UNISINOS

Programa Interdisciplinar de Pós-Graduação em

Computação Aplicada

Mestrado Acadêmico

Michele Lermen

Molps: Uma Ontologia para definição de Linha de Produto de *Software* para Gerência de Projeto com Metodologias Ágeis

São Leopoldo, 2012

MICHELE LERMEN

Molps:
Uma Ontologia para definição de Linha de Produto de *Software* para Gerência de Projeto com Metodologias Ágeis

Dissertação apresentada como requisito parcial para a obtenção do título de Mestre pelo Programa Interdisciplinar de Pós-Graduação em Computação Aplicada da Universidade do Vale do Rio dos Sinos.

Orientador: Prof. Dr. Sérgio Crespo Coelho da Silva Pinto

São Leopoldo

2012

L616m Lermen, Michele

Molps: uma ontologia para definição de linha de produto de *software* para gerência de projeto com metodologias ágeis / por Michele Lermen. -- São Leopoldo, 2012.

180 f. : il. color. ; 30 cm.

Dissertação (mestrado) – Universidade do Vale do Rio dos Sinos, Programa Interdisciplinar de Pós-Graduação em Computação Aplicada, São Leopoldo, RS, 2012.

Orientação: Prof. Dr. Sérgio Crespo Coelho da Silva Pinto, Ciências Exatas e Tecnológicas.

1.Agentes inteligentes (*software*). 2.*Software* – Desenvolvimento. 3.Desenvolvimento ágil de *software*. 4. Programação (Computadores) – Gerência. 5.Ontologias. I.Pinto, Sérgio Crespo Coelho da Silva. II.Título.

CDU 004.4
004.413
004.42

Catalogação na publicação:
Bibliotecária Carla Maria Goulart de Moraes – CRB 10/1252

AGRADECIMENTOS

Está se concluindo mais um objetivo de minha vida. Durante dois anos de mestrado me dediquei muito, horas, dias, noites adentro fazendo trabalhos e pesquisas para chegar neste momento e poder dizer que concluí mais uma etapa. Chegar neste momento e sentir que todo o esforço dedicado deu certo, não tem sentimento que explique a sensação de trabalho concluído. Nesta caminhada não estive sozinha nenhum momento, pois, colegas, amigos, professores e familiares me acompanharam dando força e incentivo para chegar ao final. Diante disso, só tenho a agradecer todas estas pessoas que me acompanharam principalmente minha família e meu orientador. Agradeço a minha família pelo apoio e paciência nestes dois anos, pois sem vocês nada disto estaria valendo a pena. Ao meu orientador professor doutor Sérgio Crespo Coelho da Silva Pinto, deixo meu agradecimento pelas orientações, conhecimento passado e disponibilidade sempre que precisei de ajuda. Aos meus colegas um agradecimento especial por tudo o que passamos juntos, principalmente a minha colega Fabiane Penteado por estar sempre ao meu lado. Por fim, deixo meus agradecimentos a todos que fazem parte do PIPCA.

RESUMO

Esta dissertação tem por objetivo desenvolver uma ontologia que represente uma linha de produto para o gerenciamento de projetos de *software* com metodologias ágeis. O modelo de domínio da linha de produto é composto por conceitos formadores das metodologias ágeis *eXtreme Programming* (XP), *Scrum*, e a *Feature Driven Development* (FDD), bem como trata dos conceitos de gerenciamento de projetos baseados no PMBOK. Através da ontologia Molps se propõe representar este domínio, possibilitando que agentes de *software* realizem consultas na ontologia, inferindo as possíveis variabilidades a serem geradas no produto. A ontologia também permitirá o gerenciamento de processo de desenvolvimento de *software*. Para evidenciar tal flexibilidade e o gerenciamento do processo de desenvolvimento de *software* realizam-se testes na ontologia desenvolvida, onde os resultados são comentados e analisados.

Palavras chave: Metodologias Ágeis, Gerenciamento de Projetos, Linha de Produto de *Software*, Ontologias.

ABSTRACT

This study aims to develop an ontology to represents a *software* product line to be used for project management based on agile methodologies. The product line domain model was built utilizing concepts coming from eXtreme Programming (XP), *Scrum*, and feature driven development (FDD), and also concepts brought from project management through PMBOK. Through the ontology Molps proposes to represent this domain, enabling *software* agents to perform queries on the ontology, inferring the possible variability in the product to be generated. The ontology will also allow the management process of *software* development. To demonstrate this flexibility and process management *software* development take place in the ontology developed tests, where results are discussed and analyzed.

Key-words: Agile Methodologies, Project Management, *Software* Product Line, Ontology

LISTAS DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AFIT – *Air Force Institute os Technology*

API – *Aplication Programming Intetface*

ASD – *Agile Software Development*

BDI – *Belief – Desire - Intention*

DL – *Description Logic*

DSDM – Método de Desenvolvimento de Sistemas Dinâmicos

FDD – *Feature Driven Development*

FIPA – *Foundation for Intelligent Phisical Agents*

FODA – *Feature-Oriented Domain Analysis*

IDE – *Integrated Development Enviroment*

JADE – *Java Agent Development Framework*

LPS – Linha de Produto de *Software*

MaSE – *Multiagent System Engeneering*

OWL –*Web Ontology Language*

PMBOK – *Porject Management Body of Knowledge*

RDF – *Resource Description Framework*

SPARQL – *Protocol and RDF Query Language*

URI – *Uniform Resource Identifier*

XML – *Extensible Markup Language*

XP – *Extreme Programming*

XPO - *Extreme ProgrammingOntology*

W3C – *World Wide WebConsortium*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo introduz esta dissertação através da contextualização do domínio do problema, das motivações para seu desenvolvimento, das definições dos objetivos a serem atingidos, da metodologia aplicada na solução e da organização da dissertação.

1.1 CONTEXTUALIZAÇÃO

É amplamente reconhecido que a produção de *software* precisa tornar-se capaz de se adaptar e responder as necessidades dos clientes com maior flexibilidade e agilidade. Os produtos devem ser entregues aos clientes em menor tempo e custo pelas empresas, valendo-se para isso da reutilização de *software* (CERAVOLO et al., 2003). Um dos principais custos de *software* é a manutenção, tornando-se mais complexa e cara na medida em que sofre modificações. A reutilização de componentes pré-acabados tem se mostrado eficaz mesmo frente a variabilidade imposta pelo mercado atualmente. Para que o produto final atenda seu objetivo em plenitude, processos de desenvolvimento de *software* concomitantes com a gerência de projeto trazem um modelo produção e manutenção padrão, que permitem a geração de artefatos que atendam a expectativa de tempo, escopo e recursos dos clientes (MARTINS, 2005).

Na área da gestão de projetos, a sua condução ao sucesso depende do que todos os envolvidos esperam deste projeto. Por muitas vezes aparecem casos de insucesso em muitas das estatísticas levantadas, seja porque não chegam ao fim, seja por não alcançar o objetivo esperado com relação a funcionalidades, prazo ou custo (VIANA, 2008).

Com isso, devido ao setor de projetos de desenvolvimento de *software* estar sujeito a constantes mudanças durante a sua execução, surgiu como alternativa ao método de gerenciamento tradicional, uma metodologia de desenvolvimento chamada “Método Ágil”. O modelo tradicional era centrado em processos e menos flexível em relação a mudanças. Já o modelo ágil objetiva atender às necessidades de rápidas mudanças impostas no decorrer do projeto, eliminando muitas das atividades antes realizadas (VIANA, 2008).

Métodos ágeis surgem em meio a tentativas de oferecer uma resposta aos anseios de uma comunidade de negócios, que procura continuamente formas mais simples e rápidas de desenvolver *softwares* que atendam às suas necessidades. Esses métodos propõem para a criação de aplicações o envolvimento de grupos de desenvolvedores, enfocados na implementação de funções prioritárias, entregando-as rapidamente, coletando *feedbacks* e reagindo prontamente às mudanças do mercado e de tecnologias (RAMOS e PENTEADO, 2007).

As iniciativas de componentização de *software* e de desenvolvimento orientado a objetos na década de 80 despertaram a comunidade de *software* para as oportunidades e vantagens da reutilização de código. O sucesso destas atividades instigou iniciativas de reuso para diversas etapas do processo e gerenciamento de desenvolvimento de *software*, incluindo subprodutos de trabalho como documentos, especificações e modelos, o que aumentaria ainda mais a perspectiva de redução de custos e ganho de produtividade (DURSCKI et al., 2004).

A evolução destas ideias levou a formulação do modelo de Linha de Produto de S*oftware*, que apresenta um deslocamento no foco do paradigma tradicional de desenvolvimento. Dentro desse novo paradigma, as organizações que anteriormente abordavam o desenvolvimento de *software* projeto a projeto, devem concentrar seus esforços

na criação e manutenção de uma linha de produto de *software*, a qual será a base para a produção de uma coleção de produtos pertencentes a uma "família" (DURSCKI et al., 2004).

Alguns estudos sobre a junção de linha de produto de *software* e metodologias ágeis foram feitos, onde se observou ganhos na qualidade do produto que é o principal objetivo no gerenciamento de *software*. Os estudos relatam que as duas abordagens se completam atingindo os objetivos esperados (HANSSEN e FARQRI, 2008).

