a ordem descrita na Seção 5.3.

1.22APRESENTAÇÃO DO SISTEMA

Nesta seção é apresentada a ferramenta ReqODE. Para isso, são mostrados os passos principais para a utilização da mesma.

Na janela principal da ferramenta (*JanPrincipalRequisitos*), mostrada na Figura 31, à esquerda temos o painel no qual são exibidos os requisitos de um projeto. Ao selecionar um desses requisitos, o painel à direita é exibido com os dados do item selecionado. Tornam-se disponíveis as funcionalidades de definição de rastreabilidade, sendo que cada aba trata um evento do caso de uso “Definir Ligações de Rastreabilidade”. No exemplo da Figura 31, destacamos a aba “Casos de Uso”, mostrando a seguir como ela é acionada.

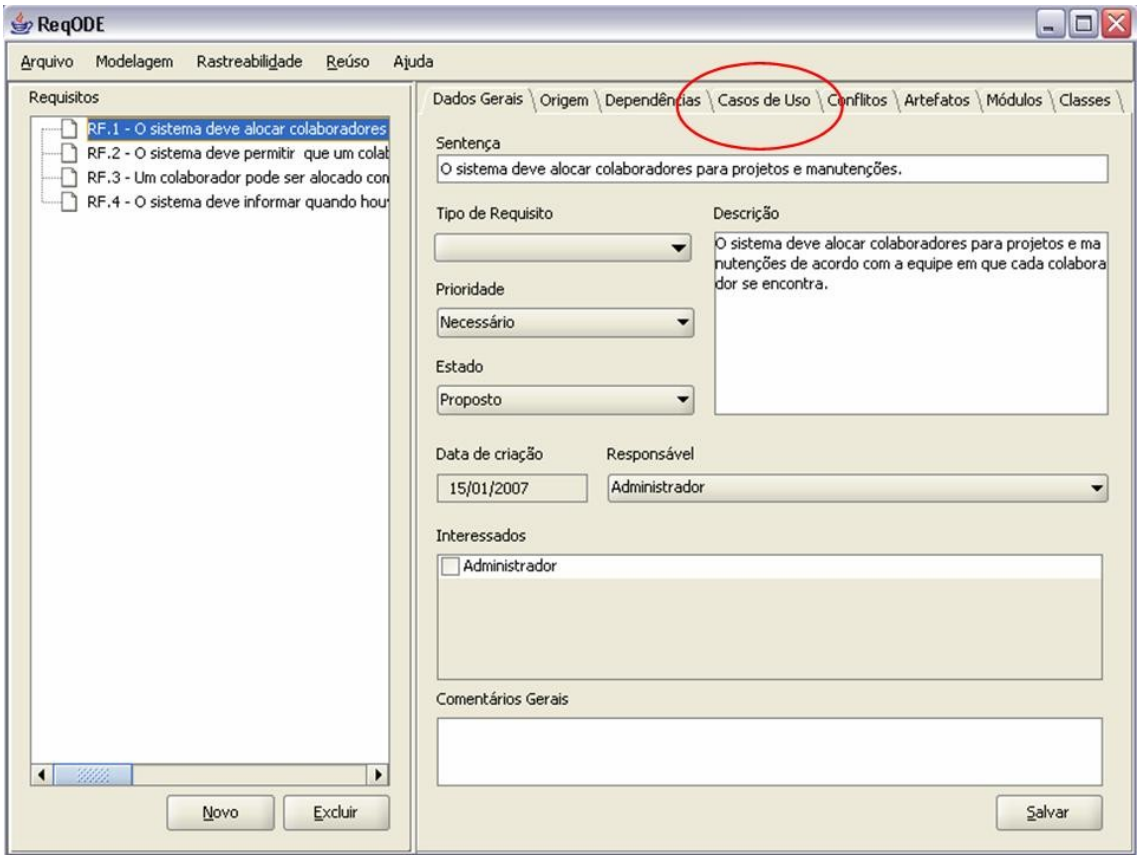


Figura 31 - Janela Principal de ReqODE

A Figura 32 apresenta a aba "Caso de Uso". Note que neste exemplo não há casos de uso cadastrados. A ferramenta de Cadastro de Elementos de Modelo pode ser acessada pela barra de menu da janela principal ou pelos botões na própria aba, como destacado na ilustração. Vale lembrar que é possível também utilizar a ferramenta de modelagem OODE para realizar os cadastros e que esta é acionada através do menu "Cadastros" (submenu OODE) ou pelo botão "Modelagem" da aba "Casos de Uso".

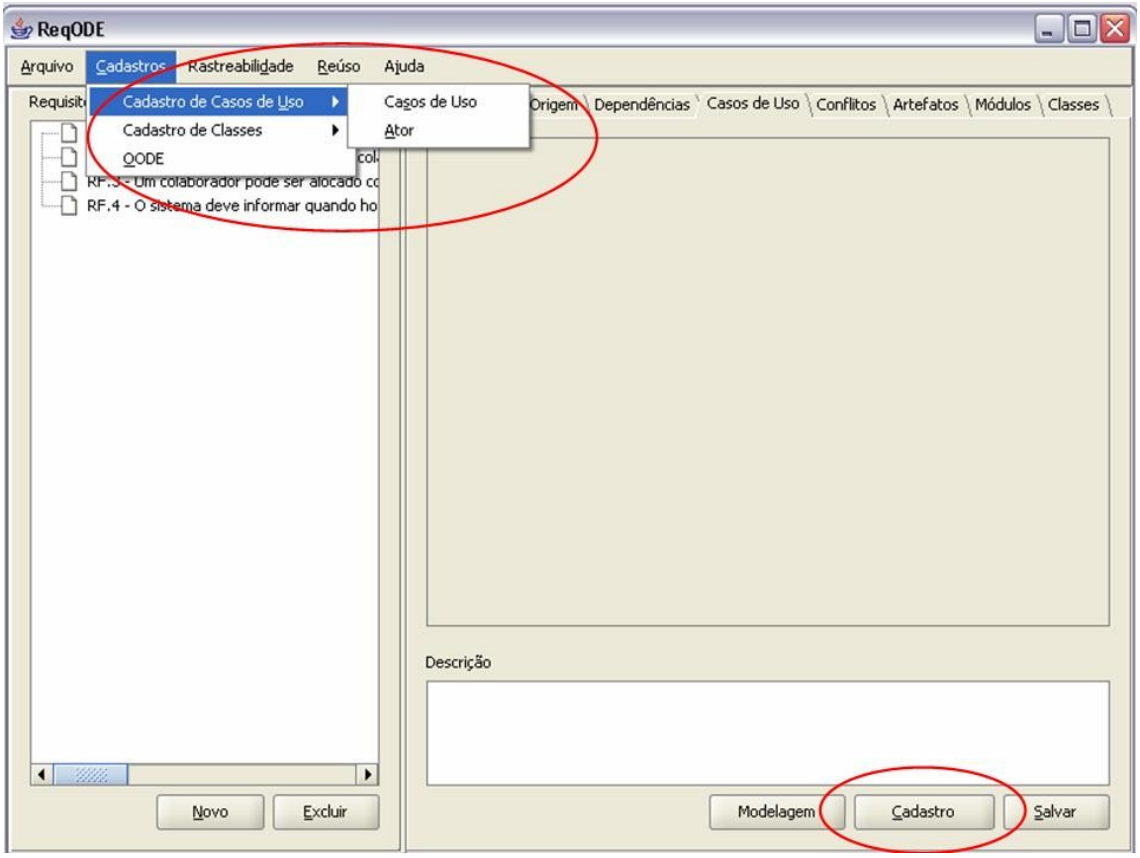


Figura 32 - Definição de rastreabilidade Requisito - Caso de Uso

A Figura 33 corresponde à janela de cadastro de Casos de Uso (JanCadastroCasoUso). Para criar um novo caso de uso, basta clicar no botão “Incluir” e a janela de dados (JanDadosCasoUso), apresentada na Figura 34, aparece para a entrada com os dados do novo caso de uso. Para salvá-lo, basta clicar no botão “Ok”.

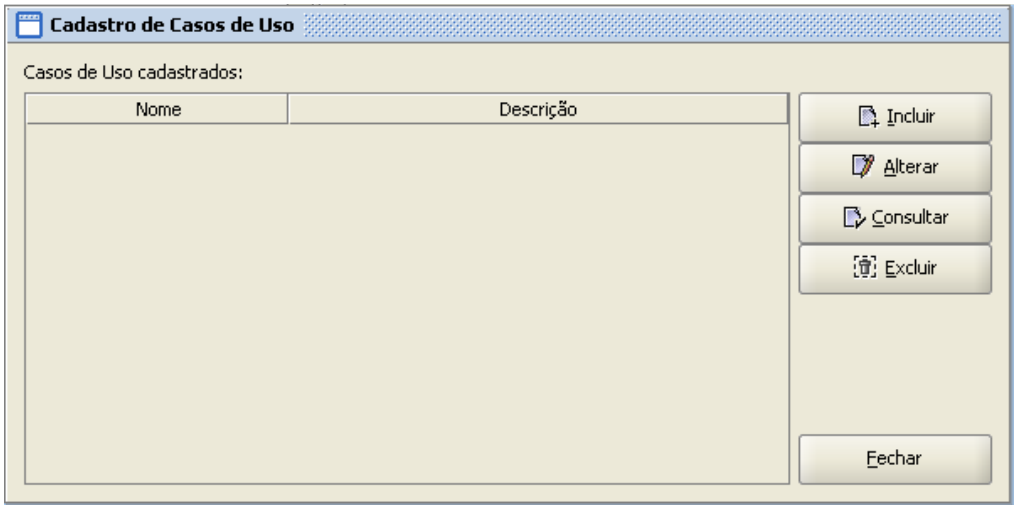


Figura 33 - Janela de Cadastro de Casos de Uso

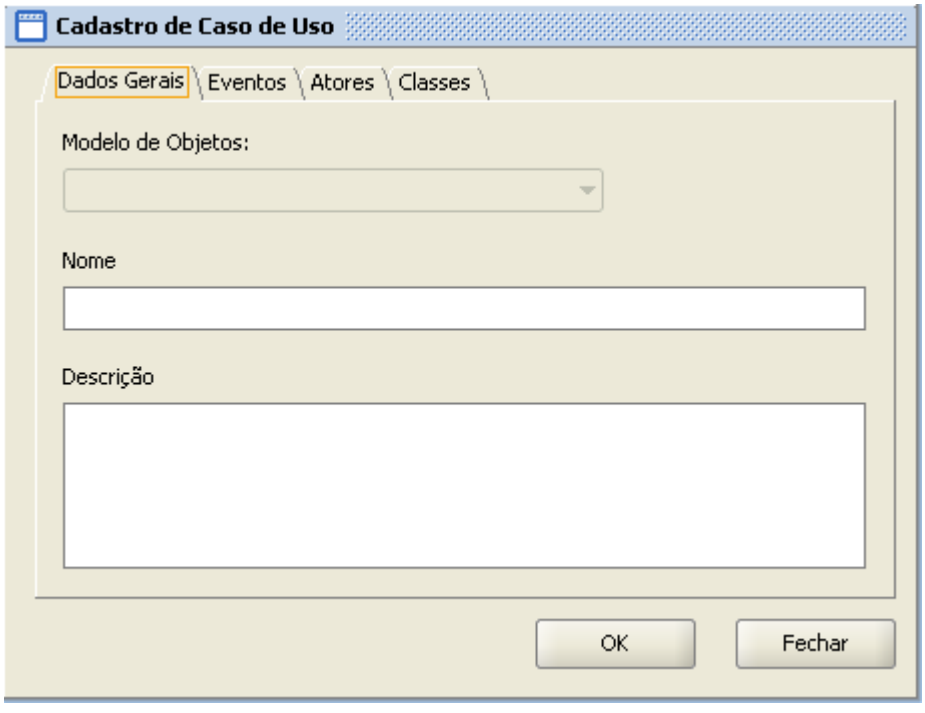


Figura 34 - Janela de Dados de Casos de Uso

Tendo sido cadastrados os casos de uso, eles aparecem na janela de cadastro, como mostra a Figura 35. Se for necessário alterar ou excluir algum desses itens cadastrados, basta selecioná-lo e clicar no botão correspondente. Para voltar a trabalhar em ReqODE, basta fechar a janela de cadastro de casos de uso.

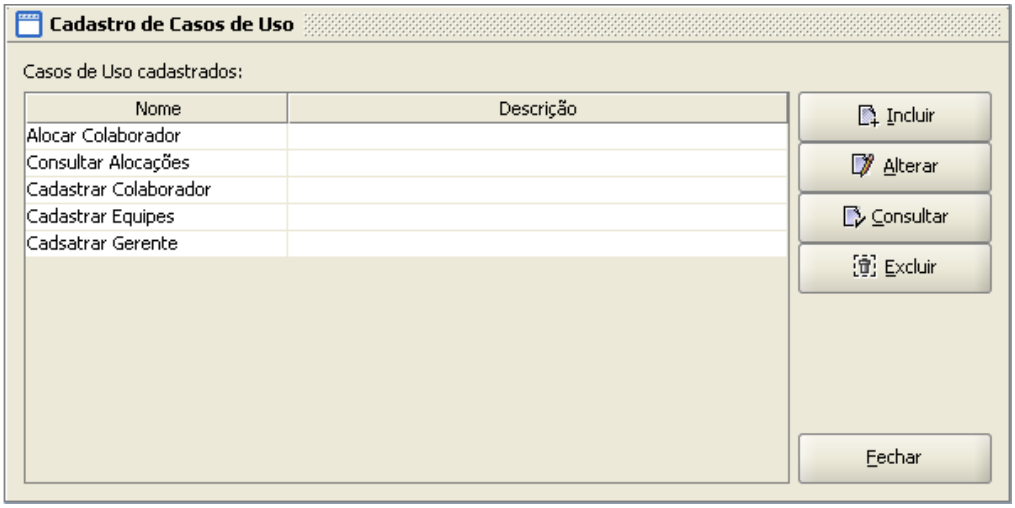


Figura 35 - Janela de Cadastro de Casos de Uso

Com os casos de uso cadastrados, basta selecionar aqueles que modelam o requisito em questão. Clicando-se no botão “Salvar” está definida a relação entre o requisito e os casos de uso selecionados, como ilustra a Figura 36.

De maneira similar a esse procedimento, também são implementadas as funcionalidades de cadastro de contexto e classes. Outros elementos, como módulos do projeto, artefatos e recursos humanos, só podem ser cadastrados nas ferramentas específicas, respectivamente, Ferramenta de Decomposição do Produto, Ferramenta de Planejamento de Documentação e Ferramenta de Alocação de Recursos.

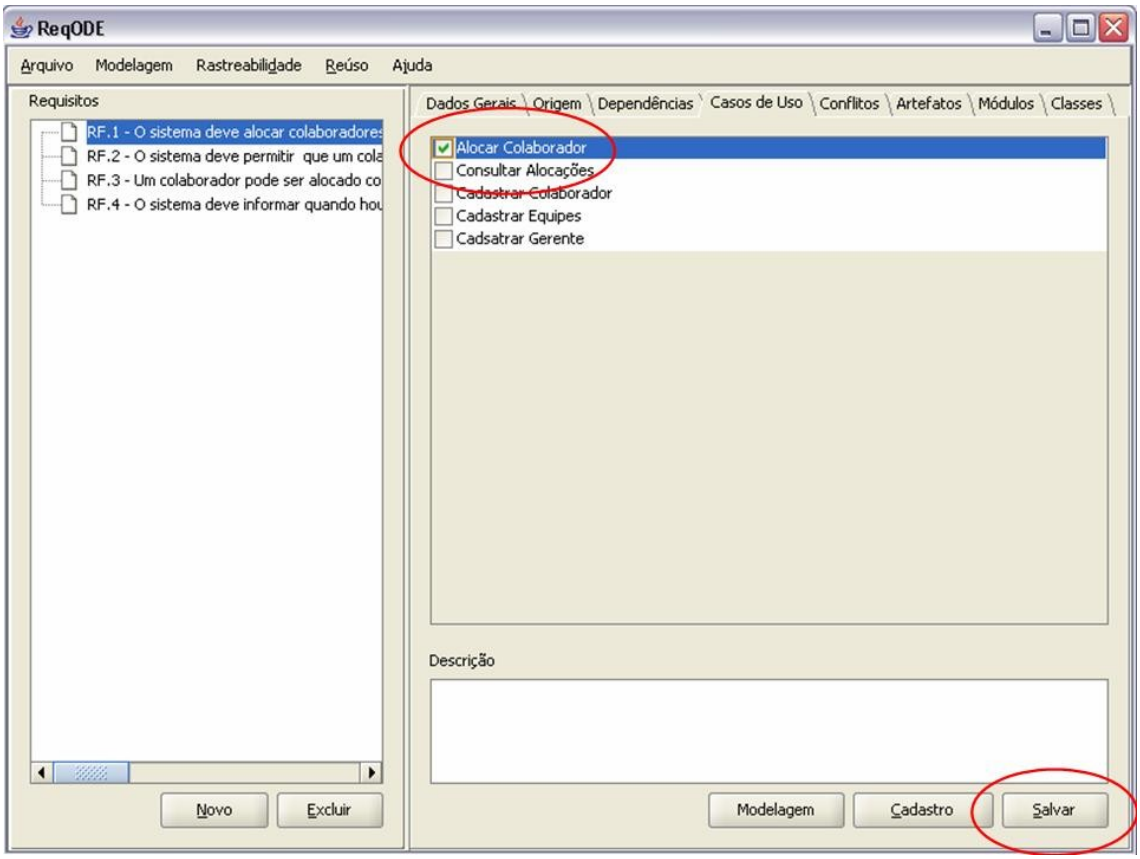


Figura 36 - Associação de um Requisito a um Caso de uso

Para verificar as ligações de rastreabilidade definidas através da ferramenta, é possível criar relatórios completos, através do menu "Rastreabilidade -> Gerar Relatório Completo", ou ainda criar relatórios customizados, neste caso, acessando o menu "Rastreabilidade -> Gerar Relatório Customizado". Acionando o último, é possível visualizar a janela mostrada na Figura 37, que corresponde à janela *JanCustomizarRelatorio* apresentada na Seção 5.2.4. Nessa janela é possível escolher quais os itens que farão parte do relatório.

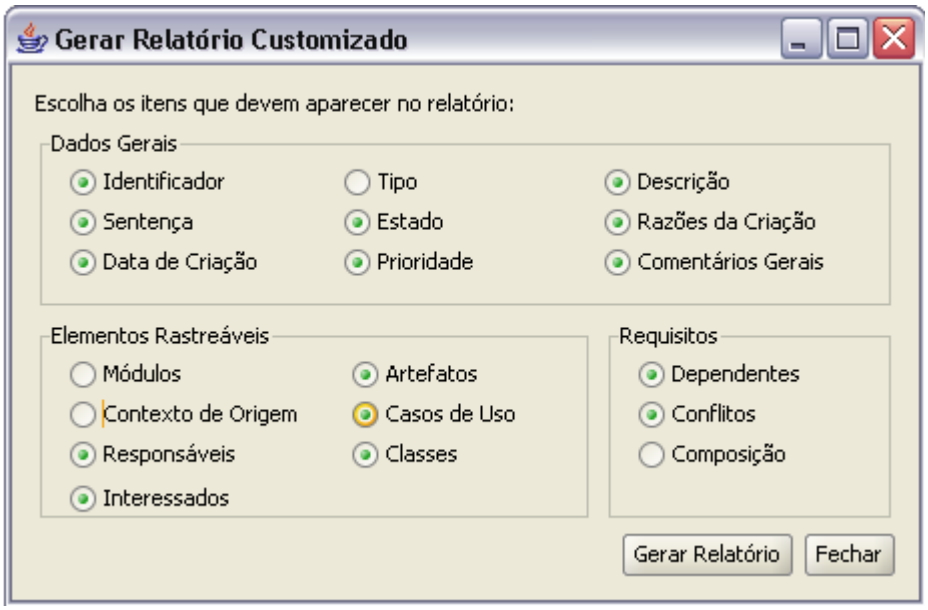
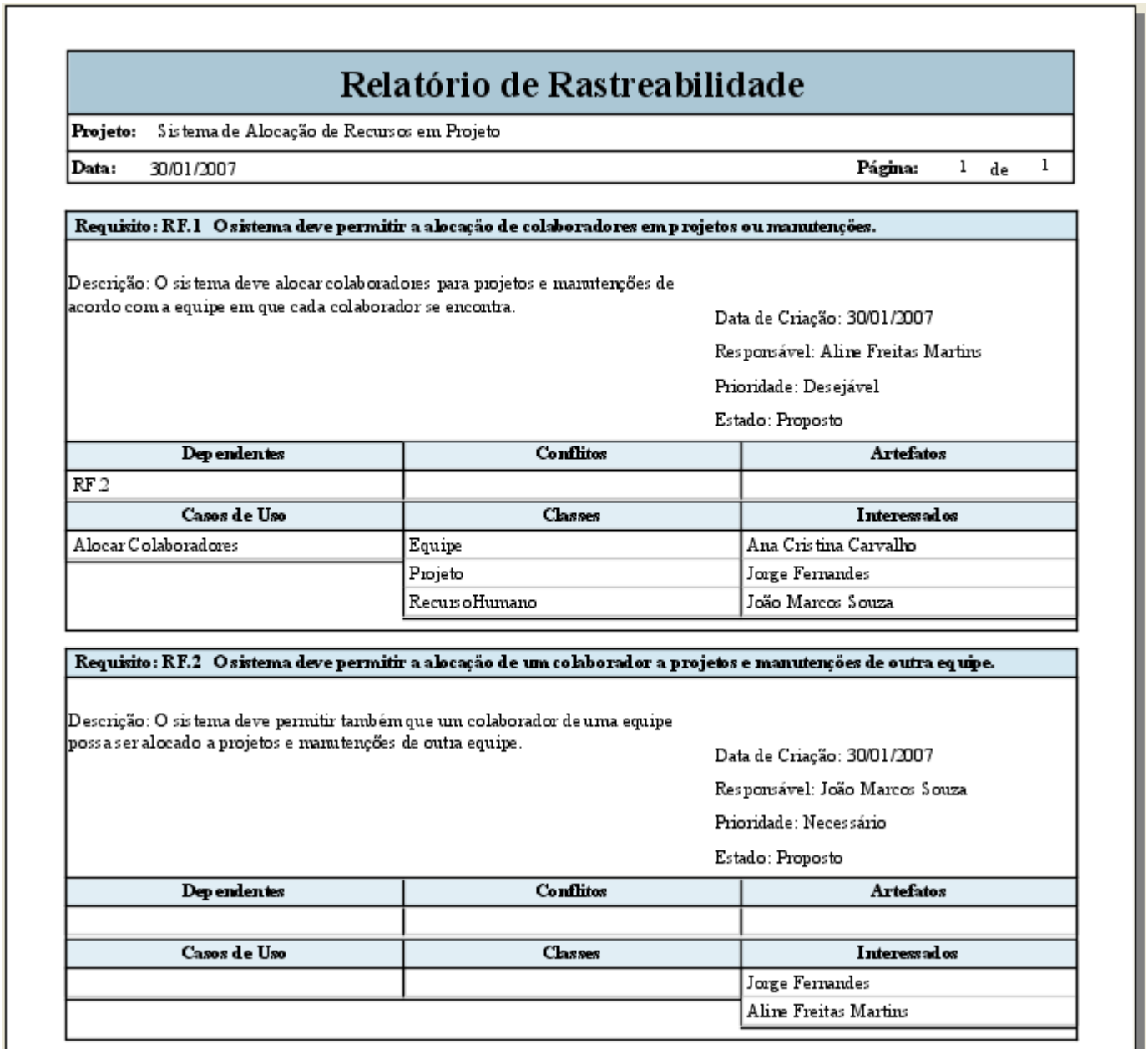


Figura 37 - Janela de criação de relatórios customizados

Depois de escolhidos os itens, basta acionar o botão "Gerar Relatório" e o relatório contendo as informações selecionadas é criado e exibido, tendo o *layout* apresentado na Figura 38.



Relatório de Rastreabilidade

Projeto: Sistema de Alocação de Recursos em Projeto

Data: 30/01/2007 **Página:** 1 de 1

Requisito: RF.1 O sistema deve permitir a alocação de colaboradores em projetos ou manutenções.

Descrição: O sistema deve alocar colaboradores para projetos e manutenções de acordo com a equipe em que cada colaborador se encontra.

Data de Criação: 30/01/2007
Responsável: Aline Freitas Martins
Prioridade: Desejável
Estado: Proposto

Dependentes	Conflitos	Artefatos
RF.2		
Casos de Uso	**Classes**	**Interessados**
Alocar Colaboradores	Equipe	Ana Cristina Carvalho
	Projeto	Jorge Fernandes
	RecursoHumano	João Marcos Souza

Requisito: RF.2 O sistema deve permitir a alocação de um colaborador a projetos e manutenções de outra equipe.

Descrição: O sistema deve permitir também que um colaborador de uma equipe possa ser alocado a projetos e manutenções de outra equipe.

Data de Criação: 30/01/2007
Responsável: João Marcos Souza
Prioridade: Necessário
Estado: Proposto

Dependentes	Conflitos	Artefatos
Casos de Uso	**Classes**	**Interessados**
		Jorge Fernandes
		Aline Freitas Martins

Figura 38 – *Layout* do relatório de rastreabilidade

CAPÍTULO 6 CONSIDERAÇÕES FINAIS

Nos capítulos iniciais foi apresentado o conteúdo bibliográfico que representou a base e a justificativa para a construção das ferramentas propostas neste trabalho. Como foco principal foi construída a ferramenta ReqODE, que permite o controle e a rastreabilidade dos requisitos de um projeto durante o seu ciclo de desenvolvimento. Para o apoio de atividades complementares, criaram-se ferramentas de Cadastro de Conhecimento sobre Tipos de Requisitos e de Cadastro de Elementos de Modelo que, respectivamente, possibilitaram o cadastro de tipos de requisito e de casos de uso, atores e classes. Este capítulo apresenta as principais conclusões e perspectivas futuras com respeito ao presente trabalho.

1.23CONCLUSÕES

Como foi apresentado, o processo de Engenharia de Requisitos (ER) é complexo e envolve grande quantidade de informações, logo é importante um eficiente acompanhamento desse processo por meio de ferramentas automatizadas. Além disso, para melhor gerenciar os requisitos de um projeto, é importante que essas ferramentas de apoio à ER estejam integradas às demais ferramentas já existentes em um Ambiente de Desenvolvimento de Software (ADS). Assim, a ferramenta ReqODE foi desenvolvida com o intuito de apoiar o processo de Engenharia de Requisitos no contexto do ambiente ODE.

Durante o levantamento de requisitos, a ReqODE dá suporte ao armazenamento dos requisitos, mantendo informações sobre sua origem e permitindo sua classificação e caracterização. Na análise é possível identificar requisitos conflitantes e gerar relatórios para apoiar negociações, definindo prioridades na resolução desses conflitos. A documentação de requisitos não tem ainda suporte automatizado, pois se optou por aguardar a conclusão da ferramenta de documentação do ambiente, atualmente em desenvolvimento. A validação é parcialmente apoiada pela possibilidade de se criar um documento de rastreabilidade que identifica origens, prioridades, conflitos e dependências dos requisitos. Assim, é possível, ainda que indiretamente, ter um suporte à identificação de inconsistências, incoerências e redundâncias. A Gerência de Requisitos é apoiada, sobretudo, pela rastreabilidade bidirecional, possibilitada pelas buscas e pelos documentos de rastreabilidade customizados ou completos.

A ferramenta ReqODE foi criada com o intuito de tornar o processo de Engenharia de Requisitos mais simples e controlado. Mantendo sempre a consistência entre elementos do processo de desenvolvimento, a ferramenta de apoio à Engenharia de Requisitos contribui para manter prazos e orçamentos bem definidos, auxiliando na construção de produtos de software estáveis e mais fáceis de manter.

