UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
DEPARTAMENTO DE INFORMÁTICA
COLEGIADO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO

DANILLO RICARDO CELINO

EODE: DESENVOLVIMENTO DA BASE DO AMBIENTE DE DESENVOLVIMENTO DE SOFTWARE ODE EM UMA PLATAFORMA DISTRIBUÍDA E EXTENSÍVEL

VITÓRIA
2014

EODE: DESENVOLVIMENTO DA BASE DO AMBIENTE DE DESENVOLVIMENTO DE SOFTWARE ODE EM UMA PLATAFORMA DISTRIBUÍDA E EXTENSÍVEL

DANILLO RICARDO CELINO

MONOGRAFIA APRESENTADA COMO EXIGÊNCIA PARCIAL PARA OBTENÇÃO DO TÍTULO DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO À UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO, NA ÁREA DE CONCENTRAÇÃO ENGENHARIA DE SOFTWARE, SOB A ORIENTAÇÃO DO PROF. VÍTOR ESTÊVÃO SILVA SOUZA.

COMISSÃO EXAMINADORA

_____

Prof. Vítor Estêvão Silva Souza, Ph.D. – Orientador
Universidade Federal do Espírito Santo

_____

Prof. Ricardo de Almeida Falbo, D.Sc.
Universidade Federal do Espírito Santo

_____

Victorio Albani de Carvalho, M.Sc.
Instituto Federal de Educação, Ciência e Tecnologia ES

VITÓRIA
2014

AGRADECIMENTOS

Agradeço a Deus por ter me dado coragem para conquistar mais essa vitória. Aos meus pais, Neilton e Ercília, pelo grande apoio, sacrifício, dedicação ... essa vitória é tão de vocês quanto minha! Aos meus irmãos, Daniel e Darlan, por sempre me alegrarem e nunca me deixarem desistir. Ao mestre, orientador e, acima de tudo, amigo Vitor Souza, por me guiar nessa jornada e pelo enorme incentivo. Um agradecimento especial Ricardo Falbo e Monalessa Perini por me ajudarem nesse projeto, a professora Rosane pelo carinho e dedicação. Ao meus amigos da UFES ao pessoal do NEMO, pelos momentos de correria, e por me tirarem tantas dúvidas. Gostaria de dividir toda a alegria com vocês. Nunca teria chegado até aqui se todos vocês não estivessem presentes em minha vida!

RESUMO

Ferramentas CASE (*Computer-Aided Software Engineering*) auxiliam em cada passo do processo de desenvolvimento de sistemas, porém trabalham isoladamente e maiores vantagens só podem ser alcançadas com sua integração em um Ambiente de Desenvolvimento de Software (ADS). Neste contexto, encontra-se o ambiente ODE (*Ontology-based software Development Environment*), um ADS centrado em processo que reúne diversas ferramentas CASE, apoiando diversas atividades. Na última década, diversos avanços foram feitos dentro do projeto ODE, incluindo esforços para criação de novas ferramentas, interoperabilidade semântica entre as mesmas, etc. No entanto, falta ao ambiente ODE uma base sólida que permita o uso distribuído do ambiente (i.e., por vários desenvolvedores de um mesmo projeto) e que facilite a extensão do ADS por meio da instalação de *plug-ins* (como já é feito em ambientes integrados de desenvolvimento – IDEs). Este projeto tem como objetivo o desenvolvimento desta plataforma de base, com vistas a sua utilização em futuros projetos de desenvolvimento dentro do contexto do projeto ODE.

Palavras chave: Ambiente de Desenvolvimento de Software, Desenvolvimento Distribuído, Extensibilidade, Interoperabilidade.

SUMÁRIO

Capítulo 1 - Introdução

O ambiente de trabalho ideal, seja qual for a especialidade, deve incluir (PRESSMAN, 2011): (a) uma coleção de ferramentas úteis que auxiliem em cada passo do processo de construção do produto; (b) disposição organizada das ferramentas que permita que elas sejam encontradas facilmente e usadas eficientemente; (c) um especialista capacitado em utilizar as ferramentas da maneira correta.

Com relação à Engenharia de Software, as ferramentas CASE atendem ao primeiro requisito e representaram um grande avanço e um enorme benefício. Porém, tais ferramentas trabalham isoladamente, tratando atividades específicas do processo de software, e maiores vantagens só podem ser alcançadas com sua integração. Essa é a ideia por traz dos Ambientes de Desenvolvimento de Software (ADSs). Um ADS é um ambiente que integra ferramentas que apoiam atividades do processo de software de maneira coordenada através de todo o ciclo de vida do software, ou pelo menos em porções significativas dele (HARRISON et al., 2000). Os ADSs atendem ao segundo requisito listado acima.

Existem vários níveis de integração de ferramentas, diferindo no suporte fornecido. Dentre eles, é possível citar (TRAVASSOS, 1994) (PFLEEGER, 2001): integração de dados, de apresentação, de controle, de plataforma, de processo e de conhecimento. O advento da integração de processo deu origem aos ADSs Centrados em Processo (ADSCP), que apoiam a criação e a exploração de modelos de processo e permitem que o desenvolvimento de um projeto seja guiado por um processo definido.

Neste contexto, encontra-se o ambiente ODE (*Ontology-based software Development Environment*) (FALBO et al., 2003). ODE é um ADS centrado em processo. Portanto, possui uma ferramenta de definição e acompanhamento de processos de software e reúne diversas ferramentas CASE que apoiam as mais diversas atividades, como análise de riscos, gerência de conhecimento, etc.

Ao longo de pouco mais de uma década, o projeto ODE vem sendo desenvolvido dentro do Núcleo de Estudos em Modelagem Conceitual e Ontologias (NEMO), formado a partir do antigo Laboratório de Engenharia de Software (LabES) do Departamento de Informática da UFES. Por meio de bolsas de iniciação científica, projetos de graduação e pesquisas de mestrado/doutorado, diversos alunos contribuíram ao longo do tempo para o ambiente. Algumas das contribuições mais recentes incluem ferramentas de definição de processos (SEGRINI, 2009), de gerência do conhecimento

(COELHO, 2010), gerência de configuração de software (CALHAU, 2011), apoio ao planejamento de processos (BRINGUENTE, 2011), etc.

No entanto, podem-se identificar algumas limitações do ambiente ODE:

- ODE foi desenvolvido inicialmente como uma aplicação desktop, utilizando uma base de dados local, o que impossibilita seu uso conjunto por uma equipe de desenvolvedores distribuídos em computadores distintos;
- Em um segundo momento, ODE foi migrado para a plataforma Web para permitir seu uso distribuído. Porém, tal plataforma não permite o mesmo nível de interação com o usuário que a aplicação desktop em diversos tipos de atividade (ex.: desenho de modelos, escrita de código, etc.);
- Apesar dos esforços empreendidos, as diferentes ferramentas não seguem os mesmos padrões de desenvolvimento, muitas vezes forçando um desenvolvedor que deseja incrementar uma determinada ferramenta a reimplementá-la em partes ou por completo;
- Analogamente, a base do ambiente não foi pensada como uma arquitetura que facilitasse a integração de diversas ferramentas como *plug-ins*, ou seja, cada integração deve ser tratada individualmente, o que prejudica a extensibilidade do ambiente.

Tais características, ausentes no projeto ODE, são comuns em ambientes integrados de desenvolvimento (*Integrated Development Environments* – IDEs) presentes no estado da prática. A plataforma Eclipse[1], por exemplo, é baseada no conceito de *plug-ins* que compartilham o mesmo núcleo da plataforma e podem acessar dados dos projetos que existem no espaço de trabalho. Em cima dessa plataforma, iniciativas como Eclipse Orion[2], Saros[3] e Egit[4], permitem o desenvolvimento colaborativo e distribuído ao colocar as informações dos projetos "na nuvem", ou seja, em servidores acessíveis via Internet ou Intranet.

Acredita-se ser importante investir na criação de uma base para o ambiente ODE que permita o desenvolvimento distribuído e, ao mesmo tempo, extensível de modo que novas ferramentas sejam adicionadas ao ambiente de modo simples e uniforme.

[1] http://www.eclipse.org
[2] http://www.eclipse.org/orion/
[3] http://www.saros-project.org
[4] http://www.eclipse.org/egit

1.1 Objetivos

O objetivo principal deste projeto é o desenvolvimento de uma nova base para o ADS ODE utilizando uma arquitetura distribuída e baseada em *plug-ins*, permitindo seu uso simultâneo por múltiplos desenvolvedores de uma mesma equipe e facilitando sua extensão, ou seja, a integração de novas ferramentas. São objetivos específicos deste projeto:

- Determinar, dentre as IDEs existentes que tenham o código aberto e sejam extensíveis/modificáveis, aquela mais adequada para servir de fundação para o desenvolvimento da base do ODE distribuído;
- Desenvolver, a partir da IDE escolhida, a base de um ambiente de desenvolvimento de software que possa ser utilizado simultaneamente por diversos desenvolvedores do mesmo projeto de forma distribuída e que tenha como uma de suas características principais a extensibilidade, permitindo a fácil integração de ferramentas ao ambiente;
- Portar funcionalidades já desenvolvidas para o ODE à nova base do ambiente (ODE distribuído) de forma a avaliar se, de fato, o ambiente permite o desenvolvimento distribuído e pode ser facilmente estendido;
- Produzir documentação que facilite a futuros desenvolvedores a integração de novas ferramentas ao ODE;
- Disponibilizar o ODE distribuído utilizando uma licença de código aberto em uma ferramenta de controle de versão na Internet de modo a promover o interesse de outros desenvolvedores a contribuírem com o projeto.

1.2 Método de Pesquisa

Este trabalho foi desenvolvido em quatro grandes fases: revisão bibliográfica, pesquisa, escolha da IDE para servir de base e desenvolvimento do ambiente.

A primeira e a segunda fases do trabalho foram iniciadas no projeto de Iniciação Científica (CELINO, 2014) com uma revisão da literatura pertinente ao escopo do projeto, incluindo ferramentas CASE, ambientes de desenvolvimento de software, desenvolvimento distribuído e arquiteturas de *plug-ins.*

Na terceira fase foi decidido, dentre os ambientes de desenvolvimento de software de código-aberto já disponíveis, qual era o mais adequado para servir de

fundação para o desenvolvimento da base do ambiente ODE distribuído.

Na quarta etapa foram realizadas as atividades de especificação de requisitos, análise, projeto, implementação e testes, segundo o paradigma da orientação a objetos, usando a linguagem de modelagem unificada (UML) e uma linguagem de programação orientada a objetos para o ambiente distribuído.

1.3 Organização da Monografia

Este documento está dividido em quatro capítulos.

O Capítulo 2 apresenta uma revisão bibliográfica sobre os principais temas relacionados ao projeto. Como fonte de pesquisa foram utilizados artigos científicos, relatórios técnicos, livros e trabalhos acadêmicos que tratam sobre plataforma de desenvolvimento de software, Ambientes de Desenvolvimento de Software, processo de software e controle de versão na Web.

O Capítulo 3 apresenta os resultados das atividades de Especificação de Requisitos, Análise, Projeto, Implementação e Testes conduzidas no contexto deste projeto.

No capítulo 4 encontram-se as conclusões e trabalhos futuros.

Capítulo 2 – Revisão Bibliográfica

Neste capítulo é apresentada a revisão da literatura, ou seja, uma análise crítica, meticulosa e ampla das publicações correntes da área para criação da ferramenta e também para saber quais são as variáveis do problema em questão.

2.1 O Ambiente de Desenvolvimento de Software ODE

ODE (*Ontology-based software Development Environment*) (FALBO et al., 2003) é um Ambiente de Desenvolvimento de Software desenvolvido no Núcleo de Modelagem Conceitual e Ontologias (NEMO/UFES), tendo por base ontologias e é um ADS centrado em processo (ADSCP), que apoia a criação e a exploração de modelos de processo, permitindo que o desenvolvimento de um projeto seja guiado por um processo definido. Portanto, possui uma ferramenta de definição e acompanhamento de processos de software e reúne diversas ferramentas CASE que apoiam as mais diversas atividades, como análise de riscos, realização de estimativas, gerência de conhecimento, etc.

2.2 Processos de Software

Um dos objetivos (provavelmente um dos principais) da Engenharia de Software é desenvolver sistemas com qualidade, dentro dos prazos estabelecidos e sem a necessidade de alocação de mais recursos.

Para que esse objetivo possa ser alcançado, não se deve focar somente nos produtos gerados, mas também no seu processo de desenvolvimento. A qualidade de um produto de software depende fortemente da qualidade do processo de software utilizado em seu desenvolvimento (FUGGETTA, 2000). Assim, a definição de um processo de software é um requisito básico para se chegar a produtos de qualidade.

Porém, a definição de um processo de software não é uma tarefa trivial. Processos devem ser definidos caso a caso, levando-se em consideração as especificidades do projeto em questão. Deve-se considerar a adequação às tecnologias envolvidas, ao tipo de software em questão, ao domínio de aplicação, ao grau de maturidade (ou capacitação) da equipe em Engenharia de Software, às características próprias da organização e às características do projeto e do grupo de desenvolvimento (ROCHA et al., 2001).

Embora diferentes projetos requeiram processos com características específicas para atender às suas particularidades, é possível estabelecer um conjunto de ativos de processo de software (*software process assets*) a ser utilizado na definição de processos de

software de uma organização. Essas coleções de ativos de processo constituem os chamados processos padrão de software. Processos para projetos específicos podem, então, ser definidos a partir da instanciação de um processo de software padrão da organização, levando em consideração suas características particulares.

Um processo de software não pode ser definido de forma universal. Ele deve ser adequado ao domínio da aplicação e ao problema específico a ser resolvido. Contudo, de maneira geral, encontramos as atividades de planejamento, especificação de requisitos, análise, projeto, implementação e testes (FALBO, 2000).

Para o desenvolvimento deste trabalho, por exemplo, definiu-se um ciclo de vida incremental, utilizando os requisitos das versões anteriores de ODE – ODE-Desktop e ODE-Web – observando-se apenas o núcleo padrão do ambiente e, a partir desse núcleo, integrar três ferramentas, a saber: controle de usuário, controle de processo e criação de projeto. A Figura 2.1 ilustra o ciclo de vida escolhido.

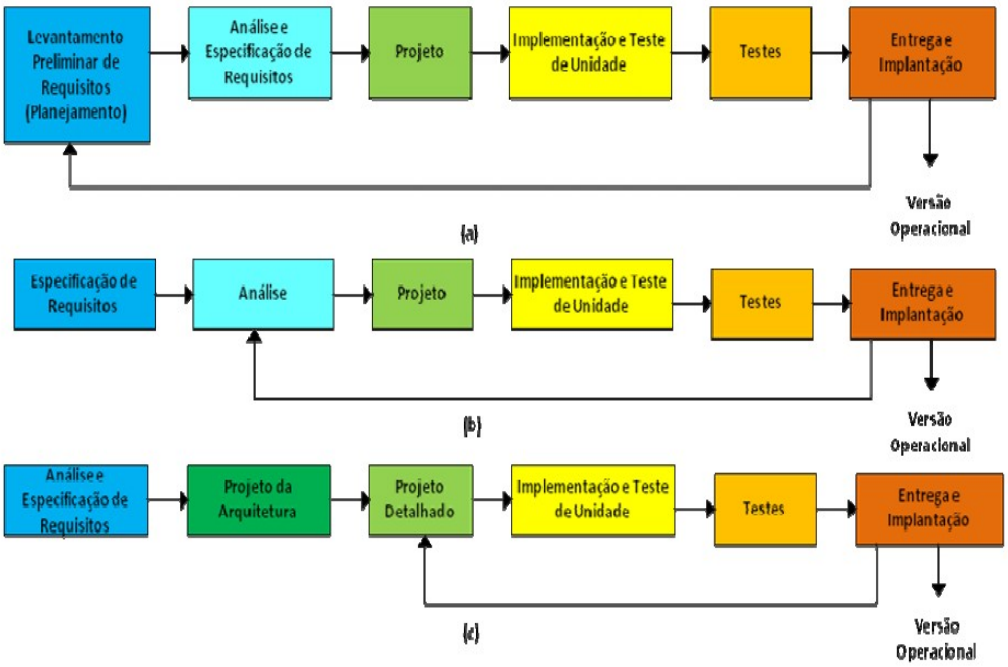

Figura 2.1. Modelo de Ciclo de Vida Incremental

2.2.1 Definição de Processos de Software

Um processo de software pode ser definido como um conjunto coerente de políticas, estruturas organizacionais, tecnologias, procedimentos e artefatos necessários para conceber, desenvolver, implantar e manter um produto de software (FUGGETTA, 2000). Um

processo de software bem definido deve indicar as atividades a serem executadas, os recursos requeridos, os artefatos consumidos e produzidos e os procedimentos a serem adotados (métodos, técnicas, modelos de documentos, entre outros) (FALBO et al., 1998).