Existem abordagens para modelar linhas de produto de *software*, dentre elas as ontologias, onde um domínio de informação é representado através de conceitos e relacionamentos. Dentre as vantagens do uso de ontologias estão a possibilidade de reuso e a utilização de mecanismos de inferência. Sendo assim as ontologias podem ajudar a representar linhas de produto, utilizando o formalismo de representação de conhecimento acessível a objetos de *software*.

Este estudo tem por objetivo o desenvolvimento de uma ontologia que represente uma linha de produto para o gerenciamento de projetos de *software* com metodologias ágeis. O conceito da linha de produto é utilizado devido as suas características de flexibilidade e reutilização de artefatos de conhecimento. O resultado se constitui por um processo de desenvolvimento de *software* que permite sua variabilidade através da utilização dos diferentes domínios das metodologias ágeis e gerenciamento de projetos.

1.2 MOTIVAÇÃO

De maneira geral, a reutilização de *software* pode ser vista como o uso de componentes existentes, ou conhecimento referente a estes, para a construção de um novo produto de *software*. Seu valor se mostra presente na produtividade, confiabilidade e custo reduzido em projetos de construção de sistemas complexos (FRAKES e KANG, 2005).

No entanto, para que a reutilização alcance os níveis desejados de produtividade e qualidade na construção de *software*, é necessário que ela seja feita de maneira sistemática, considerando todas as fases do desenvolvimento, desde a análise até a implementação (WERNER e BRAGA, 2005). Tal necessidade dá lugar às técnicas de reutilização como a Linha de Produto de *Software*.

O paradigma de Linha de Produto de *Software* tem como proposta a construção sistemática de *software* baseada em uma família de produtos, guiando as organizações tanto na construção de novas aplicações, a partir de artefatos reutilizáveis, quanto na construção desses artefatos. Além disso, permite que as organizações explorem as características comuns e variáveis dos seus produtos de *software*, como forma de alcançar economia em sua produção (CLEMENTS e NORTHROP, 2002).

Ontologias permitem a organização do conhecimento e promovem a facilidade de reuso de especificações. Dentre os estudos pesquisados e elencados como trabalhos correlatos, identificou-se uma evolução positiva no desenvolvimento deste tema. Porém, não foi encontrado estudo anterior que aborde a utilização de ontologias na representação de uma linha de produto de *software* para gerência de projetos com metodologias ágeis. Sua utilização seria relevante, pois possibilita a consulta e inferência no modelo proposto como linha de produto, por agentes de *software*, preservando a variabilidade como característica principal, beneficiando o paradigma de linha de produto de forma que o reuso ocorra de maneira sistemática, e que novos produtos possam ser facilmente configurados.

1.3 OBJETIVOS

A dissertação tem por objetivo criar uma linha de produto de *software* através de ontologia. A ontologia proposta tem por objetivo:

- Definir o domínio da linha de produto de *software* para gerência de projeto baseada em conceitos principais do PMBOK e nas metodologias ágeis XP (*Extreme Programming*), *Scrum*, e FDD (*Feature Driven Development*);
- Construir uma ontologia que representa o domínio da linha de produto especificado e que permita o gerenciamento de processos em Metodologias Ágeis;
- Construir o modelo e implementar um protótipo de um sistema baseado em agentes de *software* para realizar consultas na ontologia;
- Validar e testar o modelo proposto em um cenário de aplicação real.

1.4 QUESTÃO DE PESQUISA

A QUESTÃO DE PESQUISA DA DISSERTAÇÃO DESTE ESTUDO É:

Como modelar uma ontologia para uma Linha de Produto de *Software* que flexibilize a Gerência de Processos em Metodologias Ágeis por meio de agentes?

1.5 METODOLOGIA

Para atingir os objetivos estabelecidos nesta dissertação a seguinte metodologia é definida para conduzir os trabalhos:

- Atividade 1. Elaboração do Problema. Formulação do problema a ser resolvido com base no conhecimento e análise do domínio em questão;
- Atividade 2. Pesquisa por Trabalhos Correlatos. Levantamento e análise de publicações relacionadas ao domínio do problema;
- Atividade 3. Fundamentação Teórico-prática. Estudos e práticas sobre potenciais tecnologias aplicáveis na solução do problema;
- Atividade 4. Definição da Solução. Elaboração da solução para o problema mapeado;
- Atividade 5. Análise e Projeto da Solução. Análise e projeto da solução com base nas tecnologias elencadas para resolver o problema;
- Atividade 6. Implementação da Solução. Implementação das estruturas necessárias para resolver o problema mapeado;
- Atividade 7. Verificação e Validação da Solução. Verificação e validação das implementações em cenários simulados de uso;
- Atividade 8. Registro dos Resultados Obtidos. Elaboração da dissertação a qual registra todos os estudos realizados e resultados obtidos com o desenvolvimento do trabalho.

1.6 ORGANIZAÇÃO DA DISSERTAÇÃO

O restante desta dissertação está organizado conforme abaixo:

- Capítulo 2: Neste capítulo é realizada a fundamentação da Linha de Produto de *Software*, Metodologias Ágeis, Gerenciamento de Projeto de *Software*, Ontologia e Agentes de *Software;*
- Capítulo 3: Apresenta os trabalhos relacionados com o tema da proposta;
- Capítulo 4: Este capítulo apresenta o modelo proposto como solução ao problema, através de uma visão geral da solução, sua arquitetura, funcionalidades e elementos constitutivos;
- Capítulo 5: Este capítulo apresenta o domínio e cenário de aplicação para a solução;
- Capítulo 6: Este capítulo apresenta os aspectos de implementação da solução proposta, abrangendo suas características técnicas, sua arquitetura detalhada, e a estrutura de seus componentes;
- Capítulo 7: Neste capítulo são apresentados os experimentos de verificação e validação da solução no cenário de aplicação, são relatados experimentos de forma detalhada, apresentando e discutindo os resultados da utilização da solução proposta no cenário de aplicação;
- Capítulo 8: Neste capítulo são feitas as considerações finais sobre o trabalho, onde se discutem suas principais contribuições, e realizam-se reflexões para trabalhos futuros.

2 ARCABOUÇO TEÓRICO

Este capítulo aborda o estudo bibliográfico feito para desenvolver a temática proposta. São apresentados os estudos sobre Linha de Produto de *Software*, Gerência de *Software*, Metodologias Ágeis, Ontologias e Agentes de *Software*.

2.1 LINHA DE PRODUTO DE *SOFTWARE*

Esta seção realiza a fundamentação de Linha de Produto de *Software* através da definição e conceitos relacionados.

2.1.1 Definição

Na engenharia de *software* vêm se formando uma consciência de que para obter produtos com alta qualidade e que sejam economicamente viáveis, torna-se necessário um conjunto sistemático de processos, técnicas e ferramentas. Entre as técnicas mais relevantes desse conjunto está a reutilização. Sendo assim, reutilizando partes bem especificadas, desenvolvidas e testadas, pode-se construir *software* em menor tempo e com maior confiabilidade. Muitas técnicas que favorecem a reutilização têm sido propostas ao longo dos últimos anos, como a linha de produto de *software* (LPS) (GIMENES e TRAVASSOS, 2002).

Em Gimenes e Travassos (2002) uma LPS é definida como um conjunto de produtos de *software* com características similares para permitir a definição de estrutura comum dos itens que compõe os produtos. Contudo uma linha de produto envolve um conjunto de aplicações similares dentro de um domínio que podem ser desenvolvidas a partir de uma arquitetura genérica comum, e um conjunto de componentes que povoam a arquitetura. O objetivo é reconhecer os aspectos comuns e as diferenças entre os artefatos durante todo o processo de desenvolvimento, assim, os pontos de decisão em que devem ser ajustados os componentes para geração de produtos específicos, devem ser reconhecidos através de pontos de variabilidade ou pontos em que as características dos produtos podem diversificar.

Segundo Clements e Northrop (2001), uma LPS é um conjunto intensivo de sistemas de *software* que compartilham e gerenciam um conjunto de características comuns, satisfazem as necessidades de um segmento particular de mercado ou missão e são desenvolvidos a partir de um conjunto comum de ativos principais e de uma forma pré-estabelecida. Alguns benefícios são visíveis no uso de LPS como: redução dos custos de desenvolvimento, aumento na qualidade, redução do tempo de mercado e redução do esforço de manutenção.

Conforme Souza (2007) as atividades essenciais de uma linha de produto são:

- Desenvolvimento dos artefatos básicos (*core assets*), infra-estrutura essencial para o reuso, base para o desenvolvimento do produto em uma linha de produto. Exemplos de *core assets* são: arquitetura, componentes, especificação de requisitos, modelo do domínio, plano de testes, entre outros. Essa atividade é também conhecida como Engenharia do Domínio.
- Desenvolvimento dos produtos a partir dos *core assets* do domínio. Essa atividade é também chamada de Engenharia da Aplicação.
- Gerenciamento em nível organizacional e técnico da linha de produto.

Essas três atividades são altamente iterativas e relacionadas, como pode ser visto na Figura 1.