1.24PERSPECTIVAS FUTURAS

ReqODE foi integrada à nova versão do ambiente ODE em abril de 2006 e está sendo utilizada por uma organização de software parceira desde outubro de 2006. Assim, espera-se evoluí-la com base no *feedback* dado por essa organização. Além disso, existem outras ferramentas em desenvolvimento ou aprimoramento no ambiente ODE que poderão ser integradas à ReqODE. Entre elas, a Ferramenta de Documentação (BUBACK, 2007) que poderá ser utilizada na geração de Documentos de Especificação de Requisitos, e a Ferramenta de Gerência de Configuração (NUNES, 2004) que, se integrada à ReqODE, possibilitará um maior controle sobre as mudanças ocorridas em um requisito, mantendo informações sobre diversas versões do mesmo. Dessa forma, ReqODE se tornará mais completa e robusta, oferecendo um apoio ainda mais eficiente às várias atividades do processo de Engenharia de Requisitos.

Atualmente, existe em ODE uma infra-estrutura de Gerência de Conhecimento (GC) (NATALI et al., 2003), que foi especializada para dar suporte ao processo de Engenharia de Requisitos, por meio da reutilização de itens de conhecimento relevantes, tais como requisitos, modelos de objetos, ontologias de domínio e padrões de análise (NARDI, 2006). É possível, a partir de ReqODE, utilizar a infra-estrutura de caracterização e cálculo de similaridade entre projetos de ODE, para identificar projetos similares ao projeto corrente e, a partir deles, requisitos podem ser reutilizados. Esse trabalho já foi feito usando funcionalidades de ReqODE e pode ser aperfeiçoado, procurando tratar outros itens de conhecimento potencialmente relevantes para a ER.

Referências Bibliográficas

ARANTES, L. O. **Automatização da Gerência de Projetos em ODE**. Monografia (Graduação em Ciência da Computação) – Departamento de Informática, Universidade Federal do Espírito Santo, Vitória, 2006.

BAUER,C.; KING, G. **Hibernate em Ação**. 1 ed. Rio de Janeiro, RJ: Ciência Moderna, 2005.

CHRISSIS, M. B.; KONRAD, M.; SHRUM, S. **CMMI: Guidelines for Process Integration and Product Improvement**, Addison-Wesley Pub Co, 2003.

CHRISTEL, M.G.; KANG, K.C. **Issues in Requirements Elicitation.** Software Engineering Institute, 1992. Apud: PRESSMAN, R. S. **Engenharia de Software**. 5ª. ed., Rio de Janeiro: McGrawHill, 2002.

COELHO, A. **Apoio à Gerência de Recursos Humanos em ODE**. Monografia (Graduação em Ciência da Computação) – Departamento de Informática, Universidade Federal do Espírito Santo, Vitória, 2007.

FALBO, R.A. **Análise de Sistemas - Notas de Aula.** Departamento de Informática, UFES, 2000.

FALBO, R.A. **Projeto de Sistemas - Notas de Aula**. Departamento de Informática, UFES, 2003.

FALBO R.A. et al.**. ODE: Ontology-based Software Development Environment**. In: IX Argentine Congress on Computer Science (CACIC'2003), La Plata, Argentina, 2003, pp 1124-1135.

FALBO, R.A. et al. **Um Processo de Engenharia de Requisitos Baseado em Reutilização e Padrões de Análise**. VI Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento (JIISIC'07), Lima, Peru (aceito para apresentação e publicação), jan. 2007.

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.

IEEE-SA STANDARDS BOARD IEEE Std 1233-1998: **IEEE Guide for Developing System Requirements Specifications.** Dezembro de 1998.

IEEE-SA STANDARDS BOARD IEEE Std 830-1993: **IEEE Recommended Practice for Software Requirements Specifications**. Dezembro de 1993.

IREPORT Project Home. Disponível em: <jasperforge.org/sf/projects/ireport>. Acesso em: 3 jan. 2007.

JASPERREPORTS Project Home. Disponível em: <jasperforge.org/sf/projects/jasperreports>. Acesso em: 3 jan. 2007.

KENDAL, K. E.; KENDAL, J. E. **Systems Analysis and Design.** Prentice Hall, 1992. Apud: FALBO, R. A. **Análise de Sistemas - Notas de Aula.** Departamento de Informática, UFES, 2000.

KOTONYA, G.; SOMMERVILLE, I. **Requirements engineering: processes and techniques.** Chichester, England: John Wiley, 1998.

LOPES, P. S. N. D. **Uma Taxonomia da Pesquisa na Área de Engenharia de Requisitos**. Dissertação de mestrado, USP, São Paulo, SP, 2002.

MARTINS, A.F.; NARDI, J.C., FALBO, R.A. **ReqODE: uma Ferramenta de Apoio à Engenharia de Requisitos integrada ao Ambiente ODE**. In: XIII Sessão de Ferramentas do SBES, p. 31-36, Florianópolis, SC: out. 2006.

NARDI, J.C. **Apoio de Gerência de Conhecimento à Engenharia de Requisitos em um Ambiente de Desenvolvimento de Software**. Dissertação (Mestrado em Informática) – Programa de Pós-Graduação em Informática, Universidade Federal do Espírito Santo, Vitória, 2006.

NARDI, J. C.; FALBO, R. A. **Uma Ontologia de Requisitos de Software.** In: 9º Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software, La Plata, Argentina, 2006.

PFLEEGER, S. L. **Software Engineering – Theory and Practice.** Saddle River, New Jersey: Prentice Hall, Upper, 1998.

POSTGRESQL Global Development Group. Disponível em: <www.postgresql.org >. Acesso em: 9 jan. 2007.

PRESSMAN, R. S. **Engenharia de Software**. 5ª. ed. Rio de Janeiro: McGrawHill, 2002.

ROBERTSON, S. R.; ROBERTSON, J. **Mastering the Requirements Process.** Addison-Wesley, 1999.

ROCHA, A. R. C.; MALDONADO, J. C.; WEBER, K. C. **Qualidade de Software – Teoria e Prática.** São Paulo, SP: Prentice Hall, 2001.

SEGRINI, B. M.; BERTOLO, G., FALBO, R.A. **Evoluindo a Definição de Processos de Software em ODE**. In: XIII Sessão de Ferramentas do SBES, Florianópolis, out. 2006. p. 109-114.

SEGRINI, B. M. **Evolução do Apoio à Definição e Acompanhamento de Processos de ODE**. Monografia (Graduação em Ciência da Computação) – Departamento de Informática, Universidade Federal do Espírito Santo, Vitória, 2007.

SILVA, B. C. **Adequação ao meta-modelo da UML em OODE: Apoio à Elaboração de Diagramas de Classe e Casos de Uso**. Monografia (Graduação em Ciência da Computação) - Universidade Federal do Espírito Santo, Vitória, 2003.

SOMMERVILLE, I. **Engenharia de Software.** 6a. ed. São Paulo, SP: Addison-Wesley, 2003.

MVC - Model-view-controller. Disponível em: <java.sun.com/blueprints/patterns/MVC-detailed.html>. Acesso em: 11 dez. 2006.

DAO - Data Access Object. Disponível em: <java.sun.com/j2ee/patterns/DataAccessObject.html>. Acesso em: 8 jan. 2007.

XDOCLET Attribute Oriented Programming. Disponível em: <xdoclet.sourceforge.net>. Acesso em: 3 jan. 2007.

WIEGERS, K.E. **Software Requirements.** 2nd ed. Redmond, Washington: Microsoft Press, 2003.

Anexo A

DOCUMENTAÇÃO DO PROJETO DA FERRAMENTA DE CADASTRO DE CONHECIMENTO DE TIPOS DE REQUISITOS

Documento de Especificação de Requisitos Funcionais

Projeto: Cadastro de Conhecimento sobre Tipo de Requisitos

Responsável: Aline Freitas Martins

1. Introdução

Este documento apresenta a especificação de requisitos para o Cadastro de Conhecimento existente em ODE (*Ontology-based software Development Environment*) (FALBO et al., 2003), um Ambiente de Desenvolvimento de Software (ADS). O foco do documento está na parte relativa ao conhecimento sobre requisitos.

Essa atividade foi conduzida usando a técnica de Modelagem de Casos de Uso e, portanto, este documento contém uma descrição do propósito do sistema (descrição do mini-mundo), apresentada na seção 2, e o modelo de casos de uso, incluindo diagrama e descrições dos casos de uso (seção 3).

2. Descrição do Mini-Mundo

O objetivo deste documento é apresentar a especificação de requisitos necessários para o desenvolvimento de uma ferramenta que forneça ao usuário apoio direcionado ao conhecimento organizacional sobre requisitos. É importante manter padronizado a classificação dos requisitos de um sistema. Para isso, é essencial manter quais os tipos de requisitos serão considerados na organização.

Existem diferentes classificações de requisitos, sendo que a base é classificá-los como funcionais e não-funcionais, podendo estes apresentar subtipos. Assim é necessário manter um cadastro desses tipos, armazenando o seu nome, uma breve descrição e de quem ele é subtipo.

A ferramenta proposta é integrada ao ODE como parte do Cadastro de Conhecimento do mesmo, o que torna possível armazenar informações a respeito dos tipos de requisitos podendo associá-los a outros elementos cadastrados no ambiente.

3. Modelo de Casos de Uso

A ferramenta de cadastro de Conhecimento de ODE permite que informações organizacionais sejam mantidas no ambiente, facilitando a utilização dessas pelo usuário. No contexto da ferramenta existem diversos subsistemas responsáveis por registrar conhecimento a respeito de processos, riscos, requisitos, métricas, qualidade etc. O foco deste documento é descrever a parte responsável pelo Cadastro de Conhecimento sobre Tipo de Requisitos, mais especificamente tipos de requisitos. Analisaremos, portanto, o sub-sistema *Conhecimento.requisito*, mostrado na Figura 1.

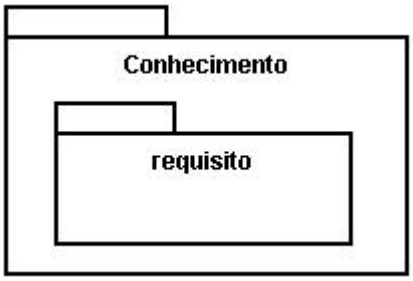


Figura 1 - Diagrama de pacotes

A Figura 2 apresenta o diagrama de casos de uso para o sub-sistema identificado, cujo caso de uso é descrito na seqüência.

Figura 2 - Diagrama de casos de uso

Sub-sistema: Conhecimento.requisito

Caso de Uso: Cadastrar Tipo de Requisito

Descrição: Este caso de uso permite o cadastro de tipos de requisitos, que permitirão a devida classificação dos requisitos tratados pelo sistema.

Cursos Normais:

Criar Novo Tipo de Requisito

O gerente de conhecimento informa o nome, o super tipo (se necessário) e uma descrição do tipo de requisito que deseja cadastrar no sistema. Caso os dados sejam válidos, a informação é registrada no sistema.

Alterar Tipo de Requisito

O gerente de conhecimento seleciona o tipo de requisito que deseja alterar e entra com os novos dados, podendo ser alterados o nome e a descrição apenas. Se os novos dados forem válidos, a alteração é efetuada.

Excluir Tipo de Requisito

O gerente de conhecimento seleciona um tipo de requisito e solicita a sua exclusão. Uma mensagem é exibida para a confirmação da exclusão. Se confirmada, o tipo escolhido é excluído e todos os seus subtipos passam a ser subtipos do seu supertipo.

Curso Alternativo:

Criar Tipo de Requisito

Se o nome estiver em branco e o gerente de conhecimento solicitar o salvamento do tipo de requisito, o sistema mostrará uma mensagem para lembrá-lo de preencher este campo.

Se já existir um tipo de requisito com este nome e com mesmo super-tipo, uma mensagem avisará ao gerente de conhecimento e solicitará que o ele informe outro nome para o tipo de requisito.

Alterar Tipo de Requisito

Se o gerente de conhecimento alterar o nome do tipo de requisito e já existir cadastrado no sistema para o mesmo super-tipo outro tipo de requisito com este nome, a ferramenta informará o ocorrido e solicitará que ele digite outro nome.

Excluir Tipo de Requisito

Se o tipo de requisito selecionado para a exclusão estiver associado a um requisito, ou seja, classificando algum requisito cadastrado no sistema, será mostrada uma mensagem avisando que existe essa associação e o tipo de requisito não será excluído.

Classes: Conhecimento, KTipoRequisito.

Referências Bibliográficas

FALBO R.A., NATALI, A.C.C., MIAN, P.G., BERTOLLO, G., RUY, F.B. **ODE: Ontology-based software Development Environment**, Proceedings of the IX Argentine Congress on Computer Science (CACIC'2003), La Plata, Argentina, 2003, pp 1124-1135

Documento de Especificação de Análise

Projeto: Cadastro de Conhecimento sobre Tipos de Requisitos

Responsável: Aline Freitas Martins

1. Introdução

Este documento apresenta a especificação de análise para o Cadastro de Conhecimento sobre Tipos de Requisitos. Essa atividade foi conduzida seguindo o método da Análise Orientada a Objetos, tendo sido elaborados diagrama de Classes, apresentados na seção 2. Vale destacar que foi adotada a notação da UML.

2. Modelo de Classes

O diagrama de pacotes da Figura 1 mostra as dependências existentes entre os pacotes contemplados nessa ferramenta.

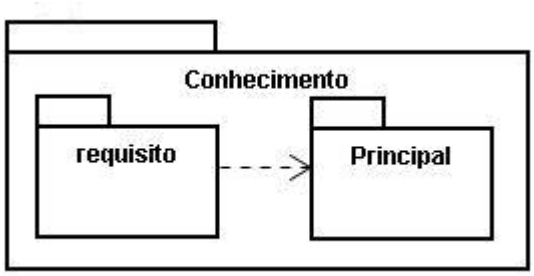


Figura 1 - Diagrama de pacotes

Dentre os pacotes acima mostrados, apenas o pacote *Conhecimento.requisito* foi criado para atender o objetivo deste sistema. O restante já existe no ambiente ODE e é utilizado com o objetivo de manter a integração e consistência do ambiente.

Foi acrescentado, portanto, apenas o pacote *Conhecimento.requisito* que se encontra dentro do pacote *Conhecimento* para respeitar as premissas do ambiente ODE e contém os elementos de modelo relativos à classificação de requisitos.

2.1 – Pacote *Conhecimento.requisito*

A Figura 2 mostra o diagrama de classes do pacote *Conhecimento.requisito*. Para representar os tipos de requisitos do ambiente, foi criada a classe *KTipoRequisito*. Como um tipo de requisito pode ser subtipo de outro, foi adicionado o auto-relacionamento representado no modelo. É importante ressaltar que essa relação é transitiva (isto é, se um tipo de requisito *tr1* é subtipo de outro *tr2*, que por sua vez é um subtipo de *tr3*, então *tr1* é também um subtipo de *tr3*) e irreflexiva (ou seja, um tipo de requisito não pode ser subtipo dele mesmo). Note que *KTipoRequisito* herda da classe *Conhecimento*; isso ocorre em ODE com todas as classes que representam algum conhecimento organizacional.

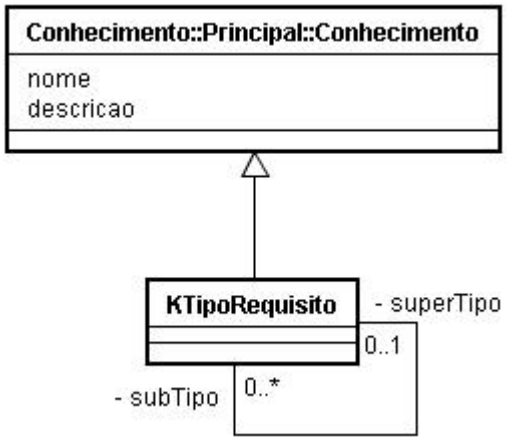


Figura 2 - Diagrama de Classes do pacote *Conhecimento.requisito*

Documento de Especificação de Projeto

Projeto: Cadastro de Conhecimento sobre Tipos de Requisitos

Responsável: Aline Freitas Martins

1. Introdução

Este documento apresenta a especificação de projeto para o Cadastro de Conhecimento sobre Tipos de Requisitos. A ferramenta proposta será implementada usando a linguagem de programação Java, o *framework* de mapeamento objeto-relacional Hibernate (BAUER, 2005) e a ferramenta XDoclet (XDOCLET, 2007) para agilizar e facilitar o mapeamento dos objetos em tabelas.

Essa atividade foi conduzida em duas etapas: Projeto Arquitetural e Projeto Detalhado. A seção 2 apresenta o Projeto Arquitetural por meio de diagramas de pacotes e a seção 3 trata do Projeto Detalhado para o pacote principal envolvido neste projeto, o pacote *Conhecimento.requisito*, apresentando os diagramas de classes de cada componente da arquitetura. Vale destacar que foi adotada a notação da UML.

2. Projeto Arquitetural

Com a finalidade de organizar o sistema, as classes do projeto foram organizadas em pacotes de acordo com o domínio do problema. A Figura 1 apresenta o diagrama de pacotes de nível mais alto. Em relação ao modelo de análise foi introduzido o pacote *Utilitario* que contém classes que são utilizadas por diversas ferramentas do ambiente.

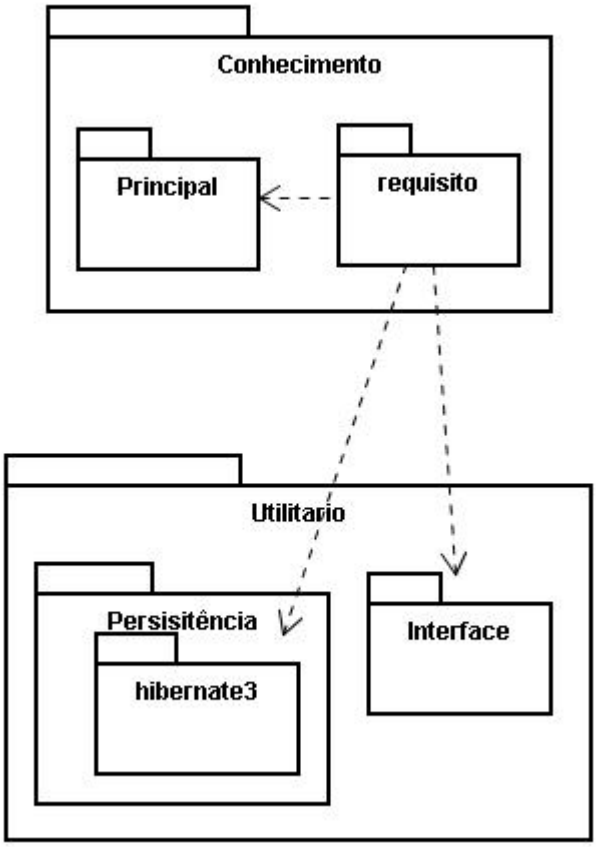


Figura 1 - Diagrama de Pacotes do pacote *Conhecimento.requisito*

A Figura 1 mostra as dependências existentes entre o pacote principal da ferramenta *Conhecimento.requisito* e os demais pacotes utilizados, já existentes no ambiente, a saber: *Conhecimento.Principal*, que contém classes que apóiam a criação de objetos relativos ao conhecimento no ambiente, *Utilitario.Persistencia*, que contém as classes base de apoio às operações envolvendo bancos de dados, e *Utilitario.Interface*, que contém interfaces com o usuário padrão do ambiente.

Complementando a organização das classes, há um próximo nível de divisão em pacotes, segundo a função que exercem no sistema, ou melhor, segundo os seus estereótipos, como mostra a Figura 2.

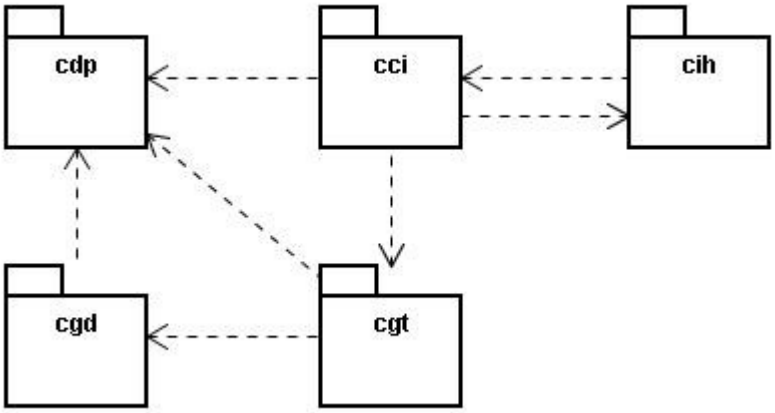


Figura 2 - Diagrama de Pacotes do pacote *Conhecimento.requisito*

O pacote *cdp* corresponde ao Componente de Domínio do Problema, cujas classes correspondem a abstrações do domínio do problema e, portanto, são derivadas do modelo de análise.

O pacote *cgd* corresponde ao Componente de Gerência de Dados que envolve as classes relativas à persistência de dados. As classes de domínio (*cdp*) que necessitam ter informações persistidas têm no topo de sua herança a classe *ObjetoPersistente*, que encontra-se no pacote *Utilitario.Persistencia.hibernate3*. Isto é importante para manter as informações necessárias para a sua persistência em bancos de dados e traz as vantagens da reutilização, como a facilidade de implementação e abstração de informações de persistência, ou seja, geração de um domínio livre de informações sobre o banco de dados. Como a infra-estrutura do ambiente é baseada no padrão DAO (*Data Access Object*) (SUN, 2006), para cada classe do domínio a ser persistida é criada uma interface DAO e uma de implementação que utiliza o *framework* Hibernate, como mostra a Figura 3.

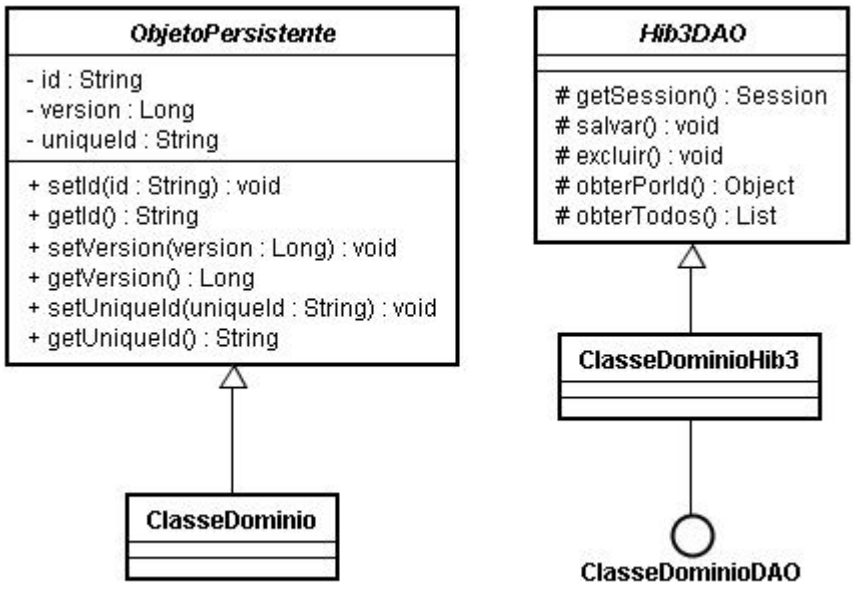


Figura 3 - Infra-estrutura de Persistência

O pacote *cgt* corresponde ao Componente de Gerência de Tarefas e contém as aplicações que implementam os casos de usos definidos na especificação de requisitos.

O pacote *cih* é o Componente de Interação Humana e possui as classes relativas às interfaces gráficas da ferramenta.

O pacote *cci* é o Componente de Controle de Interação cujas classes têm o papel principal de manter a comunicação entre as classes de interface (*cih*) e de aplicação (*cgt*). Suas classes fazem o devido tratamento dos retornos da aplicação para a exibição de resultados na interface, bem como informam às classes de aplicação o que o usuário solicitou por meio de uma interface.

3. Pacote *Conhecimento.requisito*

O pacote *Conhecimento.requisito* contém as classes que implementam as funcionalidades relacionadas ao cadastro de conhecimento de requisitos em ODE.

3.1 - Componente de Domínio do Problema (cdp)

A Figura 4 apresenta o diagrama de classes referente ao Componente do Domínio do Problema do pacote *Conhecimento.requisito*. Como se pode notar comparando com o correspondente modelo da fase de análise, não houve mudanças significativas, sendo tratados apenas tipos de dados, navegabilidades e operações.

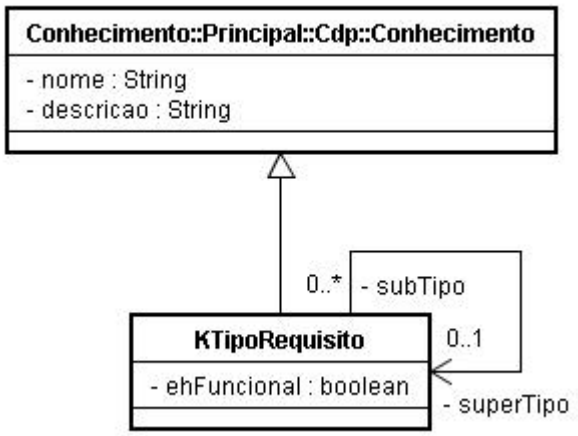


Figura 4 - Diagrama de Classes do pacote *cdp*

3.2 - Componente de Gerência de Dados (cgd)

A Figura 5 apresenta o diagrama de classes referente ao Componente de Gerência de Dados do pacote *Conhecimento.requisito*. As classes criadas são baseadas no padrão DAO (*Data Access Object*) (SUN, 2006), conforme comentado na seção 2 deste documento.

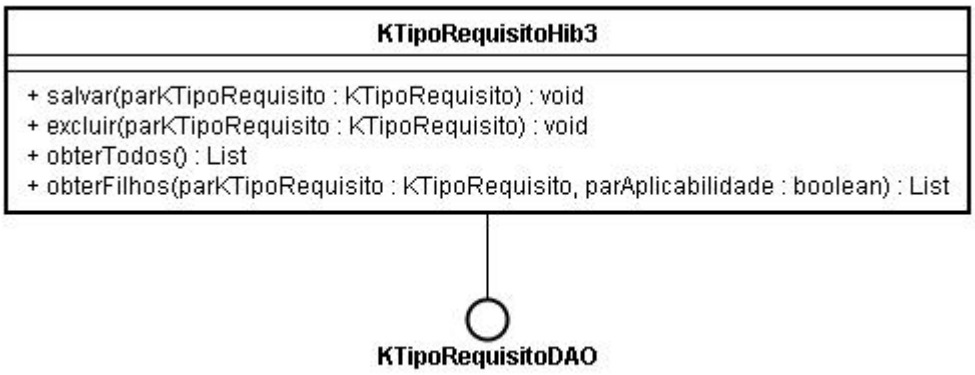


Figura 5 - Diagrama de Classes do pacote *cgd*

3.3 - Componente de Gerência de Tarefas (cgt)

A Figura 6 apresenta o diagrama de classes referente ao Componente de Gerência de Tarefas do pacote *Conhecimento.requisito*.

Figura 6 - Diagrama de Classes do pacote.*cgt*

Como o pacote *Conhecimento.requisito* apresenta apenas um caso de uso, foi criada apenas a classe de aplicação *AplCadastrarKTipoRequisito* para tratar todos os eventos relativos ao caso de uso "Cadastro Tipo de Requisito", apresentado no Documento de Especificação de Requisitos.

3.4 - Componente de Interação Humana (cih)

A Figura 7 apresenta o diagrama de classes referente ao Componente de Interação Humana do pacote *Conhecimento.requisito*.

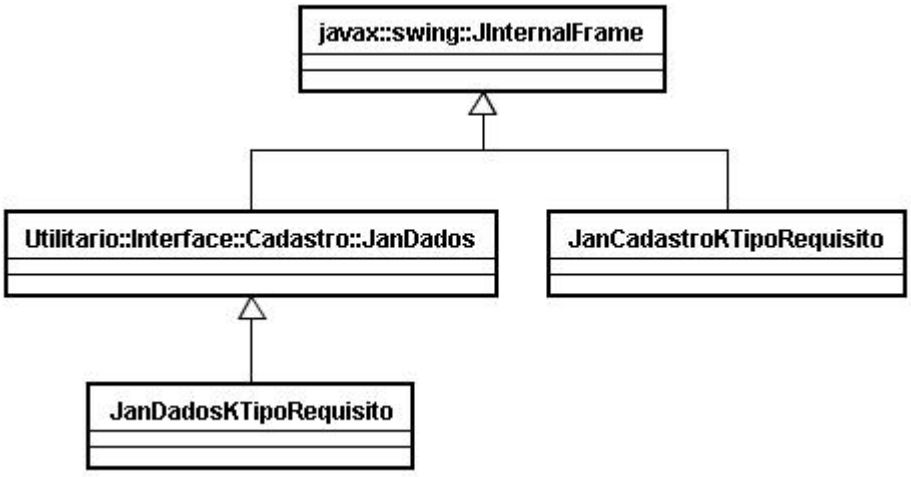


Figura 7 - Diagrama de Classes do pacote *cih*

A janela de cadastro de tipos de requisitos (*JanCadastroKTipoRequisito*) exibe os tipos de requisitos cadastrados no sistema e disponibiliza as funcionalidades de criar um novo tipo, alterar, visualizar ou excluir um tipo de requisito previamente selecionado. Para criação de um

tipo de requisito que seja filho de um dos tipos já cadastrados, é necessário que o usuário selecione o pai e acione a opção de criar um novo requisito.