Um elemento chave das normas e modelos de maturidade relativos a processos de software é a definição de um processo padrão descrevendo as atividades que devem ser realizadas em todos os projetos de software de uma organização, bem como os demais ativos de processo envolvidos, dentre eles artefatos, procedimentos, ferramentas e papéis. O uso de um processo padrão como base para o planejamento do processo de software específico de um projeto permite aos gerentes de projeto definir planos em conformidade com os padrões de qualidade e procedimentos da organização (BERGER, 2003).

Entretanto, esse tipo de esforço de padronização sofre com um problema: para acomodar todos os tipos de iniciativas de desenvolvimento em uma organização, o padrão vai inevitavelmente estar em um nível de abstração que atenda às necessidades de todos os projetos, mas não vai ser capaz de fornecer apoio específico às atividades individuais de um dado projeto.

De fato, nenhum projeto é igual ao outro e, portanto, para ser efetivo e conduzir a produtos de qualidade, um processo deve ser adequado às características específicas do projeto, considerando, dentre outros, o tipo de software a ser desenvolvido, o paradigma, o domínio de aplicação, características da equipe, tamanho e complexidade do projeto. Assim, é necessária uma abordagem flexível e configurável para a definição de processos, de modo a facilitar a adaptação de processos padrão às necessidades específicas de cada projeto.

2.3 Plataforma de Desenvolvimento de Software

Plataforma é um conjunto de ativos que podem ser usados para alavancar o reúso e o rápido desenvolvimento de novos produtos. No mínimo, ela define o ambiente operacional, a arquitetura em alto nível de todos os produtos desenvolvidos com base nesta plataforma, e um conjunto de políticas de desenvolvimento para aperfeiçoar a plataforma e o desenvolvimento de produtos (MURER, 2014).

Quando se trata de ambientes de desenvolvimento de software, os “ativos” são as ferramentas de CASE e os “produtos” são os aplicativos. Entre as ferramentas estão (MURER, 2014):

- Kits ou pacotes para o desenvolvimento de aplicativos – SDK (*Software Development Kit*);

- Emuladores ou Simuladores, que permitem ao desenvolvedor visualizar e testar seu aplicativo como se estivesse no ar;
- APIs (*Application Program Interfaces*) as quais definem padrões e especificam como os diferentes componentes da plataforma se comunicam;
- Bibliotecas e frameworks, os quais podem ser usados pelos desenvolvedores para agilizar o desenvolvimento dos aplicativos.

2.3.1 Eclipse

O Eclipse é uma plataforma de desenvolvimento de software livre extensível, baseada em Java. Por si só, é simplesmente uma estrutura e um conjunto de serviços para desenvolvimento de aplicativos de componentes de *plug-in*. Felizmente, o Eclipse vem com um conjunto padrão de *plug-ins*, incluindo as amplamente conhecidas Ferramentas de Desenvolvimento Java (JDT).

Embora a maioria dos usuários esteja satisfeita em usar o Eclipse como um ambiente de desenvolvimento integrado (IDE) Java, a plataforma ambiciona mais. O Eclipse também inclui o *Plug-in Development Environment* (PDE), que é de interesse principalmente daqueles que desejam estender o Eclipse, visto que ele permite desenvolver ferramentas que se integram perfeitamente ao ambiente do Eclipse. Como tudo no Eclipse é *plug-in*, todos os desenvolvedores de ferramentas têm um campo de ação nivelado para oferecer extensões ao Eclipse e fornecer um IDE unificado e consistente aos usuários.

Essa paridade e consistência não se limitam às ferramentas de desenvolvimento Java. Embora o Eclipse seja escrito na linguagem de programação Java, seu uso não se limita à linguagem Java. Por exemplo, estão disponíveis ou planejados *plug-ins* que incluem suporte para linguagens de programação como C/C++ e COBOL. A estrutura do Eclipse também pode ser usada como base para outros tipos de aplicativos não relacionados ao desenvolvimento de software, com sistemas de gerenciamento de conteúdo (ANISZCZYK; GALLARGO, 2014).

Originalmente, a plataforma Eclipse foi projetada para funcionar como uma plataforma de ferramentas abertas. No entanto, a partir do Eclipse 3.0, a plataforma teve sua arquitetura reformulada para que seus componentes pudessem ser usados para construir praticamente qualquer aplicação cliente. O conjunto mínimo de *plug-ins* necessário para construir uma aplicação com interface com o usuário rica é coletivamente conhecido como *Rich Client Platform* (RCP). Essas aplicações ricas ainda são baseadas em um modelo de

plug-in dinâmico e a interface gráfica com o usuário (*Graphical User Interface - GUI*) é construída usando os mesmos kits de ferramentas e pontos de extensão. No entanto, a diferença chave é que o ambiente de trabalho está sob controle de baixa granularidade do desenvolvedor do *plug-in* com aplicações RCP. Observe que o Eclipse IDE é, por si só, uma aplicação RCP (MINOCHA, 2014).

Neste projeto, decidiu-se usar o Eclipse RCP como plataforma de desenvolvimento, pois ele supre a necessidade de facilidade de integração por já ser uma plataforma feita pensada como *plug-in*, tornando o ODE facilmente extensível resolvendo uma das limitações do ambiente.

2.4 Sistema de Controle de Versão

Sistema de Controle de Versão (SCV) é um software responsável por manter o controle de várias revisões da mesma unidade de informação. Esses softwares são comumente utilizados no desenvolvimento de software para gerência do código fonte e de outros artefatos do projeto, como documentação e modelos de dados (MIKKELSEN, 1997). Controlar versões tem uma gama de vantagens na solução de problemas para gerência e rastreamento de alterações durante a fase de desenvolvimento do software, como a análise de histórico do desenvolvimento, resgate de versões antigas, ramificações do projeto, dentre outras atividades.

As ferramentas de controle de versão vêm evoluindo constantemente, e junto com elas conceitos como o de controle de versão centralizado e distribuído, acabam se tornando indispensáveis para desenvolvedores.

O controle de versão centralizado se baseia na arquitetura cliente-servidor, com um único repositório principal e várias cópias de trabalho para os desenvolvedores, que se comunicam apenas através do repositório. Os dois sistemas de controle de versão centralizado (SCVC) mais conhecidos são o CVS[5] e o Subversion[6]. Tais softwares são chamados de centralizados, pois todos os desenvolvedores do projeto trabalham juntos em um único repositório central, sendo este localizado em um diretório diferente de onde os desenvolvedores alteram seus arquivos (DE QUEIROZ, 2014).

Os softwares de controle de versão distribuído (SCVD) são qualificados por terem vários repositórios autônomos, um para cada desenvolvedor. Cada repositório possui uma área de trabalho acoplada e as operações de *commit*, onde os desenvolvedores mandam suas

[5] http://www.nongnu.org/cvs/

[6] http://subversion.apache.org/

versões atualizadas, e *update*, por meio da qual o desenvolvedor atualiza a sua versão, acontecem localmente. Tais softwares são chamados de distribuídos por não possuírem o conceito de repositório central e qualquer colaborador pode ter um ou mais repositórios. Os principais representantes desse tipo de sistema são o Mercurial[7] e o Git[8] (DE QUEIROZ, 2014). Este último, desde seu lançamento, em maio de 2007, contribui para um aumento significativo da adoção de SCVD, se tornando um dos principais concorrentes aos SCVCs, em especial ao SVN.

2.4.1 Git

Git é um sistema de controle de versões desenvolvido por Linus Torvalds, inicialmente para dar suporte ao desenvolvimento do núcleo do Linux. O primeiro protótipo foi divulgado por Torvalds em 2005, como uma alternativa aos sistemas de controle de versão existentes, pois nenhum deles atendia ao requisito de eficiência exigido por ele (TORVALDS, 2014).

Para atingir a eficiência exigida, o Git foi projetado para ter o seu repositório distribuído. Ou seja, cada desenvolvedor possui uma cópia completa do repositório, com cada versão dos arquivos do sistema gerenciado e seu histórico. Assim, o Git contempla duas características muito importantes. A primeira é a velocidade, pois qualquer operação de *commit* (envio de novas versões ao repositório), *branch* (que é o ato de criar ramos de projetos), ou consulta ao histórico é realizada apenas na máquina local, sem acessar a rede (apenas as operações de sincronização com outros repositórios acessam a rede). A segunda é a tolerância a falhas, pois, como o repositório é replicado, problemas em um repositório são facilmente recuperáveis a partir das outras cópias existentes (FERREIRA, 2009).

Além disso, o Git foi desenvolvido com o intuito de facilitar a criação de *branches*, criando ramos de projetos e a realização de *merges*, juntando ramos de projetos. Devido à quantidade de ferramentas desenvolvidas e a velocidade que essas operações são realizadas, o uso seu é muito mais frequente do que em outras ferramentas de controle de versão (DE QUEIROZ, 2008).

Existem também *plug-ins* que dão suporte ao Git nas IDEs mais populares. Assim, as operações mais comuns (como *commit*, listagem de histórico) podem ser realizadas via interface gráfica, mas algumas operações mais avançadas para a manutenção do repositório continuam tendo que ser realizadas via linha de comando. Atualmente, estão em

[7] http://www.selenic.com/mercurial/

[8] http://git-scm.com/

desenvolvimento *plug-ins* para uso com as IDEs Eclipse (EGit, descrito na seção a seguir) e NetBeans (NetBeans Git Module)[9] (DE QUEIROZ, 2008).

2.4.1.1 EGit

EGit é um *plug-in* do Eclipse para desenvolvimento em equipe para o sistema de controle de versão Git. Como o Git é um SCVD, isso significa que cada desenvolvedor tem uma cópia completa de todo o histórico de cada revisão do código, fazendo consultas no histórico rapidamente e de maneira versátil.

Por ter essas características, esse *plug-in* foi o escolhido para implementar o requisito de distribuição necessário à nova plataforma de ODE, que era uma das limitações das versões antigas do ambiente. Nesse projeto ele foi integrado à nova versão de ODE, fazendo com que o desenvolvimento seja distribuído.

[9] https://netbeans.org/kb/docs/ide/git.html

Capítulo 3 - Desenvolvimento da Ferramenta

Neste capítulo será apresentada a elaboração e a implementação da ferramenta. A ferramenta foi elaborada pensando nos processos da Engenharia de Software: Requisitos, Análise, Projeto, Implementação e Testes.

3.1. Especificação de Requisitos

A modelagem da nova base de ODE iniciou-se com a atividade de levantamento de requisitos. Durante esta atividade, foram identificados como *stakeholders* professores envolvidos com o projeto ODE e, então, conduzidas entrevistas para entendimento e modelagem do problema a ser resolvido. Além das entrevistas, foi realizada também análise de documentos (trabalhos acadêmicos) relacionados ao projeto ODE. São apresentados, a seguir, uma descrição geral do problema e de seus casos de uso principais. Para uma descrição mais detalhada, o leitor pode consultar o documento de especificação de requisitos, disponível para download no site do projeto[10]. Os modelos de caso de uso e de classes junto com a documetação do ODE Web já haviam sido desenvolvidos em outras versões do ODE, a utilizada foi a versão Web. A modificação realizada nos modelos foi traduzi-lo para o inglês.

No contexto de ODE, se faz necessária uma nova base que seja fácil de se estender e distribuída ao mesmo tempo, já que na base da versão Web de ODE, mesmo sendo distribuída, cada iniciativa de integração deve ser tratada individualmente, o que prejudica a extensibilidade do ambiente.

A base de ODE é formada por: controle de usuários, controle de processos, com definição de processos padrão, e cadastramento de projetos.

O Cadastro de Projetos em ODE permite controlar projetos, registrando seu nome e descrição e controlando a cada momento qual é o projeto atualmente aberto no ambiente.

Já o Cadastro de Recursos Humanos permite controlar os recursos humanos da organização de software que serão, posteriormente, organizados em equipes de projetos de software. De um recurso humano registra-se nome, telefone, e-mail, carga horária semanal a cumprir e o cargo que exerce.

Organizações executam projetos, os quais são realizados por equipes de recursos humanos. Projetos de organizações devem possuir uma equipe. É necessário definir a equipe

[10] http://nemo.inf.ufes.br/pt-br/eode/Requisitos_v1.0.pdf

de cada projeto, registrando o papel que será desempenhado por cada recurso humano naquela equipe.

O Cadastro de Usuários de ODE permite controlar os recursos humanos que terão acesso ao sistema. Cada recurso humano que possui acesso a ODE tem um usuário correspondente. Dessa forma, os dados necessários para efetuar o *login* no ambiente correspondem ao usuário e não ao recurso humano. De um usuário, registra-se *login*, senha e seu perfil de acesso, sendo que o último possui nome e funcionalidades para as quais seu acesso é permitido.

A base de conhecimento do ODE registra informações sobre tipos de atividades, artefatos, recursos humanos, processos, procedimentos, modelo de ciclo de vida e recursos que podem ser de software ou hardware.

No ambiente deve-se definir o processo padrão da organização. O controle de processo é feito cadastrando um processo para cada projeto, tendo um processo padrão como base. Para cada processo que está sendo definido, deve-se selecionar um projeto. Cada processo de projeto tem as mesmas características do processo padrão. Devem-se incluir nele os artefatos, recursos humanos e recursos para o projeto específico.

A nova base do ODE deve permitir a integração de ferramentas, tratando-as de forma conjunta e não individualmente, facilitando a extensibilidade, sendo possível escolher quais ferramentas se quer integrar na base. Ela deve ser distribuída para que diversos usuários possam compartilhá-la independentemente do local dos projetos em que estão alocados.

Com base nas informações levantadas e descritas acima, foram especificados os requisitos funcionais da ferramenta, utilizando os modelos de caso de uso com suas respectivas descrições para estruturar esta visão. Os requisitos capturados nesta fase modelam as funcionalidades que o sistema deve oferecer a seus usuários.

Os requisitos foram agrupados em seis pacotes principais, como mostra a Figura 3.1: *userControl*, *humanResourceControl*, *projectControl*, *processKnowledge*, *standardProcess* e *processControl*, sendo que o pacote *mainKnowledge* é usado como referência para instâncias de objetos que são conhecimento e que podem ser utilizados externamente. Os modelos foram produzidos em inglês de modo a permitir que desenvolvedores e grupos de pesquisa do exterior mostrem interesse pelo projeto (atendendo um dos objetivos elencados anteriormente na Seção 1.1).

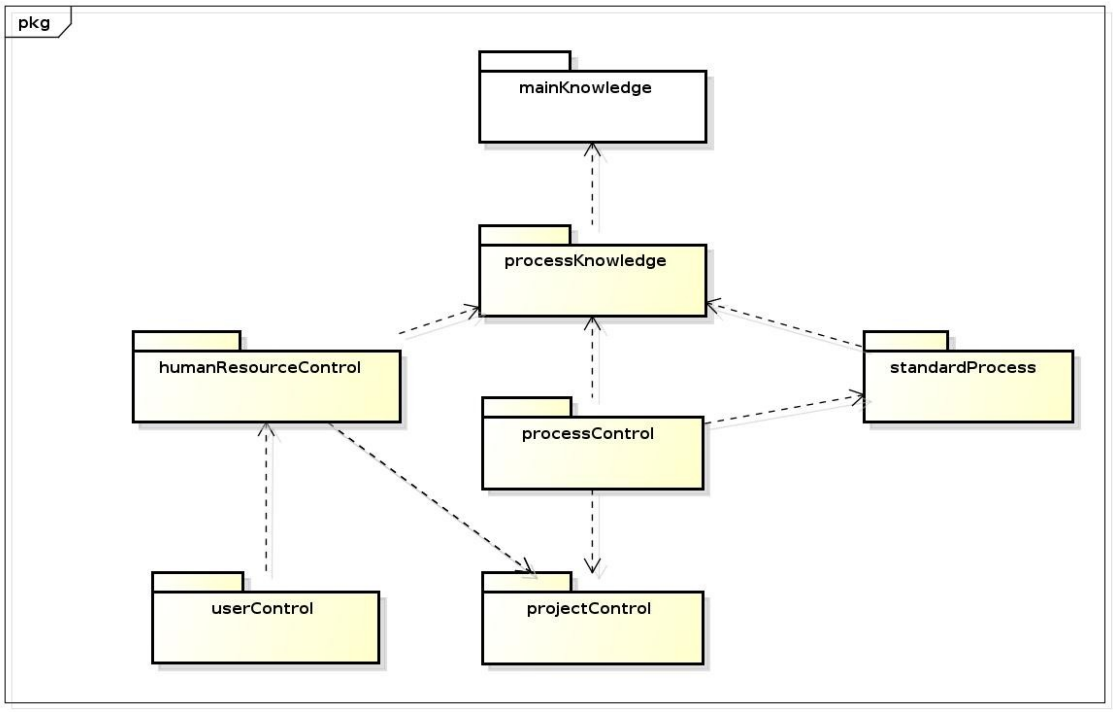

Figura 3.1 Diagrama Geral de Pacotes.