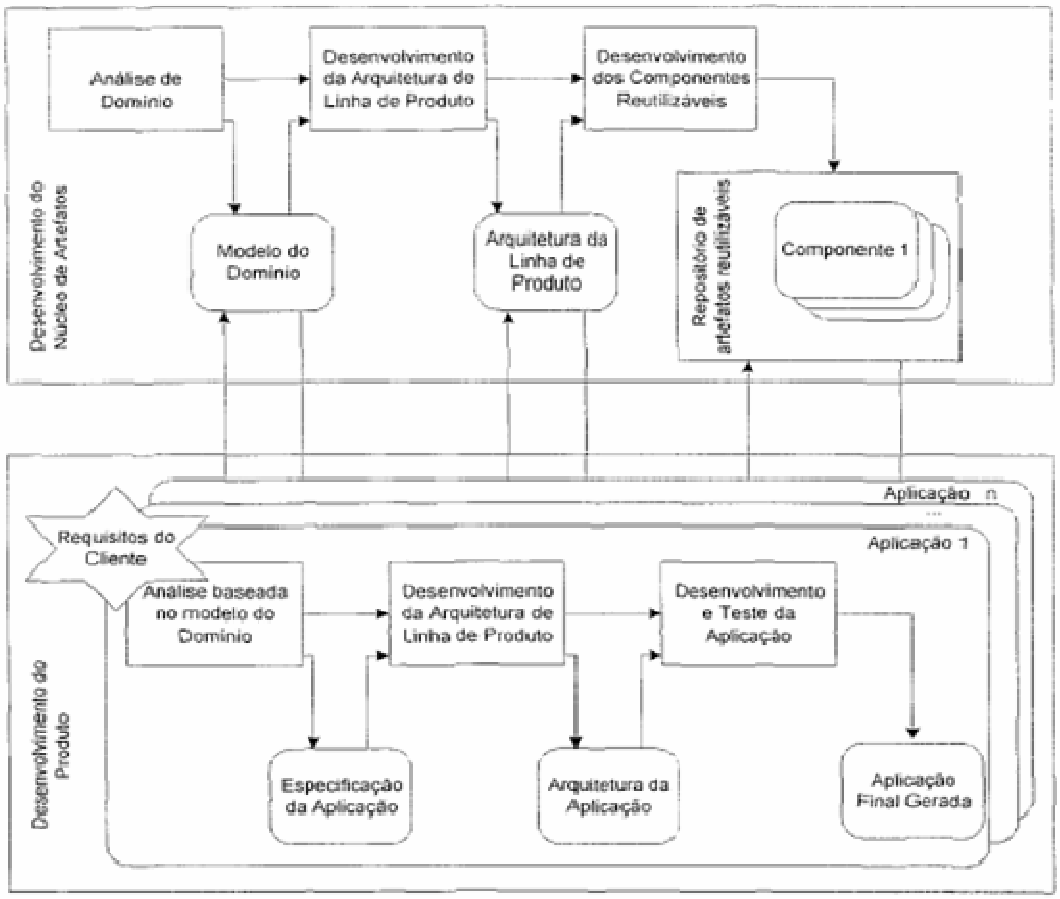
Figura 1 - Atividades de uma LPS.

Fonte: CLEMENTS e NORTHROP, 2001.

2.1.2 O processo de Desenvolvimento de uma LPS

O processo de desenvolvimento de uma LPS, de acordo com Gimenes e Travassos (2002), pode ser visto como dois modelos de ciclo de vida, Desenvolvimento do Núcleo de Artefatos (Engenharia de Domínio) e Desenvolvimento do Produto (Engenharia de Aplicação), conforme destacados na Figura 2.

Figura 2 - Processo de Desenvolvimento de uma LPS.



Fonte**:** GIMENES e TRAVASSOS, 2002.

Na Figura 2, Gimenes e Travassos (2002) definem os retângulos como sendo as etapas de desenvolvimento enquanto os retângulos arredondados representam os artefatos produzidos. A estrela apresenta os requisitos da aplicação a ser desenvolvida (produto). O modelo de desenvolvimento do núcleo de artefatos, também chamado de Engenharia de Domínio, é composto de três etapas: análise de domínio, desenvolvimento da arquitetura e desenvolvimento de componentes reutilizáveis. Estas produzem um modelo de domínio, uma arquitetura e um conjunto de componentes reutilizáveis e geradores de *software* para a linha de produto. O desenvolvimento do produto é composto das etapas: análise baseada no modelo de domínio, desenvolvimento da arquitetura de linha de produto e desenvolvimento e teste da aplicação. Estas produzem a especificação da aplicação, arquitetura da aplicação e a aplicação final gerada.

O desenvolvimento do produto, também é conhecido como engenharia de aplicação, em geral, inclui os seguintes artefatos (GIMENES e TRAVASSOS, 2002):

- O modelo do domínio, base para identificar os requisitos do cliente;
- Um *framework* de arquitetura de linha de produto, base para especificar uma arquitetura para um membro da família;
- Um conjunto de componentes reutilizáveis a partir do qual um subconjunto de componentes será integrado à arquitetura para gerar um produto.

2.1.2.1 Desenvolvimento do Núcleo de Artefatos

A idéia básica do desenvolvimento do núcleo de artefatos é disponibilizar um conjunto de artefatos que permita a construção de produtos membros de uma família. Como resultado desta atividade, obtém o domínio da linha de produto (definição do contexto), o núcleo de artefatos e o plano de produção.

2.1.2.2 Domínio da Linha de Produto

O domínio da linha de produto é a descrição dos produtos que constituirão a linha de produto ou quais produtos a linha será capaz de incluir. Conforme Czarnecki e Eisenecker (2000):

> "Um modelo de domínio é uma representação explícita das propriedades comuns e variáveis de um sistema, a semântica das propriedades e dos conceitos de domínio e as dependências entre as propriedades variáveis".

No desenvolvimento de *software* orientado à reutilização é importante que todos os elementos da equipe de desenvolvimento partilhem dos mesmos conceitos e semântica do domínio a resolver pelo sistema, e que existam restrições que limitem o desenvolvimento de elementos de *software* a reutilizar, sob o risco de serem construídos elementos de *software* reutilizáveis, mas sem utilidade futura. É nesse contexto que surge a necessidade de limitar o domínio de abrangência do *software*, através da definição de modelos de domínio, incluindo os conceitos, propriedades e âmbito do domínio (GOUVEIA, 2007).

Estes modelos suportam a análise e tomada de decisão em relação ao desenvolvimento de novas aplicações que se enquadrem dentro do âmbito do domínio. A definição dos requisitos da LPS e o modelo de domínio constituem o âmbito da LPS. Um requisito da LPS é "*uma unidade lógica de comportamento que é especificada por um conjunto de funcionalidades e requisitos de qualidade*" (BOSH, 2000). Um requisito satisfeito pela LPS é um conjunto de características funcionais ou não funcionais que são suportadas por um ou mais elementos de *software* da arquitetura da LPS (GOUVEIA, 2007).

Na definição do domínio, Kang et al.(1990) utiliza um modelo de decomposição de *features* em árvore, onde as *features* genéricas são colocadas no topo e as detalhadas em baixo. A utilização de modelos hierárquicos de *features* pode ser enquadrada nos seguintes contextos:

- Modelação de vastos domínios;
- Modelação de variabilidades em LPS;
- Encapsulamento dos requisitos do sistema;
- Contextualização e orientação no desenvolvimento da LPS;
- Planejamento e suporte à evolução da LPS;
- Comunicação entre os vários intervenientes no desenvolvimento do sistema.

2.1.2.3 Núcleo de Artefatos

O núcleo de artefatos, representado na Figura 2 como repositório de artefatos reutilizáveis, é a base para a construção de produtos em uma linha de produto. É composto da arquitetura e do conjunto de componentes de *software* que são desenvolvidos para reutilização na linha de produto. Incluí também modelos de desempenho, resultados da avaliação da arquitetura, documentação de projeto, especificação de requisitos, modelos de domínio e componentes COTS *(Commercial off-the-shelf).*

2.1.3 Desenvolvimento do Produto

Durante a fase de Engenharia de Aplicação um membro da linha de produto é desenvolvido, fazendo uso da reutilização dos artefatos de domínio e explorando a variabilidade da linha de produto. A Engenharia de Aplicação é composta pelos subprocessos (POHL et al., 2005):

- Engenharia de Requisitos de Aplicação: Identifica os requisitos dos *stakeholders* para a aplicação, em que é necessário fazer um mapeamento entre os requisitos da aplicação e os requisitos do processo de engenharia de domínio;
- Desenho de Aplicação: Engloba as atividades para produzir a aplicação de arquitetura. Usa a arquitetura de referência para instanciar a arquitetura de aplicação. Seleciona e configura as partes necessárias da arquitetura de referência e incorpora adaptações específicas da aplicação;
- Realização de Aplicação: Os principais objetivos são a seleção e configuração de componentes de *software* reutilizáveis, assim como a realização de artefatos de aplicação específica;
- Testes de Aplicação: Compreende as atividades necessárias para validar e verificar uma aplicação, contra a sua especificação.

2.1.4 Variabilidade

Variações dos artefatos de produtos são necessários representar em uma LPS. Os produtos de uma LPS existem simultaneamente e podem se diferenciar em termos de comportamento (conjunto de ações), atributos de qualidade, plataforma, configuração física, fatores de escala, entre muitos outros (CLEMENTS e NORTHROP, 2001).