Para visualizar ou alterar os dados de um tipo de requisito ou mesmo inserir informações de um novo foi criada, a janela de dados *JanDadosKTipoRequisito* é utilizada. Ela aparecerá sempre que as opções de cadastro forem solicitadas na janela de cadastro (*JanCadastroKTipoRequisito*). Pode-se notar que a janela *JanDadosKTipoRequisit*o herda de *JanDados*, que se encontra no pacote *Utilitario.Interface.Cadastro.* Isto é feito visando à padronização e ao reúso.

3.5 - Componente de Controle de Interação (cci)

A Figura 8 apresenta o diagrama de classes referente ao Componente de Controle de Interação do pacote *Conhecimento.requisito.*

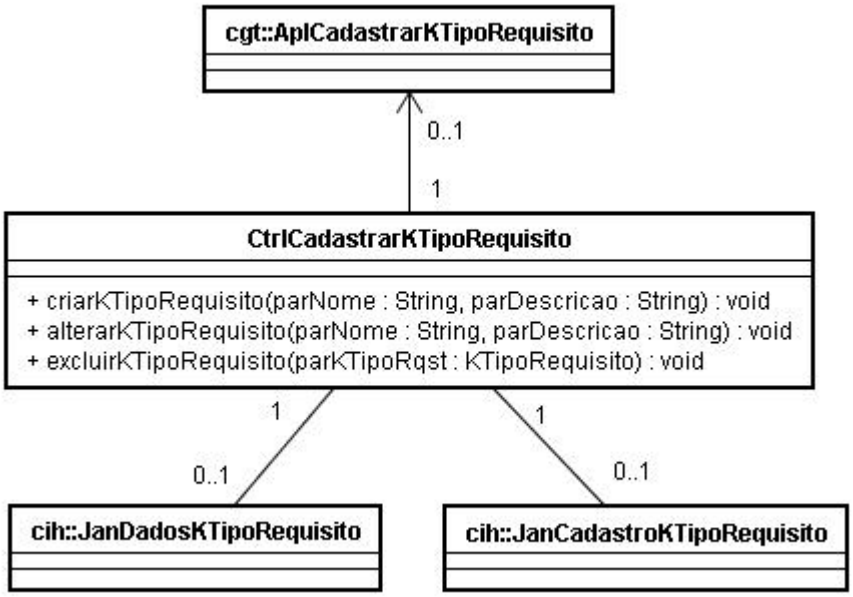


Figura 8 - Diagrama de Classes do pacote *cci*

A classe *CtrlCadastrarKTipoRequisito* tem o papel principal de fazer a comunicação entre os objetos das camadas de aplicação (*cgt*) e de interface (*cih*) desta ferramenta. Desta forma as interfaces ficam isoladas, facilitando alterações. Permite, também, que as janelas não tenham acesso direto a métodos da classe de aplicação, isolando funcionalidades distintas.

A classe de controle apresentada neste pacote tem em seus relacionamentos com navegabilidade dupla com as janelas do pacote *Conhecimento.requisito.cih*, a saber *JanDadosKTipoRequisito e JanCadastrokTipoRequisito.* Desta forma o controlador (*CtrlCadastrarKTipoRequisito*) poderá exibir ou atualizar as janelas nos momentos devidos e

as janelas poderão invocar métodos do controlador para chamar outra janela ou mesmo solicitar a realização de operações implementadas nas aplicações (*cgt*).

O relacionamento com a classe de aplicação só é navegável no sentido controlador - aplicação, ou seja, o controlador conhece a classe de aplicação, mas o contrário não é verdadeiro. Isto porque, com a arquitetura estabelecida, o controlador verifica os retornos da aplicação e invoca seus métodos.

Referências Bibliográficas

BAUER,C.; KING, G. **Hibernate em Ação**. 1 ed. Rio de Janeiro, RJ: Ciência Moderna, 2005.

FALBO R.A., NATALI, A.C.C., MIAN, P.G., BERTOLLO, G., RUY, F.B. **ODE: Ontology-based software Development Environment**, Proceedings of the IX Argentine Congress on Computer Science (CACIC'2003), La Plata, Argentina, 2003, pp 1124-1135

MVC Model-view-controller. Disponível em: <java.sun.com/blueprints/patterns/MVC-detailed.html>. Acesso em: 11 dez. 2006.

DAO Data Access Object. Disponível em: <java.sun.com/j2ee/patterns/DataAccessObject.html>. Acesso em: 8 jan. 2007.

XDOCLET Attribute Oriented Programming. Disponível em: <xdoclet.sourceforge.net>. Acesso em: 3 jan. 2007.

Anexo B

DOCUMENTAÇÃO DO PROJETO DA FERRAMENTA DE CADASTRO DE ELEMENTOS DE MODELO

Documento de Especificação de Requisitos Funcionais

Projeto: Cadastro de Elementos de Modelo

Responsável: Aline Freitas Martins

1. Introdução

Este documento apresenta a especificação de requisitos para a ferramenta de cadastro de elementos de modelos (neste documento, apenas casos de uso, atores e classes) criados no desenvolvimento de software. Essa atividade foi conduzida usando a técnica de Modelagem de Casos de Uso e, portanto, este documento contém uma descrição do propósito do sistema (descrição do mini-mundo), apresentada na seção 2, e o modelo de casos de uso, incluindo diagramas e descrições dos casos de uso (seção 3).

2. Descrição do Mini-Mundo

Durante o processo de desenvolvimento de software, vários artefatos são produzidos para que informações importantes não sejam perdidas e para facilitar a visualização de problemas e soluções. Dentre esses artefatos, podemos citar diagramas de classes, diagramas de casos de uso, diagramas de pacotes e diagramas de componentes, dentre outros. Para a construção desses diagramas, devem ser definidos os elementos básicos que os compõem, denominados elementos de modelo, segundo a UML.

No ambiente ODE existe a ferramenta de modelagem UML OODE, usada normalmente para definir Modelos de Objeto e Elementos de Modelo. Porém, muitas vezes, é necessário criar alguns desses elementos em outro momento do desenvolvimento, como por exemplo, nas estimativas por pontos de casos de uso, na qual podem ser informados casos de uso, atores e classes. Logo, para dar maior flexibilidade e agilidade, é útil ter um cadastro mais simples, no qual esses elementos não são modelados, e sim, cadastrados no sistema.

Para esse cadastro algumas informações apenas são requeridas: das classes deve-se informar o nome e a descrição, podendo ser informados, ainda, atributos e operações; dos casos de uso, precisa-se saber o nome, a descrição, os atores que o realizam, eventos que fazem parte do mesmo e as classes que estão relacionadas ao caso de uso; de atores, é necessário saber nome e descrição.

3. Modelo de Casos de Uso

No contexto da ferramenta de Cadastro de Elementos de Modelo foram identificados dois sub-sistemas principais: CasosUso e Nucleo.

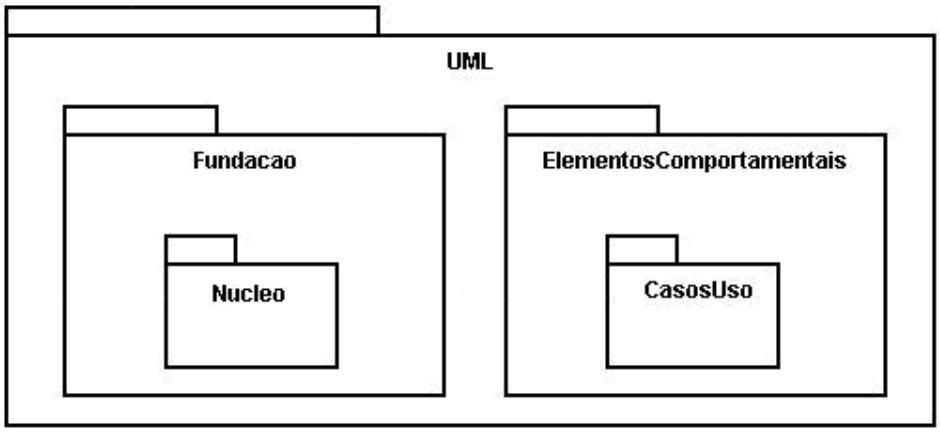


Figura 1 - Diagrama de pacotes

6.1.1.13.1 Modelo de Casos de Uso do pacote *UML.ElementosComportamentais.CasosUso*

A Figura 2 apresenta o diagrama de casos de uso para o pacote *CasosUso*, cujos casos de uso são descritos na seqüência.

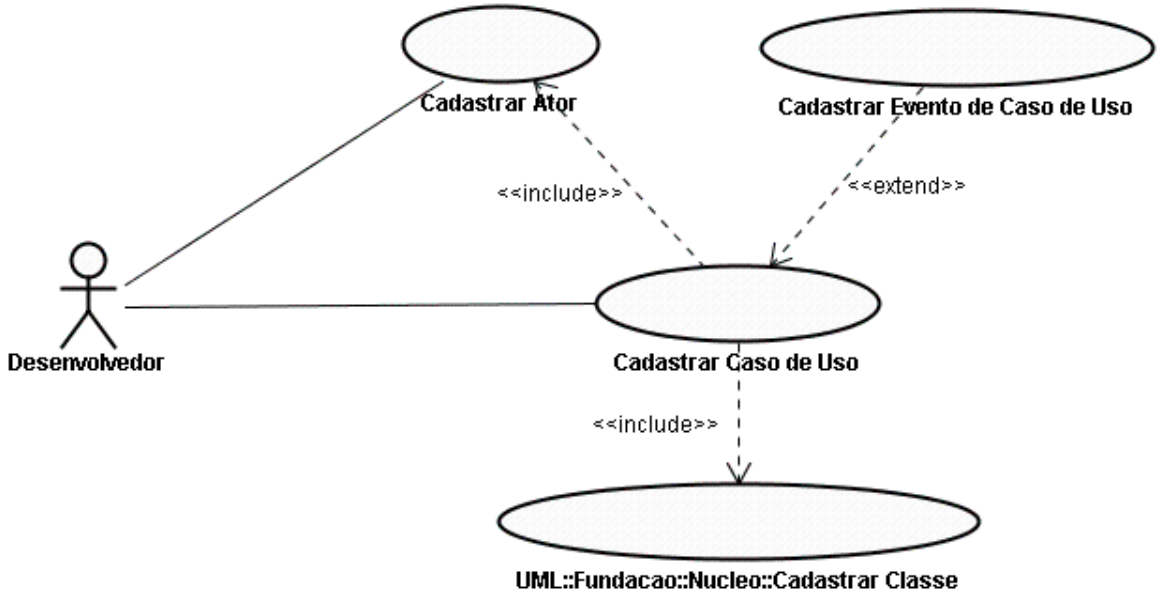


Figura 2 - Diagrama de casos de uso do pacote *CasosUso*.

Sub-sistema: UML.ElementosComportamentais.CasosUso
Caso de Uso: Cadastrar Caso de Uso
Descrição: Este caso de uso é responsável pelo cadastro de casos de uso.

Cursos Normais:

Criar Caso de Uso

O desenvolvedor informa o nome, a descrição, os atores e as classes envolvidos. Caso o ator desejado não esteja cadastrado, é possível cadastrá-lo realizando o caso de uso *Cadastrar Ator.* O mesmo ocorre com as classes: se não estiverem cadastradas, poderão sê-las por meio da realização do caso de uso *Cadastrar Classe.* Por fim, pode-se cadastrar eventos do caso de uso, realizando o caso de uso *Cadastrar Evento de Caso de Uso*.

Quando da criação de um novo caso de uso, associa-se o mesmo a um modelo de objetos para manter a relação com o projeto a que pertence. Se não existir um Modelo de Objetos para o projeto corrente, cria-se um modelo novo com o nome "Modelo de Objetos do projeto <nome do projeto>". Se existir apenas um Modelo de Objetos no projeto corrente, o caso de uso é automaticamente atribuído a este. Por fim, se existirem vários modelos de objetos no projeto corrente, o desenvolvedor deverá escolher um deles ao qual o caso de uso deverá pertencer.

Caso os dados sejam válidos (o nome não pode estar em branco ou já existir outro caso de uso com o mesmo nome), o caso de uso é criado.

Alterar Caso de Uso

Dado um caso de uso, o desenvolvedor informa os novos dados (nome, descrição, eventos, atores e classes) e solicita a alteração do caso de uso. Caso os dados informados sejam válidos, a alteração é registrada.

Excluir Caso de Uso

Dado um caso de uso, o desenvolvedor solicita sua exclusão. Uma mensagem de confirmação é exibida e, caso o desenvolvedor confirme a exclusão, o caso de uso e seus eventos são excluídos.

Cursos alternativos

Criar Caso de Uso / Alterar Caso de Uso

Se o nome estiver em branco e o desenvolvedor solicitar o salvamento do caso de uso, o sistema mostrará uma mensagem para lembrá-lo de preencher este campo.

Se já existir um caso de uso no projeto com este nome, uma mensagem avisará ao desenvolvedor e solicitará que o desenvolvedor informe outro nome para o caso de uso.

Excluir Caso de Uso

Se o caso de uso selecionado para a exclusão estiver associado a uma estimativa, uma mensagem informará ao desenvolvedor que não será possível concluir a operação, pois existem objetos associados ao caso de uso.

Classes: Caso de Uso, Ator, EventoCasoUso, Classe.

Sub-sistema: UML.ElementosComportamentais.CasosUso
Caso de Uso: Cadastrar Ator
Descrição: Este caso de uso é responsável pelo cadastro de atores.

Cursos Normais:

Criar Ator

O desenvolvedor informa o nome, a descrição e o ator do qual ele herda, se for o caso. Caso os dados sejam válidos (o nome não pode estar em branco nem pode existir outro ator com o mesmo nome), o ator é criado.

Alterar Ator

Dado um ator, o desenvolvedor informa os novos dados e solicita a alteração do ator. Caso os dados informados sejam válidos, a alteração é registrada.

Excluir Ator

Dado um ator, o desenvolvedor solicita sua exclusão. Uma mensagem de confirmação é exibida e, caso o desenvolvedor confirme a exclusão, o ator é excluído.

Cursos alternativos

Criar/Alterar Ator

Se o nome estiver em branco e o desenvolvedor solicitar o salvamento do ator, o sistema mostrará uma mensagem para lembrá-lo de preencher este campo.

Se já existir um ator no projeto com este nome, uma mensagem avisará ao desenvolvedor e solicitará que o desenvolvedor informe outro nome para o ator.

Excluir Ator

Se o Ator selecionado para a exclusão estiver associado a uma estimativa, uma mensagem informará ao desenvolvedor que não será possível concluir a operação, pois existem objetos associados ao Ator.

Classes: Ator.

Sub-sistema: UML.ElementosComportamentais.CasosUso
Caso de Uso: Cadastrar Evento de Caso de Uso
Descrição: Este caso de uso é responsável pelo cadastro de eventos de um caso de uso.

Cursos Normais:

Criar Evento de Caso de Uso

O desenvolvedor informa o nome e a descrição do evento de caso de uso. Caso os dados sejam válidos (o nome não pode estar em branco ou ser repetido), o evento é criado e associado ao caso de uso pré-selecionado (Caso de Uso *Cadastrar Caso de Uso*).

Alterar Evento de Caso de Uso

Dado um evento, o desenvolvedor informa os novos dados (nome e descrição) e solicita a alteração. Caso os dados informados sejam válidos, a alteração é registrada.

Excluir Evento de Caso de Uso

Dado um evento, o desenvolvedor solicita a sua exclusão. Uma mensagem de confirmação é exibida e, caso o desenvolvedor confirme a exclusão, o evento é excluído.

Cursos Alternativos

Criar / Alterar Evento de Caso de Uso

Se o nome estiver em branco e o desenvolvedor solicitar o salvamento do evento o sistema mostrará uma mensagem para lembrá-lo de preencher este campo.

Se já existir um evento no caso de uso com este nome uma mensagem avisará ao desenvolvedor e solicitará que o desenvolvedor informe outro nome para o evento.

Classes: Caso de Uso, EventoCasoUso.

3.2 Modelo de Casos de Uso do pacote UML.Fundacao.Nucleo

A Figura 3 apresenta o diagrama de casos de uso para o pacote *UML.Fundacao.Nucleo*, cujos casos de uso são descritos na seqüência.

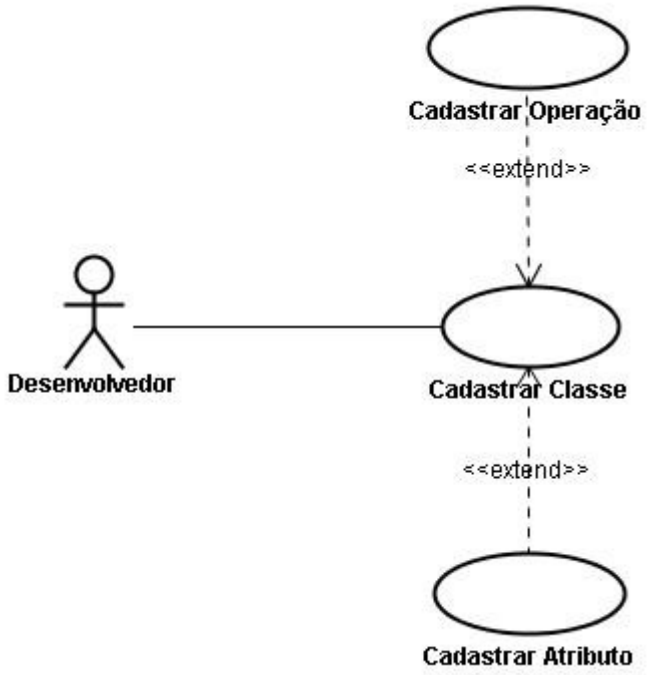


Figura 3 - Diagrama de casos de uso do pacote *Nucleo*.

Sub-sistema: UML.Fundacao.Nucleo

Caso de Uso: Cadastrar Classe

Descrição: Este caso de uso é responsável pelo cadastro de classes.

Cursos Normais:

Criar Classe

O desenvolvedor informa o nome, a descrição, os atributos (realizando o caso de uso *Cadastrar Atributo*) e as operações (realizando o caso de uso *Cadastrar Operação*). Caso os dados sejam válidos (o nome não pode estar em branco nem pode existir outra classe com esse nome no projeto), a classe é criada.

Quando da criação de uma nova classe, associa-se a mesma a um modelo de objetos para manter a relação com o projeto a que pertence. Se não existir um Modelo de Objetos para o projeto corrente, cria-se um modelo novo com o nome "Modelo de Objetos do projeto <nome do projeto>". Se existir apenas um Modelo de Objetos no projeto corrente, o caso de uso é automaticamente atribuído a este. Por fim, se existirem vários modelos de objetos no projeto corrente, o desenvolvedor deverá escolher um deles ao qual a classe deverá pertencer.

Caso os dados sejam válidos (o nome não pode estar em branco ou já existir outra classe com o mesmo nome), a classe é criada.

Alterar Classe

Dada uma classe, o desenvolvedor informa os novos dados (nome e descrição). É possível também realizar os casos de uso *Cadastrar Atributo* e *Cadastrar Operação* para criar, alterar ou excluir atributos e operações da classe. Caso os dados informados sejam válidos, a alteração é registrada.

Excluir Classe

Dada uma classe, o desenvolvedor solicita sua exclusão. Uma mensagem de confirmação é exibida e caso o desenvolvedor confirme a exclusão, a classe é excluída, juntamente com seus atributos e operações.

Cursos alternativos

Criar / Alterar Classe

Se o nome estiver em branco e o desenvolvedor solicitar o salvamento da classe, uma mensagem será exibida para lembrá-lo de preencher este campo.

Se já existir uma classe no projeto com este nome, uma mensagem avisará ao desenvolvedor e solicitará que o desenvolvedor informe outro nome para esta.

Classes: Classe, Atributo, Operação.

Sub-sistema: UML.Fundacao.Nucleo
Caso de Uso: Cadastrar Atributo
Descrição: Este caso de uso é responsável pelo cadastro de atributos de uma classe.

Cursos Normais:

Criar Atributo

O desenvolvedor informa o nome e a descrição do atributo. Caso os dados sejam válidos (o nome não pode estar em branco nem pode existir um outro atributo com o mesmo nome para a classe corrente), o atributo é criado e associado à classe previamente selecionada no caso de uso *Cadastrar Classe*.

Alterar Atributo

Dado um atributo, o desenvolvedor informa os novos dados (nome e descrição) e solicita a alteração. Caso os dados informados sejam válidos, a alteração é registrada.

Excluir Atributo

Dado um atributo, o desenvolvedor solicita a sua exclusão. Uma mensagem de confirmação é exibida e, caso o desenvolvedor confirme a exclusão, o atributo é excluído.

Cursos alternativos

Criar / Alterar Atributo

Se o nome estiver em branco e o desenvolvedor solicitar o salvamento do atributo, será exibida uma mensagem para lembrá-lo de preencher este campo.

Se já existir um atributo na classe com este nome, uma mensagem avisará ao desenvolvedor e solicitará que o desenvolvedor informe outro nome para o atributo.

Classes: Classe, Atributo.

Sub-sistema: UML.Fundacao.Nucleo
Caso de Uso: Cadastrar Operação
Descrição: Esse caso de uso é responsável pelo cadastro de operações de uma classe.

Cursos Normais:

Criar Operação

O desenvolvedor informa o nome e a descrição da operação. Caso os dados sejam válidos (o nome não pode estar em branco nem pode existir uma outra operação com o mesmo nome para a classe corrente), a operação é criada e associada à classe pré-selecionada no caso de uso Cadastrar Classe.

Alterar Operação

Dada uma operação, o desenvolvedor informa os novos dados (nome e descrição) e solicita a alteração. Caso os dados informados sejam válidos, a alteração é registrada.

Excluir Operação

Dada uma operação, o desenvolvedor solicita a sua exclusão. Uma mensagem de confirmação é exibida e, caso o desenvolvedor confirme a exclusão, a operação é excluída.

Cursos alternativos

Criar Operação / Alterar Operação

Se o nome estiver em branco e o desenvolvedor solicitar o salvamento da operação, uma mensagem será exibida para lembrá-lo de preencher este campo.

Se já existir uma operação na classe com este nome, uma mensagem avisará ao desenvolvedor e solicitará que o desenvolvedor informe outro nome para a operação.

Classes: Classe, Operação.

Referências Bibliográficas

FALBO R.A., NATALI, A.C.C., MIAN, P.G., BERTOLLO, G., RUY, F.B. **ODE: Ontology-based software Development Environment**, Proceedings of the IX Argentine Congress on Computer Science (CACIC'2003), La Plata, Argentina, 2003, pp 1124-1135

Documento de Especificação de Análise

Projeto: Cadastro de Elementos de Modelo

Responsável: Aline Freitas Martins

1. Introdução

Este documento apresenta a especificação de análise para a ferramenta de cadastro de elementos de modelos (neste documento, apenas casos de uso, atores e classes). Essa atividade foi conduzida seguindo o método da Análise Orientada a Objetos, tendo sido elaborados diagrama de classes, apresentados na seção 2, utilizando a notação da UML.

2. Modelo de Classes

O diagrama de pacotes da Figura 1 mostra as dependências existentes entre os pacotes contemplados nessa ferramenta.

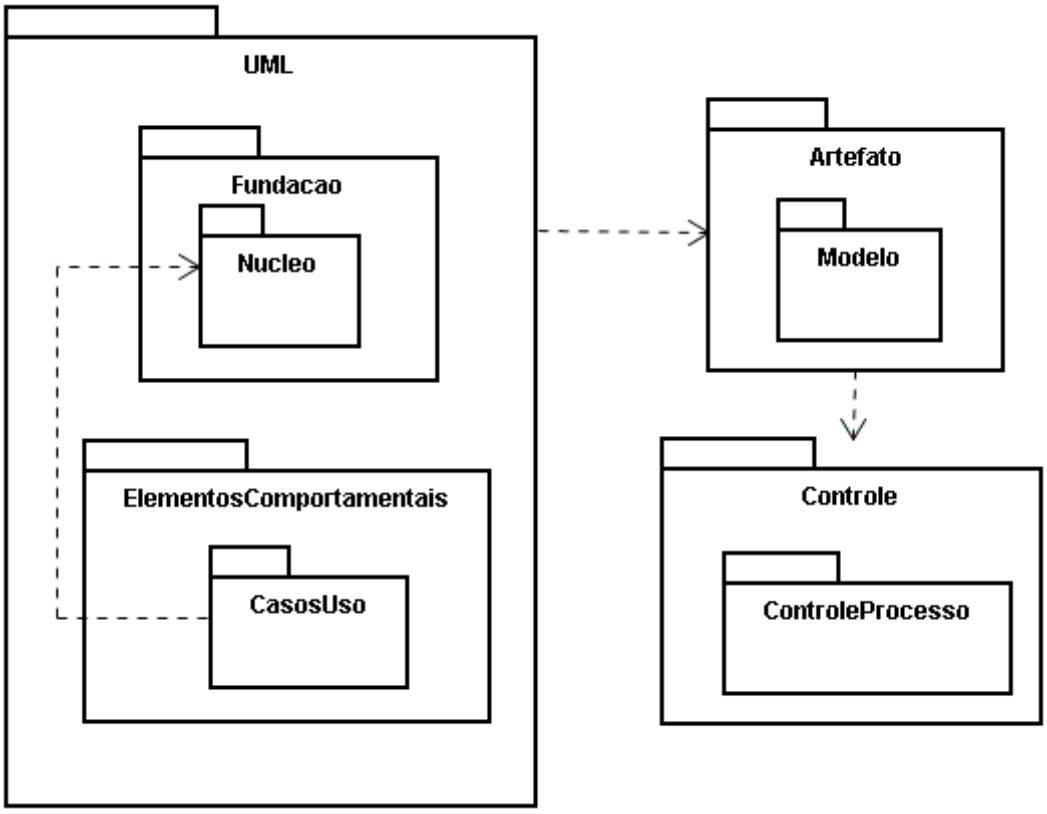


Figura 1 - Diagrama de Pacotes.

Dentro do pacote UML estão todas as classes relativas ao cadastro de elementos de modelo. De nosso interesse direto, temos apenas algumas classes dos pacotes *Nucleo* e *CasoUso*. Nos pacotes que restam, a saber *Controle.Processo* e *Artefato.Modelo*, existem as classes responsáveis por armazenar informações sobre os Modelos de Objetos que armazenam os elementos de modelo de nosso interesse e os relacionam ao Projeto.

A seguir são apresentados os diagramas de classe dos dois primeiros pacotes citados (*Nucleo* e *CasoUso*) e do último (*Artefato.Modelo*), os mais importantes para a ferramenta descrita neste documento.

2.1 – Pacote *UML.Fundacao.Nucleo*

A Figura 2 apresenta o diagrama de classes de parte do pacote *UML.Fundacao.Nucleo*. Esse pacote foi desenvolvido em (SILVA, 2003), estando em conformidade com o meta-modelo da UML.

A hierarquia apresentada tem como raiz a classe *Elemento* que nada mais é que um elemento básico produzido em um processo de software. Um elemento específico de modelagem de sistemas é denominado Elemento de Modelo, que, por sua vez, é classificado em Parâmetro, Propriedade, Espaço de Nome, Elemento Generalizável e Relacionamento. Neste trabalho consideramos apenas as classes *Propriedade* e *ElementoGeneralizavel*, pois em sua descendência estão as classes de nosso interesse.

Um Elemento Generalizável, como o próprio nome diz, é aquele passível de ser generalizado por meio de um relacionamento de generalização (não mostrado na Figura 2). Um classificador é um elemento generalizável que descreve propriedades. Classes, atores e casos de uso são subtipos de classificadores, enquanto operações e atributos são herdeiros de *Propriedade*, sendo operação uma Propriedade Comportamental e atributo um tipo de Propriedade Estrutural.