As dependências entre pacotes como mostrado na Figura 3.1 são entre classes as quais são especificadas na Seção 3.2.

3.1.1 Especificação de Requisitos do Controle de Usuários (*userControl*)

O diagrama de casos de uso mostrado na Figura 3.2 modela as funcionalidades disponíveis para controle de usuários e foi construído por (CHAVES, 2011a) e por mim traduzido para inglês.

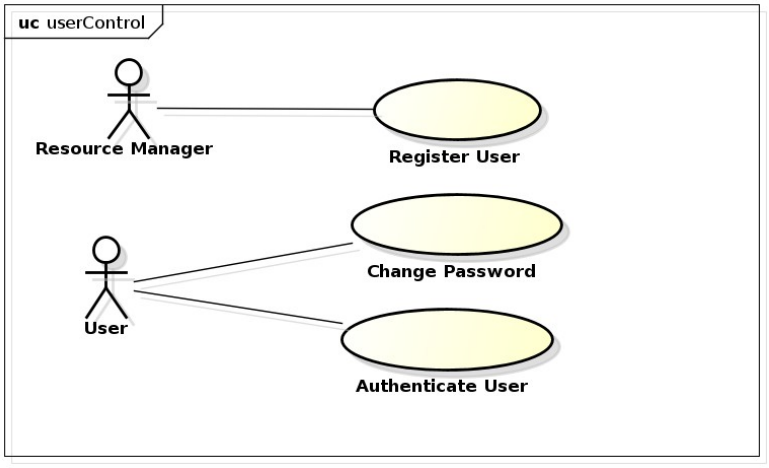

Figura 3.2 Diagrama de caso de uso userControl

O ator *Resource Manager* (gerente) representa usuários do sistema que possuem permissão para realizar o cadastro de usuários, enquanto *User* representa qualquer usuário de ODE. Neste subsistema, o gerente pode cadastrar usuários (*Register User*), inserindo novos usuários, visualizando/alterando dados ou excluindo usuários existentes. Usuários, por sua vez, podem autenticar-se (*Authenticate User*) e mudar suas próprias senhas (*Change Password*).

3.1.2 Especificação de Requisitos do Controle de Recursos Humanos (*humanResourceControl*)

O diagrama de casos de uso mostrado na Figura 3.3 modela as funcionalidades disponíveis para controle de recursos humanos.

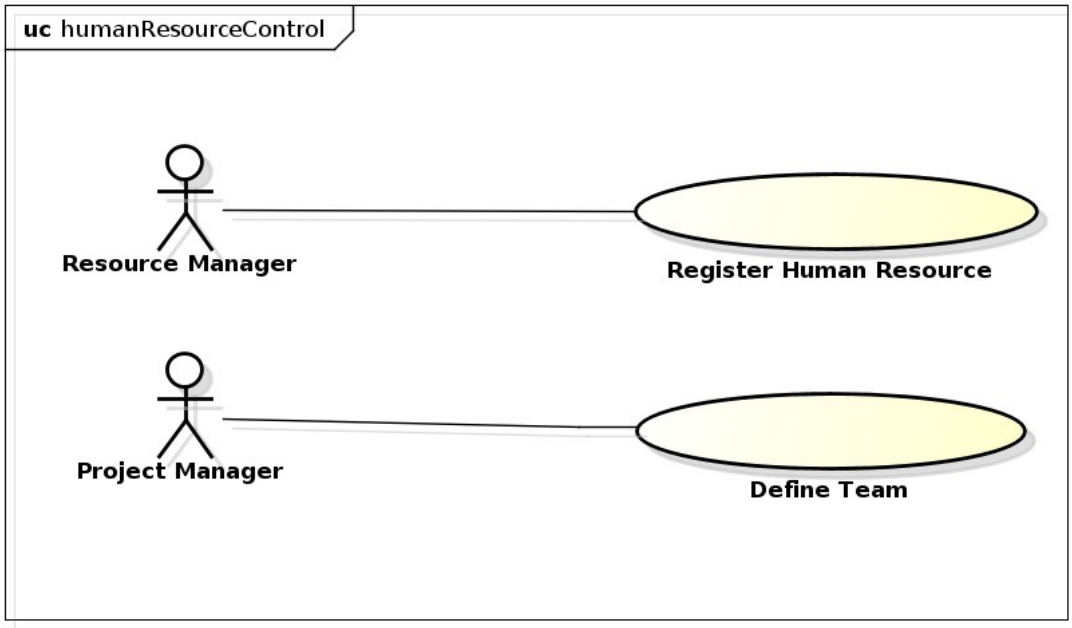

Figura 3.3 Diagrama de caso de uso humanResourceControl

O ator *Resource Manager* (gerente de recursos), representa quem tem permissão para realizar o cadastro de recursos (*Register Human Resource)*. *Project Manager* é o ator que representa usuários que podem gerenciar projetos da organização. Nesse subsistema, *Resource Manager* pode cadastrar recursos humanos *(Register Human Resource)*, inserindo novos recursos humanos, alterando, consultando e excluindo recursos humanos.

Project Manager pode definir equipe (*Define Team)*, permitindo que o gerente de projeto defina quais recursos humanos deverão compor a equipe de um projeto.

3.1.3 Especificação de Requisitos do Controle de Projetos (*projectControl*)

O diagrama de casos de uso mostrado na Figura 3.4 modela as funcionalidades disponíveis para controle de projetos.

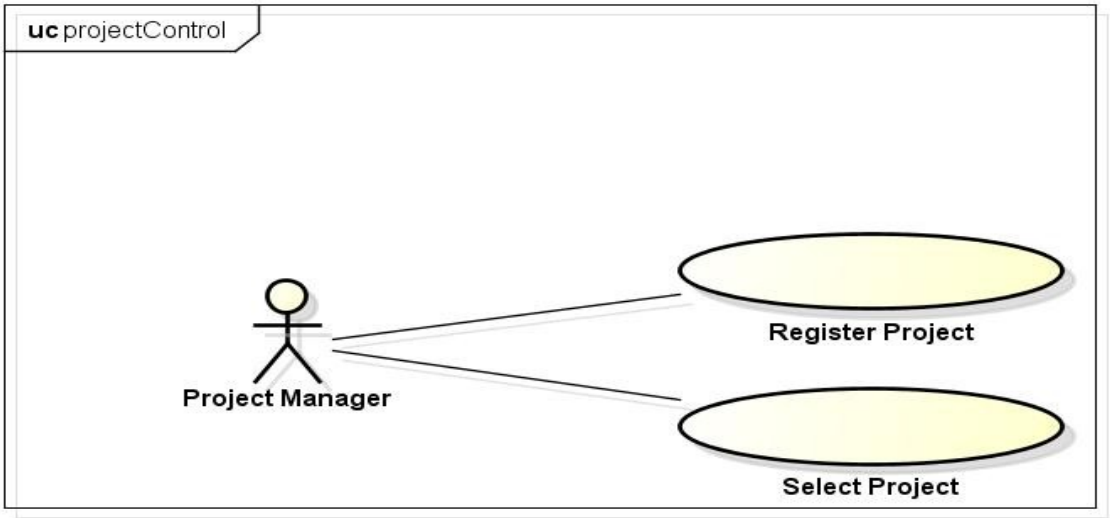

Figura 3.4 Diagrama de caso de uso projectControl

O gerente de projetos pode cadastrar projeto *(Register Project)*, incluindo novos projetos, alterando, consultando e excluindo um projeto no sistema. Para incluir um projeto, registra-se nome e descrição. O gerente também pode selecionar projetos (*Select Project)*, que significa consultar um projeto, selecionando-o para a sessão vigente do sistema para ser usado em outras funcionalidades que o requerem.

3.1.4 Especificação de Requisitos da Gerência de Conhecimento sobre Processos (*processKnowledge)*

O diagrama de casos de uso mostrado na Figura 3.5 modela as funcionalidades disponíveis para gerência de conhecimento sobre processos.

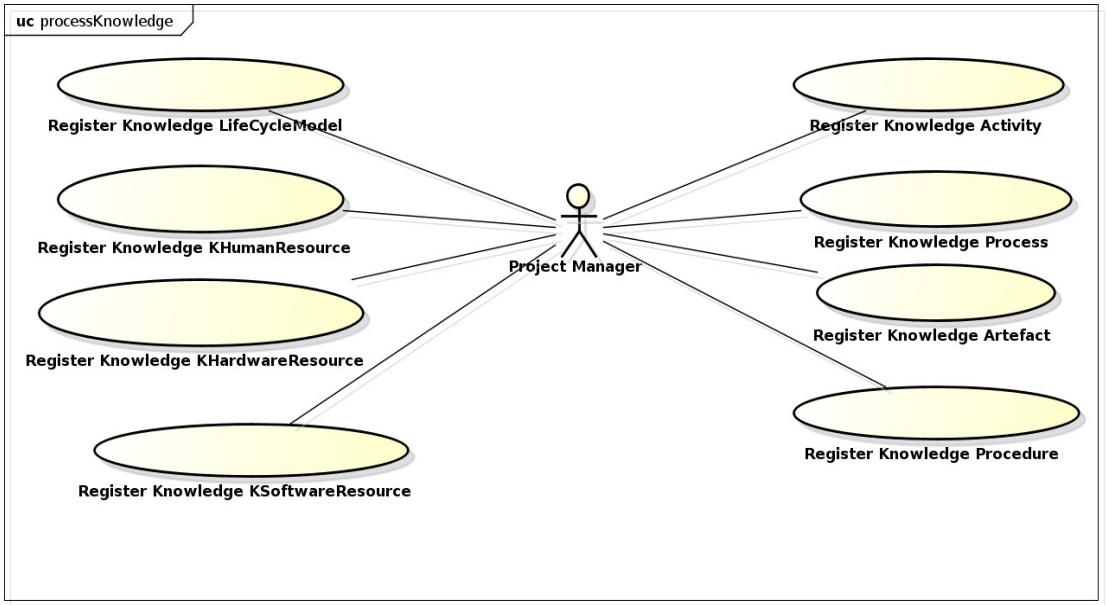

Figura 3.5 Diagrama de caso de uso processKnowledge

O gerente de projetos pode cadastrar conhecimentos *(Register Knowledge Activity), (Register Knowledge Process), (Register Knowledge Artefact), (Register Knowledge Procedure), (Register Knowledge LifeCycleModel), (Register Knowledge KHumanResource), (Register Knowledge KHardwareResource) e (Register Knowledge KSotfwareResource)*, incluindo novos conhecimentos, alterando, consultando e excluindo um conhecimento no sistema.

3.1.5 Especificação de Requisitos da Gerência de Processos Padrão (*standardProcess*)

O diagrama de casos de uso mostrado na Figura 3.6 modela as funcionalidades disponíveis para gerência de processos padrão.

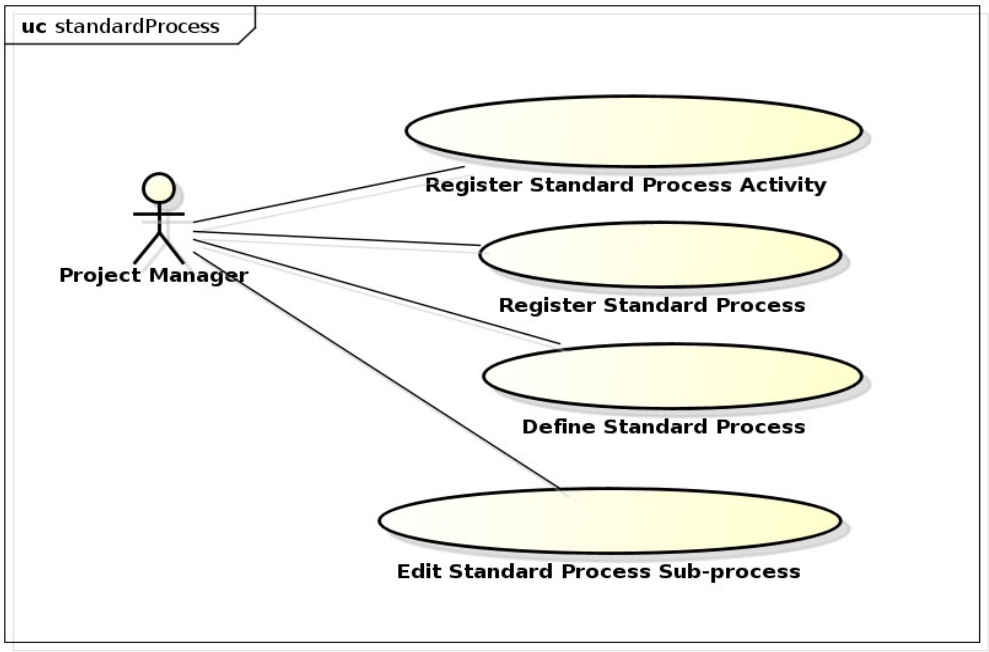

Figura 3.6 Diagrama de caso de uso standardProcess

O ator gerente de projetos pode cadastrar as atividades de um processo padrão *(Register Standard Process Activity)*, incluindo novas atividades, alterando, consultando e excluindo uma atividade de processo padrão no sistema. É importante saber que na inclusão registra-se nome, o tipo que é conhecimento atividade, sub e préatividades. Ele também pode cadastrar processos padrão (*Register Standard Process)*, incluindo novos processos padrão, editando, consultando e excluindo processos padrão do sistema neste caso o projetista só poderá incluir nome e descrição do processo padrão. Ele também pode editar subprocessos de processo padrão (*Edit Standard Process Sub-Process)*, incluindo a definição das macro atividades, a definição do tipo de interação entre o subprocesso em definição e o subprocesso de engenharia e a exclusão de um subprocesso do processo padrão.

E por fim o ator pode definir processo padrão *(Define Standard Process)*, definindo processos padrão com suas características como sub-processos, atividades, artefatos, procedimentos e entre outros, sendo que estes podem ser processos padrão especializados.

3.1.6 Especificação de Requisitos do Controle de Processos (*processControl*)

O diagrama de casos de uso mostrado na Figura 3.7 modela as funcionalidades disponíveis para controle de processos de projeto.