A variabilidade entre produtos que podem ser reveladas e distribuídas entre os artefatos da linha de produto, sejam eles na arquitetura, nos componentes, nas interfaces entre componentes ou as conexões entre componentes (GIMENES e TRAVASSOS, 2002). Em qualquer fase do ciclo de desenvolvimento as variações podem aparecer, a começar na fase de análise de requisitos. A expressão de uma variação pode ser obtida pela introdução de parâmetros instanciáveis em tempo de construção associada a componentes, subsistemas ou coleção de subsistemas a partir dos quais um produto pode ser configurado atribuindo-se um conjunto de valores a esses parâmetros (CLEMENTS e NORTHROP, 2001). Um tipo de variação simples de ser representado é a escolha de componentes diferentes para uma mesma arquitetura. Assim, produtos podem ter maior ou menor capacidade, ou características

diferentes dependendo do tipo de componente escolhido para a arquitetura (GIMENES e TRAVASSOS, 2002).

Um produto em uma LPS é definido a partir de uma seleção de *features*, que são atributos que caracterizam as funcionalidades do produto. As *features* dos diferentes pontos de variação um produto são o conjunto de características onde se diferem produtos uns dos outros na linha. Uma *feature* pode ser um requisito específico, uma seleção entre os requisitos opcionais e alternativos; ou pode estar relacionada a certas características do produto como funcionalidade, usabilidade e desempenho, ou pode estar relacionado às características de implementação como o tamanho, a plataforma de execução ou compatibilidade com certos padrões (GIMENES e TRAVASSOS, 2002).

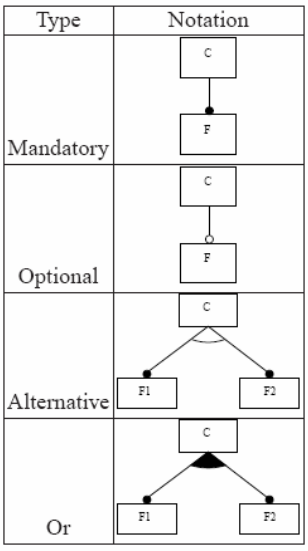
O modelo de *features* foi introduzido como parte do método *Feature-Oriented Domain Analysis* (FODA) (KANG et al., 1990), e representa uma hierarquia de propriedades de conceitos do domínio. É uma técnica bem fundamentada para facilitar a reutilização de artefatos de *software*. O modelo de *features* é uma representação hierárquica, que visa captar os relacionamentos estruturais entre as *features* de um domínio de aplicação. O modelo também representa as *features* comuns e variáveis de instâncias de conceitos e dependências entre as *features* variáveis. Um modelo de *features* consiste em um diagrama composto de *features* e alguma informação adicional, tais como descrições semânticas de cada *feature*, pontos variáveis, motivos para cada *feature*, prioridades e regras de dependência.

No contexto das LPS, um modelo de *features* representa a própria linha de produto.

Uma *feature* pode ser de um dos seguintes tipos, conforme a Figura 3:

- Obrigatória: A *feature* tem de estar presente em todos os membros da linha de produtos.
- Opcional: A *feature* pode ou não estar presente em um membro da linha de produtos.
- Alternativa: É uma *feature* que é composta de um conjunto de *features* das quais se escolhe uma ou mais, devendo-se indicar se é necessário escolher apenas uma ou se pode escolher mais que uma. Nestas *features* é necessário fazer a distinção de alternativas *OR*, que permite mais do que uma *feature*, e *XOR*, que mostra a exclusão mútua.

Figura 3 - Tipos de *Features*.



Fonte: KANG et. al., 1990.

2.2 METODOLOGIAS ÁGEIS

Esta seção realiza a fundamentação de Metodologias Ágeis através da definição e tipos de metodologias existentes.

2.2.1 Definição

A prática do desenvolvimento de *software* é uma atividade difícil em sua maior parte, normalmente caracterizada pela expressão "codificar e consertar". O *software* é escrito sem um plano definido e o projeto do sistema é repleto de várias decisões de curto prazo. Isso funciona muito bem se o sistema é pequeno, mas à medida que o sistema cresce, torna-se cada vez mais difícil adicionar novos recursos a ele. Defeitos subsequentes se tornam cada vez mais dominantes e cada vez mais difíceis de serem eliminados. Um sinal típico de um sistema desse tipo é uma longa fase de testes depois que o sistema está pronto. Esta longa fase de testes entra em confronto direto com o cronograma, pois testes e depuração são atividades cujos tempos são impossíveis de serem estimados (FOWLER, 2003).

Alternativas como metodologias impõem um processo disciplinado no desenvolvimento de *software*, com o objetivo de torná-lo mais previsível e mais eficiente. Elas fazem isso desenvolvendo um processo detalhado com uma forte ênfase em planejamento (FOWLER, 2003).

Metodologias estão disponíveis há muito tempo. Elas não têm sido percebidas como sendo particularmente bem-sucedidas. Elas têm sido notadas menos ainda por serem populares. A crítica mais freqüente é que estas metodologias são burocráticas (FOWLER, 2003).

Como uma reação a tais metodologias, um novo grupo delas surgiu nos últimos anos. Durante algum tempo elas foram conhecidas como metodologias leves, mas agora o termo mais aceito é metodologia ágil. Para muitas pessoas o apelo das metodologias ágeis é a reação delas à burocracia das metodologias monumentais. Estas novas metodologias tentam criar um equilíbrio entre nenhum processo e muitos processos, provendo apenas o suficiente de processo para obter um retorno razoável (FOWLER, 2003).

O resultado disso tudo é que os métodos ágeis têm algumas mudanças de ênfase significativas em relação aos métodos de engenharia. A diferença mais evidente é que metodologias ágeis são menos centradas em documentação, normalmente enfatizando uma quantidade menor de documentos para uma dada tarefa. De várias maneiras, elas são mais voltadas ao código fonte do programa: seguindo um caminho que diz que a parte-chave da documentação é o próprio código-fonte (FOWLER, 2003).

Metodologias ágeis são adaptativas ao invés de predeterminantes. As outras metodologias de engenharia tendem a tentar planejar uma grande parte do processo de desenvolvimento detalhadamente por um longo período de tempo. Isso funciona bem até as coisas mudarem. Então a natureza de tais métodos é a de resistir à mudança. Para os métodos ágeis, entretanto, mudanças são bem-vindas. Eles tentam ser processos que se adaptam e se fortalecem com as mudanças, até mesmo ao ponto de se auto modificarem (FOWLER, 2003).

Métodos ágeis são orientados a pessoas ao invés de serem orientados a processos. O objetivo dos métodos de engenharia é de definir um processo que irá funcionar bem,

independentemente de quem os estiverem utilizando. Métodos ágeis afirmam que nenhum processo jamais será equivalente à habilidade da equipe de desenvolvimento. Portanto, o papel do processo é dar suporte à equipe de desenvolvimento e seu trabalho (FOWLER, 2003).

A agilidade possui algumas características como a flexibilidade, equilíbrio dinâmico, adapta-se às circunstâncias específicas, atende às mudanças e auto-aperfeiçoamento. Gestão de projeto ágil gerência projetos que enfrentam constantes mudanças e incertezas durante o projeto. Agilidade é uma atitude ao invés de processo, onde os gerentes de projeto devem prestar atenção ajustando sua equipe na adaptação às mudanças, dedicando-se ao produto, coordenando com os clientes e focando na comunicação (ZHI-GEN et al., 2009).

Gerenciamento de projetos ágeis possui valores, são orientados a princípios e as práticas concretas, formando uma equipe de desenvolvimento colaborativo. Com a combinação de "pensamento sistêmico", "teoria das restrições e "produção enxuta", os princípios fundamentais de métodos ágeis são demonstrados, ou seja, incentivando a total descentralização da gestão de pessoas com qualidade, de tamanho pequeno, de gestão com ciclo de produção curto, data de entrega obrigatória, e a utilização de melhores práticas para redução de variáveis e incertezas (ZHI-GEN et al., 2009).

Métodos Ágeis promovem um processo empírico para o desenvolvimento de *software*. Essa abordagem exige um ciclo constante de inspeção, adaptação e melhoria. Encontrar maneiras eficazes de avaliar o processo e a equipe de desenvolvimento não é uma tarefa simples. Isso leva a uma proliferação de medidas baseadas na premissa de que se cada parte do processo for otimizada, os resultados do processo como um todo serão otimizados também. No entanto, essa premissa nem sempre é verdadeira. Ao tentar aperfeiçoar partes de um sistema por meio de diversas métricas, o verdadeiro objetivo se perde em meio a tantos substitutos e a equipe perde sua capacidade de tomar decisões de balanceamento (PRESSMAN, 2006).

Pressman (2006) afirma que os métodos ágeis foram desenvolvidos em 2001 por programadores experientes e consultores em desenvolvimento de processos leves. Alguns de seus princípios são:

- Indivíduo e interação, ou seja, tem mais importância do que os processos e ferramentas;
- O *software* funcionando é mais importante do que uma vasta documentação;
- A colaboração do cliente é mais importante do que a negociação de contratos;
- Adaptação a mudanças é mais importante do que seguir um plano;
- Maior prioridade na satisfação do cliente, entregando *software* com valor e em tempo hábil;
- Preparar-se quanto às mudanças de requisitos, mesmo que elas apareçam na fase mais avançada do desenvolvimento;
- Entregar versões funcionais com frequência e de preferência no menor espaço de tempo;
- As equipes devem trabalhar juntas durante todo o projeto;

- Simplicidade.

Métodos Ágeis oferecem ao desenvolvedor total flexibilidade e aproximam a equipe de tecnologia da informação do usuário final do *software*, seja ele um cliente interno ou externo. Com esse tipo de metodologia, a homologação dos projetos é feita em etapas, o que resulta em tempos de entrega mais curtos, geralmente de três a seis semanas, e a capacidade de promover alterações rapidamente (CAETANO, 2009).