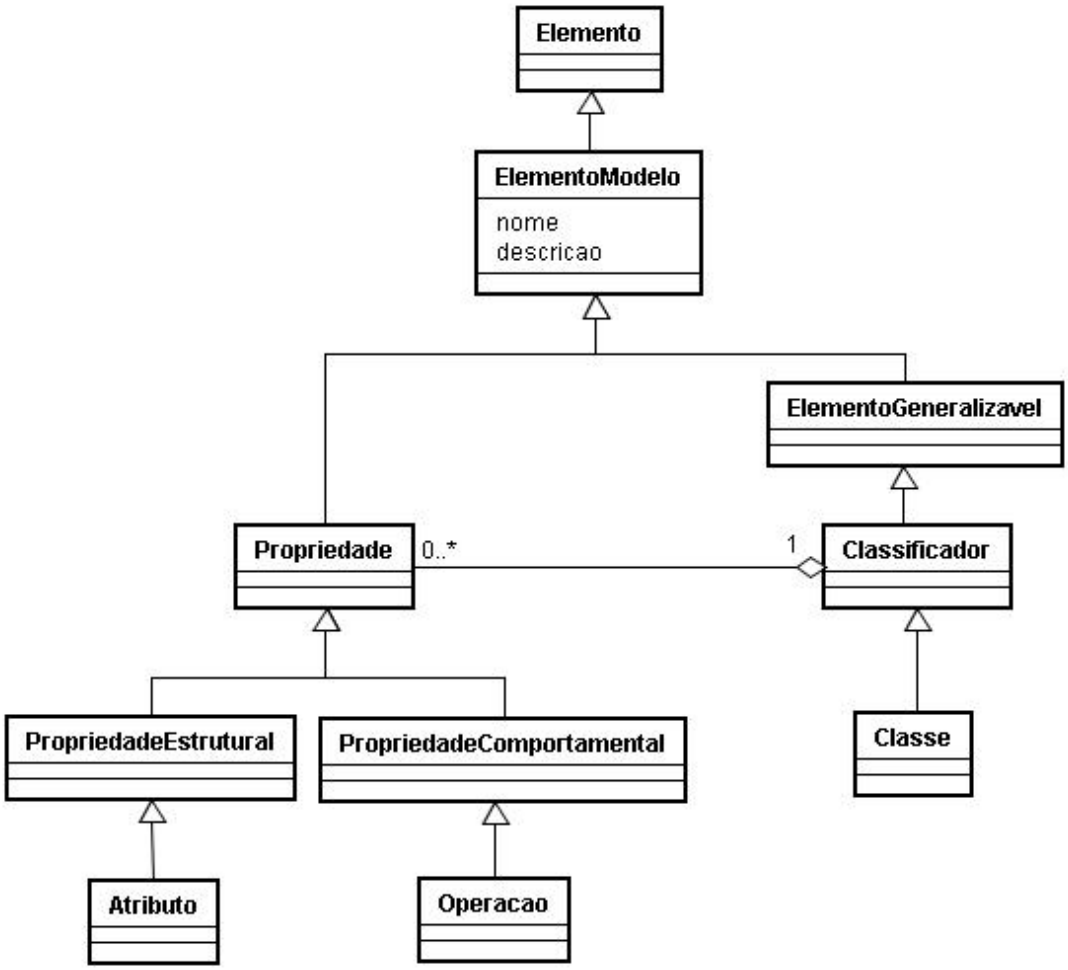


Figura 2 - Diagrama de Classes parcial do pacote *Nucleo*.

2.2 – Pacote *UML.ElementosComportamentais.CasoUso*

O pacote *UML.ElementosComportamentais* contém a infra-estrutura da linguagem que permite especificar o comportamento dinâmico nos modelos. Dentro dele está o pacote *Casos de Uso*, que especifica o comportamento usando atores e casos de uso, parcialmente mostrado na Figura 3. Conforme citado anteriormente, assim como uma classe, casos de uso e atores são classificadores. Já eventos de casos de uso são tipos de elemento de modelo, herdando diretamente de *ElementoModelo*.

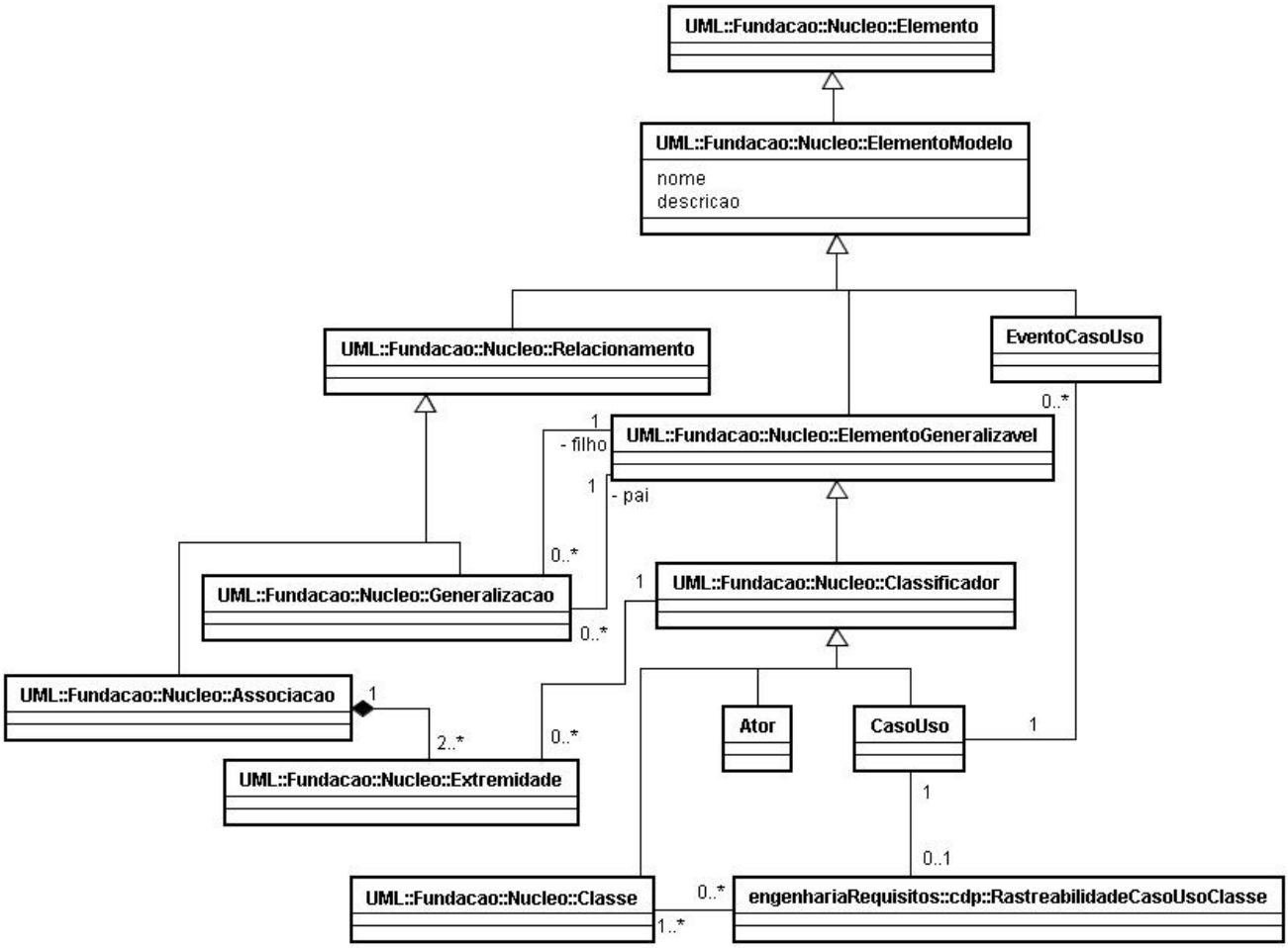


Figura 3 - Diagrama de Classes do pacote CasoUso

2.3 – Pacote *Artefato.Modelo*

Ao longo de um processo de software diversos artefatos são gerados no contexto de um projeto. Um tipo de artefato é retratado pela classe *Modelo,* que é um artefato que pode agregar vários elementos de modelo. Essa classe representa quaisquer modelos que um desenvolvedor pode construir, como, por exemplo, modelos estruturados e modelos orientados a objetos. Além disso, a classe *Modelo* possui a especialização *ModeloObjetos*, que representa um modelo direcionado, unicamente, à modelagem orientada a objetos.

A associação existente entre *ElementoModelo* e *Modelo* é de grande importância no contexto da ferramenta, pois é ela que permite descobrir qual o projeto de um caso de uso, classe ou ator. Bem como, é possível obter esses elementos a partir do projeto.

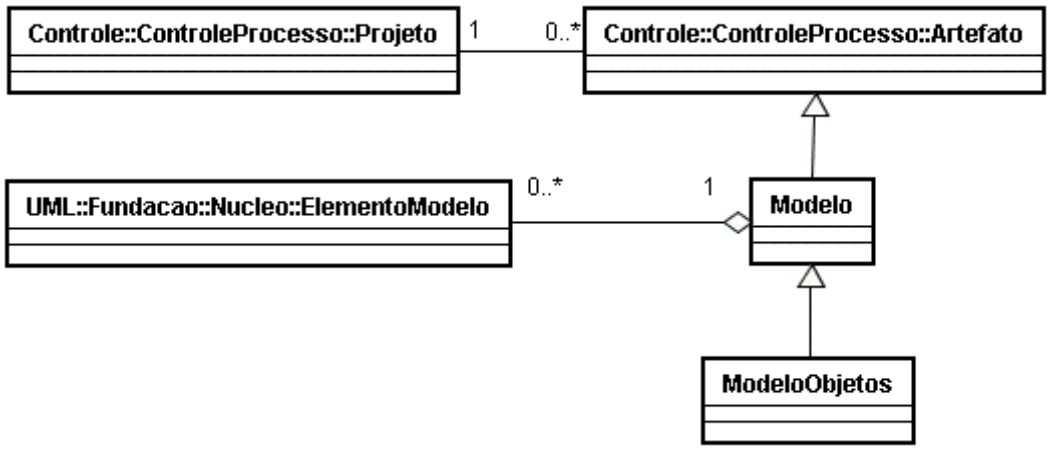


Figura 4 - Diagrama de Classes do pacote *Modelo.*

Referências Bibliográficas

SILVA, B. C. **Adequação ao meta-modelo da UML em OODE: Apoio à Elaboração de Diagramas de Classe e Casos de Uso**. Monografia (Graduação em Ciência da Computação) - Universidade Federal do Espírito Santo, Vitória, 2003.

Documento de Especificação de Projeto

Projeto: Cadastro de Elementos de Modelo

Responsável: Aline Freitas Martins

1. Introdução

Este documento apresenta a especificação de projeto para o Cadastro de Elementos de Modelo. A ferramenta proposta será implementada usando a linguagem de programação Java, o *framework* de mapeamento objeto-relacional Hibernate (BAUER, 2005) e a ferramenta XDoclet (XDOCLET, 2007) para agilizar e facilitar o mapeamento dos objetos em tabelas.

Essa atividade foi conduzida em duas etapas: Projeto Arquitetural e Projeto Detalhado. A seção 2 apresenta o Projeto Arquitetural por meio de diagramas de pacotes. As seções 3, 4 e 5 tratam do Projeto Detalhado para os pacotes *UML.Fundacao.Nucleo*, *UML.ElementosComportamentais.CasoUso* e *Artefato.Modelo*, respectivamente, apresentando os diagramas de classes de cada componente da arquitetura. Vale destacar que foi adotada a notação da UML.

2. Arquitetura do sistema

Com a finalidade de estabelecer níveis de abstração para o sistema, as classes do projeto foram organizadas em pacotes de acordo com o domínio do problema. A Figura 1 apresenta o diagrama de pacotes de nível mais alto. Em relação ao modelo de análise foi introduzido o pacote *Utilitario* que contém classes que são utilizadas por diversas ferramentas do ambiente.

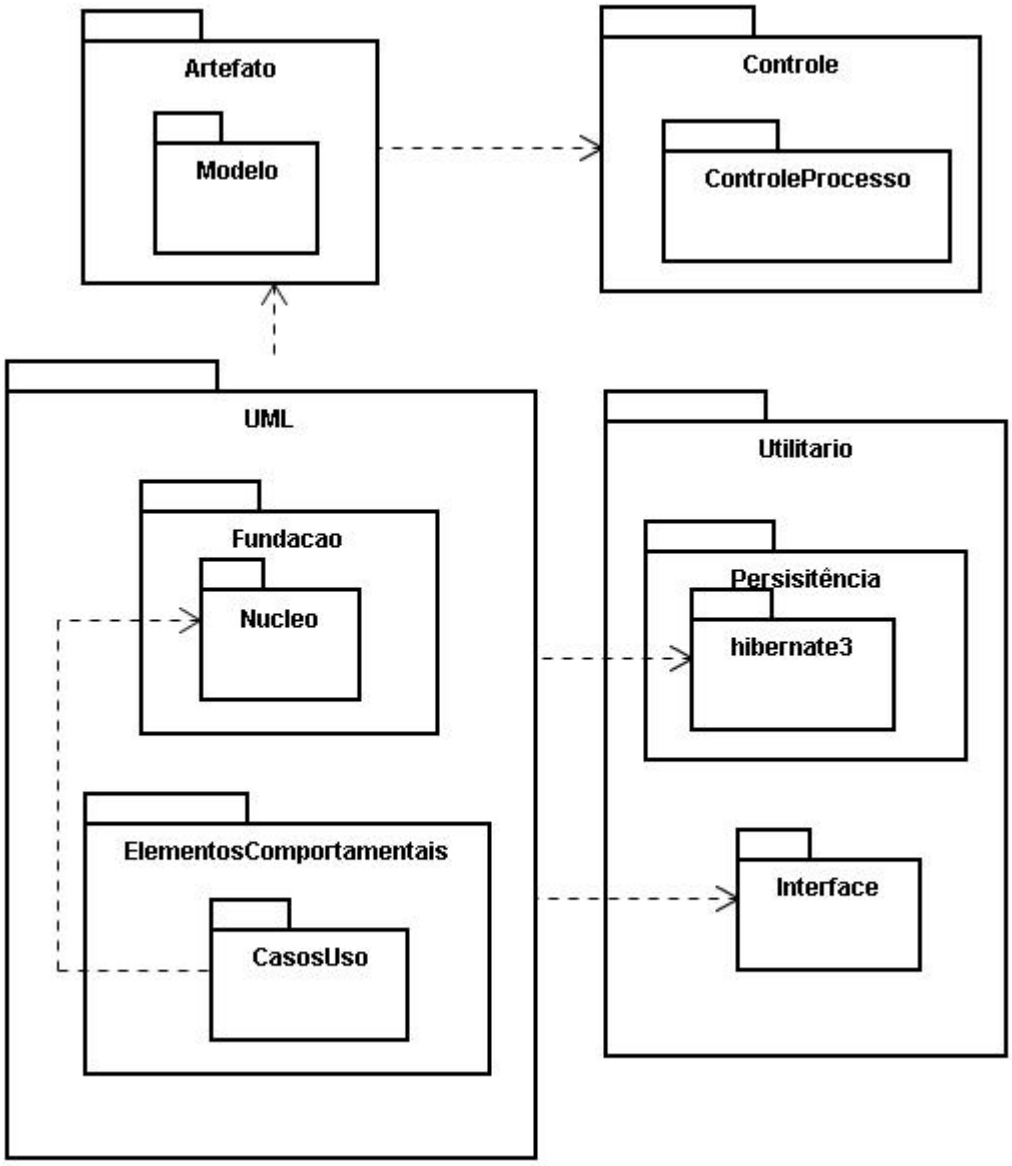


Figura 1 - Diagrama de pacotes da ferramenta

A Figura 1 mostra as dependências entre os pacotes principais da ferramenta, a saber *UML.Fundacao.Nucleo*, *UML.ElementosComportamentais.CasoUso* e *Artefato.Modelo*, e os demais pacotes do ambiente: *Controle.ControleProcesso*, que contém as classes base para armazenamento de informações sobre os artefatos gerados, e *Utilitario.Persistencia*, que contém as classes base de apoio às operações envolvendo bancos de dados.

Complementando a organização das classes, há um outro nível de divisão em pacotes, segundo a função que as classes exercem no sistema, ou seja, segundo os seus estereótipos, mostrado na Figura 2.

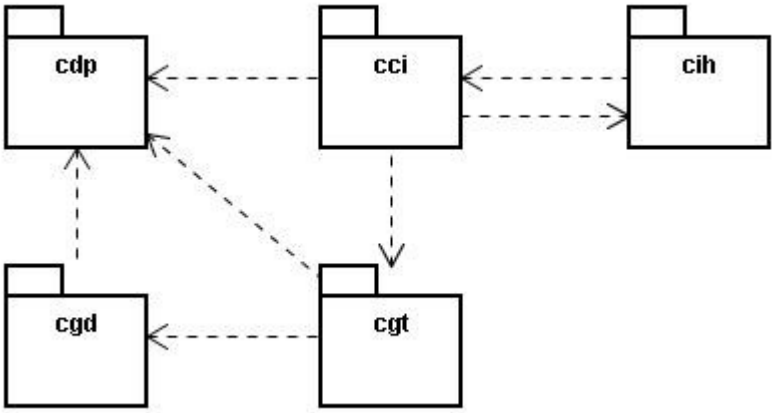


Figura 2 - Diagrama de Pacotes do pacote *Conhecimento.requisito*

O pacote *cdp* corresponde ao Componente de Domínio do Problema, cujas classes correspondem a abstrações do domínio do problema e, portanto, são derivadas do modelo de análise.

O pacote *cgd* corresponde ao Componente de Gerência de Dados e envolve as classes relativas à persistência de dados. As classes de domínio (*cdp*) que necessitam ter informações persistidas têm no topo de sua herança a classe *ObjetoPersistente*, que encontra-se no pacote *Utilitario.Persistencia.hibernate3*. Isto é importante para manter as informações necessárias para a sua persistência em bancos de dados e traz as vantagens da reutilização, como a facilidade de implementação e abstração de informações de persistência, ou seja, geração de um domínio livre de informações sobre o banco de dados. Como a infra-estrutura do ambiente é baseada no padrão DAO (*Data Access Object*) (SUN, 2006), para cada classe do domínio a ser persistida é criada uma interface DAO e uma de implementação que utiliza o *framework* Hibernate, como mostra a Figura 3.

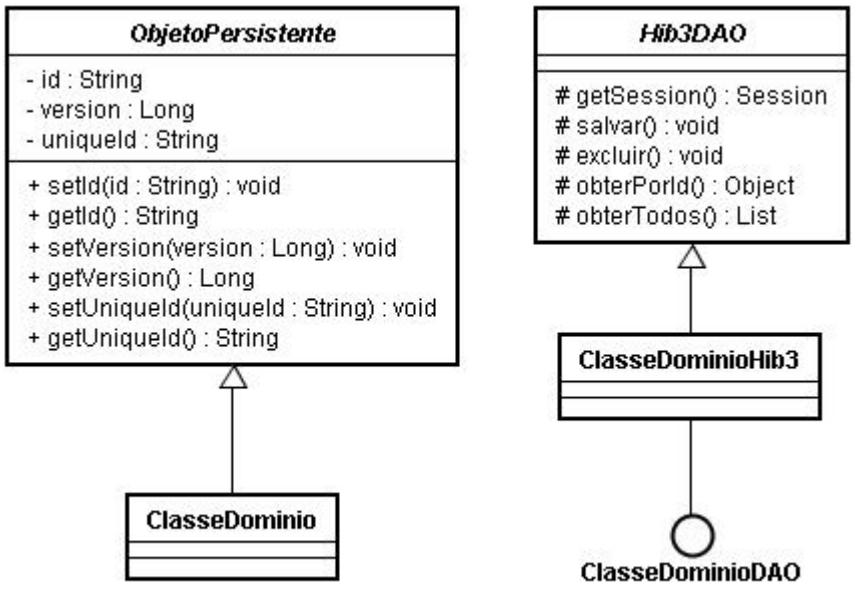


Figura 3 - Infra-estrutura de Persistência

O pacote *cgt* corresponde ao Componente de Gerência de Tarefas e contém as aplicações que implementam os casos de usos definidos na especificação de requisitos.

O pacote *cih* é o Componente de Interação Humana e possui as classes relativas às interfaces gráficas da ferramenta.

O pacote *cci* é o Componente de Controle de Interação cujas classes têm o papel principal de manter a comunicação entre as classes de interface (*cih*) e de aplicação (*cgt*). Suas classes fazem o devido tratamento dos retornos da aplicação para a exibição de resultados na interface, bem como informam às classes de aplicação o que o usuário solicitou por meio de uma interface.

3. Pacote *UML.Fundacao.Nucleo*

Este pacote contém as classes base para a criação de Elementos de Modelo em ODE. Esse pacote foi desenvolvido em (SILVA, 2003), estando em conformidade com o meta-modelo da UML. Neste documento apenas as classes relevantes para a ferramenta de Cadastro de Elementos de Modelo são apresentadas.

3.1 - Componente de Domínio do Problema (cdp)

A Figura 4 apresenta o diagrama de classes parcial referente ao Componente do Domínio do Problema do pacote *UML.Fundacao.Núcleo*. Como se pode notar, comparando com o correspondente modelo da fase de análise, foram adicionados tipos de dados, navegabilidades e operações, bem como foi alterado o relacionamento entre as classes C*lassificador* e *Propriedade*. *Classificador* passa, então, a conhecer as classes *Atributo* e *Operacao* que se encontram na base da hierarquia de *Propriedade.*

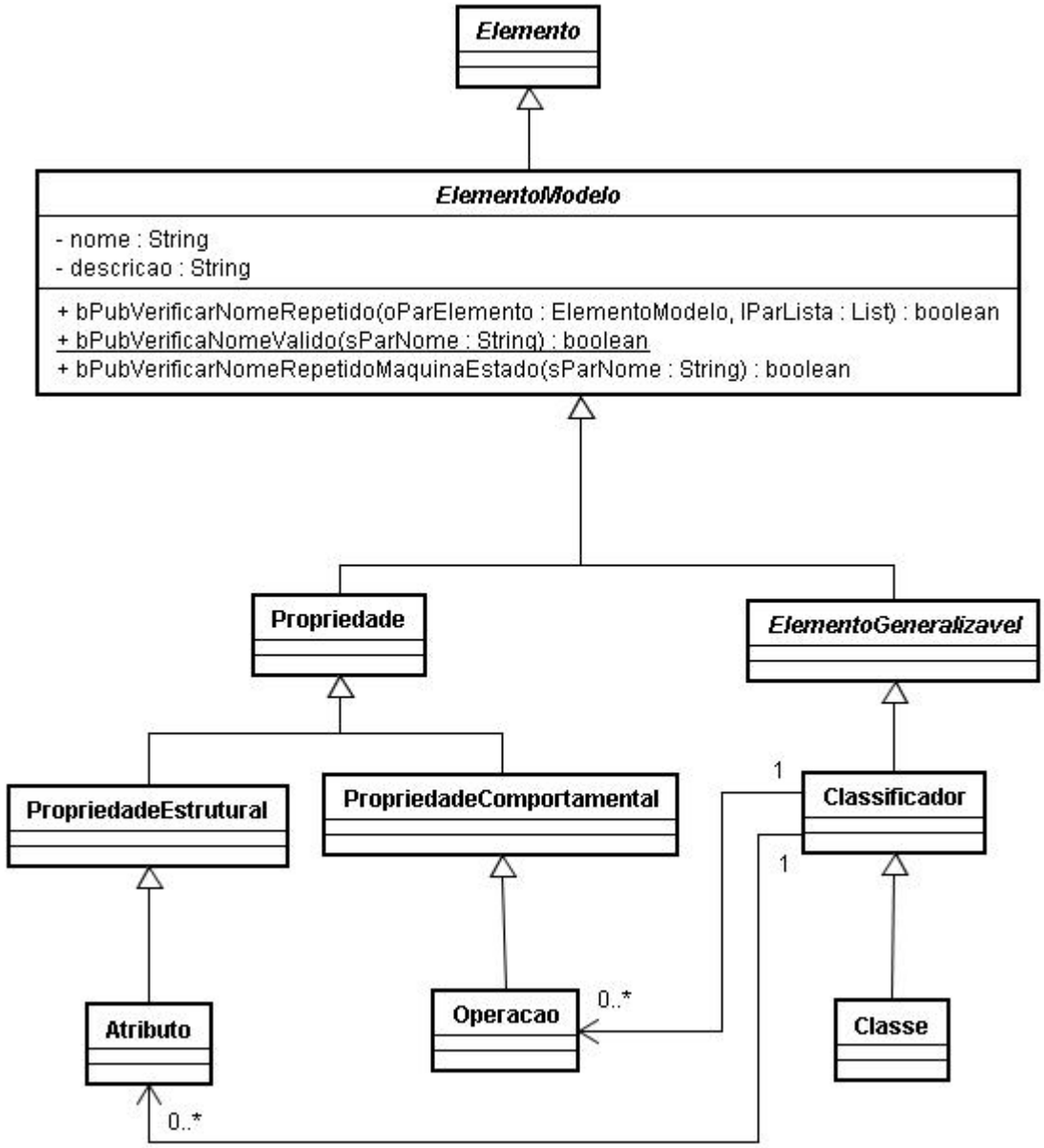


Figura 4 - Diagrama de Classes do pacote *cdp*

3.2 - Componente de Gerência de Dados (cgd)

A Figura 5 apresenta o diagrama de classes do Componente de Gerência de Dados do pacote *UML.Fundacao.Nucelo*, relativo à ferramenta de Cadastro de Elementos de Modelo. As classes criadas são baseadas no padrão DAO (*Data Access Object*) (SUN, 2006), conforme comentado na seção 2 deste documento.

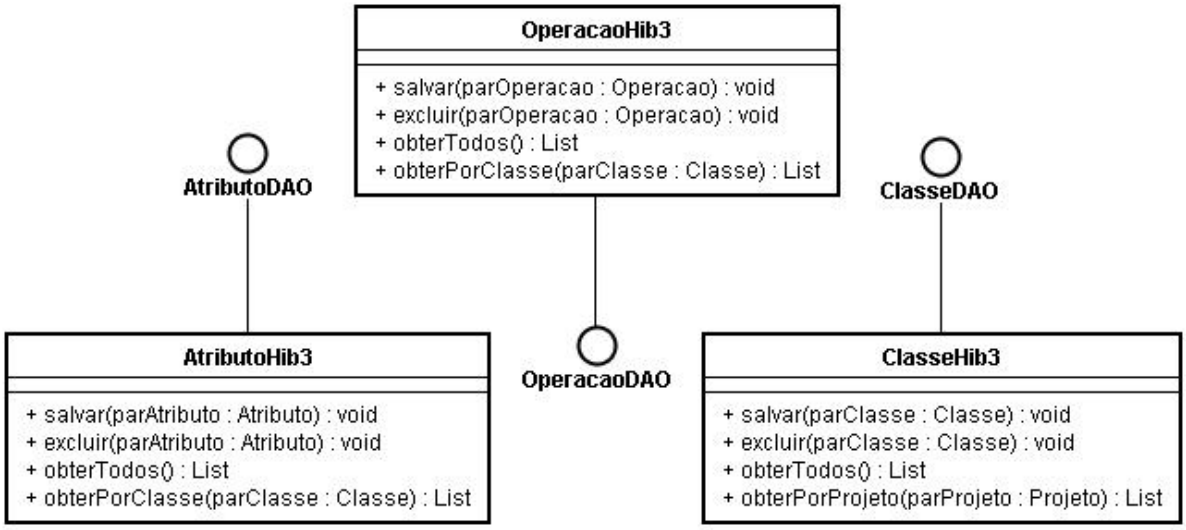


Figura 5 - Diagrama de Classes do pacote *cgd*

Foram criadas interfaces DAO e classes de persistência apenas para as classes concretas da hierarquia apresentada no diagrama da Figura 4, pois estas necessitavam serem feitas persistentes.

3.3 - Componente de Gerência de Tarefas (*cgt*)

A Figura 6 apresenta o diagrama de classes do Componente de Gerência de Tarefas do pacote *UML.Fundacao.Nucleo*, relativo à ferramenta de Cadastro de Elementos de Modelo.