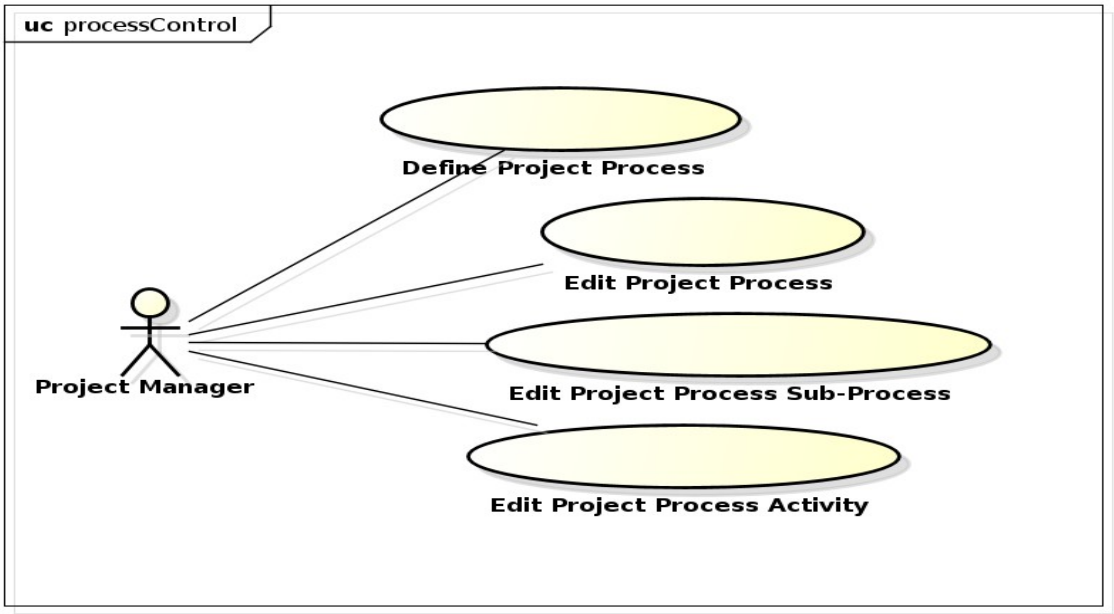

Figura 3.7 Diagrama de caso de uso processControl

O ator gerente de projetos pode definir processo de projeto (*Define Project Process)*, ou seja, o processo para o projeto corrente**,** editar o processo de projeto *(Edit Project Process*), alterando a definição de subprocessos e a exclusão do processo de projeto. O ator também pode editar subprocessos do processo de projeto *(Edit Project Process Sub-Process)* permite a edição de subprocessos do processo de projeto, incluindo a definição de novas macro-atividades e a exclusão de subprocessos. E por fim o ator pode editar atividades de processo de projeto *(Edit Project Process Activity)*, que permite a edição das atividades de um subprocesso do projeto, incluindo a edição de artefatos produzidos e requeridos, procedimentos requeridos, recursos necessários, pré-atividades, sub-atividades e propriedades da atividade, além do planejamento da documentação dos produtos da atividade e a exclusão de atividades.

3.1.7 Requisitos Não funcionais

Finalmente, durante o levantamento de requisitos foram levantados e especificados os requisitos não-funcionais para a nova base do ODE. Foram considerados como requisitos não-funcionais relevantes: segurança de acesso, facilidade de aprendizado e de operação, manutenabilidade, reusabilidade e facilidade de extensão. Mais detalhes podem ser vistos no documento de especificação de requisitos.

3.2 Análise

A atividade de análise, dentro do paradigma da Orientação a Objetos, tem por objetivo identificar objetos do mundo real, características desses objetos e relacionamentos entre eles que são relevantes para o problema a ser resolvido, especificando e modelando o problema de forma que seja possível criar um projeto orientado a objetos efetivo (PRESSMAN, 2001). A seguir, são apresentados os modelos de classes como um dos resultados desta fase.

As classes foram divididas nos mesmos subsistemas (pacotes UML) apresentados na Figura 3.1. Para cada diagrama, é apresentada uma descrição breve das classes e seu papel no sistema. Para uma descrição mais detalhada, o leitor pode consultar o documento de especificação de análise, disponível para download no site do projeto[11].

3.2.1 Pacote *userControl*

A Figura 3.8 apresenta o diagrama de classes do subsistema *userControl*, esse diagrama foi retirado de (CHAVES, 2011a), mas nesse caso traduzido para o inglês.

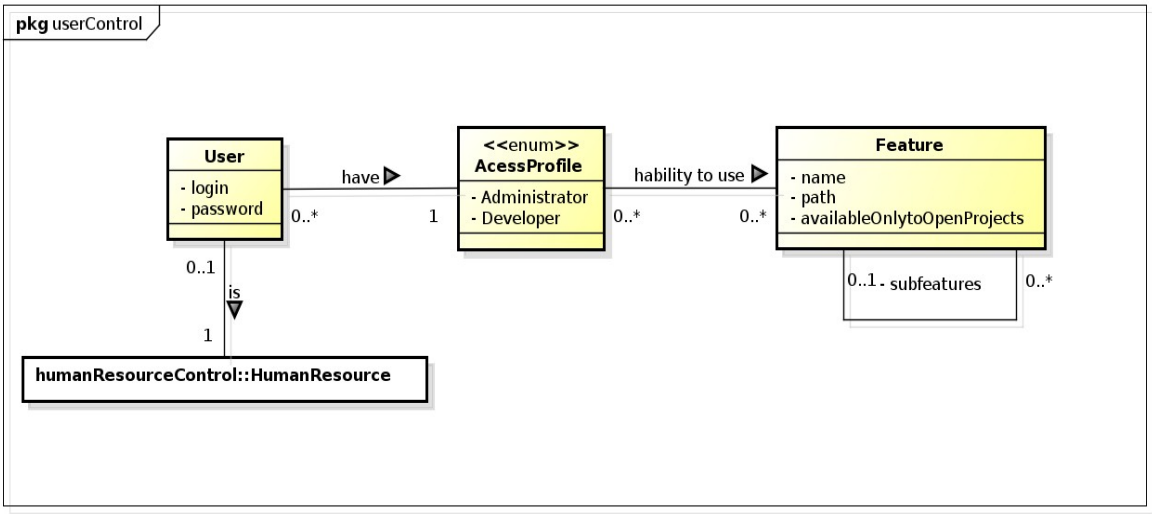

Figura 3.8 Diagrama de classes userControl

Pelo diagrama, observa-se que um usuário (*User*) possui um perfil de acesso (*AccessProfile*) que o habilita a acessar várias funcionalidades (*Feature*). Uma funcionalidade, por sua vez, pode possuir subfuncionalidades. Um usuário representa um determinado recurso humano no ambiente ODE.

[11] https://nemo.inf.ufes.br/pt-br/projetoode/eode/Especificação_v1.0.pdf

3.2.2 Pacote *humanResourceControl*

A Figura 3.9 apresenta o diagrama de classes do subsistema *humanResourceControl*.

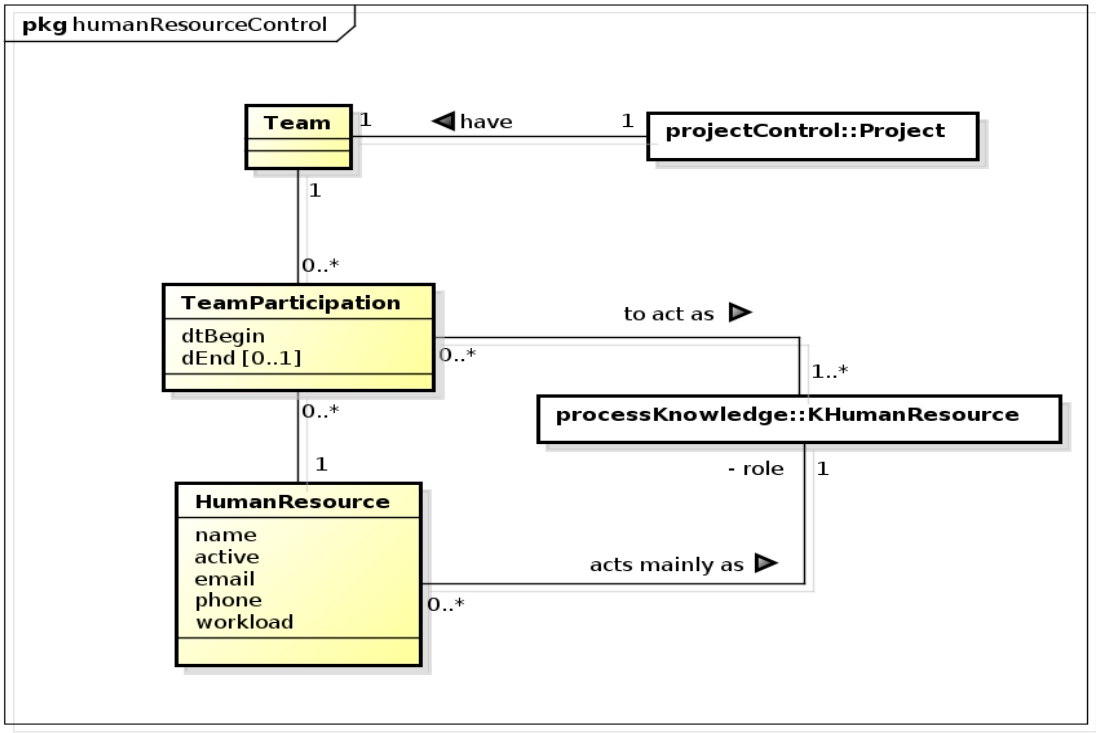

Figura 3.9 Diagrama de classes humanResourceControl

Pelo diagrama, cada recurso humano (*HumanResource*) atua como um conhecimento sobre recurso humano (*KHumanResource*) e participa (*TeamParticipation*) de uma equipe (*Team*) por tempo previamente determinado. Cada equipe está relacionada a somente um projeto (*Project*) mesmo que um recurso humano esteja em várias equipes.

3.2.3 Pacote *projectControl*

A Figura 3.10 apresenta o diagrama de classes do subsistema *projectControl*, que consiste simplesmente da classe *Project*, representando um projeto.

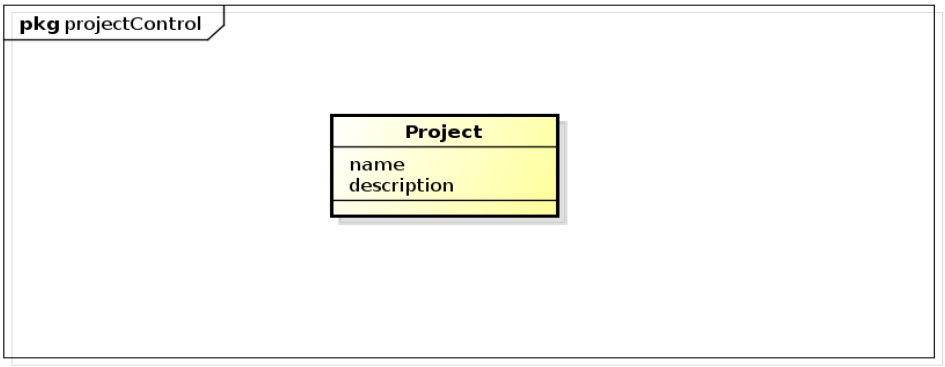

Figura 3.10 Diagrama de classes projectControl

3.2.4 Pacote *processKnowledge*

A Figura 3.11 apresenta o diagrama de classes do subsistema *processKnowledge* esse diagrama foi realizado por (SEGRINI, 2009) em português e traduzido para esse projeto para o inglês.

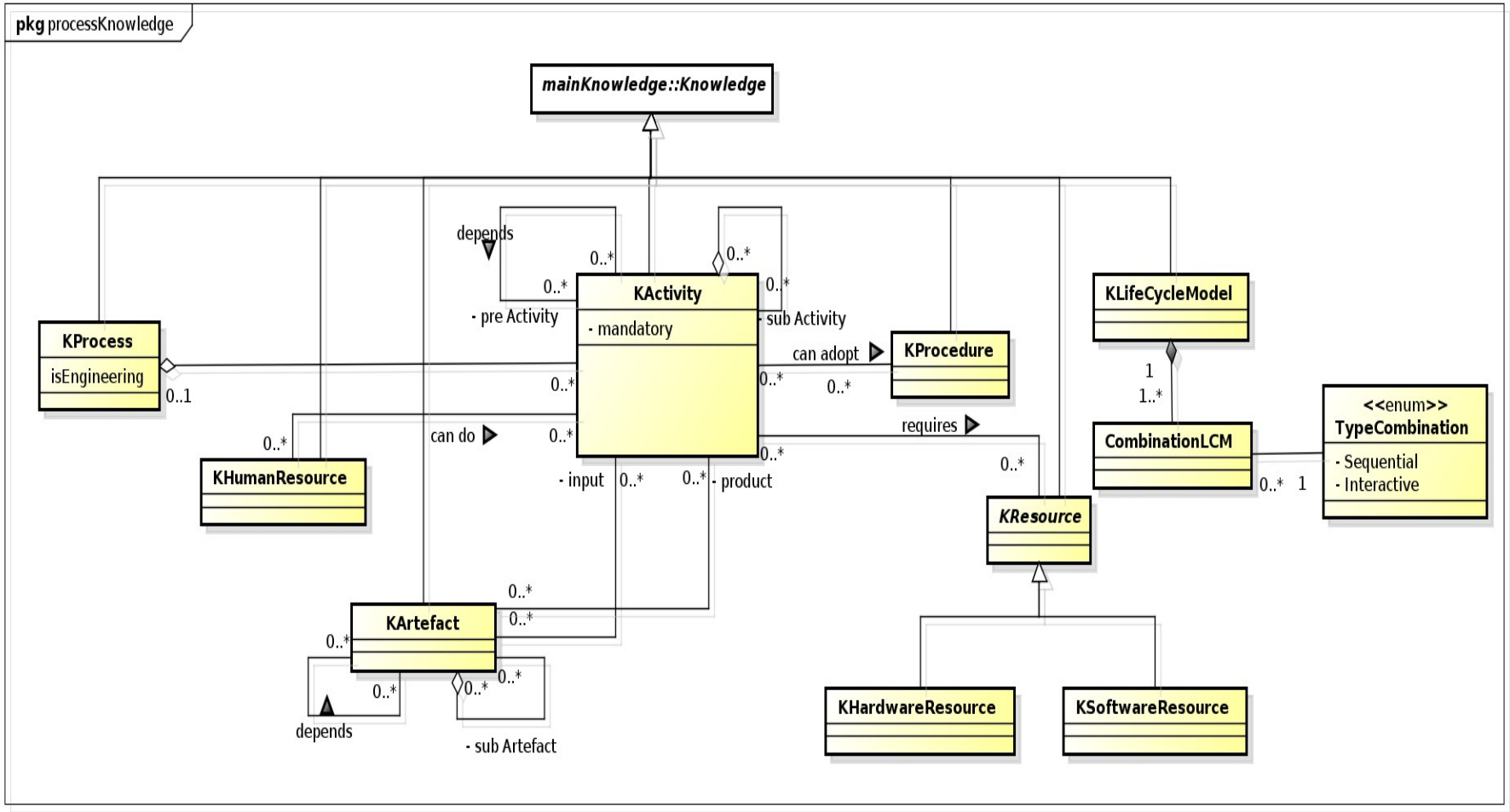

Figura 3.10 Diagrama de classes projectControl

De conhecimento registra-se nome e descrição. Para conhecimento de atividade devem-se registrar, também, se é obrigatória e os conhecimentos de sub e pré-atividades, artefatos envolvidos que podem ser insumos ou produtos, recursos humanos, recursos de hardware e software, procedimentos e o processo envolvido. De conhecimento de artefato registram-se também dependentes e subartefatos. Para conhecimento de processo registra-se também se o processo é um processo de engenharia.

A classe *KProcess*, derivada do conceito de processo de software, representa tipos de processos, tais como processo de desenvolvimento de software, processo de gerência de projeto etc.

A classe *KActivity* representa o conhecimento acerca das atividades que podem fazer parte de um determinado processo. Para uma atividade neste nível, são definidos os possíveis artefatos (*KArtefacts*) consumidos ou produzidos, recurso humanos a serem utilizados (*KHumanResource*), recursos (*KResource*) utilizados e procedimentos (*KProcedure*) adotados que podem ser selecionados na definição do processo.

3.2.5 Pacote *standardProcess*

A Figura 3.12 apresenta o diagrama de classes do subsistema *standardProcess*. O diagrama foi feito por (SEGRINI, 2007) em português e traduzido para esse projeto para o inglês.