2.2.2 Evolução das Metodologias Ágeis

Metodologias ágeis têm sido apontadas como uma alternativa às abordagens tradicionais para o desenvolvimento de *software*. As metodologias tradicionais, conhecidas também como pesadas ou orientadas a planejamentos, devem ser aplicadas apenas em situações em que os requisitos do sistema são estáveis e requisitos futuros são previsíveis. Entretanto, em projetos em que há muitas mudanças, em que os requisitos são passíveis de alterações, onde refazer partes do código não é uma atividade que apresenta alto custo, as equipes são pequenas, as datas de entrega do *software* são curtas e o desenvolvimento rápido é fundamental, não pode haver requisitos estáticos, necessitando então de metodologias ágeis. Além disso, o ambiente das organizações é dinâmico, não permitindo então que os requisitos sejam estáticos (SOARES, 2011).

Processos orientados a documentação para o desenvolvimento de *software* são, de certa forma, fatores limitadores aos desenvolvedores e muitas organizações não possuem recursos ou inclinação para processos pesados de produção de *software*. Por esta razão, as organizações pequenas acabam por não usar nenhum processo. Isto pode levar a efeitos desastrosos na qualidade do produto final, além de dificultar a entrega do *software* nos prazos e custos predefinidos (SOARES, 2011).

As metodologias ágeis surgiram com a proposta de aumentar o enfoque nas pessoas e não nos processos de desenvolvimento. Além disso, existe a preocupação de gastar menos tempo com documentação e mais com resolução de problemas de forma iterativa (SOARES, 2011).

2.2.3 Tipos de Metodologias Ágeis

Abaixo são abordadas algumas metodologias ágeis, as quais serão utilizadas para dissertação.

2.2.3.1 *Extreme Programming*

O *Extreme Programming* (XP) é um modelo de desenvolvimento de *software*, criado em 1996, por *Kent Bech*, no Departamento de Computação da montadora de carros *Daimler Crysler*, que possui muitas diferenças em relação a outros modelos, podendo ser aplicado a projetos de alto risco e com requisitos dinâmicos. O XP é um conjunto bem definido de regras, que vem ganhando um grande número de adeptos e por oferecer condições para que os desenvolvedores respondam com eficiência a mudanças no projeto, mesmo nos estágios finais do ciclo de vida do processo, devido a quatro lemas adotados por seus seguidores, que correspondem a quatro dimensões a partir das quais os projetos podem ser melhorados. São eles (SOUZA, 2007):

- Comunicação;
- Simplicidade;
- *Feedback*;
- Coragem.

O sucesso e popularidade adquiridos por XP se devem principalmente aos relatos de bons resultados obtidos em projetos, a motivação dos profissionais envolvidos com XP e também devido a sua natureza simples e objetiva por se basear em práticas que já provaram sua eficiência no cenário do desenvolvimento de *software*. Essas práticas têm como objetivo entregar funcionalidades de forma rápida e eficiente ao cliente. Além disso, XP foi criado considerando que mudanças são inevitáveis e que devem ser incorporadas constantemente (GEBER, 2011).

O XP possui doze práticas que consistem no núcleo principal do processo e que foram criadas com base nos ideais pregados pelos valores e princípios apresentados anteriormente. Segundo um dos criadores de XP, estas práticas não são novidades, mas sim práticas que já vêm sendo utilizadas há muitos anos, com eficiência, em projetos de *software*. Muitas das práticas de XP não são unanimidades dentro da comunidade de desenvolvimento de *software*, como por exemplo, programação em pares. No entanto, o valor e benefícios de tais práticas devem ser avaliados em conjunto e não individualmente, pois elas foram criadas para serem usadas coletivamente, de forma a reduzir as fraquezas umas das outras. As doze práticas de XP são comentadas abaixo (GEBER, 2011).

- **O jogo de planejamento:** Os planejamentos de um *release* e das iterações são feitos com base nas histórias (casos de uso simplificados) e conta com a colaboração de toda a equipe de desenvolvimento, inclusive o cliente, dividida em dois papéis:
  - **Negócio:** Participam as pessoas que entendem sobre o negócio, e que possam estabelecer prioridades para as funcionalidades a serem entregues.
  - **Técnico:** Participam as pessoas que irão implementar as funcionalidades descritas.
- ***Release*s pequenos:** Isso possibilita ter *release*s frequentes o que resulta em maior *feedback* para clientes e programadores, facilitando o aprendizado e a correção dos defeitos do sistema.
- **Metáfora:** A intenção da metáfora é oferecer uma visão geral do sistema, em um formato simples, que possa ser compartilhada por clientes e programadores. A idéia da metáfora é que seja feita uma analogia entre o sistema que está sendo desenvolvido e um sistema, não necessariamente de *software*, que todos entendam, com o objetivo de obter um "vocabulário comum" para a posterior criação de nomes de classes, subsistemas, métodos, etc.
- **Projeto simples:** Pode-se explicar esta prática em duas partes: A primeira diz que devem ser projetadas as funcionalidades que já foram definidas e não as que poderão ser definidas futuramente. A segunda diz que deve ser feito o melhor projeto que possa entregar tais funcionalidades. Esta prática tem o intuito de enfatizar que o projeto simples deve se concentrar em soluções simples e bem estruturadas para os problemas de hoje e que não se deve perder tempo investindo em soluções genéricas que procurem atender a funcionalidades futuras, pois como

os requisitos mudam constantemente tais soluções genéricas podem não ser mais a realidade do futuro.

- **Testes constantes:** Os testes em XP são feitos antes da programação. Existem dois tipos de teste: teste de unidade e teste funcional. Os testes de unidade são feitos para verificar tudo que possa dar errado. Os testes unitários são automatizados, e toda vez que o programador escrever código, ele irá verificar se o sistema passa em todos os testes. Os testes funcionais são usados para verificação, junto ao cliente, do sistema como um todo.
- **Refatoramento:** São constantes melhorias no projeto do *software* para aumentar sua capacidade de se adaptar a mudanças.
- **Programação em pares:** Todo o código produzido em XP é escrito por um par de programadores, que possuem papéis distintos, sentados lado a lado e olhando para o computador.
- **Propriedade coletiva do código:** A programação em pares encoraja duas pessoas a trabalharem juntas procurando atingir o melhor resultado possível. A propriedade coletiva encoraja a equipe inteira a trabalhar mais unida em busca de qualidade no código fazendo melhorias e refatoramentos em qualquer parte do código a qualquer tempo.
- **Integração contínua:** O código das funcionalidades implementadas pode ser integrado várias vezes ao dia. Um modo simples de fazer isso é ter uma máquina dedicada para integração.
- **Semana de quarenta horas:** Essa não é uma regra que obriga as equipes em projetos XP a trabalharem somente 40 horas por semana.
- **Cliente no local:** O cliente tem um papel importante dentro de um projeto XP já que ele participa do planejamento do projeto escrevendo as histórias e priorizando-as.
- **Padrões de codificação:** O objetivo é que todos programem da mesma forma, facilitando o entendimento do código e as alterações.

2.2.3.2 *SCRUM*

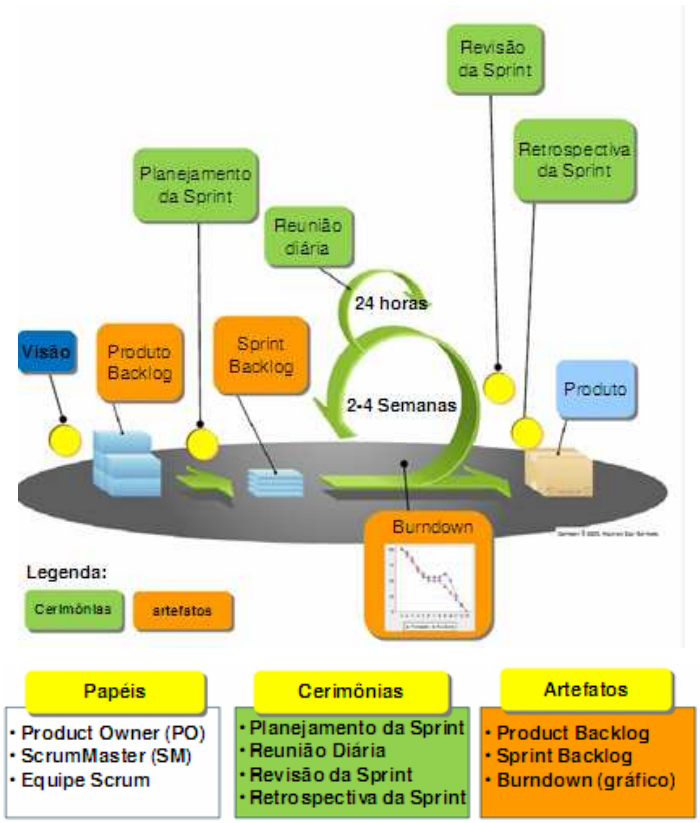
O *Scrum* é um método que aceita que o desenvolvimento de *software* é imprevisível e formaliza a abstração, sendo aplicável a ambientes voláteis (FOWLER, 2011). Ele se destaca dos demais métodos ágeis pela maior ênfase dada ao gerenciamento do projeto. Reúne atividades de monitoramento e *feedback*, em geral, reuniões rápidas e diárias com toda a equipe, visando à identificação e correção de quaisquer deficiências e/ou impedimentos no processo de desenvolvimento (JARDIM, 2010). O *Scrum* assume a premissa de que o desenvolvimento de *software* é muito complexo e imprevisível para ser planejado totalmente inicialmente. Ao invés disso, deve-se usar controle do processo empírico para garantir a visibilidade, inspeção e adaptação. O método baseia-se ainda, em princípios como: equipes pequenas de, no máximo, sete pessoas; requisitos que são pouco estáveis ou desconhecidos; e iterações curtas. Divide o desenvolvimento em intervalos de tempos de no máximo 30 dias, também chamados de *Sprints* (MARÇAL et al., 2007).