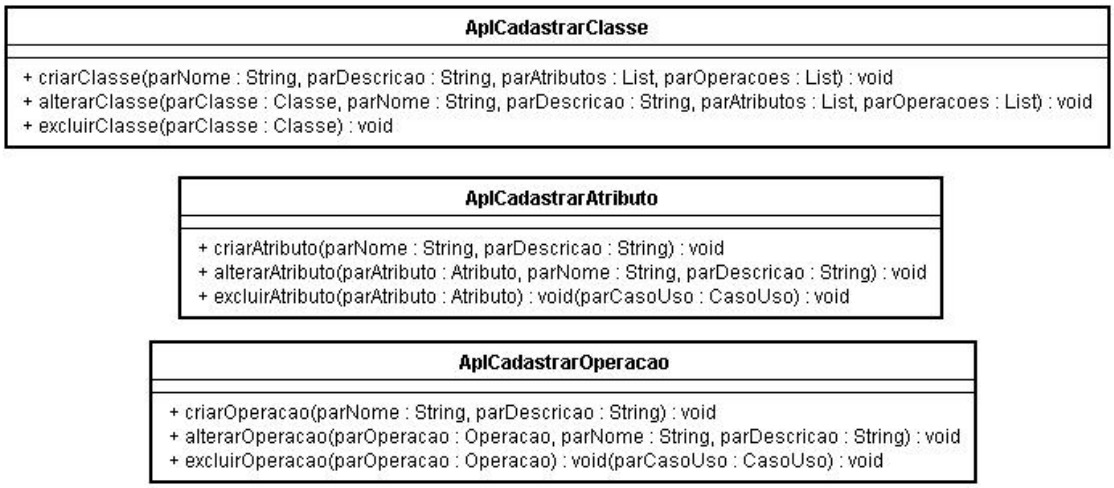


Figura 6 - Diagrama de Classes do pacote *cgt*

Foram criadas as classes de aplicação *AplCadastrarClasse, AplCadastrar Atributos e AplCadastrarOperacoes* para tratar, respectivamente, os casos de uso "Cadastrar Classe", "Cadastrar Atributo" e "Cadastrar Operação", descritos no Documento de Especificação de Requisitos da ferramenta de Cadastro de Elementos de Modelo.

3.4 - Componente de Interação Humana (cih)

A Figura 7 apresenta o diagrama de classes do Componente de Interação Humana do pacote *UML.Fundacao.Nucleo* relativo à ferramenta de Cadastro de Elementos de Modelo.

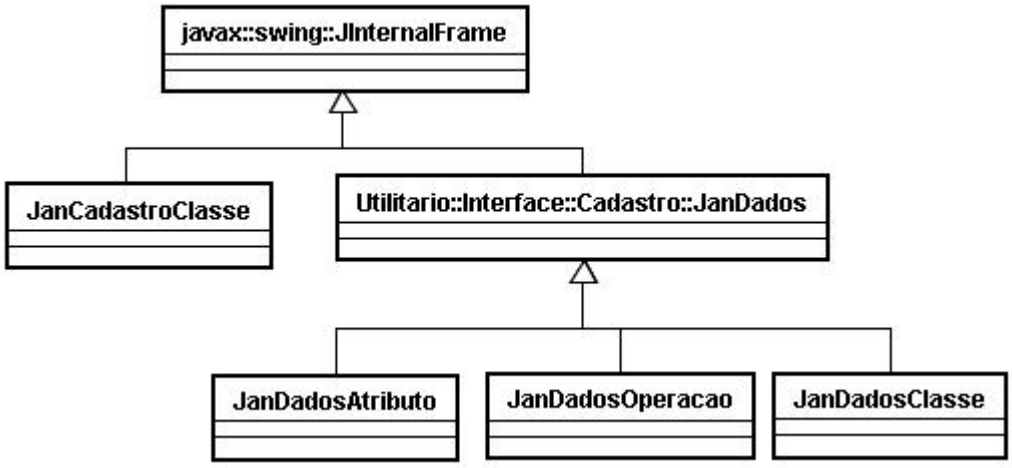


Figura 7 - Diagrama de Classes do pacote *cih*

A janela de cadastro de classes (*JanCadastroClasse*) exibe todas as classes cadastradas no sistema e disponibiliza as funcionalidades de criar uma nova classe, alterar, visualizar ou

excluir uma classe previamente selecionada. Para visualizar ou alterar os dados de uma classe, ou mesmo inserir informações de uma nova classe, foi criada a janela de dados *JanDadosClasse*. Ela aparece sempre que as opções de cadastro forem solicitadas na janela *JanCadastroClasse*. Em *JanDadosClasse* é possível visualizar os atributos e operações da classe selecionada e ativar as funcionalidades de cadastro desses itens, sendo possível, portanto, criar, alterar, consultar ou excluir um atributo ou operação. Para visualização ou inclusão de informações sobre atributos é acionada a janela *JanDadosAtributo* e para realizar o mesmo com operações, a janela acionada é *JanDadosOperacao.* Note que as janelas de dados *JanDadosClasse, JanDadosAtributo* e *JanDadosOperacao* herdam de *JanDados*, que se encontra no pacote *Utilitario.Interface.Cadastro.* Isto é feito visando à padronização e ao reúso.

3.5 - Componente de Controle de Interação (cci)

A Figura 8 apresenta o diagrama de classes do Componente de Controle de Interação do pacote *UML.Fundacao.Nucleo*, relativo à ferramenta de Cadastro de Elementos de Modelo.

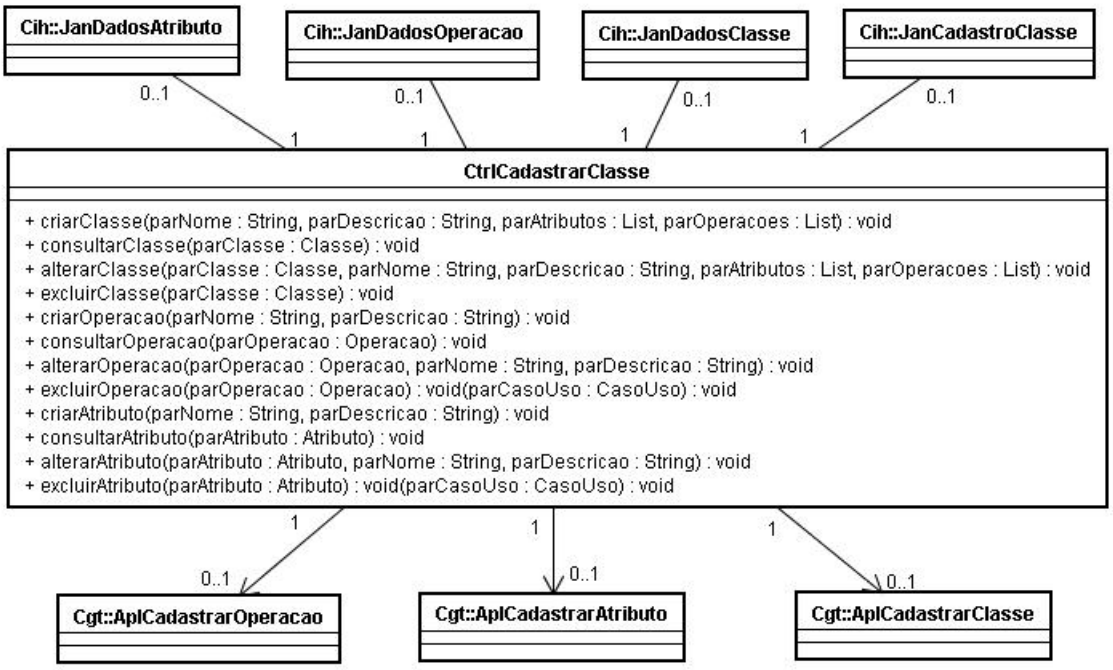


Figura 8 - Diagrama de Classes do pacote *cci*

A classe *CtrlCadastrarClasse* tem o papel principal de fazer a comunicação entre os objetos das camadas de aplicação (*cgt*) e de interface (*cih*) da porção da ferramenta de

Cadastro de Elementos de Modelo que lida com classes. Desta forma, as interfaces ficam isoladas, facilitando alterações. Permite, também, que as janelas não tenham acesso direto a métodos das classes de aplicação, isolando funcionalidades distintas.

A classe de controle apresentada tem seus relacionamentos com navegabilidade dupla com as classes de janelas do pacote *cih*, a saber *JanCadastroClassse, JanDadosClasse, JanDadosAtributo e JanDadosOperacao.* Desta forma o controlador pode exibir ou atualizar as janelas nos momentos devidos e as janelas podem invocar métodos do controlador para chamar outra janela ou mesmo para solicitar a realização de operações implementadas nas aplicações (*cgt*).

Os relacionamentos com as classes de aplicação só são navegáveis no sentido controlador - aplicação, ou seja, o controlador conhece as classes de aplicação, mas o contrário não é verdadeiro. Isto porque, com a arquitetura estabelecida, o controlador verifica os retornos da aplicação e invoca seus métodos.

4. Pacote *UML.ElementosComportamentais.CasoUso*

O pacote *UML.ElementosComportamentaisCasoUso* contém as classes que tratam de atores e casos de uso.

4.1 - Componente de Domínio do Problema (cdp)

A Figura 9 apresenta o diagrama de classes do Componente do Domínio do Problema do pacote *UML.ElementosComportamentaisCasoUso*, relativo à ferramenta de Cadastro de Elementos de Modelo. Como se pode notar comparando com o correspondente modelo da fase de análise, não houve mudanças significativas, sendo tratados apenas tipos de dados, navegabilidades e operações.

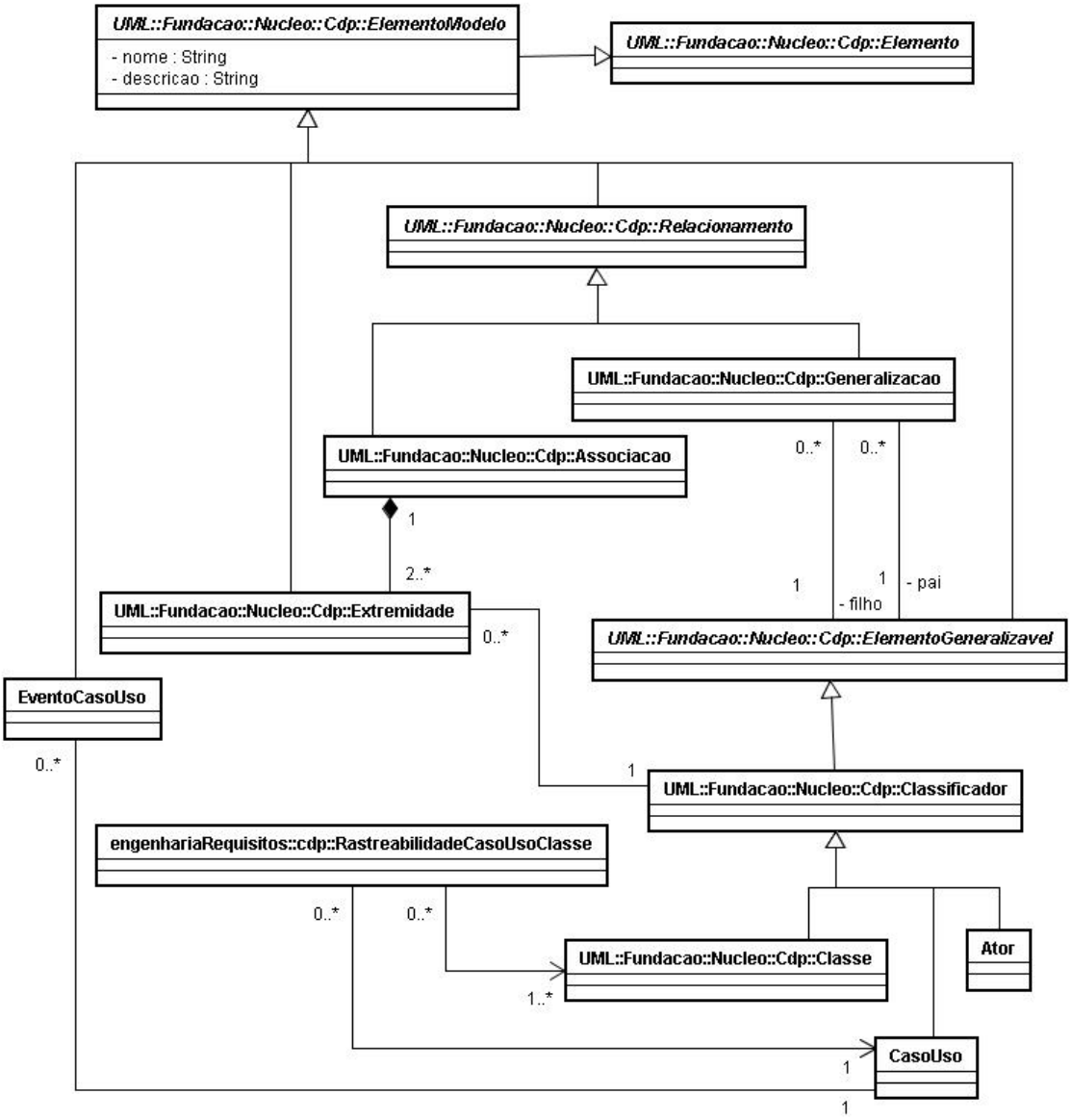


Figura 9 - Diagrama de Classes do pacote *cdp*

4.2 - Componente de Gerência de Dados (cgd)

A Figura 10 apresenta o diagrama de classes do Componente de Gerência de Dados do pacote *UML.ElementosComportamentaisCasoUso*, relativo à ferramenta de Cadastro de Elementos de Modelo. As classes criadas são baseadas no padrão DAO (*Data Access Object*) (SUN, 2006), conforme comentado na seção 2 deste documento.

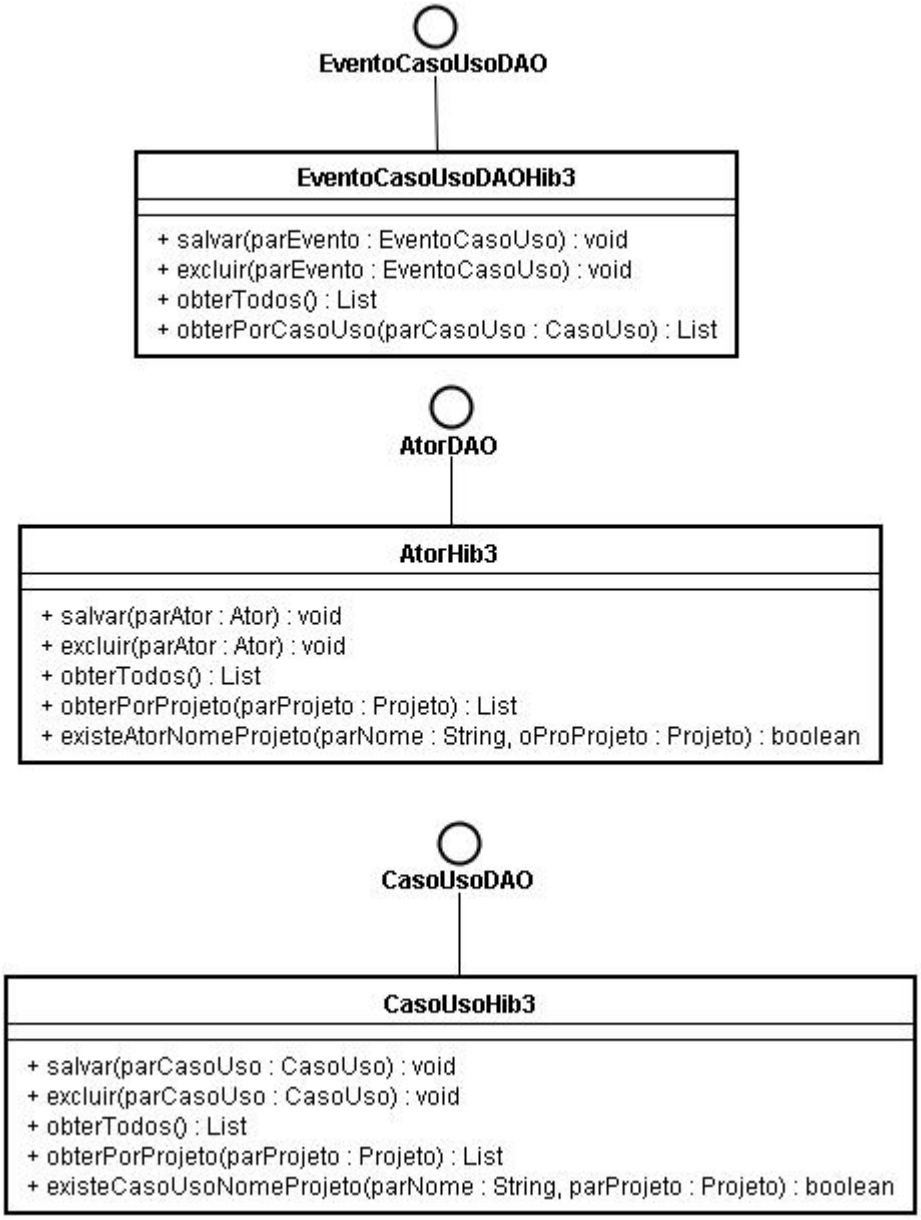


Figura 10 - Diagrama de Classes do pacote *cgd*

Criaram-se interfaces DAO e classes de persistência do Hibernate apenas para as classes que necessitavam serem persistentes, que correspondem às classes concretas da hierarquia apresentada no diagrama da Figura 9, a saber *CasoUso, Ator* e *EventoCasoUso.*

4.3 - Componente de Gerência de Tarefas (*cgt*)

A Figura 11 apresenta o diagrama de classes do Componente de Gerência de Tarefas do pacote *UML.ElementosComportamentais.CasoUso*, relativo à ferramenta de Cadastro de Elementos de Modelo.

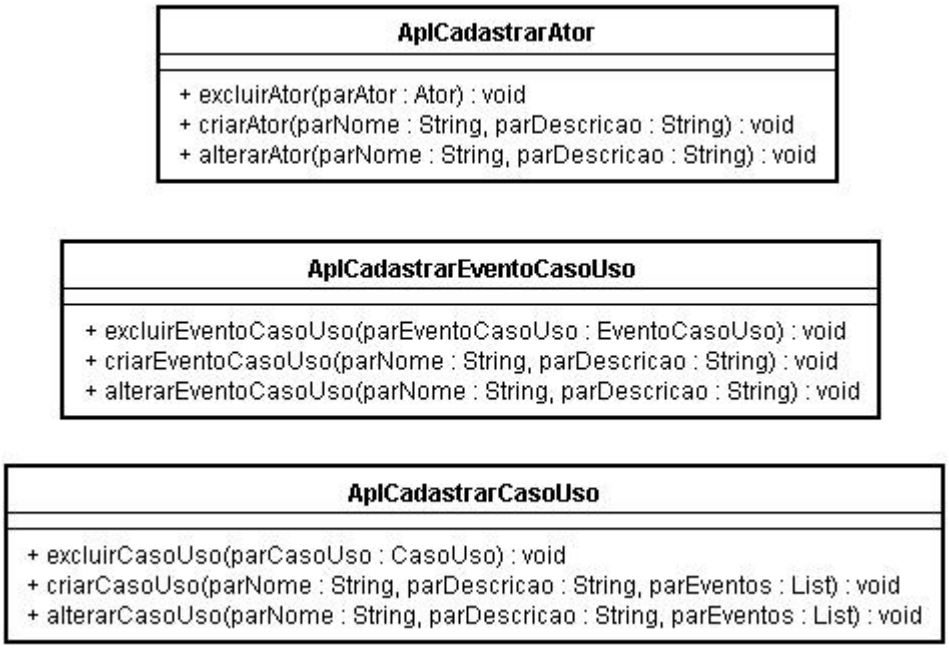


Figura 11 - Diagrama de Classes do pacote *cgt*

Foram criadas as classes de aplicação *AplCadastrarCasoUso, AplCadastrarEventoCaosUso e AplCadastrarAtor* para tratar, respectivamente, os casos de uso "Cadastrar Caso de Uso", "Cadastrar Evento de Caso de Uso" e "Cadastrar Ator", descritos no Documento de Especificação de Requisitos da ferramenta de Cadastro de Elementos de Modelo.

4.4 - Componente de Interação Humana (cih)

A Figura 12 apresenta o diagrama de classes do Componente de Interação Humana do pacote *UML.ElementosComportamentaisCasoUso* relativo à ferramenta de Cadastro de Elementos de Modelo.

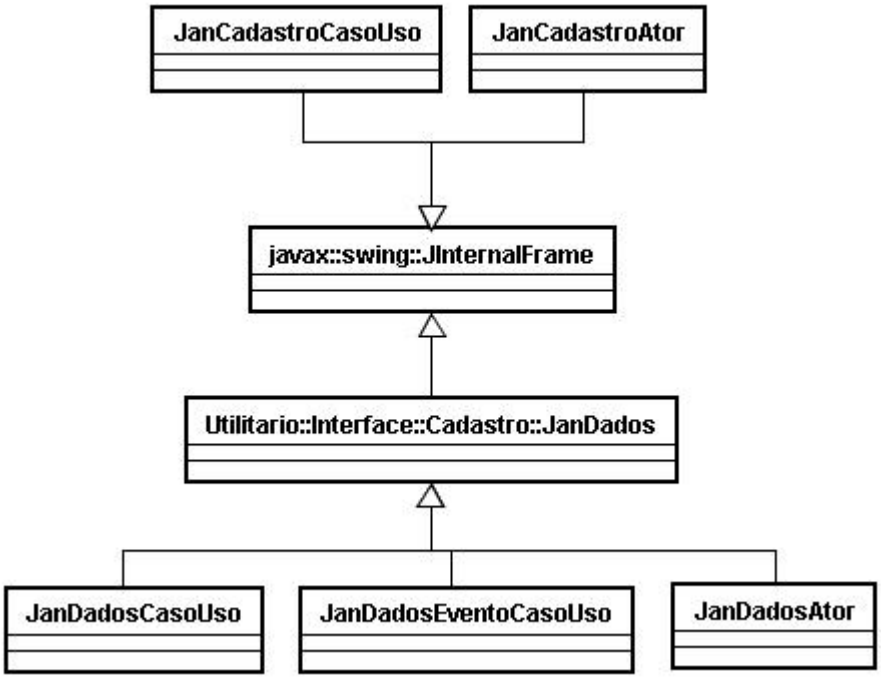


Figura 12 - Diagrama de Classes do pacote *cih*

A janela de cadastro de casos de uso (*JanCadastroCasoUso*) exibe todos os casos de uso cadastrados no sistema e disponibiliza as funcionalidades de criar um novo caso de uso, alterar, visualizar ou excluir um caso de uso previamente selecionado. Para visualizar ou alterar os dados de um caso de uso, ou mesmo inserir informações de um novo caso de uso, foi criada a janela de dados *JanDadosCasoUso*. Ela aparece sempre que as opções de cadastro forem solicitadas na janela *JanCadastroCasoUso*. Em *JanDadosCasoUso* é possível visualizar todos os eventos, atores e classes relacionadas ao caso de uso selecionado, sendo possível ativar as funcionalidades de cadastro desses itens, ou seja, criar, alterar, consultar ou excluir um evento, ator ou classe. Para visualização ou inclusão de informações sobre eventos é acionada a janela *JanDadosEventoCasoUso,* para atores a janela acionada é *JanDadosAtores* e para classes *JanDadosClasse*, descrita na seção 3.4. Note que as janelas de dados *JanDadosCasoUso, JanDadosEventoCasoUso* e *JanDadosAtor* herdam de *JanDados*, que se encontra no pacote *Utilitario.Interface.Cadastro.* Isto é feito visando à padronização e ao reúso.

Foi criada, também, a janela de cadastro de atores (*JanCadastroAtor*), que possui as opções de criar, alterar, consultar ou excluir um ator. Neste caso também é utilizada a janela de dados *JanDadosAtor* para inserção, alteração ou visualização dos dados de um ator.

4.5 - Componente de Controle de Interação (cci)

A Figura 13 apresenta o diagrama de classes do Componente de Controle de Interação do pacote *UMLElementosComportamentais.CasoUso*, relativo à ferramenta de Cadastro de Elementos de Modelo.

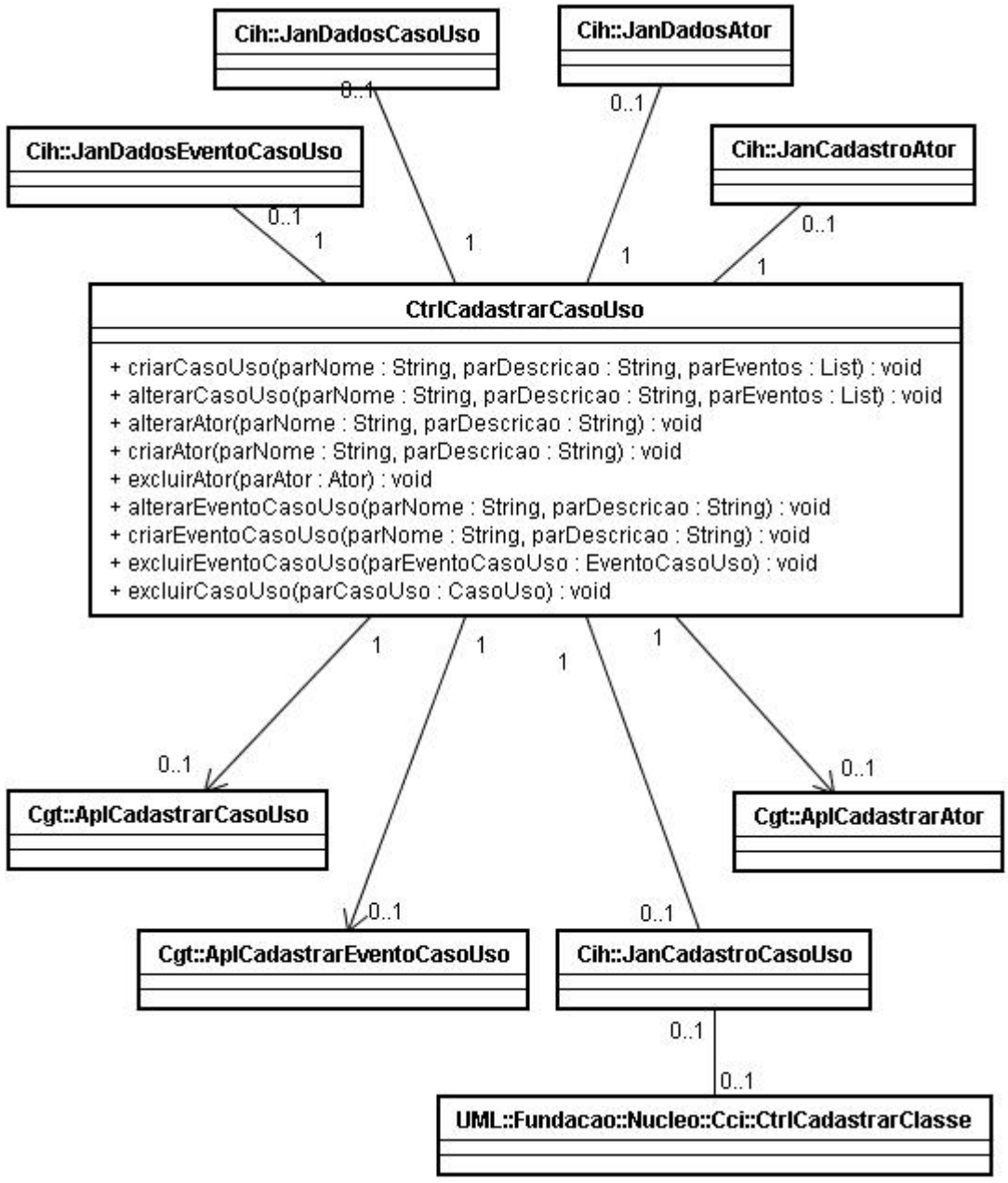


Figura 13 - Diagrama de Classes do pacote *cci*

A classe de controle *CtrlCadastrarCasoUso* é responsável pela comunicação entre os objetos das camadas de aplicação (*cgt*) e de interface (*cih*) da porção da ferramenta de Cadastro de Elementos de Modelo que lida com casos de uso. Ela apresenta relacionamentos de navegabilidade dupla com as janelas do pacote *cih*, a saber *JanCadastroCasoUso, JanDadosCasoUso, JanCadastroAtor, JanDadosAtor e JanDadosEventoCasoUso.* Desta

forma o controlador (*CtrlCadastrarCasoUso*) pode exibir ou atualizar as janelas nos momentos devidos e as janelas podem invocar métodos do controlador para chamar outra janela ou mesmo para solicitar a realização de operações implementadas nas aplicações (*cgt*).