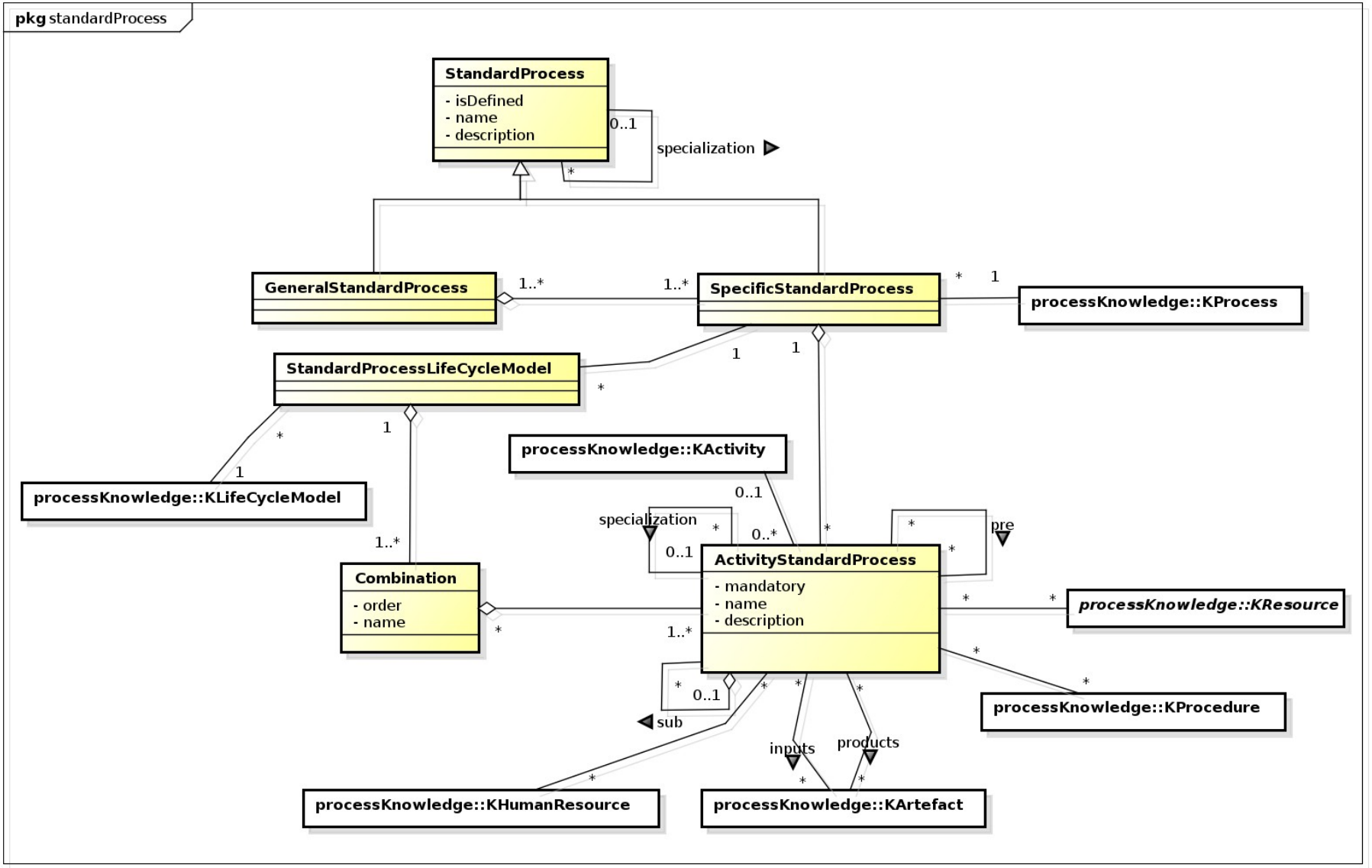

Figura 3.12 Diagrama de classes standardProcess.

Um processo padrão definido para uma organização é dito um processo padrão geral (*GeneralStandardProcess*) e é composto por subprocessos, ditos processos padrão específicos (*SpecificStandardProcess*). Estes, por sua vez, têm um tipo (*KProcess*) e são compostos por atividades (*ActivityStandardProcess*). Uma atividade pode, ainda, ser decomposta em sub-atividades.

Dependendo do tipo de um processo padrão específico, ele pode ser de engenharia (tipicamente os processos de desenvolvimento e manutenção) ou não (os demais processos). Em um processo padrão geral só pode haver um único processo padrão específico que seja de engenharia. Já para um processo padrão específico de engenharia, podem-se definir modelos de ciclo de vida (*StandardProcessLifeCycleModel*) que podem ser aplicados quando de sua instanciação. Um modelo de ciclo de vida definido para um processo padrão tem um tipo (*KLifeCycleModel*) e é composto por combinações de atividades (*Combination*). As atividades dentro de uma combinação devem obedecer a uma ordem de precedência. Em outras palavras, um modelo de ciclo de vida somente pode ser definido para um processo padrão específico de engenharia, enquanto um tipo de interação somente pode ser definido para um processo padrão específico que não seja de engenharia.

Para cada atividade do processo padrão devem ser definidos um tipo (*KActivity*) e seus ativos, que são os tipos de artefatos produzidos e consumidos (*KArtefact*), os procedimentos a serem adotados (*KProcedure*), os tipos de recursos requeridos (*KResource*) e a ordem de precedência entre as atividades.

Ainda é possível especializar um processo padrão e reutilizar um processo padrão específico. Quando se especializa um processo padrão, instancia-se um novo processo padrão realizando-se uma cópia fiel do processo padrão de origem. A única ressalva é com relação aos modelos de ciclo de vida, que devem ser redefinidos para o processo padrão especializado. Um processo padrão especializado deve ser caracterizado. Na caracterização de um processo, informam-se as características que foram levadas em consideração para especializá-lo.

Já quando se decide reutilizar um processo padrão específico, um novo processo padrão deste tipo é criado, realizando-se uma cópia fiel do processo padrão específico a ser reutilizado. Só é possível especializar, reutilizar ou instanciar um processo padrão para um projeto caso ele tenha tido sua definição finalizada.

3.2.6 Pacote *processControl*

A Figura 3.13 apresenta o diagrama de classes do subsistema *processControl*.

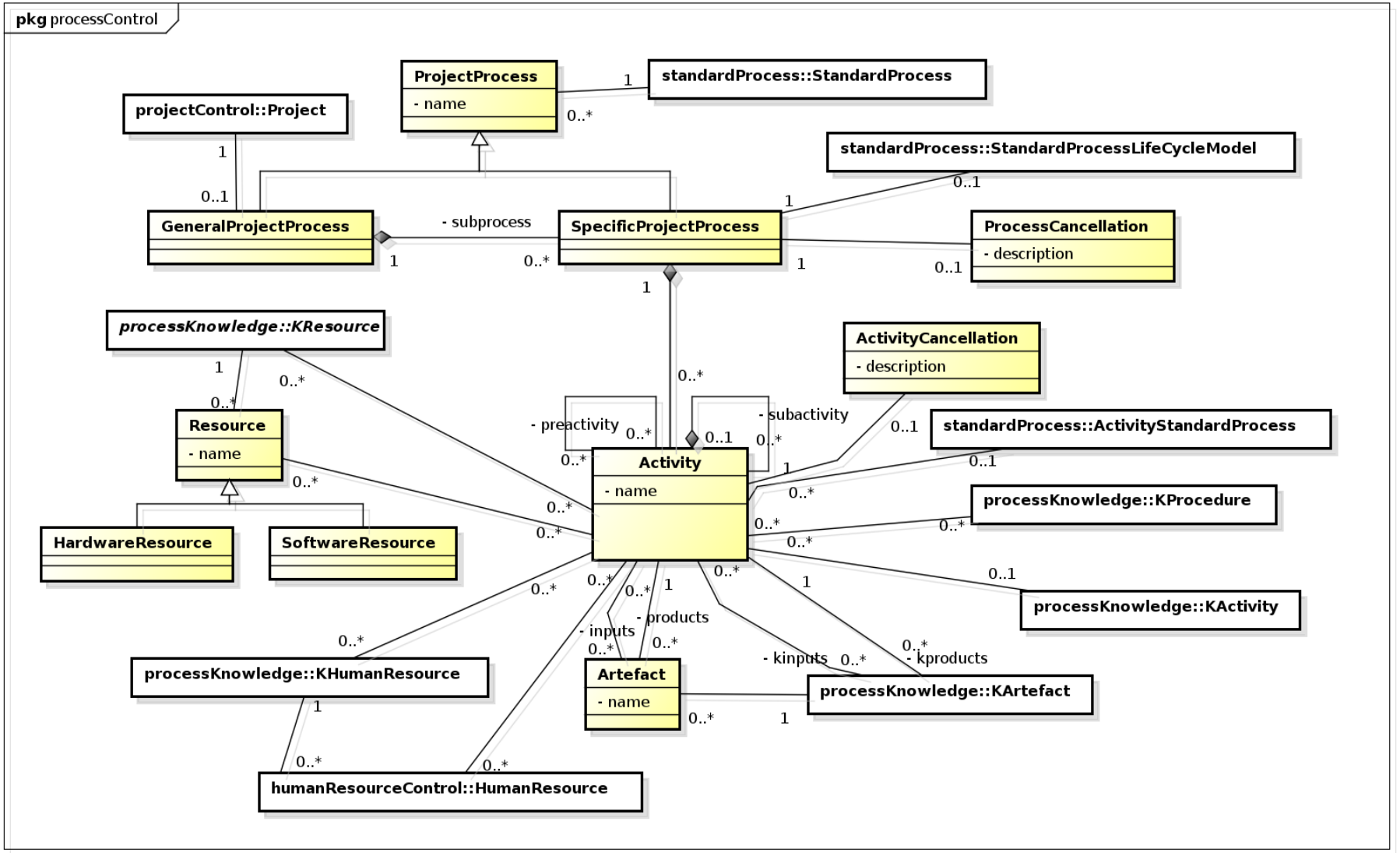

Figura 3.13 Diagrama de classes processControl.

O diagrama foi feito por (SEGRINI, 2007) em português e traduzido para esse projeto para o inglês. Este pacote trata da definição de processos de projeto. Um processo de projeto é definido para um projeto (*Project*), tomando por base um processo padrão (*StandardProcess*) e um modelo de ciclo de vida (*StandardProcessLifeCycleModel*) para o processo de engenharia do processo padrão. De forma análoga ao processo padrão, um processo de projeto pode ser composto por subprocessos ou por atividades (*Activity*). Quando um processo de projeto é composto por processos, deve ser uma instância da classe *GeneralProjectProcess*, definido com base em um *GeneralStandardProcess*. Quando o processo é composto por atividades, deve ser uma instância da classe *SpecificProjectProcess*, tendo como base um *SpecificStandardProcess*.

Definido o processo de projeto inicial com base no processo padrão, o gerente de projeto pode alterar os elementos do processo, considerando as características específicas do projeto. Assim, é possível incluir, alterar ou excluir processos, atividades, artefatos (*Artefact*), recursos (*Resource*) e procedimentos (*Procedure*).

Na abordagem de definição de processos de ODE, o Gerente de Projeto tem total liberdade para alterar os ativos de processo previamente definidos com base no processo padrão. Contudo, alguns eventos são registrados de forma a permitir avaliar a conformidade do processo definido com o processo padrão. Dessa forma, as exclusões de processos inteiros e atividades obrigatórias descritas no processo padrão devem conter uma justificativa do gerente de projeto descrevendo os motivos da exclusão. As classes *ProcessCancellation* e *ActivityCancellation* foram criadas para conter o registro de exclusão de processos e atividades, respectivamente.

3.3 Projeto e Implementação

O projeto é uma extensão do modelo de análise visando sua implementação em uma plataforma de implementação específica. Essa fase visa decidir como sistema irá operar em termos de: hardware, software e infraestrutura de rede, interface de usuário, formulários e relatórios, os programas específicos, bancos de dados e arquivos que serão necessários. Na análise são desconsideradas características tecnológicas, desse modo se diminui a distância do imaginário e real. Neste capítulo é apresentado o projeto de arquitetura do sistema (Seção 3.3.1). Em seguida, serão apresentadas as arquiteturas completas apenas das partes de Gerência de Processos Padrão (Seção 3.3.2) e Controle de Processos (Seção 3.3.3). Estes subsistemas foram escolhidos pelo fato de ODE ser centrado em processos e, portanto, serem partes fundamentais do núcleo do ambiente. Finalmente, as fases de implementação e testes são dis-

cutidas (Seção 3.3.4) e o protótipo implementado é apresentado (Seção 3.3.5). Para uma descrição mais detalhada, o documento de projeto completo encontra-se (https://www.nemo.inf.ufes.br/pt-br/projetoode/eode/ Projeto_v1.0).

3.3.1 Arquitetura do Sistema

Esta fase se inicia com a definição da Arquitetura do Sistema, que mapeia os conceitos modelados na fase de análise para a plataforma tecnológica de desenvolvimento, acrescenta aspectos de implementação e diminui a abstração, aproximando-se do nível de implementação em uma linguagem de programação.

A primeira atividade da elaboração do projeto de arquitetura do sistema é a divisão das classes em pacotes. Desta forma, são estabelecidos níveis de abstração, que são organizados em camadas e tratados separadamente durante esta fase (FALBO, 2000).

No caso do Projeto ODE distribuído e extensível, tal plataforma é composta pela utilização do Eclipse como base para extensão, o *plug-in* EGit para distribuição, a linguagem de programação *Java* e um *web service* com funcionalidades do ODE (SALVATORE, 2014).

A arquitetura de software do ambiente ODE é organizada em três camadas, como mostra a Figura 3.14.

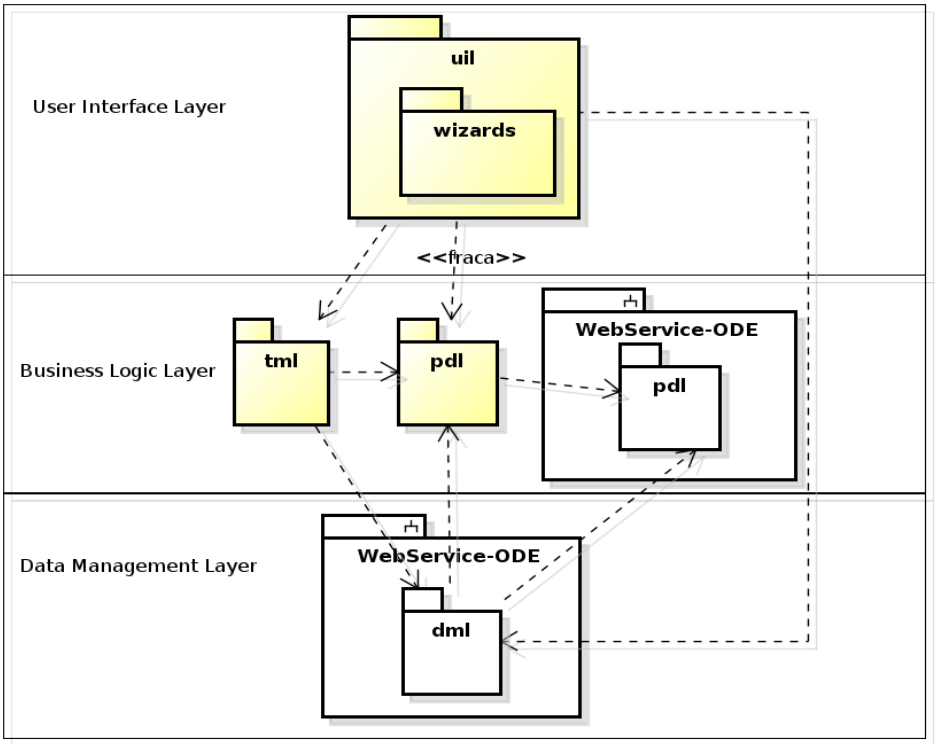

Figura 3.14 Diagrama da Arquitetura do Sistema núcleo ODE.

Para organizar a *user interface layer (*camada de Interface com o Usuário) e sua comunicação com a *business logic layer (*Camada de Lógica de Negócio), é usado o padrão arquitetônico MVC (Modelo-Visão-Controlador). Assim, o pacote *uil* contém tanto classes desempenhando o papel de visão, enquanto classes desempenhando o papel de controlador. Todas as classes do pacote *uil* que desempenhem papel de controladores são nomeadas começando com o prefixo **Ctrl**.

Para organizar a camada de Lógica de Negócio, é usado o padrão arquitetônico Camada de Serviço, o qual considera dois tipos de lógica de negócio: a Lógica de Domínio, que trata das classes de domínio capturadas na fase de análise e são agrupadas no *problem domain layer* (*pdl*), e a Lógica de Aplicação, que se refere à lógica de negócio descrita pelos casos de uso e tratada pelo *task management layer* (*tml*). Uma vez que as classes do pacote *tml* capturam lógica de aplicação, elas devem ser nomeadas começando com o prefixo **Appl**.

As classes de visão do *uil* chamam os métodos básicos das classes do *pdl* para montar as interfaces e para criar objetos do *pdl* a serem passados como parâmetro na comunicação com as classes controladoras (pacote *uil*) e com as classes de *tml*.