O *Scrum* é uma metodologia ágil para gerenciamento de projetos, muito utilizada atualmente no desenvolvimento e manutenção de *software*. Em seus primórdios o *Scrum* foi

concebido como um estilo de gerenciamento de projetos em empresas de fabricação de automóveis e produtos de consumo.

O *Scrum*, conforme Figura 4, permite a criação de equipes auto-organizadas, encorajando a comunicação verbal entre todos os membros da equipe e entre todas as disciplinas que estão envolvidas no projeto.

Figura 4 - Ciclo *Scrum*.

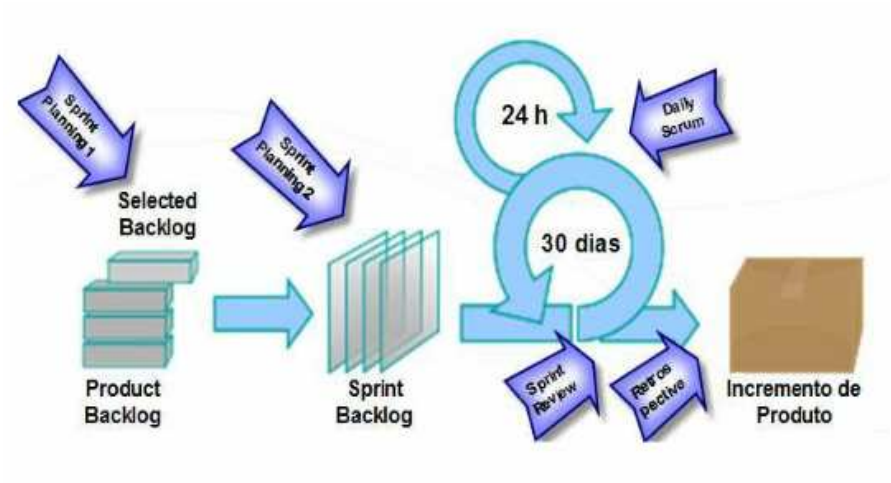


Fonte: QUEIROZ, 2010.

Para que a metodologia *Scrum* consiga atingir todos os seus objetivos, definem-se alguns processos básicos, divididos em papéis, cerimônias e artefatos, que constituem a essência do processo *Scrum*, conforme detalhado na Figura anterior e abaixo (VERNON, 2010):

- *Scrum*: *Scrum* não é uma sigla. A analogia óbvia é que o trabalho que está sendo realizado é movido para a frente da equipe e toda a equipe deve trabalhar em conjunto para fazer progressos reais. Trabalho em equipe é a essência do desenvolvimento ágil de *software*.
- *Sprint*: a Sprint, de acordo com a Figura 5, é um período definido de tempo ou de uma iteração de desenvolvimento.

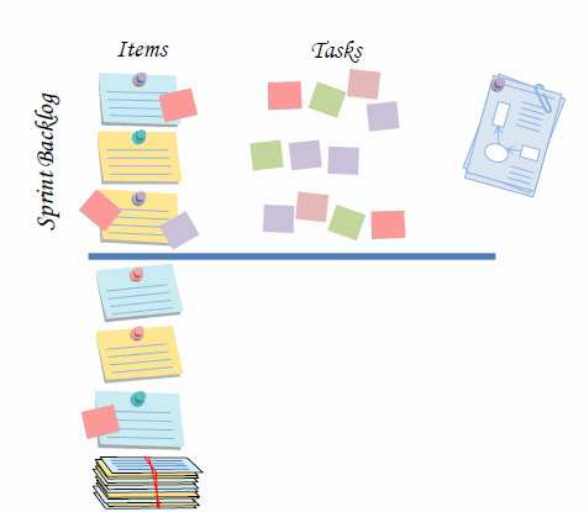
Figura 5 - *Sprint.*



Fonte: RAWSTHORNE, 2009.

- *Scrum Master*: Individuo, que é responsável por facilitar as reuniões *sprint*, acompanhando o progresso e ajudar as equipes superar quaisquer obstáculos.
- *Product Backlog*: O *product backlog* é uma lista priorizada de todo o trabalho a ser feito sobre o produto.
  - *Sprint Backlog*: O *sprint backlog*, conforme Figura 6, é uma lista estimada de todo o trabalho a ser realizado no sprint atual.

Figura 6 - Representação de *SprintBacklog*.



Fonte: RAWSTHORNE, 2009.

- *Product Owner*: O proprietário do produto é responsável pelo entendimento do produto, mantendo o *product backlog*, priorização e dar um *feedback* para a equipe sobre o trabalho realizado durante o *sprint*.

- *Sprint Planning Meeting*: A reunião do novo *sprint*, onde o proprietário do produto faz comentários do atraso e da equipe, estimando que eles podem realizar.
- *Burndown*: Gráfico que mostra o progresso da equipe que está fazendo sobre a realização do itens do *backlog*.
- *Daily Scrum*: *Scrum* diário é um encontro com toda a equipe onde cada membro da equipe aborda três questões: o que foi realizado ontem, o que vai ser trabalhado hoje e quais os obstáculos estão no caminho da realização do trabalho.
- Retrospectiva: A retrospectiva é uma reunião de toda a equipe no final do *sprint*, onde discutem o que correu bem durante o *sprint*, o que não correu bem e que pode ser melhorado para o próximo *sprint*.
- *Sprint Expo*: Um encontro de toda a equipe, os proprietários do produto e outras partes interessadas em que a equipe demonstra o que foi realizado no último *sprint*.

2.2.3.2.1 Fluxo de Desenvolvimento

No *Scrum*, um projeto se inicia com uma visão do produto que será desenvolvido. A visão contém a lista das características do produto estabelecidas pelo cliente, além de algumas premissas e restrições. Em seguida, o *Product Backlog* é criado contendo a lista de todos os requisitos conhecidos. O *Product Backlog* é então priorizado e dividido em *releases* (MARÇAL et al., 2007).

No *Scrum*, são realizadas iterações chamadas de *Sprints*. Cada *Sprint* inicia-se com uma reunião de planejamento (*Sprint Planning Meeting*), na qual o *Product Owner* e o Time decidem em conjunto o que deverá ser implementado (*Selected Product Backlog*). A reunião é dividida em duas partes. Na primeira parte (*Sprint Planning 1*), o *Product Owner* apresenta os requisitos de maior valor e prioriza aqueles que devem ser implementados (MARÇAL et al., 2007).

O Time então define colaborativamente o que poderá entrar no desenvolvimento da próxima *Sprint*, considerando sua capacidade de produção. Na segunda parte (*Sprint Planning 2*), o time planeja seu trabalho, definindo o *Sprint Backlog*, que são as tarefas necessárias para implementar as funcionalidades selecionadas no *Product Backlog*. Nas primeiras *Sprints*, é realizada a maioria dos trabalhos de arquitetura e de infra-estrutura. A lista de tarefas pode ser modificada ao longo da *Sprint* pelo Time e as tarefas podem variar entre 4 a 16 horas para a sua conclusão (MARÇAL et al., 2007).

Na execução das *Sprints*, diariamente o time faz uma reunião de 15 minutos para acompanhar o progresso do trabalho e agendar outras reuniões necessárias. Na reunião diária (*Daily Scrum Meeting*), cada membro do time responde a três perguntas básicas: O que eu fiz no projeto desde a última reunião? O que irei fazer até a próxima reunião? Quais são os impedimentos? Ao final da *Sprint*, é realizada a reunião de revisão (*Sprint Review Meeting*) para que o Time apresente o resultado alcançado na iteração ao *Product Owner*. Neste momento as funcionalidades são inspecionadas e adaptações do projeto podem ser realizadas (MARÇAL et al., 2007).

Em seguida o *ScrumMaster* conduz a reunião de retrospectiva (*Sprint Retrospective Meeting*), com o objetivo de melhorar o processo/time e/ou produto para a próxima *Sprint* (MARÇAL et al., 2007).

O monitoramento do progresso do projeto é realizado através de dois gráficos principais: *Product Burndown* e *Sprint Burndown*. Estes gráficos mostram ao longo do tempo a quantidade de trabalho que ainda resta ser feito, sendo um excelente mecanismo para visualizar a correlação entre a quantidade de trabalho que falta ser feita (em qualquer ponto) e o progresso do time do projeto em reduzir este trabalho (MARÇAL et al., 2007).

2.2.3.3 *Feature Driven Development*

Feature Driven Development (FDD) é uma metodologia de desenvolvimento de *software* que inclui alguns benefícios de processos rigorosos como modelagem, planejamento prévio, controle do projeto, assim como contém características de processos ágeis como foco na programação, interação constante com o cliente e entrega freqüente de versão do produto. Prevêem práticas apenas para o desenvolvimento de *software* em si não se preocupando com outros fatores como a escolha de tecnologias e ferramentas, a definição de procedimentos de aquisição, dentre outros (SILVA et al., 2009).