Note que a janela de cadastro de casos de uso (*JanCadastroCasoUso*) tem relacionamento com o controlador *CtrlCadastrarClasse*. Isso é vantajoso, pois não é necessário adicionar código de cadastro de classes no controlador de cadastro de casos de uso (*CtrlCadastrarCasoUso*). Assim, caso seja necessário cadastrar uma classe no ato de cadastro de um caso de uso, a janela passa o controle para o controlador de cadastro de classes, que exibe as janelas necessárias e executa os devidos métodos da aplicação de cadastro de classes.

Os relacionamentos com as classes de aplicação só são navegáveis no sentido controlador - aplicação, ou seja, o controlador conhece as classes de aplicação, mas o contrário não é verdadeiro. Isto porque, com a arquitetura estabelecida, o controlador verifica os retornos da aplicação e invoca seus métodos.

5. Pacote *Artefato.Modelo*

A associação existente entre *ElementoModelo* e *Modelo* é de grande importância no contexto da ferramenta de Cadastro de Elementos de Modelo, pois ela permite descobrir a qual projeto um caso de uso, classe ou ator pertence. Para a implementação desta ferramenta, foram importantes apenas os pacotes *cdp* e *cgd* do pacote *Artefato.Modelo*, que são descritos a seguir.

5.1 - Componente de Domínio do Problema (cdp)

A Figura 14 apresenta o diagrama de classes do Componente do Domínio do Problema do pacote *Artefato.Modelo* relevante para a ferramenta de Cadastro de Elementos de Modelo. Como se pode notar comparando com o correspondente modelo da fase de análise, foram tratados tipos de dados, navegabilidades e operações e foi necessária a alteração do relacionamento entre *Modelo* e *ElementoModelo.* Note que o relacionamento passou para níveis mais baixos da hierarquia. Temos que o Modelo de Objetos conhece diretamente as classes (*Classe*), atributos (*Atributo*), operações (*Operacao*), generalizações (*Generalizacao*), casos de uso (*CasoUso*) e atores (*Ator*), que estão na base da hierarquia de *ElementoModelo.*

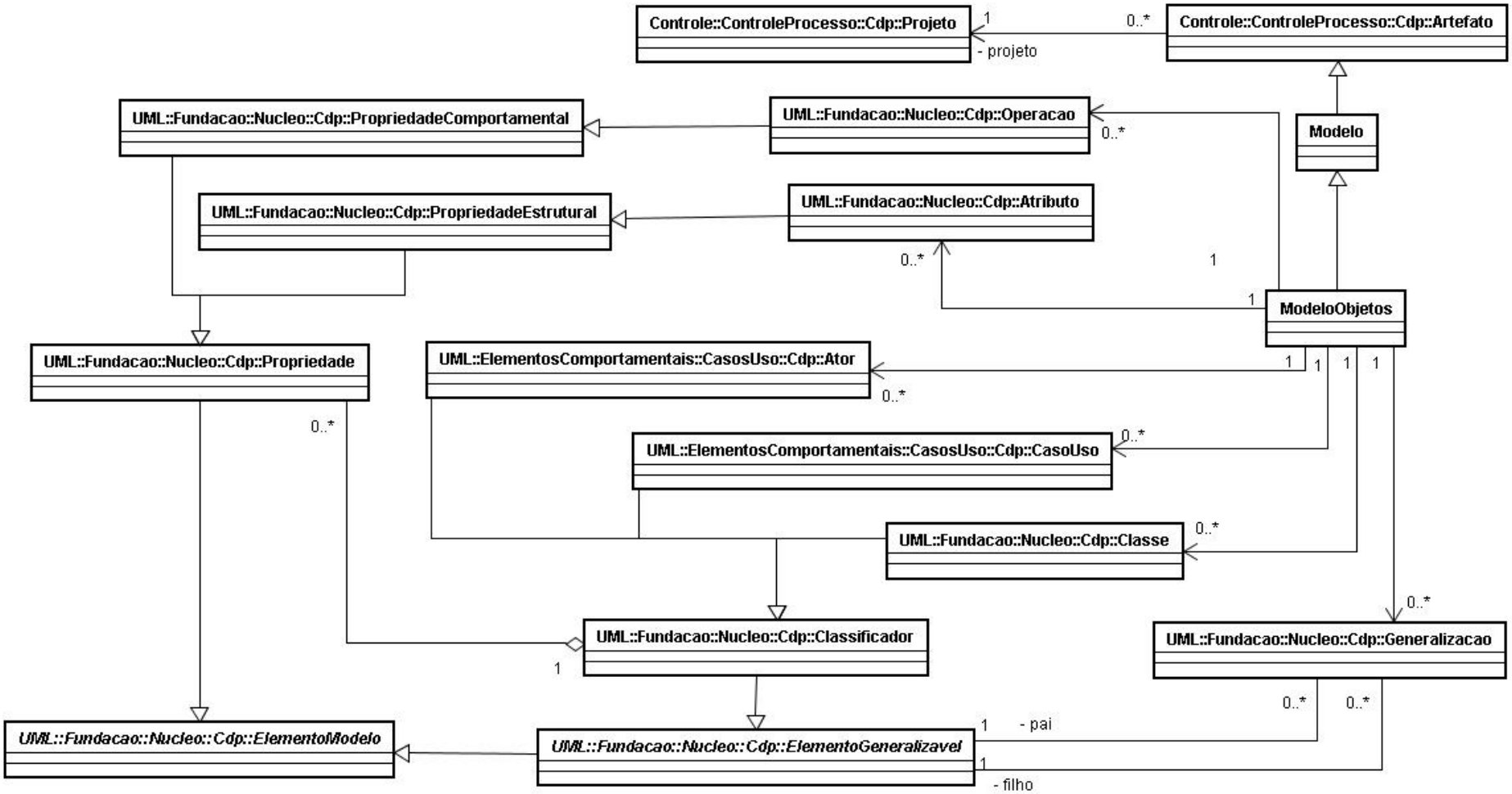


Figura 14 - Diagrama de Classes do pacote *cdp*

5.2 - Componente de Gerência de Dados (cgd)

A Figura 15 apresenta o diagrama de classes do Componente de Gerência de Dados do pacote *Artefato.Modelo*, relativo à ferramenta de Cadastro de Elementos de Modelo As classes criadas são baseadas no padrão DAO (DAO, 2005), conforme comentado na seção 2 deste documento.

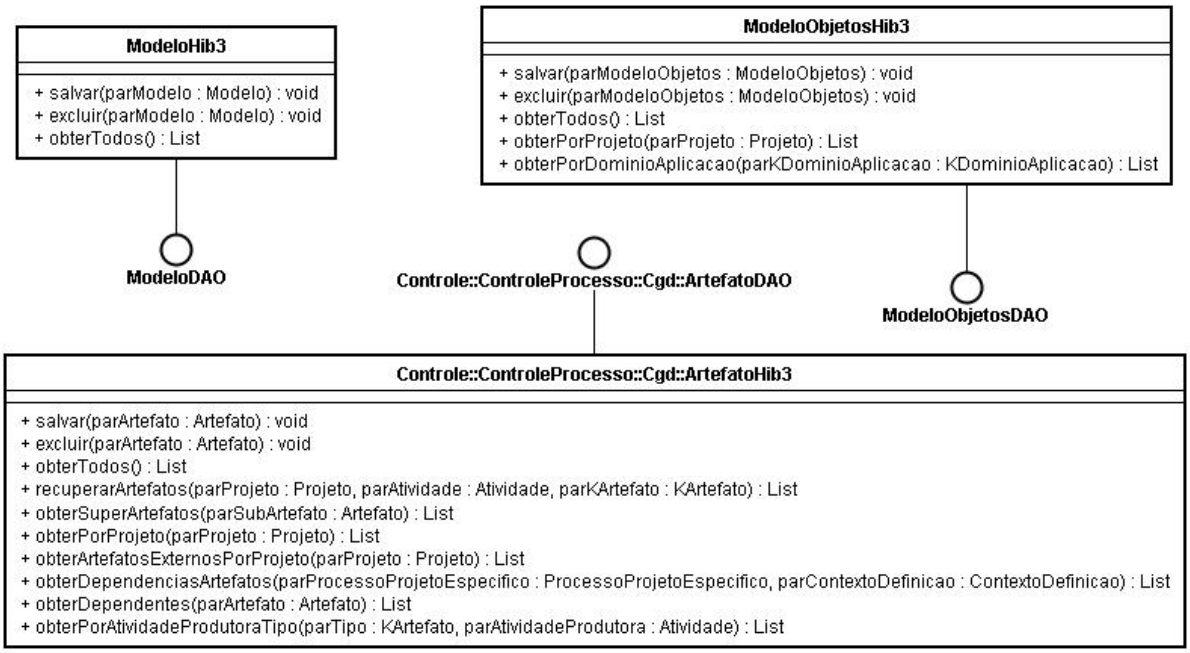


Figura 15 - Diagrama de Classes do pacote *cgd*

Referências Bibliográficas

BAUER,C.; KING, G. **Hibernate em Ação**. 1 ed. Rio de Janeiro, RJ: Ciência Moderna, 2005.

FALBO R.A., NATALI, A.C.C., MIAN, P.G., BERTOLLO, G., RUY, F.B. **ODE: Ontology-based software Development Environment**, Proceedings of the IX Argentine Congress on Computer Science (CACIC'2003), La Plata, Argentina, 2003, pp 1124-1135

SUN DEVELOPER NETWORK, **Data Access Object**. Disponível em: <java.sun.com/j2ee/patterns/DataAccessObject.html>. Acesso em: 8 jan. 2007.

XDOCLET Attribute Oriented Programming. Disponível em: <xdoclet.sourceforge.net>. Acesso em: 3 jan. 2007.

Anexo C

DOCUMENTAÇÃO DO PROJETO DA FERRAMENTA REQODE

Documento de Especificação de Requisitos Funcionais

Projeto: ReqODE – Ferramenta de Apoio à Engenharia de Requisitos

Responsável: Aline Freitas Martins

1. Introdução

Este documento apresenta a especificação de requisitos para a ferramenta de apoio à Engenharia de Requisitos. Essa atividade foi conduzida usando a técnica de Modelagem de Casos de Uso e, portanto, este documento contém uma descrição do propósito do sistema (descrição do mini-mundo), apresentada na seção 2, e o modelo de casos de uso, incluindo diagramas de casos de uso (um para cada subsistema) associados às descrições dos casos de uso (seção 3).

2. Descrição do Mini-Mundo

A Engenharia de Requisitos é o ramo da Engenharia de Software que envolve todas as atividades relativas a desenvolver, documentar e dar manutenção ao conjunto de requisitos de um sistema. O processo de Engenharia de Requisitos possui como principais atividades (Kotonya e Sommerville, 1998):

- Levantamento de requisitos: envolve a descoberta dos requisitos. É uma atividade complexa que pode ser feita utilizando diversas técnicas, tais como entrevistas, questionários, análise de documentos e/ou observação;
- Análise de Requisitos: tem o objetivo de garantir que os requisitos estejam consistentes, sem ambigüidades e problemas. Para tal, diversos tipos de modelos e diagramas podem ser usados, tais como Modelos de Entidades e Relacionamentos, Diagramas de Classes, Diagramas de Estado etc.
- Documentação de Requisitos: oficializa formalmente os resultados do processo de engenharia de requisitos, documentando os requisitos capturados e modelados em um Documento de Especificação de Requisitos.
- Verificação e Validação de Requisitos: verifica se os artefatos gerados estão em conformidade com as necessidades do usuário.

- Gerência de Requisitos: corresponde ao controle de mudanças, objetivando manter a consistência entre documentos, modelos e os requisitos alterados. Sendo usado o mecanismo de rastreabilidade para manter essa consistência.

Neste contexto surgem muitas informações acerca de um requisito e estas devem ser mantidas para que esse processo seja eficiente. Dentre elas, destacam-se:

- Identificador: gerado automaticamente para controle e identificação de um requisito.
- Sentença: descreve em poucas palavras do que trata o requisito.
- Descrição: descrição detalhada do requisito.
- Data da criação: data em que o requisito foi registrado.
- Prioridade: indica o nível de importância do requisito.
- Estado: informa em qual etapa encontra-se o desenvolvimento do requisito.
- Razões da Criação: justificativas e importância do requisito.
- Comentários Gerais: qualquer outra informação acerca do requisito que seja relevante para o processo de desenvolvimento.
- Dependentes: requisitos que dependem do requisito em questão.
- Tipo Primitivo: identifica se o requisito é funcional ou não funcional.
- Conflitos: Requisitos que conflitam, tendo que ser escolhidos quais devem ser priorizados.
- Projeto: projeto ao qual pertence o requisito.
- Tipo do Requisito: uma classificação organizacional mais detalhada do que o tipo primitivo (funcional ou não funcional).
- Recursos Humanos: incluindo os interessados (*stakeholders*) no requisito e os responsáveis pelo mesmo.
- Contexto de origem: define o contexto em que o requisito foi capturado.
- Casos de Uso: casos de uso que foram derivados do requisito.
- Classe: classes que implementam um requisito.
- Artefatos: artefatos que foram gerados durante o desenvolvimento tendo como base o requisito.

A Engenharia de Requisitos é um processo complexo e por isso é importante prover uma ferramenta para apoiá-lo. Para um melhor controle de mudanças, é importante que essa ferramenta esteja integrada às demais utilizadas no desenvolvimento de software. Neste documento apresentamos a Especificação de Requisitos de ReqODE, uma ferramenta de

apoio à Engenharia de Requisitos, integrada a ODE (*Ontology-based software Development Environment*) (FALBO et al., 2003), um Ambiente de Desenvolvimento de Software (ADS) Centrado em Processo.

3. Modelo de Casos de Uso

No contexto da ferramenta de apoio à Engenharia de Requisitos foram definidos dois sub-sistema principais: *engenhariaRequisitos* e *contexto*, como mostra a Figura 1.

Figura 1 - Diagrama de pacotes

3.1 Modelo de Casos de Uso do pacote *engenheriaRequisitos*

A Figura 2 apresenta o diagrama de casos de uso para o pacote *engenhariaRequisitos*, cujos casos de uso são descritos na seqüência.

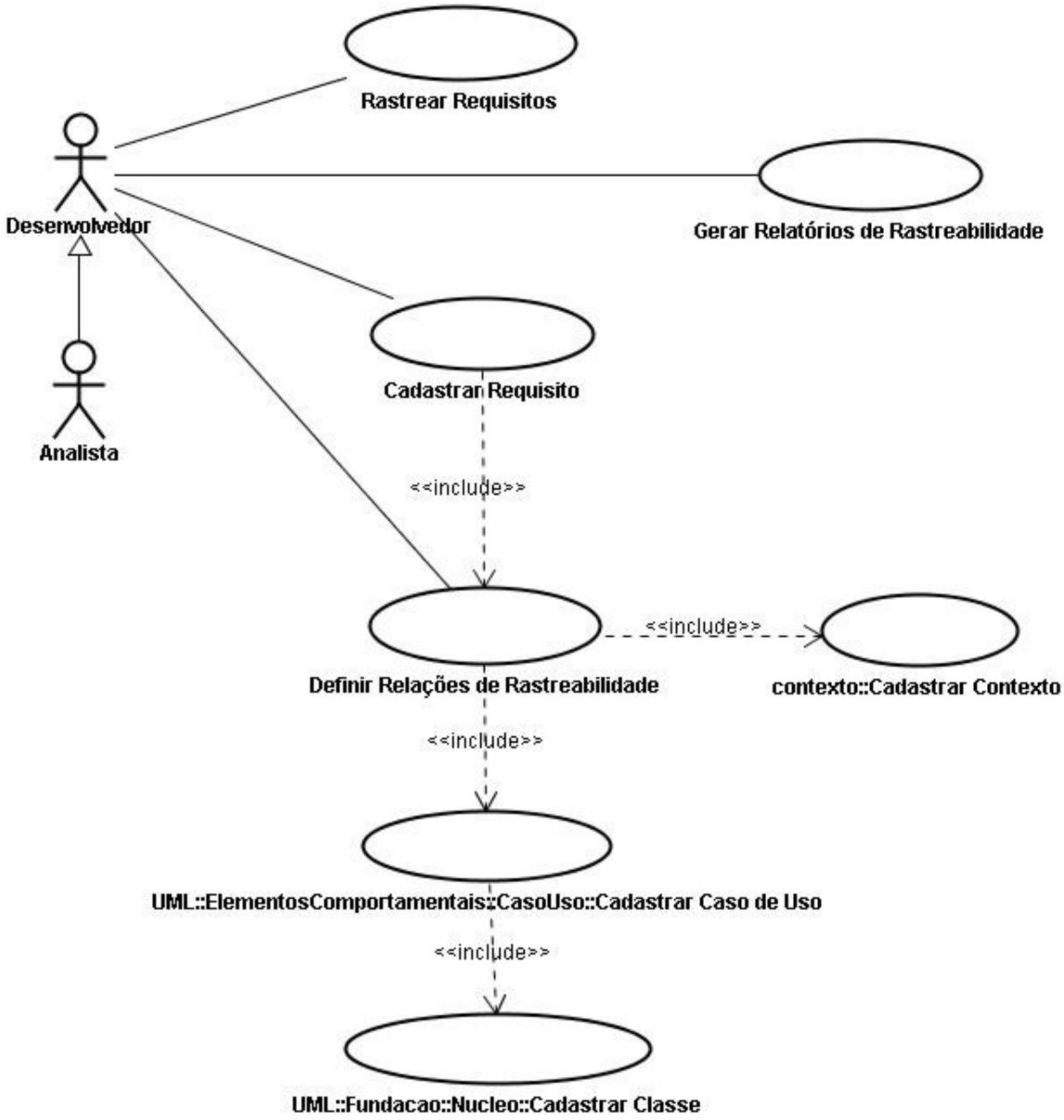


Figura 2 - Diagrama de casos de uso do pacote *engenhariaRequisitos*.

No processo de engenharia de requisitos encontramos como principais atores o analista e o desenvolvedor, como mostra a Figura 2. O desenvolvedor pode realizar os eventos dos casos de uso mais gerais, que envolvem consulta de informações sobre requisitos, geração de documentos e cadastro de contexto de origem, sendo que o analista, por herdar de desenvolvedor, também pode realizar esses eventos. Porém, além dessas atividades, o analista tem a permissão de cadastrar requisitos, associá-lo a outros elementos do desenvolvimento e pode, ainda, criar alguns desses elementos. Nas seções que se seguem são apresentadas as descrições dos casos de uso e seus respectivos eventos, deixando explícito qual ator pode realizá-los.

Sub-sistema: engenhariaRequisitos
Caso de Uso: Cadastrar Requisitos
Descrição: Este caso de uso permite que o analista crie um requisito, alterar dados gerais de um requisito e exclua um requisito.

Cursos Normais:

Criar Requisito

O analista informa a sentença, descrição, o tipo de requisito (se é funcional, não funcional e uma classificação organizacional), o requisito do qual ele é derivado, dito super-requisito, a prioridade, o responsável por sua criação, os recursos humanos interessados (*stakeholders*), comentários gerais e as razões de sua criação. Caso os dados informados sejam válidos, o requisito é criado. Neste momento um identificador único para ele é gerado, sua data de criação é definida como a data corrente, o estado do requisito é considerado "proposto" e ele é associado ao escopo do projeto corrente.

O identificador do requisito é gerado da seguinte maneira: <prefixo><nº super-requisito>.<nº seqüencial de requisitos derivados no super-requisito>, onde prefixo trata-se de RF se o requisito for funcional e RNF se for não funcional.

Caso deseje, o analista pode definir relações úteis para a rastreabilidade, realizando o caso de uso *Definir Relações de Rastreabilidade*.

Consultar Dados de Requisito

O desenvolvedor seleciona um requisito e são exibidas as seguintes informações: identificador, sentença, descrição, data de criação, prioridade, estado, tipo, comentários gerais, razões de criação, responsável, interessados, contexto de criação, casos de uso, classes, módulos, requisitos dependentes, requisitos que o compõem, requisitos conflitantes e artefatos relacionados.

Alterar Dados Gerais de Requisito

Dado um requisito, o analista informa os novos dados (sentença, descrição, tipo do requisito, prioridade, razões de criação, responsável, estado, interessados e comentários

gerais) e solicita a alteração do requisito. Caso os dados informados sejam válidos, a alteração é registrada.

Excluir Requisito:

Dado um requisito, o analista solicita sua exclusão. Uma mensagem de confirmação é exibida e, caso o analista confirme, o requisito é excluído, sendo também excluídos os requisitos que o compõem.

Restrições

1) O responsável por um requisito deve estar alocado a uma equipe do projeto no qual o requisito foi definido.
2) Os interessados em um requisito devem estar alocados a uma equipe do projeto no qual o requisito foi definido.

Cursos Alternativos:

Criar / Alterar Requisito

Caso a sentença do requisito não seja preenchida, uma mensagem é exibida avisando ao analista que esse campo deve ser preenchido.

Caso o tipo primitivo (Funcional e Não Funcional) não seja escolhido, uma mensagem é exibida avisando ao analista que a escolha de um tipo é obrigatória.

Classes: Requisito, KTipoRequisito, RecursoHumano, Modulo, Contexto, Escopo, CasoUso, Classe, Artefato.

Sub-sistema: engenhariaRequisitos

Caso de Uso: Definir Relações de Rastreabilidade

Descrição: Esse caso de uso permite definir as ligações utilizadas para manter a rastreabilidade entre requisitos e entre requisitos e artefatos gerados ao longo do processo de software.

Cursos Normais:

Definir Dependências

Dado um requisito, o analista informa de quais requisitos ele depende.

Definir Conflitos

Dado um requisito, o analista informa os requisitos que são conflitantes com o mesmo.

Definir Alocação a Módulo

Dado um requisito, o analista seleciona, dentre os módulos do projeto corrente, a quais o requisito está alocado.

Definir Contexto de Origem

O analista seleciona um contexto de origem para um dado requisito, os dados desse contexto são apresentados (nome, descrição, atividade, participantes e artefatos analisados) e caso confirmado, o contexto é definido como contexto de origem do requisito.

Caso o analista deseje criar um novo contexto de origem para um requisito ou alterar um contexto existente, ele pode realizar o caso de uso *Cadastrar Contexto*.

Definir Ligações com Caso de Uso

Dado um requisito, o analista informa, dentre os casos de uso do projeto corrente, quis tratam o requisito. Caso o analista deseje criar um novo caso de uso, alterar ou excluir um caso de uso existente, ele pode realizar o caso de uso *Cadastrar Caso de Uso*.

Definir Ligações com Artefatos

Dado um requisito, o analista seleciona, dentre os artefatos do projeto corrente, tais como Documento de Especificação de Análise, Documentação de Especificação de Projeto e Código Fonte, quais possuem relação com aquele requisito.

Definir Ligações com Classes

Dado um requisito, o analista seleciona, dentre as classes referentes aos casos de usos já relacionados, aquelas que implementam o requisito informado.

Classes: Requisito, RecursoHumano, CasoUso, Artefato, Contexto, Modulo, Escopo, Projeto, Classe, RastreabilidadeCasoUsoClasses.

Sub-sistema: engenhariaRequisitos **Caso de Uso:** Gerar Relatórios de Ratreabilidade **Descrição:** Esse caso de uso trata da geração de relatórios de rastreabilidade.

.

Cursos Normais:

Gerar Relatório Completo:

O desenvolvedor solicita a geração do relatório completo que é, então, gerado. Neste relatório são informados o projeto e todos os seus requisitos. Para cada requisito, as seguintes informações são apresentadas:

- Identificador
- Sentença
- Descrição
- Tipo do Requisito
- Data da criação
- Prioridade
- Estado
- Responsável
- Interessados
- Razões da Criação
- Comentários Gerais
- Casos de uso relacionados.
- Classes relacionadas.
- Artefatos relacionados e classificados por tipo de artefato (Especificação de Análise e Especificação de Projeto, por exemplo).
- Requisitos dependentes.
- Requisitos conflitantes.
- Requisitos que o compõem.
- Módulos que tratam os requisitos

Gerar Relatório Customizado:

O desenvolvedor seleciona, dentre as informações abaixo, quais ele quer que apareça no relatório e o relatório é gerado.

- Identificador
- Sentença
- Descrição
- Tipo do Requisito
- Projeto
- Data da criação
- Prioridade
- Estado
- Interessados
- Responsável
- Razões da Criação
- Comentários Gerais
- Casos de uso relacionados
- Classes relacionadas
- Artefatos relacionados
- Requisitos dependentes
- Requisitos conflitantes
- Requisitos que o compõem
- Módulos que tratam os Requisitos

Classes: Requisito, RecursoHumano, CasoUso, Artefato, Contexto, Modulo, Escopo, Projeto, Classe, RastreabilidadeCasoUsoClasses.

Sub-sistema: engenhariaRequisitos

Caso de Uso: Rastrear Requisitos

Descrição: Esse caso de uso permite buscar requisitos de modo a garantir a rastreabilidade bidirecional e a identificação de pontos de mudança durante o processo de software.

Cursos Normais:

Buscar Requisitos por Caso de Uso

O sistema informa os casos de uso que possuem ligações com algum requisito do projeto corrente. O desenvolvedor seleciona um dentre esses e o sistema, então, apresenta os requisitos que estão associados a ele. Para tornar mais fácil a localização do caso de uso deve ser possível filtrar, informando o modelo de objetos ao qual pertence.

Buscar Requisitos por Classe

O sistema informa quais as classes implementam algum requisito do projeto corrente. O desenvolvedor seleciona uma dessas classes e o sistema apresenta os requisitos que ela implementa. È possível, também, filtrar as classes informando a qual caso de uso ela está relacionada ou o modelo de objetos a que ela pertence.

Buscar Requisitos por Contexto de Criação

O desenvolvedor informa um contexto e o sistema apresenta os requisitos que foram derivados neste contexto. O sistema deve prover uma forma de filtrar os contextos de criação por meio de uma atividade geradora, um artefato de origem e/ou um recurso humano presente na criação do requisito.

Buscar Requisitos por Artefato Gerado

O sistema informa quais artefatos foram originados a partir de requisitos do projeto corrente, sendo possível filtrar os artefatos exibidos informando o processo específico, a atividade produtora e/ou o tipo de artefato. O desenvolvedor seleciona um dentre esses e o sistema apresenta os requisitos que foram base para a elaboração daquele artefato.

Buscar Requisitos por Responsável

O sistema informa os recursos humanos que são responsáveis por quaisquer dos requisitos cadastrados para o projeto corrente. É possível, ainda, informar um tipo de recurso humano para filtrar os recursos humanos a serem listados. O desenvolvedor, então, seleciona um dentre esses recursos humanos e o sistema apresenta os requisitos pelos quais ele é responsável.

Buscar Requisitos por Interessados

O sistema informa os recursos humanos que são interessados por quaisquer dos requisitos cadastrados para o projeto corrente. É possível, ainda, informar um tipo de recurso humano para filtrar os recursos humanos a serem listados. O desenvolvedor, então, seleciona um recurso humano e o sistema apresenta os requisitos em que ele figura dentre os interessados.

Buscar Requisitos de um Módulo

O desenvolvedor seleciona um módulo do projeto e o sistema apresenta os requisitos associados a ele.

Classes: Requisito, RecursoHumano, CasoUso, Artefato, Classe, Contexto, Modulo, Projeto, Escopo.

3.2 Modelo de Casos de Uso do pacote *contexto*

A Figura 3 apresenta o diagrama de casos de uso para o pacote *contexto*, cujos casos de uso são descritos na seqüência.

Figura 3 - Diagrama de casos de uso do pacote *contexto*.

Sub-sistema: contexto **Caso de Uso:** Cadastrar Contexto **Descrição:** Esse caso de uso permite criar, alterar, consultar e excluir contextos.

Cursos Normais:

Criar Contexto

O desenvolvedor informa o nome, a descrição, a atividade, os participantes e os artefatos analisados. Caso os dados sejam válidos (o nome não pode estar em branco), um novo contexto é criado.