Por fim, a *data managenent layer (*camada de Gerência de Dados) é organizada seguindo o padrão DAO (*Data Access Object* - Objeto de Acesso a Dados).

Para utilização das funcionalidades de ODE, foram usados serviços Web (*Web Services*). *Web Service* é uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes através da Web. Toda comunicação passa a ser dinâmica e principalmente segura. Ele faz com que os recursos da aplicação do software estejam disponíveis sobre a rede (W3C, 2014).

Como observado, a *dml* (camada de gerência de dados) é realizada por meio de acesso ao WebService-ODE encontrado em http://dev.nemo.inf.ufes.br:8080/ode-web/servicos e implementados por SALVATORE (2014).

Também foi usado na *uil* um subpacote *wizards*. *Wizards* são assistentes que permitem capturar a entrada do usuário de uma forma estruturada, sequencial. Um assistente guia passo a passo o usuário através de páginas diferentes para cumprir uma tarefa específica (VOGEL, 2014), assim como já é usado na plataforma Eclipse.

A seguir, são detalhados dois subsistemas: *standardProcess* na seção 3.3.2 e *controlProcess* na seção 3.3.3. Essas seções apresentam as classes de utilitário que foram usadas na nova base do ODE.

3.3.2 Pacote *standardProcess*

Conforme apresentado no Capítulo 2, este pacote contém as classes que modelam processos padrão.

3.3.2.1 Camada de Lógica de Negócio (*Business Logic Layer*)

Camada de Domínio do Problema

As classes da Camada de Domínio do Problema são mostradas na Figura 3.15 com seus atributos. Nesse diagrama as mudanças realizadas no modelo de classes na fase de análise foi especificada a navegabilidade e os tipos dos atributos.

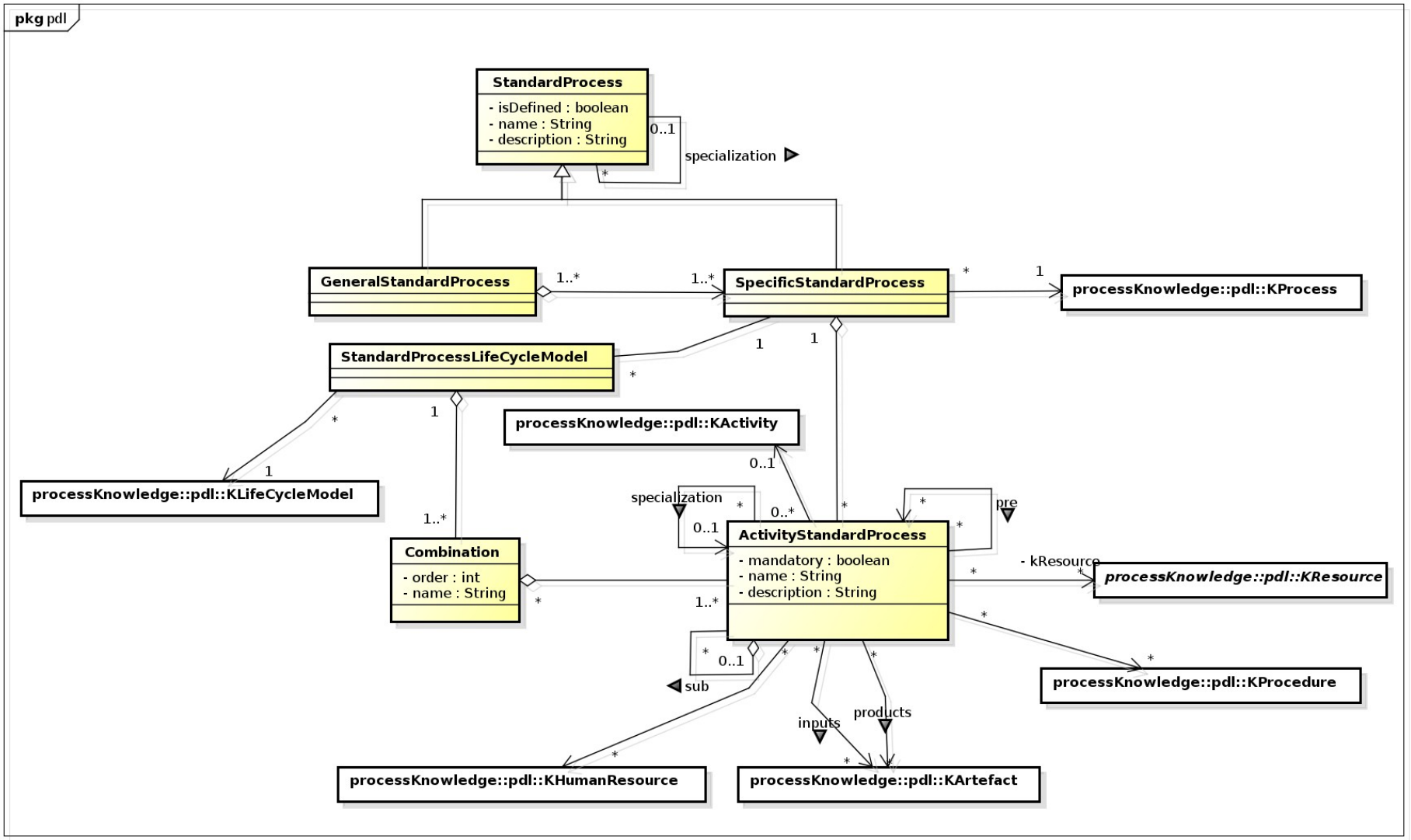

Figura 3.15 Diagrama de classes standardProcess.

Camada de Gerência de Tarefas

A abordagem escolhida para a camada de gerência de tarefas para a nova base ODE é a criação de uma classe de controle para cada caso de uso. Tais classes, também chamadas de classes de aplicação, tem um método para cada cenário do caso de uso que implementa. O pacote *standardProcess* possui classes de aplicação para todos os casos de uso do pacote, definido na especificação de requisitos (Capítulo 2).

Uma vez que o projeto da camada de gerência de tarefas está fortemente relacionado ao projeto da camada de iteração humana, um único diagrama foi elaborado, o qual é mostrado nas figuras 3.16 e 3.17.

As classes ApplCRUDSpecificStandardProcess, ApplCRUDGeneralStandardProcess e ApplCRUDActivityStandardProcess são classes de aplicação e registram, respectivamente, cadastros de processo padrão específico e geral e cadastro de atividade de processo padrão.

3.3.2.2 Camada de Interface com Usuário (*User Interface Layer*)

A camada de interface com usuário do pacote *standardProcess* reúne os elementos diagramáticos, ou seja, os desenhos que representam os elementos para cadastrar atividades de processo padrão e definição de processo padrão. Para uma melhor visualização, o diagrama dessa camada foi dividido em duas figuras nas figuras 3.16 e 3.17.

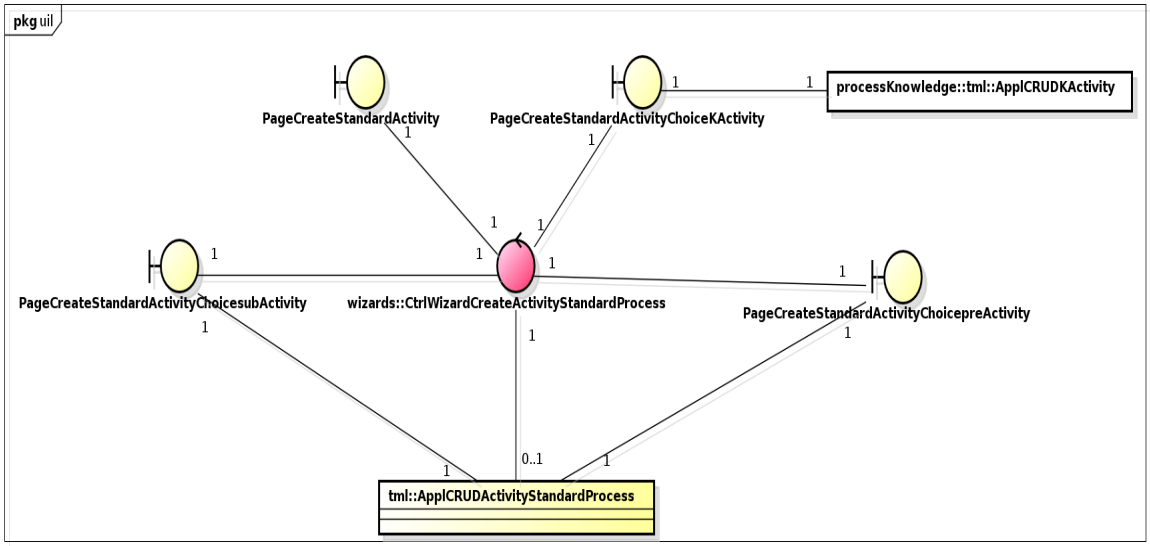

Figura 3.16 Diagrama da camada de iterface com usuário do pacote standardProcess (parte 1).

Nessa primeira parte pode-se observar pela figura como é feito o cadastro de atividades de processo padrão e o modelo visão para esse caso de uso. A classe ApplCRUDActivityStandardProcess tem ligação com

PageCreateStandardActivityChoicepreActivity e PageCreateStandardActivityChoicesubActivity para pegar as atividades de processos padrão disponível para cadastrar como sub ou pré-atividades e com WizardCreateActivityStandardProcess para cadastrar a atividade em questão. Assim como PageCreateStandardActivityChoiceKActivity tem ligação com ApplCRUDKActivity para escolher dentro das KActivity cadastradas o seu tipo, ou seja, a sua KActivity.

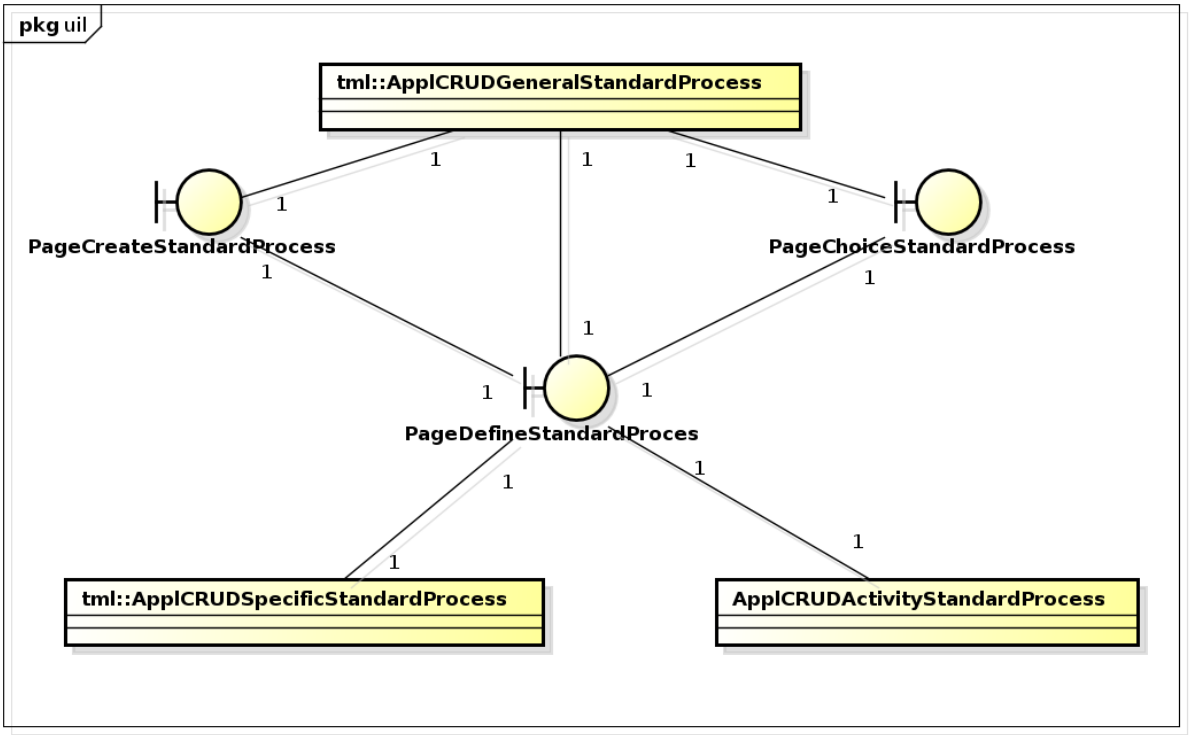

Figura 3.17 Diagrama da camada de iterface com usuário do pacote standardProcess (parte 2).

Nessa segunda parte pode-se observar pela figura como é feito o cadastro e escolha do processo padrão para definir e a definição de processo padrão e o modelo de visão para esses casos de uso. A classe ApplCRUDGeneralStandardProcess tem ligação com PageCreateStandardProcess para cadastrar um processo padrão geral a ser definido e com PageChoiceStandardProcess para escolher, dentre os processos padrão criados, qual definir naquela sessão e com PageDefineStandardProcess para definir o processo padrão escolhido.

As classes ApplCRUDSpecificStandardProcess e ApplCRUDActivityStandardProcess são usadas por PageDefineStandardProcess para, respectivamente, escolher subprocessos e escolher macro atividades para esses subprocessos.

A Figura 3.18 mostra como funciona a definição de processos na nova ferramenta.

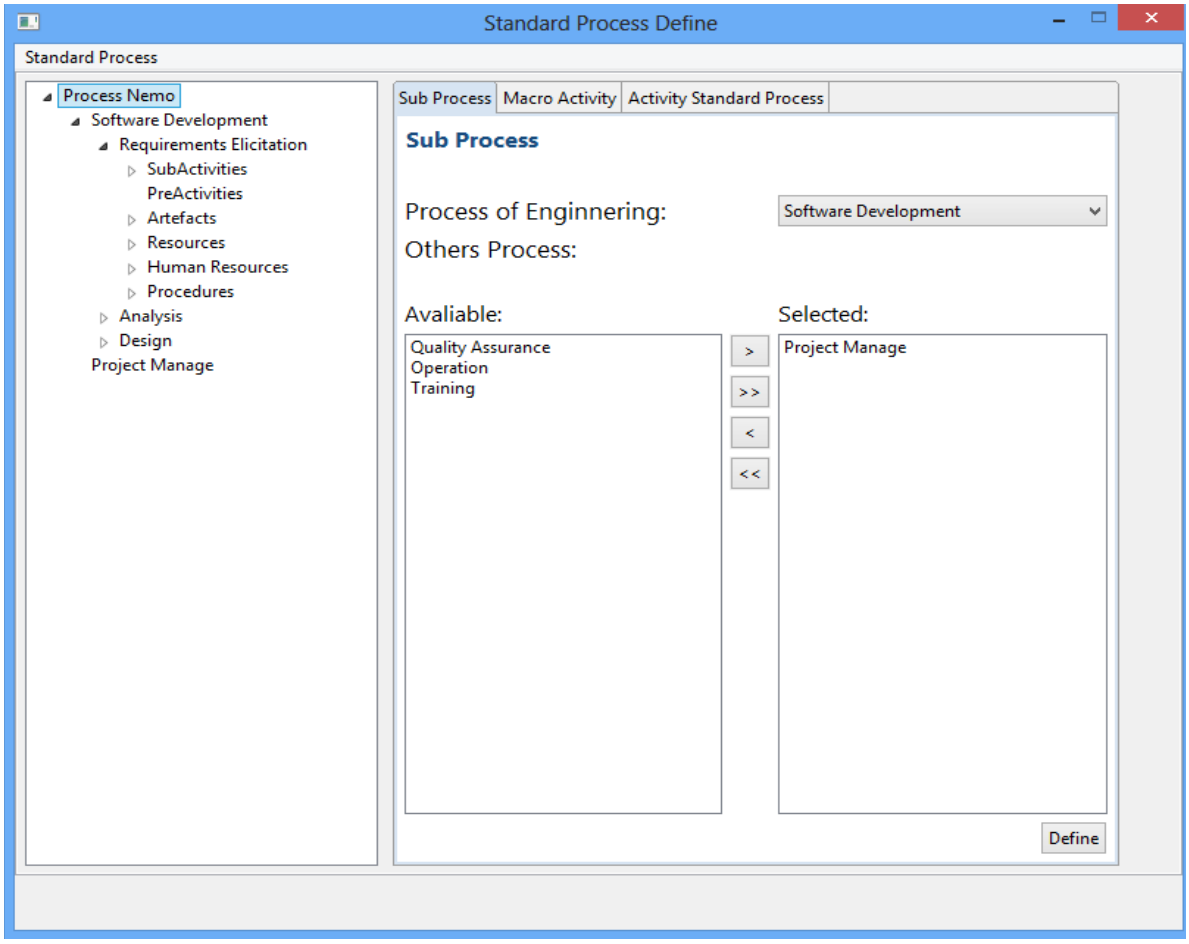

Figura 3.18 Janela de Definição de Processo Padrão.