Embora não seja tão orientada à documentação, em FDD relatórios que controlam o estado e o progresso das atividades são previstos. Os artefatos principais são o plano de projeto, a lista de funcionalidades e o diagrama de sequência. O plano de projeto é o documento principal de saída a ser aprovado pelo cliente, nele está definido o escopo, a lista de funcionalidades, os riscos, as métricas para controle do projeto, os critérios de aceitação, dentre outras informações pertencentes ao domínio da aplicação. A lista de funcionalidades é usada para planejar, dirigir, rastrear e reportar o progresso do projeto e está organizada hierarquicamente com requisitos funcionais. O diagrama de sequência serve para mostrar os participantes de uma interação e a sequência das mensagens trocadas entre eles (SILVA et al., 2009).

São cinco os processos da metodologia ágil FDD:

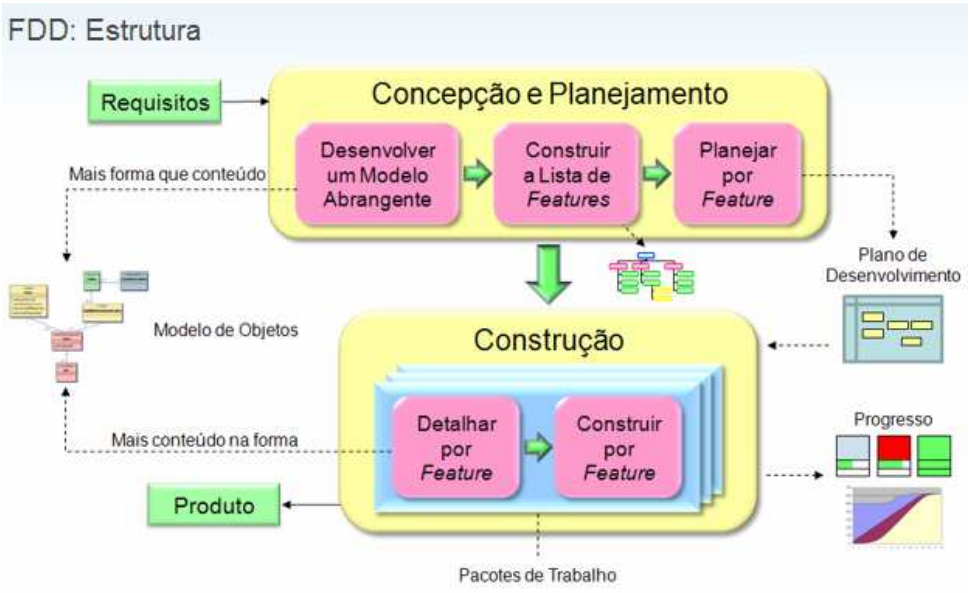
- Desenvolver um Modelo Abrangente;
- Construir uma Lista de Funcionalidades;
- Planejar através de Funcionalidades;
- Projetar através de Funcionalidades;
- Construir Através de Funcionalidades.

O processo Desenvolver um Modelo Abrangente é responsável pelo estudo detalhado sobre o domínio do negócio e pela definição do escopo do projeto. Segue-se o Construir uma Lista de Funcionalidades, onde todas as funcionalidades necessárias ao cumprimento das necessidades do cliente são levantadas. Os itens desta lista são ordenados por prioridade de desenvolvimento no processo Planejar através de Funcionalidades, considerando inclusive se a funcionalidade é funcional ou não. Ao final deste processo é gerada uma lista das classes e estas são associadas aos desenvolvedores responsáveis. Um plano de projeto é elaborado pelo arquiteto chefe e aprovado pelo cliente (SILVA et al., 2009).

Iniciam-se então várias iterações que compreende os dois processos finais. No processo Projetar através de Funcionalidades, para cada funcionalidade da lista é definida uma atividade a ser realizada. Neste processo o modelo da interface do usuário é esboçado e os diagramas de sequência e de classe são gerados. Já no processo Construir através de Funcionalidades o código é gerado, produzindo-se a cada iteração, para cada funcionalidade

definida, uma função que agregue valor ao cliente, este chamado de dono do produto (SILVA et al., 2009). A Figura 7 permite uma visão geral dos processos da metodologia FDD:

Figura 7 - Visão do processo FDD.



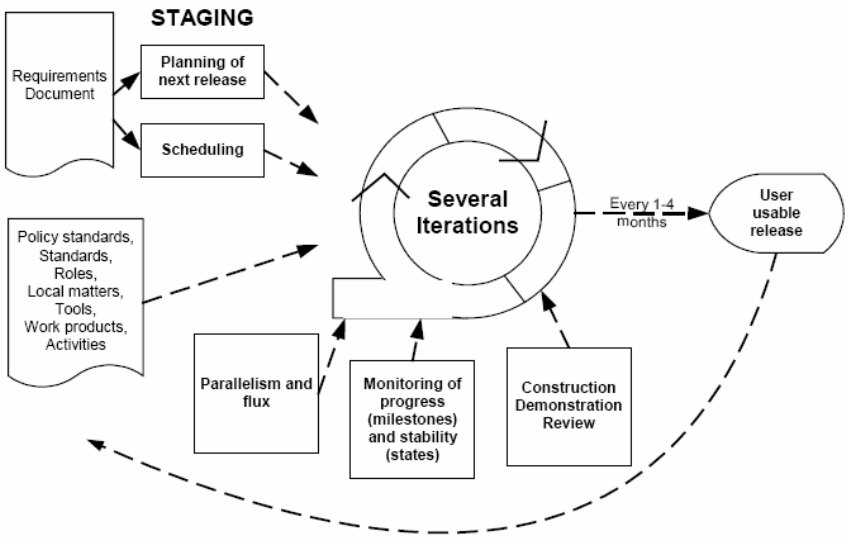
Fonte: HEPTAGON, 2011.

2.2.3.4 Família *Crystal*

As Metodologias do *Crystal* são uma família de metodologias diferentes, das quais as metodologias adequadas podem ser escolhidas para cada projeto. Os diferentes membros da família podem ser adaptados para atender diferentes circunstâncias. Os membros da família *Crystal* são indexados por cores diferentes para indicar o "peso": amarelo claro, laranja, vermelho, magenta, azul, violeta, e assim por diante. Quanto mais escura for a cor, mais "pesada" será a metodologia (SANTOS et al., 2011). Um projeto com oitenta pessoas precisa de metodologias mais pesadas do que um com apenas dez pessoas (ABRAHAMSSON et al., 2002).

Existem algumas características comuns à família *Crystal*, tais como o desenvolvimento incremental com ciclos de no máximo quatro meses, ênfase maior na comunicação e cooperação das pessoas, não limitação de quaisquer práticas de desenvolvimento, ferramentas ou produtos de trabalho e incorporação de objetivos para reduzir produtos de trabalho intermediários e desenvolvê-los como projetos evoluídos (SANTOS et al., 2011).

Figura 8 - Um incremento do *Crystal* Laranja.



Fonte: SANTOS et.al., 2011.

O ciclo de vida desta família de metodologia, conforme a Figura 8 é baseada nas seguintes práticas (SANTOS et al., 2011):

- *Staging*: Planejamento do próximo incremento do sistema. A equipe seleciona os requisitos que serão desenvolvidos na iteração e o prazo para sua entrega;
- Edição e revisão: Construção, demonstração e revisão dos objetivos do incremento;
- Monitoramento: O processo é monitorado com relação ao progresso e estabilidade da equipe. É medido em marcos e em estágios de estabilidade;
- Paralelismo e fluxo: No *Crystal* laranja, as diferentes equipes podem operar com máximo paralelismo. Isto é permitido através do monitoramento da estabilidade e da sincronização entre as equipes;
- Inspeções de usuários: são sugeridas duas a três inspeções feitas por usuários a cada incremento;
- *Workshops* refletivos: são reuniões que ocorrem antes e depois de cada iteração com objetivo de analisar o progresso do projeto;
- *Work Products* (Produtos de Trabalho): sequência de lançamento, modelos de objetos comuns, manual do usuário, casos de teste e migração de código;
- Padrões: padrões de notação, convenções de produto, formatação e qualidade usadas no projeto;
- Ferramentas: ferramentas mínimas utilizadas.

2.2.3.5 Método de Desenvolvimento de Sistemas Dinâmicos

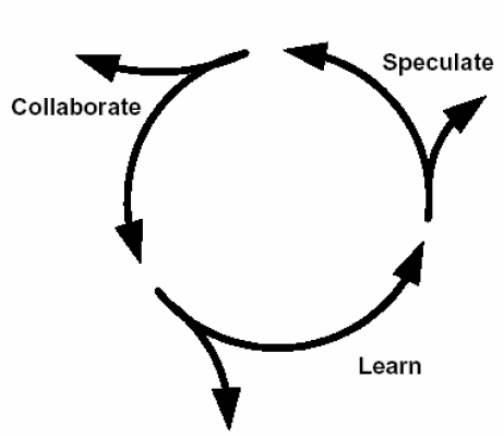
O *framework* Método de Desenvolvimento de Sistemas Dinâmicos (DSDM) consiste em três fases sequenciais: Pré-Projeto, Projeto e Pós-Projeto. A fase de Projeto do DSDM é a mais elaborada das três fases. Ela consiste em cinco níveis formados por uma abordagem passo a passo e iterativa no desenvolvimento do *software*. As três fases e os correspondentes níveis são: (COSTA, 2010).