Alterar Contexto

Dado um contexto, o desenvolvedor informa os novos dados (nome, descrição, atividade, participantes e artefatos analisados) e solicita a alteração do contexto. Caso os dados informados sejam válidos, a alteração é registrada.

Consultar Contexto

O desenvolvedor informa um contexto e os dados do mesmo são apresentados.

Excluir Contexto

Dado um contexto, o desenvolvedor solicita sua exclusão. Uma mensagem de confirmação é exibida e, caso o usuário confirme, o contexto é excluído.

Cursos Alternativos

Criar / Alterar Contexto

Se o nome estiver em branco e o desenvolvedor solicitar o salvamento do contexto, o sistema mostrará uma mensagem para lembrá-lo de preencher este campo.

Se já existir um contexto no projeto com este nome, uma mensagem será exibida avisando ao desenvolvedor e solicitando que ele informe outro nome.

Excluir Contexto

Se o contexto estiver associado a um requisito não será permitida a sua exclusão.

Classes: Contexto, Atividade, Artefato, RecursoHumano.

Referências Bibliográficas

KOTONYA, G., SOMMERVILLE, I. **Requirements engineering: processes and techniques**. Chichester, England: John Wiley, 1998.

FALBO R.A., NATALI, A.C.C., MIAN, P.G., BERTOLLO, G., RUY, F.B. **ODE: Ontology-based software Development Environment**, Proceedings of the IX Argentine Congress on Computer Science (CACIC'2003), La Plata, Argentina, 2003, pp 1124-1135.

Documento de Especificação de Análise

Projeto: ReqODE – Ferramenta de Apoio à Engenharia de Requisitos

Responsável: Aline Freitas Martins

1. Introdução

Este documento apresenta a especificação de análise para a ferramenta de apoio à Engenharia de Requisitos integrada a ODE (*Ontology-based software Development Environment*) (FALBO et al., 2003). Essa atividade foi conduzida seguindo o método da Análise Orientada a Objetos, tendo sido elaborados diagramas de classes, apresentados na seção 2, e um diagrama de estados, apresentado na seção 3, todos utilizando a notação da UML.

2. Modelo de Classes

O diagrama de pacotes da Figura 1 mostra as dependências existentes entre os pacotes contemplados nessa ferramenta.

O pacote principal da ferramenta é o pacote *engenhariaRequisitos*, que contém as classes que, de fato, guardam as informações dos requisitos. Outro pacote desenvolvido neste trabalho é o pacote *contexto*, utilizado para registrar o contexto em que um requisito foi criado. O restante dos pacotes já existia em ODE e os pacotes *engenhariaRequisitos* e *contexto* apresentam as dependências mostradas na Figura 1, mantendo a integração com o ambiente e permitindo manter associações com os demais elementos já tratados no mesmo.

2.1 Pacote *engenhariaRequisitos*

A Figura 2 mostra o diagrama de classe do pacote *engenhariaRequisito*. Esse diagrama aponta que requisitos podem ser decompostos em outros requisitos, bem como podem ser dependentes ou estar em conflito com outros requisitos. Visando à rastreabilidade, requisitos podem ser associados a casos de uso, classes e artefatos produzidos em um projeto, e a um contexto no qual o mesmo foi levantado (o que inclui a atividade do processo de software, documentos analisados e participantes que atuaram em sua identificação). Por fim, para

facilitar o gerenciamento do projeto, requisitos são alocados a módulos que definem o escopo do projeto e recursos humanos são indicados como responsáveis e interessados.

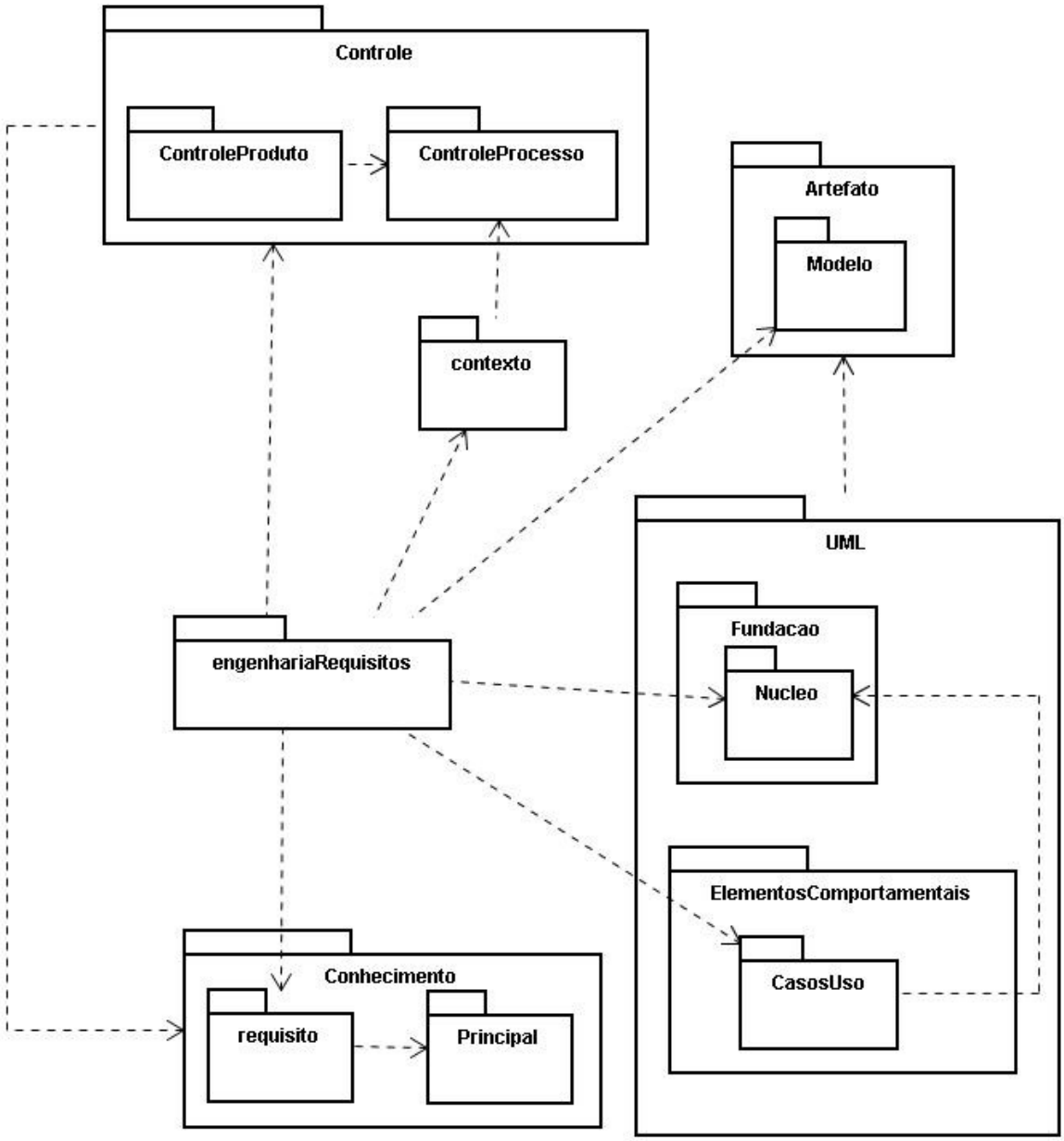


Figura 1 - Diagrama de pacotes

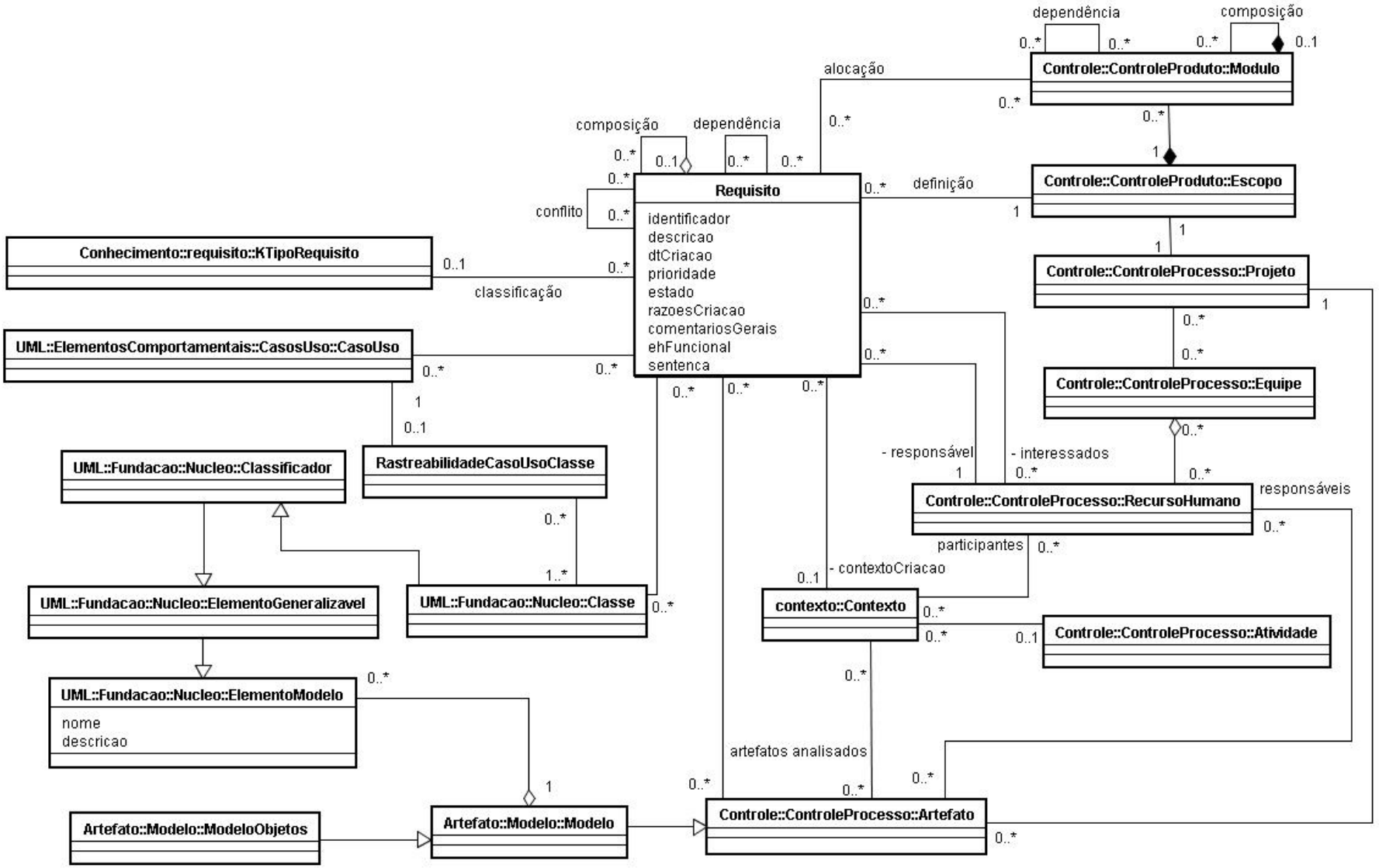


Figura 2 - Diagrama de Classes do pacote *engenhariaRequisito*.

2.2 Pacote *contexto*

A Figura 3 apresenta o diagrama de classe do pacote *contexto*. Neste estão especificadas as classes que permitem guardar informações sobre o contexto em que um requisito foi levantado, incluindo a atividade (*Atividade*) do processo de software, os documentos (*Artefato*) analisados e os participantes (*RecursoHumano*) que atuaram na identificação de um requisito.

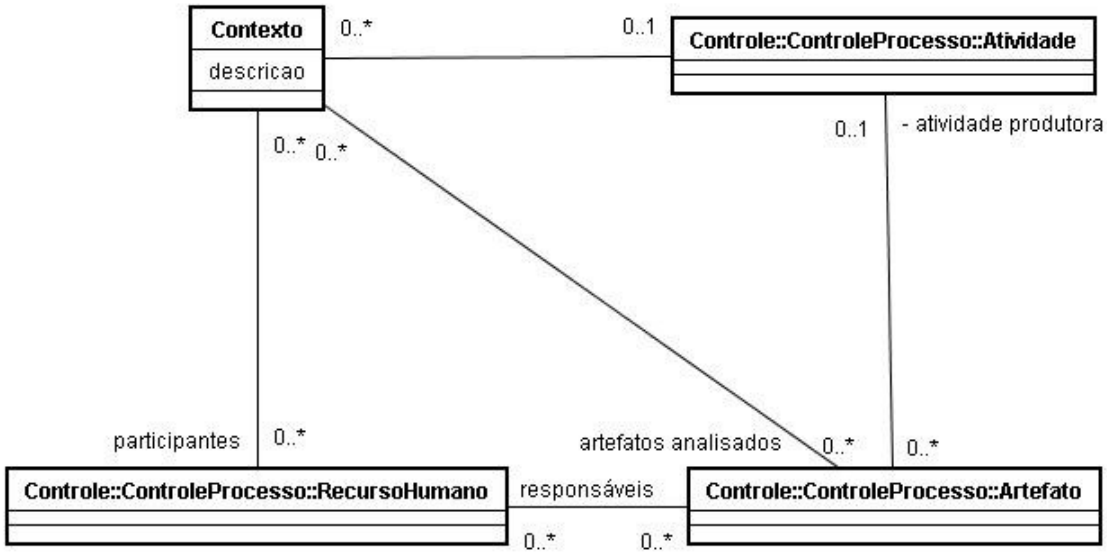


Figura 3 - Diagrama de Classes do pacote *contexto*.

3. Diagrama de Estados

Durante todo o processo de engenharia de requisitos um requisito pode sofrer alterações. Ele é criado durante o levantamento de requisitos e pode sofrer alterações ou pode mesmo ser rejeitado durante outras atividades, como verificação e validação, por exemplo. A Figura 4 mostra o diagrama de estados da classe *Requisito*.

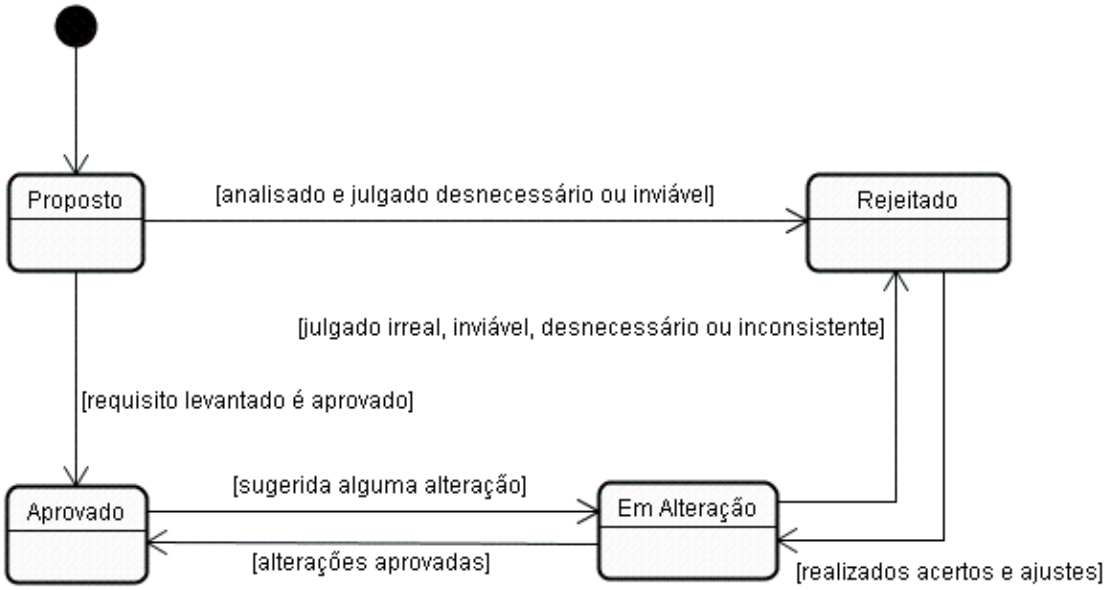


Figura 4 - Diagrama de Estados de um objeto da classe *Requisito*.

Em vários momentos do processo de desenvolvimento de um software, requisitos podem ser criados, alterados ou mesmo rejeitados. Inicialmente, ao ser criado, um requisito é tido como *Proposto*. Durante a etapa de análise ele pode ser *Aprovado*, ou mesmo julgado desnecessário ou inviável, quando seu estado muda para *Rejeitado*. Um requisito *Aprovado* ainda pode sofrer alterações, passando para o estado *Em Alteração*, e estas podem ser rejeitadas num processo de análise, negociação, verificação ou validação, tornando o requisito *Rejeitado*. No estado *Rejeitado* um requisito pode, ainda, ser revisto e se pode propor alterações para que ele seja avaliado novamente passando ao estado *Em Alteração*. Todas essas transições de estados são provocadas pela realização do evento de caso de uso "Alterar Dados Gerais de Requisito" do caso de uso "Cadastrar Requisitos". As condições de guarda mostradas não são, de fato, tratadas pelo sistema, cabendo ao analista analisá-las.

Referências Bibliográficas

FALBO R.A., NATALI, A.C.C., MIAN, P.G., BERTOLLO, G., RUY, F.B. **ODE: Ontology-based software Development Environment**, Proceedings of the IX Argentine Congress on Computer Science (CACIC'2003), La Plata, Argentina, 2003, pp 1124-1135

Documento de Especificação de Projeto

Projeto: ReqODE – Ferramenta de Apoio à Engenharia de Requisitos

Responsável: Aline Freitas Martins

1. Introdução

Este documento apresenta a especificação de projeto para a ferramenta ReqODE – Ferramenta de Apoio à Engenharia de Requisitos. A ferramenta proposta será implementada usando a linguagem de programação Java, o *framework* de mapeamento objeto-relacional Hibernate (BAUER, 2005) e a ferramenta XDoclet (XDOCLET, 2007) para agilizar e facilitar o mapeamento dos objetos em tabelas.

Essa atividade foi conduzida em duas etapas: Projeto Arquitetural e Projeto Detalhado. A seção 2 apresenta o Projeto Arquitetural por meio de diagramas de pacotes, a seção 3 trata do Projeto Detalhado para o pacote principal envolvido neste projeto, o pacote *engenhariaRequisitos* e a seção 4 apresenta o Projeto Detalhado do pacote *contexto*, apresentando para ambos os diagramas de classes de cada componente da arquitetura. Vale destacar que foi adotada a notação da UML.

2. Projeto Arquitetural

Com a finalidade de estabelecer níveis de abstração para o sistema, as classes do projeto foram organizadas em pacotes de acordo com o domínio do problema. A Figura 1 apresenta o diagrama de pacotes de nível mais alto. Em relação ao modelo de análise foi introduzido o pacote *Utilitario* que contém classes que são utilizadas por diversas ferramentas do ambiente.

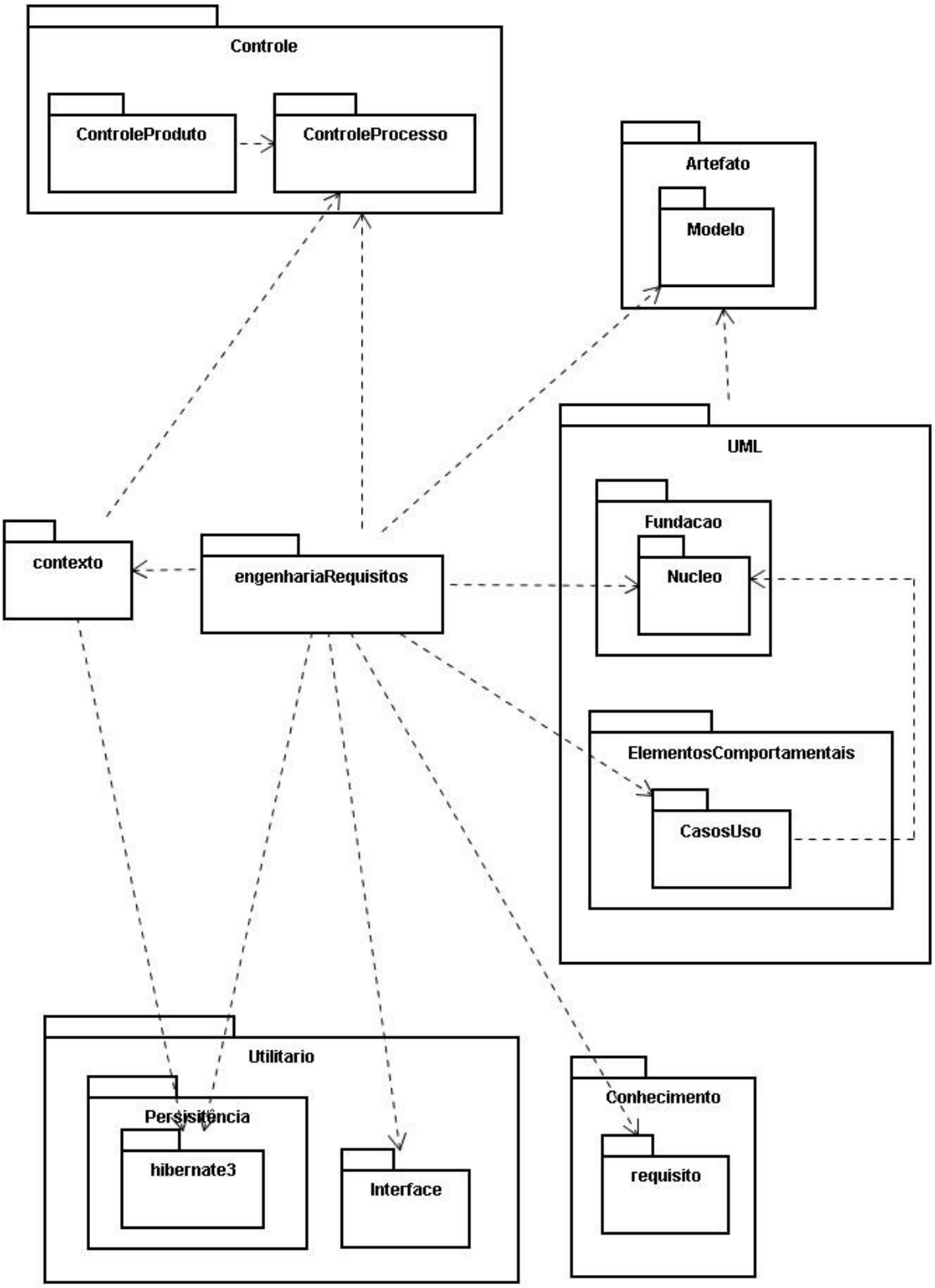


Figura 1 - Diagrama de Pacotes de ReqODE

A Figura 1 mostra as dependências entre os pacotes principais da ferramenta, a saber *engenhariaRequisitos* e *contexto*, e os demais pacotes já existentes no ambiente: *Controle.ControleProcesso*, que contém as classes base para o armazenamento de informações sobre os artefatos gerados, *Controle.ControleProduto*, que possui as classes que representam os módulos de um projeto, *UML.ElementosComportamentais.CasoUso* e *UML.Fundacao.Nucleo*, que contém as classes relativas aos elementos de modelo tipicamente associados a um requisito, *Utilitario.Persistencia.hibernate3*, que contém as classes base de apoio às operações envolvendo bancos de dados e *Utilitario.Interface*, que possui classes de interface padrão do ambiente ODE.

Complementando a organização das classes, há um outro nível de divisão em pacotes, segundo a função que as classes exercem no sistema, ou seja, segundo os seus estereótipos, mostrado na Figura 2.

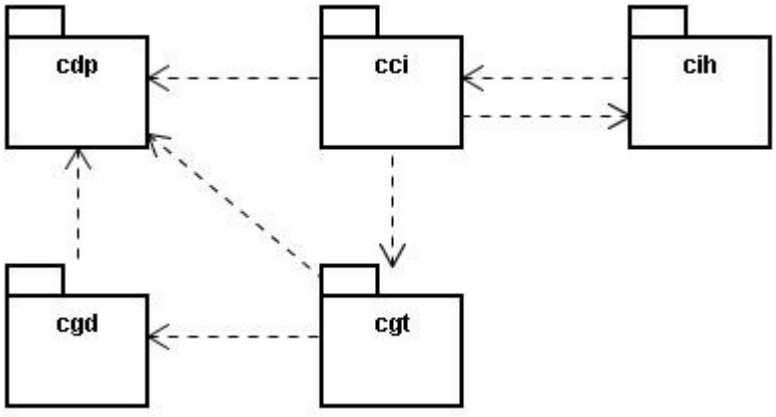


Figura 2 - Componentes de cada pacote da ferramenta

O pacote *cdp* corresponde ao Componente de Domínio do Problema, cujas classes correspondem a abstrações do domínio do problema e, portanto, são derivadas do modelo de análise.

O pacote *cgd* corresponde ao Componente de Gerência de Dados e envolve as classes relativas à persistência de dados. As classes de domínio (*cdp*) que necessitam ter informações persistidas têm no topo de sua herança a classe *ObjetoPersistente*, que encontra-se no pacote *Utilitario.Persistencia.hibernate3*. Isto é importante para manter as informações necessárias para a sua persistência em bancos de dados e traz as vantagens da reutilização, como a facilidade de implementação e abstração de informações de persistência, ou seja, geração de um domínio livre de informações sobre o banco de dados. Como a infra-estrutura do ambiente é baseada no padrão

DAO (*Data Access Object*) (SUN, 2006), para cada classe do domínio a ser persistida é criada uma interface DAO e uma de implementação que utiliza o *framework* Hibernate, como mostra a Figura 3.

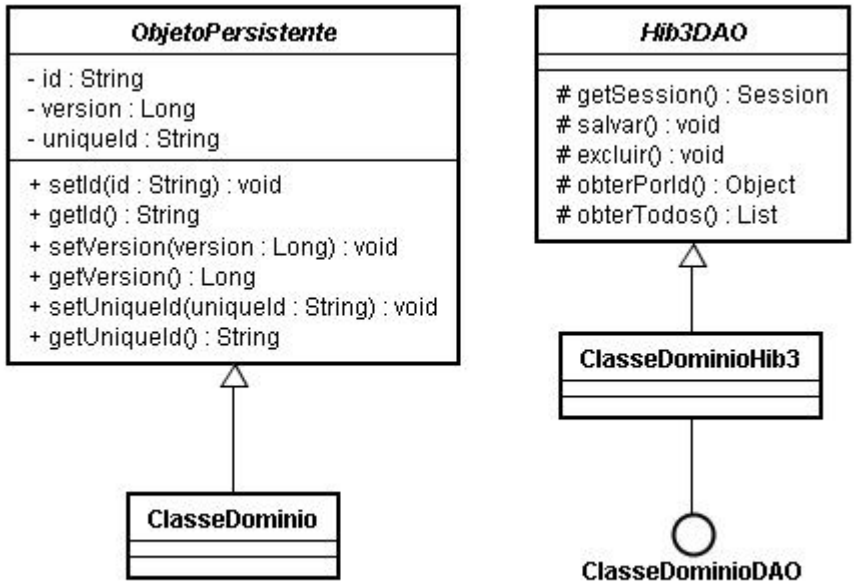


Figura 3 - Infra-estrutura de Persistência

O pacote *cgt* corresponde ao Componente de Gerência de Tarefas e contém as aplicações que implementam os casos de usos definidos na especificação de requisitos.

O pacote *cih* é o Componente de Interação Humana e possui as classes relativas às interfaces gráficas da ferramenta.

O pacote *cci* é o Componente de Controle de Interação cujas classes têm o papel principal de manter a comunicação entre as classes de interface (*cih*) e de aplicação (*cgt*). Suas classes fazem o devido tratamento dos retornos da aplicação para a exibição de resultados na interface, bem como informam às classes de aplicação o que o usuário solicitou por meio de uma interface.

3. Pacote *engenhariaRequisitos*

O pacote principal da ferramenta é o pacote *engenhariaRequisitos*, que contém as classes que, de fato, guardam as informações dos requisitos de um projeto.

3.1 - Componente de Domínio do Problema (cdp)

A Figura 4 apresenta o diagrama de classes do Componente do Domínio do Problema do pacote *engenhariaRequisisto*s. Como se pode notar comparando com o correspondente modelo da fase de análise, não houve mudanças significativas, sendo tratados apenas tipos de dados, navegabilidades e operações.