A definição do processo padrão é feita na janela *PageDefineStandardProcess*. Essa janela é dividida em duas partes. Na primeira parte, localizada no lado esquerdo da janela, o processo padrão vai sendo exibido em forma de árvore. Já na segunda parte, localizada no lado direito da janela, são mostrados em abas os painéis para definição do processo padrão.

3.3.3 Pacote *processControl*

O pacote *processControl* é o núcleo de ODE. Por ODE ser um ADS Centrado em Processo, a definição de um processo de software para um projeto é essencial e as funcionalidades referentes a essa atividade são definidas nesse pacote.

3.3.3.1 Camada de Lógica de Negócio (*Business Logic Layer*)

Camada de Domínio do Problema

As classes da Camada de Domínio do Problema são aquelas que foram identificadas na fase de análise (Figura 3.13), mas agora com respectivos atributos e navegabilidade, mostrados na Figura 3.19.

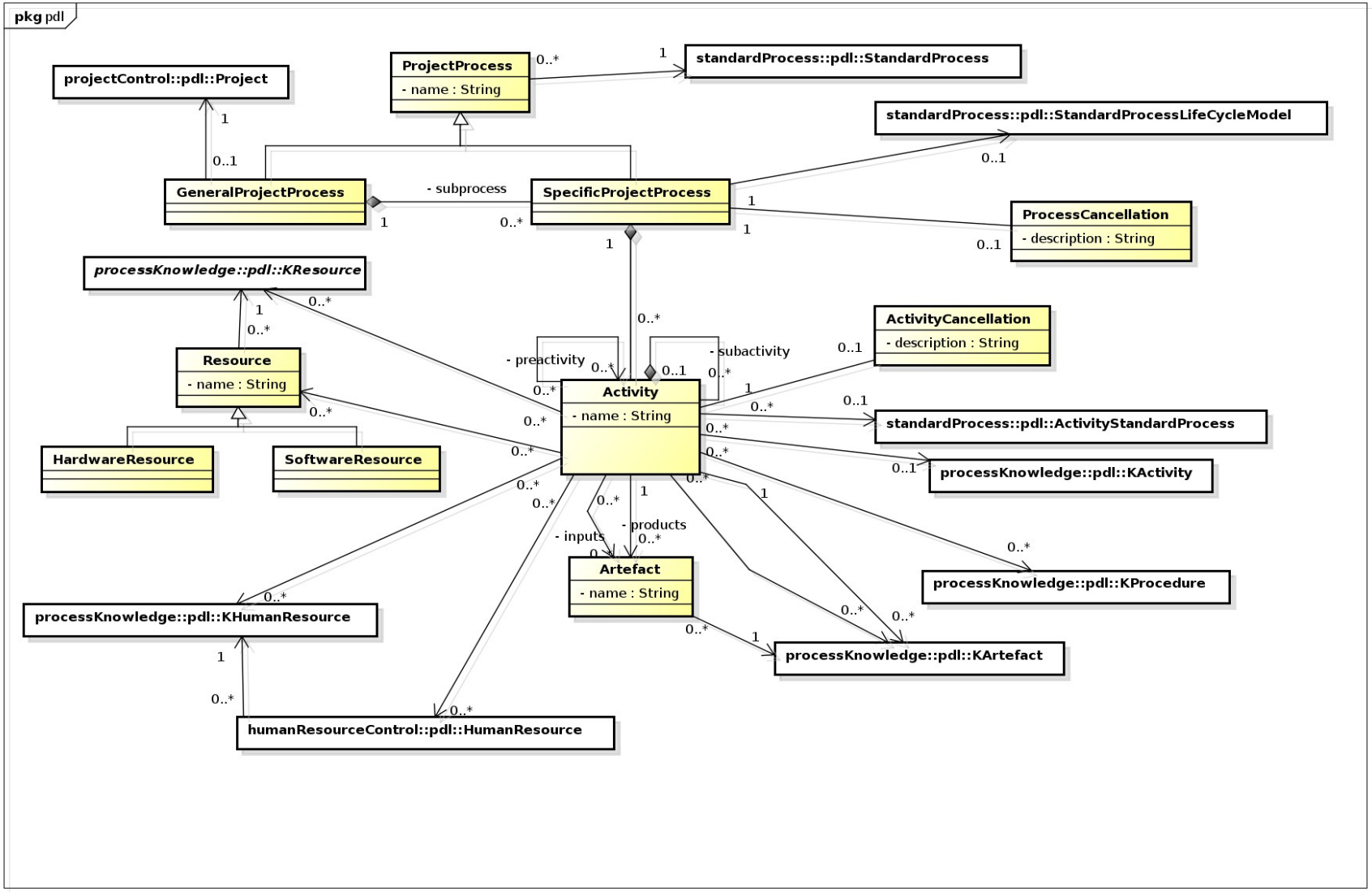

Figura 3.19 Diagrama de classes processControl.

Camada de Gerência de Tarefas

A camada de gerência de tarefas desse pacote possui três classes, que são responsáveis por realizar os casos de uso definidos. A divisão dos casos de uso em três classes de aplicação foi realizada com o propósito de tornar a manutenção do código mais fácil. Os casos de uso *Define Project Process* e *Edit Project Process* estão agrupados na classe ApplCRUDGeneralProjectProcess. O caso de uso *Edit Project Process Sub-Process* é realizado pela classe APPlCRUDSpecificProjectProcess. Já o caso de uso *Edit Project Process Activity* é realizado pela classe ApplCRUDActivity.

Uma vez que o projeto da camada de gerência de tarefas está fortemente relacionado ao projeto da camada de iteração humana, um único diagrama foi elaborado, mostrado na Figura 3.20.

3.3.3.2 Camada de Interface com Usuário (*User Interface Layer*)

A camada de interface com usuário do pacote *processControl* reúne os elementos diagramáticos, ou seja, os desenhos que representam os elementos para definir um processo de projeto. Para uma melhor visualização, o diagrama está na figura 3.20.

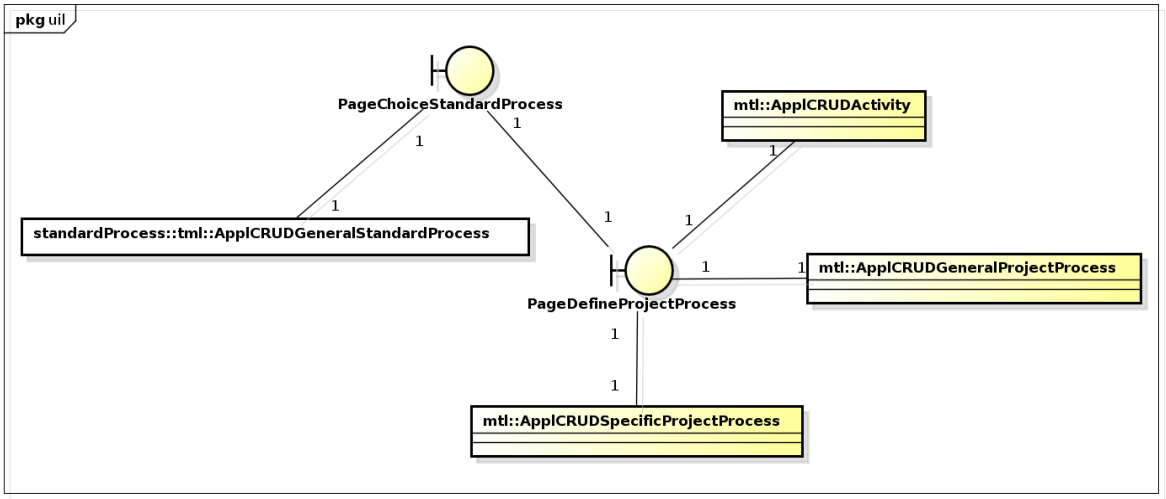

Figura 3.20 Diagrama da camada de iteração humana do pacote processControl.

A janela *PageDefineProjectProcess* provê acesso às funcionalidades relativas à definição de processos de projeto. Todos os projetos nesta primeira versão da ferramenta têm como modelo de ciclo de vida o modelo em cascata. Portanto não há, como nas versões anteriores, como escolher e editar o modelo de ciclo de vida. A Figura 3.21 mostra essa janela.

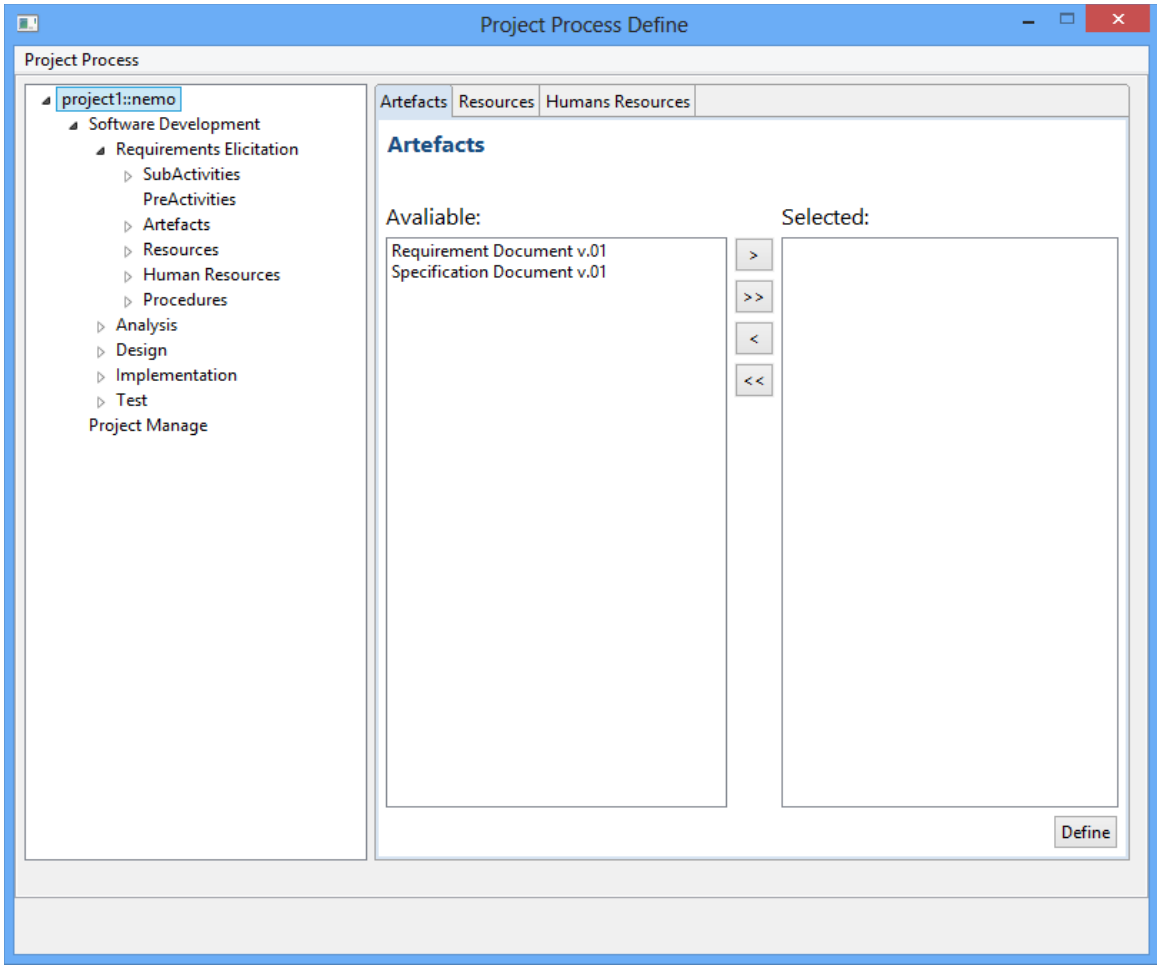

Figura 3.21 Janela para definição de processo do projeto.

A definição do processo do projeto é feita na janela *PageDefineProjectProcess*. Essa janela é dividida em duas partes. Na primeira parte, localizada no lado esquerdo da janela, o processo do projeto replicado do processo padrão escolhido vai sendo exibido em forma de árvore. Já na segunda parte, localizada no lado direito da janela, são mostrados em abas os painéis para definição do processo do projeto para que se escolham os artefatos, procedimentos, recursos de hardware, de software e recursos humanos alocados ao projeto, esses já previamente cadastrados.

3.3.4 Implementação e Testes

A ferramenta foi implementada utilizando a arquitetura descrita na Seção 3.3.1, com exceção, é claro, da camada de gerência de dados (DML), implementada juntamente com os *web services* que proveem as funcionalidades de persistência em ODE, desenvolvidos concorrentemente com a nova base do ODE por SALVATORE (2014).

A abordagem de testes adotada neste trabalho foi baseada nos testes dos eventos de cada caso de uso implementado. Primeiramente, foi testado cada evento de caso de uso, re-

alizando simulações de entradas de dados. Como exemplo, tome o caso de uso *Register User*: foi criado um teste para o fluxo normal desse caso uso e outros forçando as exceções elencadas em sua documentação.

Depois, buscou-se testar a integração dos casos de usos, realizando-se execuções das ferramentas num contexto geral que simulasse a utilização da ferramenta no dia a dia. Além disso, os testes também foram feitos no *web service*.

Codificação e testes foram realizados de maneira iterativa, seguindo uma ordem das funcionalidades a serem implementadas, cada uma delas sendo testada e depurada antes de se passar à seguinte. Por exemplo, primeiro buscou-se desenvolver o controle de usuários, cadastrando usuários, e o controle de projetos, cadastrado projetos, para finalmente integrar essas ferramentas e criar equipes de projetos. Na sequência foi implementado e testado o controle de processos, começando pela definição do processo padrão e seguindo com a especialização de processo padrão.

Após a definição e especialização de processos, foi desenvolvida a instanciação de um processo de projeto a partir de um processo padrão especializado ou não. Em seguida, testou-se a definição dinâmica de processos de projeto e deu-se uma grande ênfase no teste da alocação de recursos (de hardware, de software e humanos), ao cadastro de artefatos e à disponibilidade desse artefato.

3.3.5 Protótipo Implementado

Nesta subseção é mostrada parte da ferramenta implementada, inicialmente mostrando o acesso à ferramenta através de *login.*

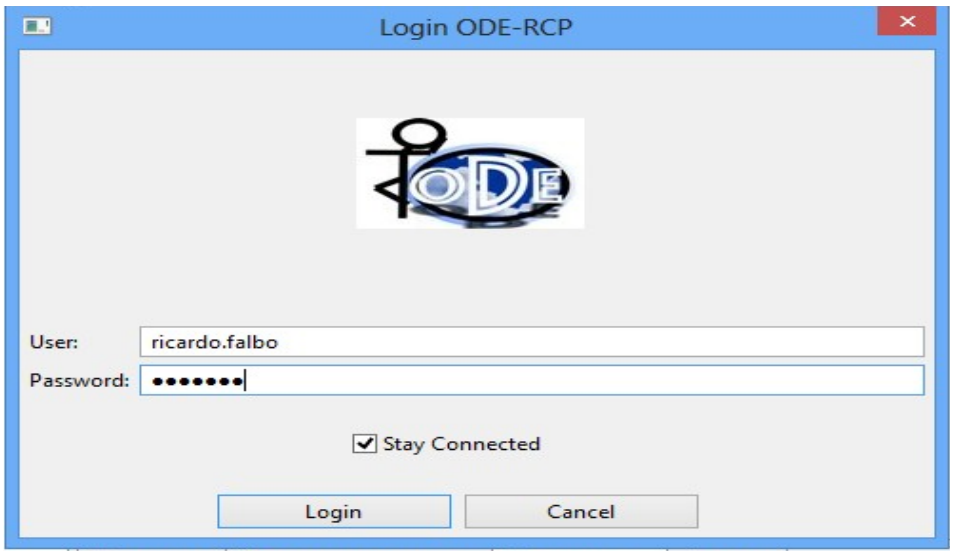

Figura 3.22 Janela para autenticar usuário.

Nessa janela o usuário digita o *login* e a senha previamente cadastrados para acesso ao ambiente. Já que o controle de acesso é uma das qualidades exigidas do ODE, essa autenticação se faz necessária.

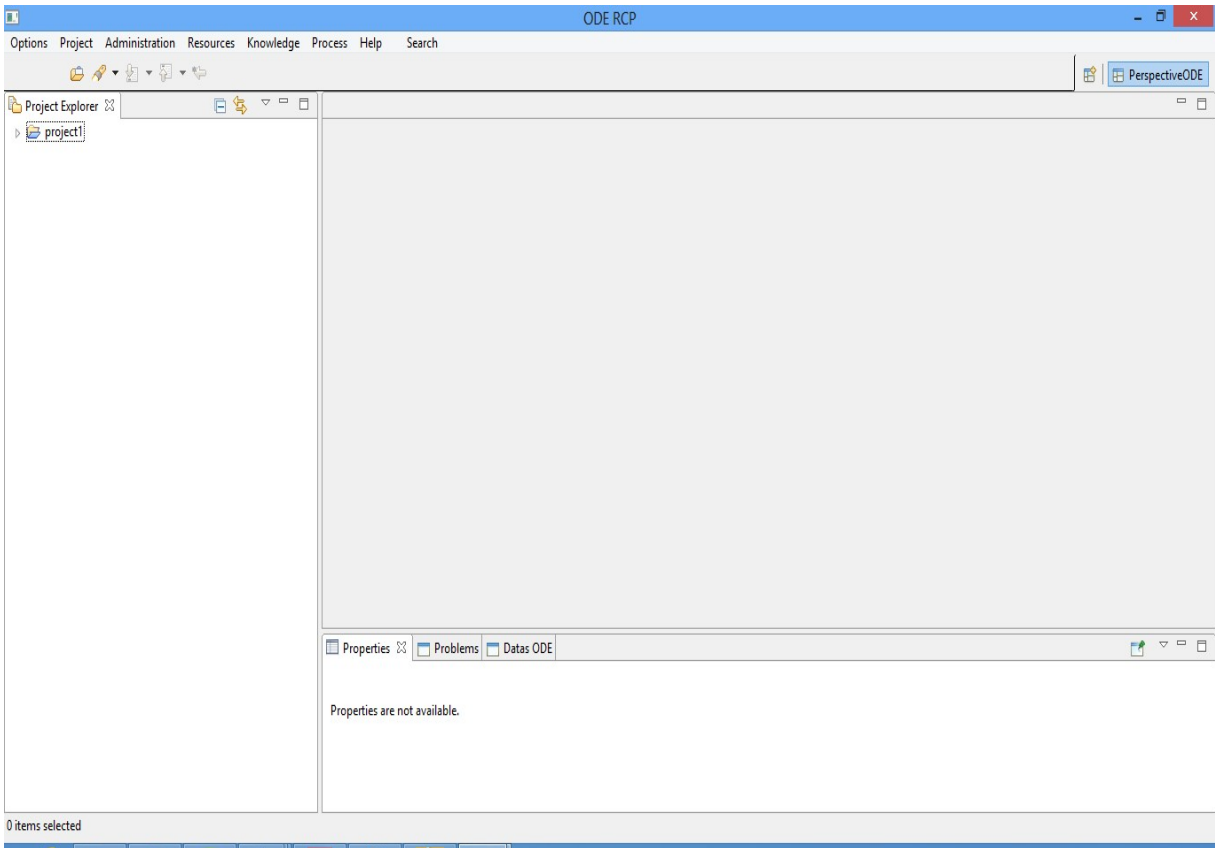

Figura 3.23 Janela do Ambiente ODE.

Na Figura 3.23 pode-se observar a entrada do ambiente para usuários administradores, os quais podem cadastrar projetos, conhecimento, usuários e recursos humanos, além de definir processos.

No canto superior direito observa-se também a seleção da palavra *PerpectiveODE*. O eclipse tem várias perspectivas para que plug-ins se personalizem. Foi criada uma perspectiva específica para o ODE, facilitando a utilização.

Para consulta, alteração e exclusão de alguns dados foi utilizado *Multi-Page Editor* do Eclipse, que é um editor que usa várias páginas de edição, como mostra a Figura 3.24.

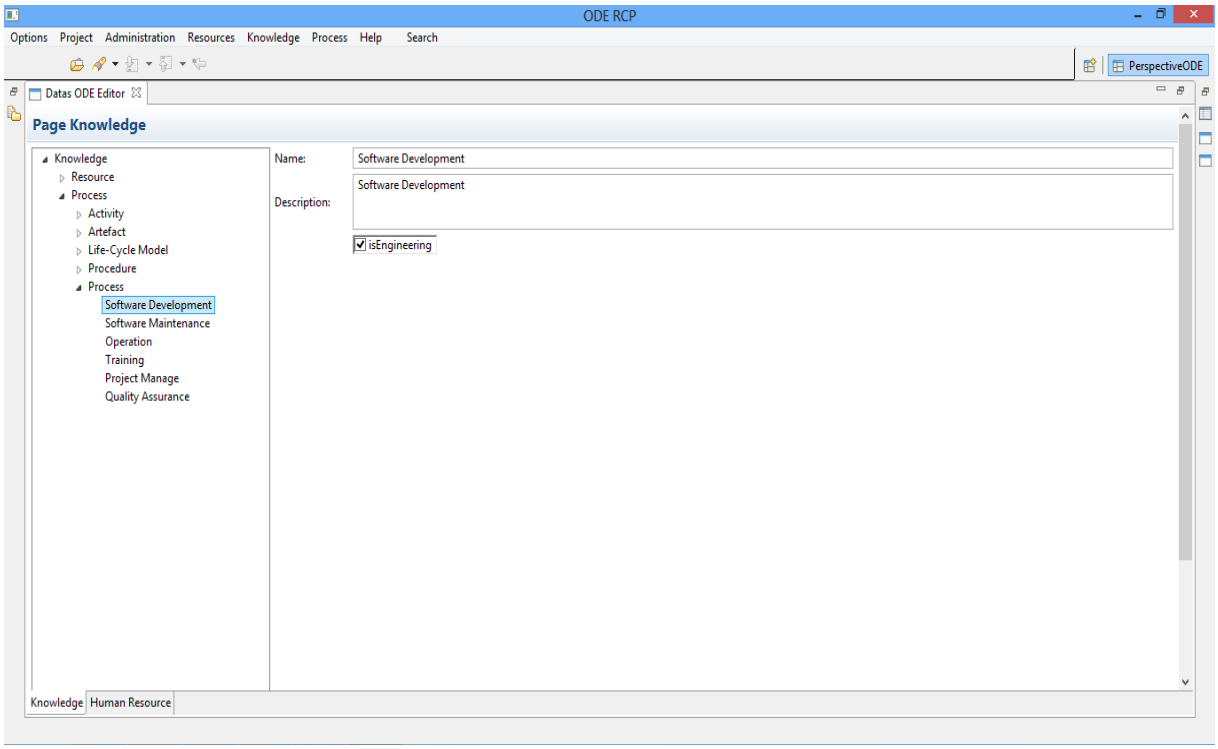

Figura 3.24 Janela de edição de dados em ODE.

Nessa janela observa-se que é possível navegar nos dados cadastrados, neste caso de Conhecimento em ODE, em uma árvore para consultá-los e editá-los. Tomando como base esses editores, pode-se fazer isso com outros dados.

Capítulo 4 – Considerações Finais

Neste capítulo são apresentadas as considerações finais a respeito do projeto desenvolvido neste trabalho. A Seção 4.1 discute as conclusões, a experiência adquirida e as contribuições oferecidas ao ambiente ODE. Na Seção 4.2, são apresentadas as perspectivas para trabalhos futuros.

4.1 Conclusões

As organizações desenvolvedoras de software entenderam que a qualidade no desenvolvimento de software tornou-se essencial. Neste contexto, a existência de um ambiente que contenha ferramentas CASE e as integre com facilidade de uso é essencial.

A nova base do ODE (EODE) cumpre essa necessidade, pois usa a facilidade de uso da plataforma Eclipse, podendo também utilizar outras ferramentas CASE já feitas para o Eclipse trazendo mais robustez ao ambiente.

A nova base do ODE deve permitir a integração de ferramentas, tratando-as de forma conjunta e não individualmente, facilitando a extensibilidade, sendo possível escolher quais ferramentas se quer integrar na base. Ela deve ser distribuída para que diversos usuários possam compartilhá-la independentemente do local dos projetos em que estão alocados.

Para que se entegre outras ferramentas personalizadas todas devem ser feitas como *plug-in* seguindo os padrões de desenvolvimento citados acima. Uma grande contribuição da nova base é que será mais fácil utilizar ferramentas que já existem como *plug-in* para eclipse na ferramenta como por exemplo diversos *plug-ins* que existem de modelagem UML para eclipse.

Este trabalho teve por objetivo implementar o ambiente ODE em uma nova plataforma que facilitasse a integração utilizando a nova base como *plug-in* e que fosse também distribuída integrando a nova base com o *plug-in* Egit de modo a facilitar a busca pela qualidade do ambiente atingindo esse objeto com a plataforma Eclipse.

Em termos de experiência adquirida, o trabalho foi muito importante, pois permitiu a consolidação dos conceitos vistos durante o curso de graduação em Ciência da Computação, como, por exemplo, modelagem conceitual, especificação de requisitos, projeto e implementação na linguagem de programação Java. Além disso, permitiu um aprofundamento em tópicos não muito abordados durante o curso, como, por exemplo, qualidade e processos de software.

4.2 Perspectivas Futuras

ODE, em versões mais antigas, tinha outras funcionalidades que faziam parte do núcleo, como o acompanhamento de projetos, além de ferramentas separadas que podiam ser acopladas, como a ferramenta de análise de ponto de função ou a de gerência de riscos. Tais ferramentas podem, agora, ser integradas como *plug-ins* da nova base do ODE.

Se faz necessário também evoluir a ferramenta de definição de processos com processos de software com mais ciclos de vida, mais tipos de procedimentos a serem acoplados ao conhecimento do processo e ao controle de processos, etc. Tais novas funcionalidades já existiam, inclusive, nas versões antigas do ODE, porém não foi possível incluí-las no escopo deste projeto.

Referências

ANISZCZYK, C; GALLARGO, D. **Introdução à plataforma Eclipse**. Disponível em <http://www.ibm.com/developerworks/br/library/os-eclipse-platform/> acessado em 22 de julho de 2014.

BERGER, P. **Instanciação de Processos de Software em Ambientes Configurados na Estação TABA**. Dissertação de Mestrado, COPPE/UFRJ, Rio de Janeiro, 2003.

BRINGUENTE, A.C.O. **Reengenharia de uma Ontologia de Processo de Software e seu Uso para a Integração de Ferramentas de Apoio ao Planejamento de Projetos**. Dissertação de Mestrado (Mestrado em Informática), Universidade Federal do Espírito Santo, Vitória - Brasil, 2011.

CALHAU, R.F. **Uma Abordagem Baseada em Ontologias para a Integração Semântica de Sistemas**. Dissertação de Mestrado (Mestrado em Informática), Universidade Federal do Espírito Santo, Vitória - Brasil, 2011.

CHAVES, R.G. **Documento de especificação de requisitos: Cadastro de usuários no Ambiente ODE**, NEMO-UFES, Vitória, Espírito Santo, Brasil, 2011a.

CHAVES, R.G. **Documento de especificação de requisitos : Cadastro de Recursos Humanos do Ambiente ODE (Versão Simplificada)**, NEMO-UFES, Vitória, Espírito Santo, Brasil, 2011b.

CHAVES, R.G. **Documento de projeto: Cadastro de usuários no Ambiente ODE**, NEMO-UFES, Vitória, Espírito Santo, Brasil, 2011c.

CHAVES, R.G. **Documento de projeto : Cadastro de Recursos Humanos do Ambiente ODE (Versão Simplificada)**, NEMO-UFES, Vitória, Espírito Santo, Brasil, 2011d.

CELINO, D.R. **Desenvolvimento da base do ambiente de desenvolvimento de software ODE em uma plataforma distribuída e extensível,** Projeto de iniciação científica UFES em 2014.

COAD, P.; Yourdon, E. **Projeto Baseado em Objetos**. Editora Campus: 1993.

COELHO, A.G.N. **Uma Infraestrutura de Gerência de Conhecimento em Organizações de Software Aplicada à Gestão de Riscos**. Dissertação de Mestrado (Mestrado em Informática), Universidade Federal do Espírito Santo, Vitória Brasil, 2010.

COLLINS-SUSSMAN, B; Fitzpatrick, B. W; Pi-lato, C. M. **Version Control with Subversion**, O'Reilly Media, June 2004.

DE QUEIROZ, F.B.; Ferreira, F.S.; Da Silva, L.S. **Análise de usabilidade dos Sistemas de**

Controle de Versão Subversion e Git. Disponível em < http://estatistica.googlecode.com/svn-history/r88/trunk/docs/especificacao/especificacao.pdf> acessado em 23 de julho de 2014.

FALBO, R.A. **A Experiência na Definição de um Processo Padrão Baseado no Processo Unificado**, Anais do II Simpósio Internacional de Melhoria de Processo de Software, SIMPROS'200. São Paulo, SP, Setembro/2000.

FALBO, R.A.; Natali, A. C. C.; Mian, P. G.; Bertollo, G.; Ruy, F. B. **ODE: Ontology-based software Development Environment**, In: IX CONGRESO ARGENTINO DE CIENCIAS DE LA COMPUTACIÓN, La Plata, Argentina, 2003, p. 1124-1135.

FUGGETTA, A. **Software Process: A Roadmap**. In Proc. of The Future of Software Engineering, ICSE'2000, Ireland, 2000.

HARRISON, W.; Ossher, H.; Tarr, P. **Software Engineering Tools and Environments: A Roadmap**, In: 22TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE-00), ACM Press, 2000.

FERREIRA, L. O. **Desenvolvimento de middleware para integração do repositório compweb com sistemas de controle de versão**. Monografia de conclusão de curso em Sistemas de Informação, ULBRA 2009.

MINOCHA, S. **Desenvolvendo seu primeiro Aplicativo em Eclipse RCP**. Disponível em <http://www.ibm.com/developerworks/br/library/os-ecl-rcpapp/> acessado em 22 de julho de 2014.

MIKKELSEN, T ; Pherigo, S, **Practical Software Configuration Management: The Latenight Developer's Handbook**, Prentice Hall PTR, Upper Saddle River, NJ, EUA, 1997.

MURER, R. **Introdução às plataformas de software**. Disponível em <http://webinsider.com.br/2012/06/16/introducao-as-plataformas-de-software> acessado em 22 de julho de 2014.

PFLEEGER, S.L. **Software Engineering: Theory and Practice**, 2nd Edition, New Jersey, USA, Prentice Hall, 2001.

PRESSMAN, R.S. **Software Engineering: A Practitioner's Approach**. 5th edition. New York, USA, McGraw Hill, 2001.

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

SALVATORE, T. **Uma Arquitetura Orientada a Serviços para o Ambiente de Desenvolvimento de Software ODE,** Projeto de iniciação científica UFES em 2014.

SEGRINI, B.M. E**volução do apoio à definição e ao acompanhamento de processos de ode. Projeto de Graduação** (Bacharel em Ciência da Computação), Universidade Federal do Espírito Santo, Vitória - Brasil, 2007.

SEGRINI, B.M. **Definição de Processos Baseada em Componentes. Dissertação de Mestrado** (Mestrado em Informática), Universidade Federal do Espírito Santo, Vitória - Brasil, Agosto/2009.

TRAVASSOS, G.H. **O Modelo de Integração de Ferramentas da Estação TABA**. Tese de Doutorado, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Rio d Janeiro, RJ, Março/1994.

TORVALDS, L.; **"The Linux Home Page at Linux Online"**. Disponível em HTTP://linux.org/, Acessado em 21 de julho 2014.

VOGEL, L. **Creating Eclipse Wizards - Tutorial**. Disponível em <http://www.vogella.com/tutorials/EclipseWizards/article.html /> acessado em 25 de julho de 2014.

W3C **Web Services Architecture**, Disponível em <http://www.w3.org/TR/ws-arch>, Acessado em 25 de julho 2014.