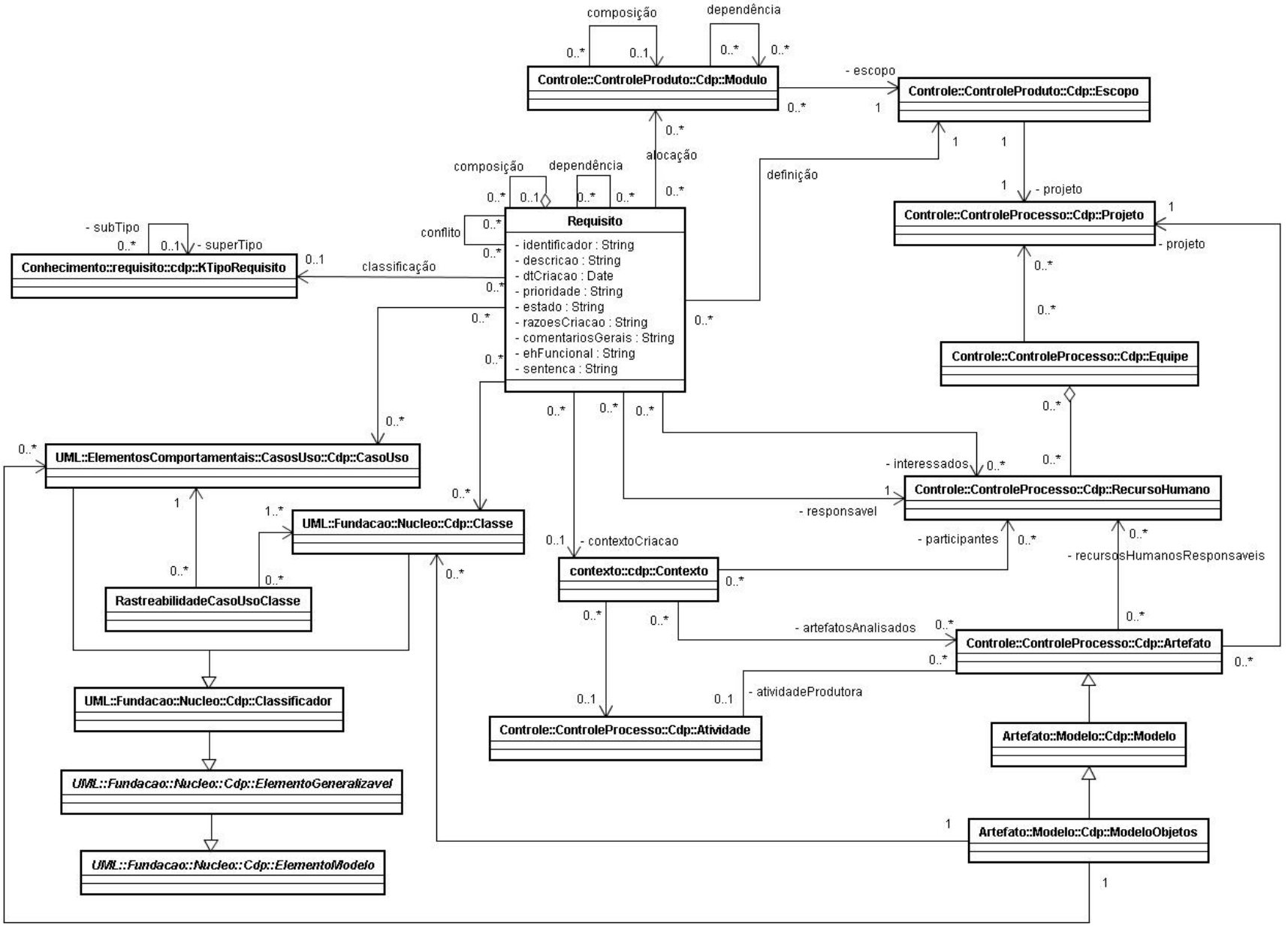


Figura 4 - Diagrama de Classes do pacote *cdp*

3.4 - Componente de Gerência de Dados (cgd)

A Figura 5 apresenta o diagrama de classes do Componente de Gerência de Dados do pacote *engenhariaRequisitos*. Como apenas a classe *Requisito* é nativa deste pacote, há somente uma classe neste componente (*RequisitoHib3*), criada com base no padrão DAO (*Data Access Object*) (SUN, 2006), conforme comentado na seção 2 deste documento.

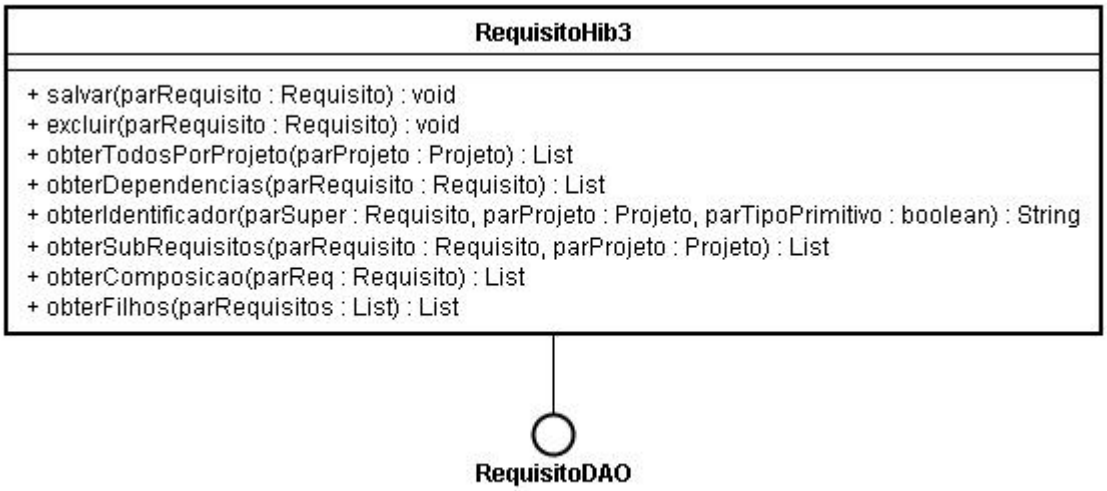


Figura 5 - Diagrama de Classes do pacote *cgd*

3.3 - Componente de Gerência de Tarefas (cgt)

A Figura 6 apresenta o diagrama de classes do Componente de Gerência de Tarefas do pacote *engenhariaRequisitos*.

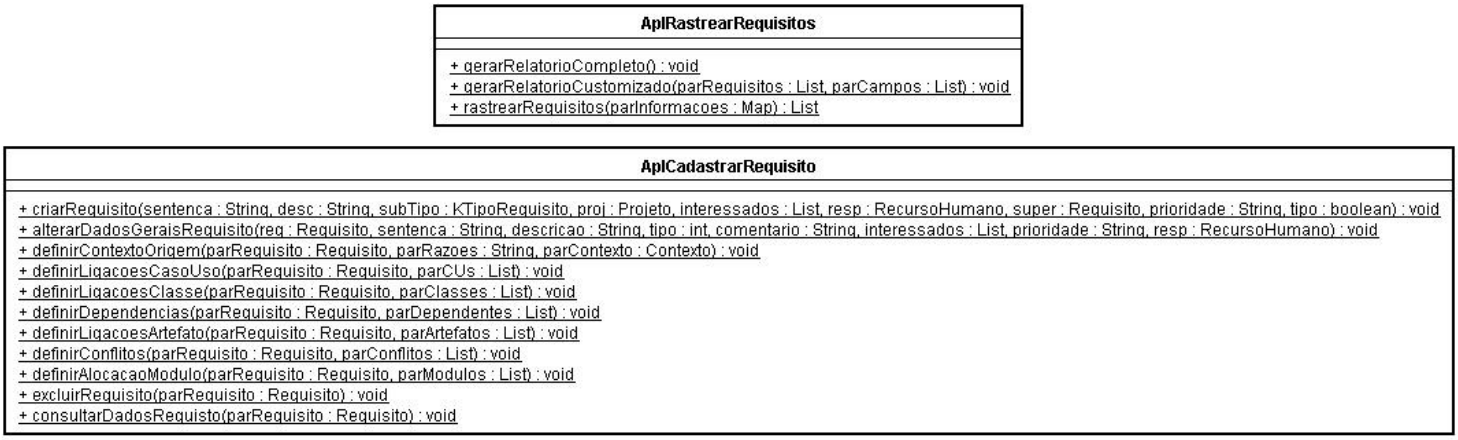


Figura 6 - Diagrama de Classes do pacote *cgt*

Foram criadas duas classes de aplicação: *AplCadastrarRequisito* e *AplRastrearRequisitos*. A primeira trata os casos de uso relativos ao cadastro de informações sobre requisitos, a saber "Cadastrar Requisito" e "Definir Relações de Rastreabilidade", apresentados no Documento de Especificação de Requisitos. Já a segunda aplicação trata os casos de uso que permitem rastrear requisitos, seja por buscas ou por geração de documentos, que são o caso de uso "Gerar Relatórios de Rastreabilidade" e "Rastrear Requisitos". Os eventos de caso de uso, do caso de uso "Rastrear requisitos", geraram apenas uma operação na aplicação, a saber "rastrearRequisitos", isso porque a linguagem utilizada permite fazer um método genérico que abranja todas as situações levantadas na especificação de requisitos.

3.4 - Componente de Interação Humana (cih)

A Figura 7 apresenta o diagrama de classes do Componente Interação Humana do pacote *engenhariaRequisitos*.

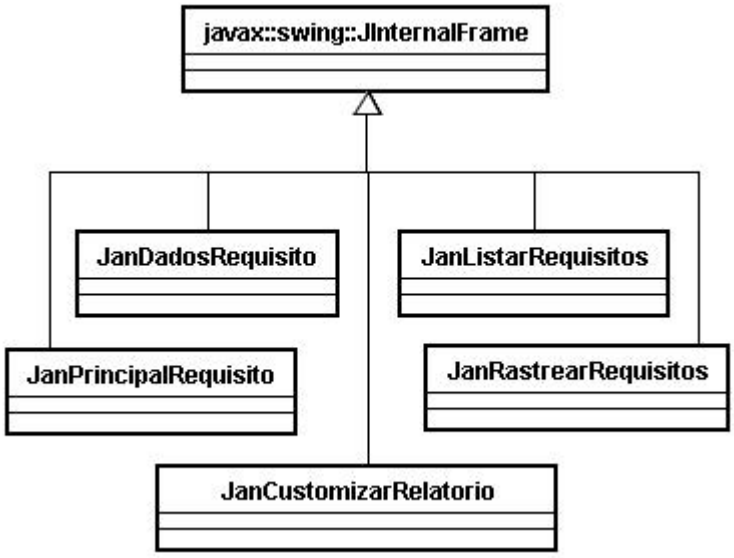


Figura 7 - Diagrama de Classes do pacote *cih*

A janela de principal da ferramenta (*JanPrincipalRequisito*) exibe, em seu painel principal, os requisitos cadastrados no sistema e disponibiliza as funcionalidades de criar um novo requisito, alterar, definir ligações de rastreabilidade, visualizar ou excluir um requisito previamente selecionado. Na criação de um requisito são informados alguns dados básicos, por meio da janela *JanDadosRequisito*.

Na barra de menu da janela principal, assim como no painel principal, é possível acessar a Ferramenta de Cadastro de Elementos de Modelo e a Ferramenta de Modelagem OODE,

para cadastrar ou modelar casos de uso, classe e atores. Pelo menu é possível, ainda, gerar relatórios de rastreabilidade completos e visualizar a janela de criação de relatórios customizados (*JanCustomizarRelatório*), onde é possível escolher os itens a serem apresentados no documento. Existe, ainda, um menu para acessar a janela de buscas de requisitos (*JanRastrearRequisitos*), que possibilita rastrear os requisitos de acordo com filtros estabelecidos pelo usuário. Os resultados dessas buscas são exibidos na janela *JanListarRequisitos*.

3.5 - Componente de Controle de Interação (cci)

A Figura 8 apresenta o diagrama de classes do Componente de Controle de Interação do pacote *engenhariaRequisitos.*

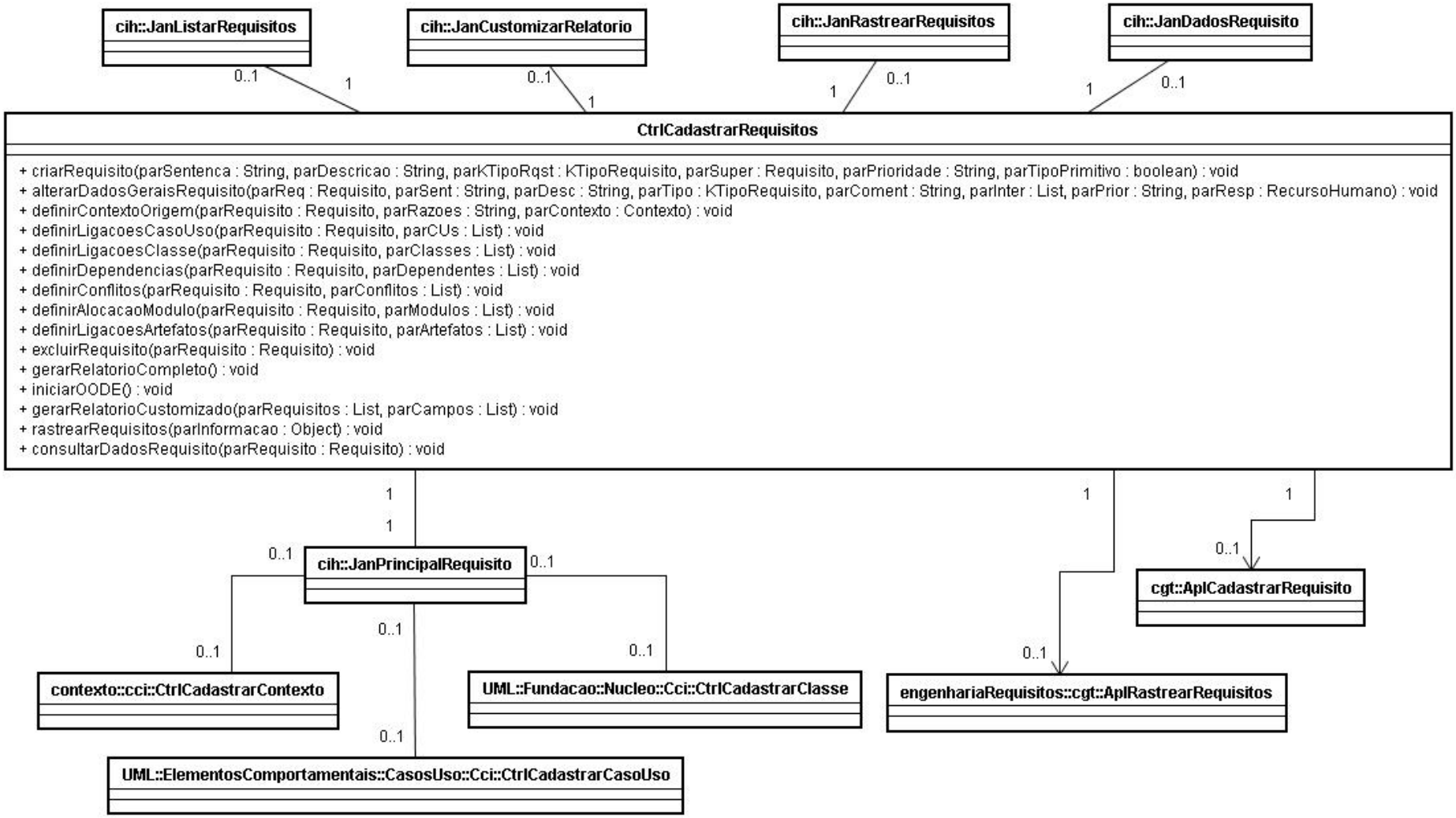


Figura 8 - Diagrama de Classes do pacote *cih*

A classe *CtrlCadastrarRequisitos* tem o papel principal de fazer a comunicação entre os objetos das camadas de aplicação (*cgt*) e de interface (*cih*) deste pacote. Desta forma as interfaces ficam isoladas, facilitando alterações, já que as interfaces não têm acesso direto a métodos da aplicação.

A classe de controle apresentada neste pacote tem seus relacionamentos com navegabilidade dupla com as janelas do *cih*, a saber *JanDadosRequisito, JanPrincipalRequisito, JanRastrearRequisitos, JanListarRequisitos e JanCustomizarRelatorio.* Desta forma, o controlador (*CtrlCadastrarRequisitos*) pode exibir ou atualizar as janelas nos momentos devidos e as janelas podem invocar métodos do controlador para chamar outra janela ou mesmo para solicitar a realização de operações implementadas nas aplicações (*cgt*) *AplCadastrarRequisito* e *AplRastrearRequisitos*.

Note que a janela principal de requisitos (*JanPrincipalRequisito*) tem relacionamento com os controladores *CtrlCadastrarClasse, CtrlCadastrarCasoUso e CtrlCadastrarContexto*. Isso é vantajoso, pois não é necessário adicionar código de cadastro de classes, casos de uso e contextos de origem no controlador de cadastro de requisitos (*CtrlCadastrarRequisito*). Assim, caso seja necessário cadastrar quaisquer um desses elementos, a janela principal passa o controle para o controlador correspondente, que exibe as janelas necessárias e executa os devidos métodos da aplicação correspondente.

4. Pacote *contexto*

O pacote *contexto é* utilizado para registrar o contexto em que um requisito foi criado.

4.1 - Componente de Domínio do Problema (cdp)

A Figura 9 apresenta o diagrama de classes do Componente de Domínio do Problema do pacote *contexto*. Como se pode notar comparando com o correspondente modelo da fase de análise, não houve mudanças significativas, sendo tratados apenas tipos de dados, navegabilidades e operações.

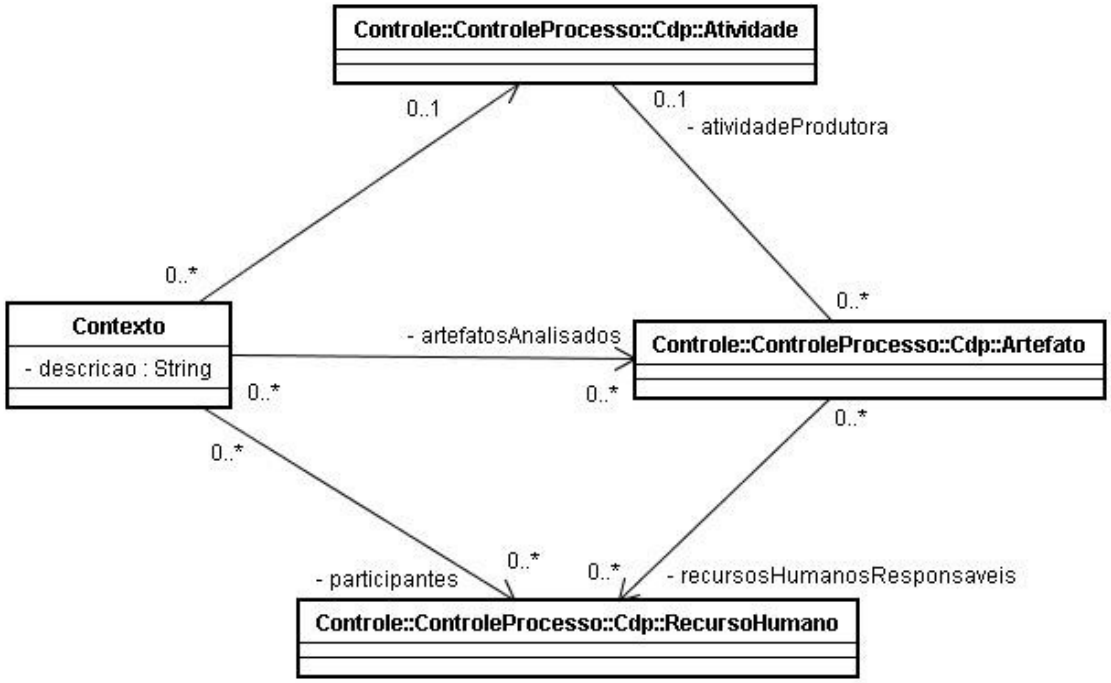


Figura 9 - Diagrama de Classes do pacote *cdp*

3.2 - Componente de Gerência de Dados (cgd)

A Figura 10 apresenta o diagrama de classes do Componente de Gerência de Dados do pacote *contexto***.** Como apenas a classe *Contexto* é nativa deste pacote, há somente uma classe neste componente (*ContextoHib3*), criada com base no padrão DAO (*Data Access Object*) (SUN, 2006), conforme comentado na seção 2 deste documento.

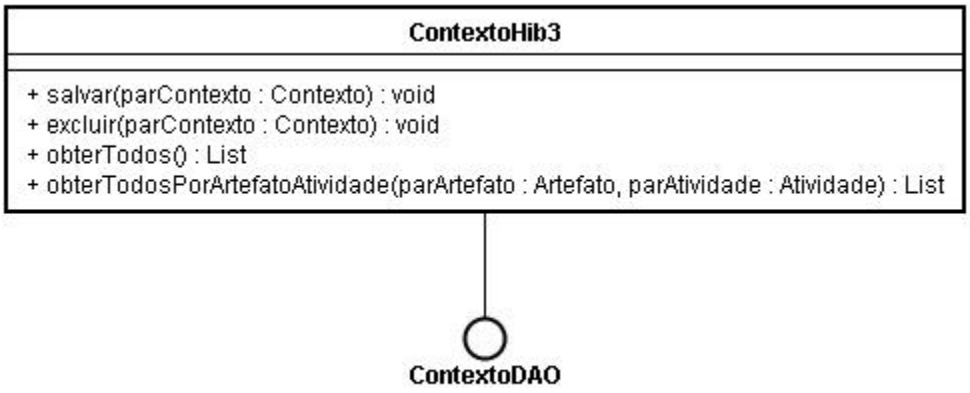


Figura 10 - Diagrama de Classes do pacote *cgd*

4.3 - Componente de Gerência de Tarefas (cgt)

A Figura 11 apresenta o diagrama de classes do Componente de Gerência de Tarefas do pacote *contexto*.

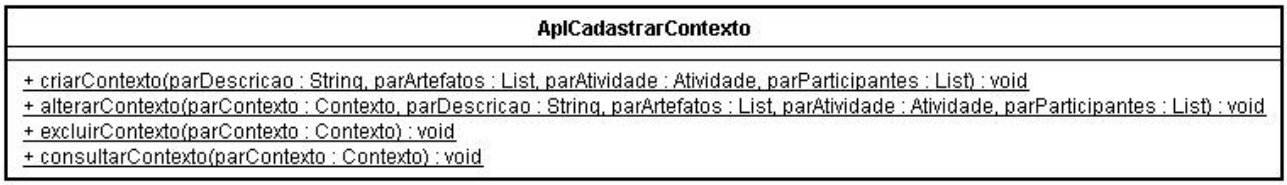


Figura 11- Diagrama de Classes do pacote *cgt*

Como o pacote *contexto* apresenta apenas um caso de uso, foi criada apenas a classe de aplicação *AplCadastrarContexto* para tratar os eventos do caso de uso "Cadastrar Contexto", apresentado no Documento de Especificação de Requisitos.

4.4 - Componente de Interação Humana (cih)

A Figura 12 apresenta o diagrama de classes do Componente Interação Humana do pacote *contexto*.

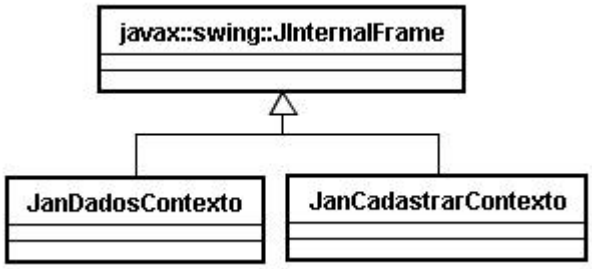


Figura 12 - Diagrama de Classes do pacote *cih*

A janela de cadastro de contexto (*JanCadastroContexto*) exibe contextos cadastrados no sistema e disponibiliza as funcionalidades de criar um novo contexto, alterar, visualizar ou excluir um contexto previamente selecionado.

Para visualizar ou alterar os dados de um contexto, ou mesmo inserir informações de um novo contexto, a janela de dados *JanDadosContexto* é utilizada. Ela é exibida sempre que as opções de cadastro forem solicitadas na janela de cadastro (*JanCadastroContexto*).

4.5 - Componente de Controle de Interação (cci)

A Figura 13 apresenta o diagrama de classes do Componente de Controle de Interação do pacote *contexto.*

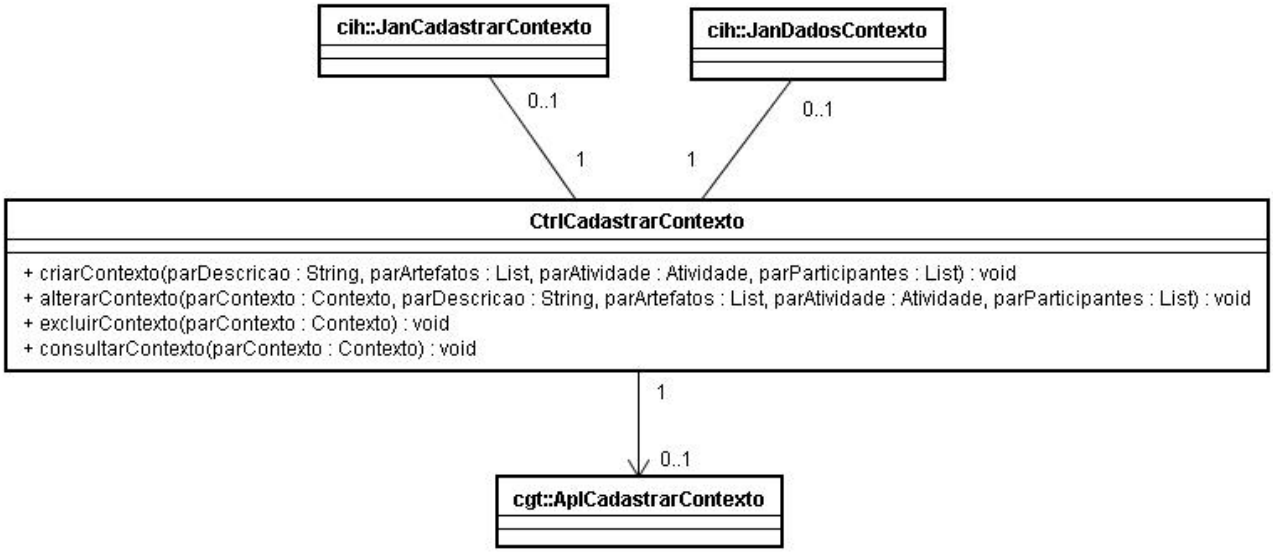


Figura 13 - Diagrama de Classes do pacote *cci*

O controlador *CtrlCadastrarContexto* tem o papel principal de fazer a comunicação entre os objetos das camadas de aplicação (*cgt*) e de interface (*cih*) deste pacote. Desta forma as interfaces ficam isoladas, facilitando alterações, já que as interfaces não têm acesso direto a métodos da aplicação.

A classe de controle apresentada neste pacote tem seus relacionamentos com navegabilidade dupla com as janelas do pacote *contexto.cih*, a saber *JanDadosContexto e JanCadastroContexto.* Desta forma o controlador pode exibir ou atualizar as janelas nos momentos devidos e as janelas podem invocar métodos do controlador para chamar outra janela ou mesmo para solicitar a realização de operações implementadas na aplicação *AplCadastrarContexto*.

Referências Bibliográficas

BAUER,C.; KING, G. **Hibernate em Ação**. 1 ed. Rio de Janeiro, RJ: Ciência Moderna, 2005.

FALBO R.A., NATALI, A.C.C., MIAN, P.G., BERTOLLO, G., RUY, F.B. **ODE: Ontology-based software Development Environment**, Proceedings of the IX Argentine Congress on Computer Science (CACIC'2003), La Plata, Argentina, 2003, pp 1124-1135

SUN DEVELOPER NETWORK, **Data Access Object**. Disponível em: <java.sun.com/j2ee/patterns/DataAccessObject.html>. Acesso em: 8 jan. 2007.

XDOCLET Attribute Oriented Programming. Disponível em: <xdoclet.sourceforge.net>. Acesso em: 3 jan. 2007.