- **Fase 1 -** O Pré-Projeto: Na fase do pré-projeto, o projeto candidato é identificado, tratando-se depois do seu plano de financiamento e sendo assegurado um compromisso de realização. Tratar destas questões numa fase inicial evita problemas futuros em fases mais avançadas do desenvolvimento do projeto.
- **Fase 2 -** O Ciclo de Vida do Projeto: A visão geral de um processo DSDM, presente na Figura 8, representa o ciclo de vida do projeto nesta segunda fase da metodologia. Ela mostra os cinco níveis que a equipe de desenvolvimento terá de percorrer para criar um *software*. Os dois primeiros níveis, o Estudo de Viabilidade e o Estudo de Negócio, são fases sequenciais que se complementam. Depois de estas fases estarem concluídas, o sistema é desenvolvido iterativamente e de forma incremental nos níveis de análise funcional, projeto e implementação.
- **Fase 3** - Pós-Projeto: A fase de pós-projeto assegura um sistema eficiente. Isto é desenvolvido através da manutenção e melhoramentos de acordo com os princípios da DSDM. Até mesmo a iniciação de novos projetos, para atualizar o sistema existente ou desenvolver um novo sistema, é possível.

2.2.3.6 Desenvolvimento de *Software* Adaptativo (ASD)

Os princípios do ASD são provenientes de pesquisas sobre o desenvolvimento iterativo. ASD fornece uma estrutura para o desenvolvimento de sistemas grandes e complexos com orientação suficiente para impedir os projetos de cair no caos. O método estimula o desenvolvimento iterativo e incremental com prototipagem constante. O processo ASD contém três fases que se sobrepõem: especulação, colaboração e aprendizagem (ver Figura 9) (COSTA, 2010).

Figura 9 - Processo ASD.



Fonte: HIGHSMITH, 1996.

Highsmith (1996) diz que os nomes das fases enfatizam a diferença do desenvolvimento de *software* tradicional. "Especulação" em vez de "planejamento", porque o planejamento sempre indica que não há incertezas. A importância do trabalho em equipe é destacada pela "colaboração". Em um ambiente imprevisível, pessoas têm a necessidade de se comunicar para serem capazes de lidar com as incertezas. "Aprendizagem" salienta que os requisitos mudam durante o desenvolvimento e existe a necessidade de um conhecimento sobre isso. Na ASD desvios não são falhas do sistema, mas levarão para uma solução correta.

Cada projeto começa com a definição da missão do projeto, uma breve declaração indicando o curso do projeto. Durante a fase de iniciação do projeto, o cronograma geral bem como os prazos e os objetivos para os ciclos de desenvolvimento são definidos. Um ciclo em ASD dura entre quatro e oito semanas (COSTA, 2010).

Como ASD é orientada a componentes em vez de orientada a tarefas, ela foca resultados e sua qualidade. A próxima fase é o planejamento do ciclo adaptativo que contém a fase de colaboração onde o ASD aborda a visão orientada a componentes. Ciclos do planejamento são repetidos quando ocorrem novas exigências e os componentes têm de ser refinados. Revisões com a presença do cliente demonstram a funcionalidade do *software* durante o ciclo. A fase de *release* é a última fase de um projeto ASD. Não há recomendações de como esta fase deve ser realizada, mas a ênfase está na importância de capturar as lições aprendidas (COSTA, 2010).

2.3 GERÊNCIA DE PROJETO DE *SOFTWARE*

Esta seção aborda a definição de gerenciamento de projeto de *software* bem como um método para realizar a gerência de projeto: o PMBOK.

2.3.1 Definição

Projeto é um empreendimento temporário com o objetivo de criar um produto ou serviço único (SOUZA, 2007). É um trabalho que visa à criação de um produto ou à execução de um serviço específico, temporário, não repetitivo e que envolve certo grau de incerteza na

sua realização (GEBER, 2011). Normalmente, é caracterizado por uma sequência de atividades (o processo do projeto), sendo executada por pessoas dentro de limitações de tempo, recursos e custos (GEBER, 2011).

Em termos gerenciais, o projeto é considerado como um tipo de empreendimento, esforço ou atividade, que tem começo e fim predeterminados, e que se compõe de uma sequência de etapas logicamente ordenadas, dirigidas por pessoas com a finalidade de cumprir metas estabelecidas dentro de certos parâmetros tais como custo, qualidade, tempo e recursos (FALBO, 2011).

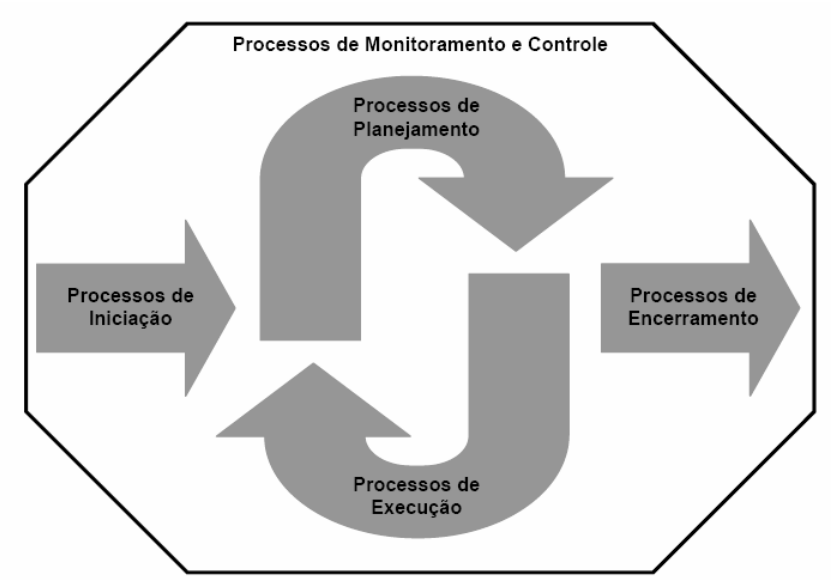
Desde o ponto de vista da engenharia, projeto é o conjunto de informações, especificações, desenhos, recursos entre outros, necessários para a criação e desenvolvimento de novos produtos. Definido o que é um projeto, segundo *Project Management Institute* (1996) gerenciamento de projetos é considerado como a aplicação de conhecimentos, habilidades, ferramentas e técnicas para as atividades do projeto com a finalidade de satisfazer as necessidades e expectativas dos interessados pelo projeto (FALBO, 2011).

Assim sendo, a Gerência de Projetos de *Software* envolve, dentre outros, o planejamento e o acompanhamento das pessoas envolvidas no projeto, do produto sendo desenvolvido e do processo seguido para evoluir o *software* de um conceito preliminar para uma implementação concreta e operacional (FALBO, 2011).

Processo de *software* é composto de diversos processos, dentre eles o processo de gerência de projetos. Tipicamente, um processo de gerência de projetos envolve quatro atividades principais:

- Fase conceitual: Os objetivos são analisados no contexto do ambiente de negócios, alternativas são definidas e avaliadas, e estimativas preliminar de custo, prazo e risco são feitos (JURISON, 1999).
- Planejamento: O planejamento do projeto deve tratar da definição do escopo do *software*, da definição do processo de *software* do projeto, da realização de estimativas, da elaboração de um cronograma e da identificação e tratamento dos riscos associados ao projeto (FALBO, 2011).
- Acompanhamento: conforme anteriormente apontado, no início do projeto há pouca informação disponível, o que pode comprometer a precisão do escopo identificado, das estimativas realizadas e, por conseguinte, do cronograma elaborado (FALBO, 2011).
- Encerramento: terminado o projeto, a gerência ainda tem um importante trabalho a fazer: fazer uma análise crítica do que deu certo e o que não funcionou, procurando registrar lições aprendidas de sucesso e oportunidades de melhoria (FALBO, 2011).

Figura 10 - Processos de Monitoramento de Controle.



Fonte: http://www.techoje.com.br/site/techoje/categoria/detalhe_artigo/675

O relacionamento entre os grupos de processos é apresentado na Figura 10, onde as setas representam os fluxos de entrada e saída entre os grupos de processos.

A gerência de projetos de *software* tem algumas particularidades em relação aos projetos genéricos. Para realizar um projeto de *software* com sucesso, deve-se ter bem claro o seu escopo, os riscos que poderão ocorrer, as tarefas a serem executadas, os pontos de controle a serem acompanhados, os recursos humanos, financeiros e materiais (*hardware*, *software*) necessários, o esforço empregado e o cronograma a ser seguido. De posse dessas informações é possível ter uma estimativa de custo confiável, uma correta divisão das tarefas e programação do projeto (SCHNEIDER, 2010).

A falta de métricas de projeto prejudica de forma geral o seu acompanhamento, uma vez que pode haver um problema, mas ele não está sendo percebido por aqueles que podem direcionar esforços para a sua solução. Assim, métricas têm um importante papel na rápida identificação e correção de problemas ao longo do desenvolvimento do projeto. Com a sua utilização, fica muito mais fácil justificar e defender as decisões tomadas, afinal, o gerente de projeto não decidiu apenas com base em seu sentimento e experiência, mas também fundamentado na avaliação de indicadores que refletem uma tendência de comportamento futuro (VASQUEZ et al., 2005).

2.3.2 PMBOK

O *Project Management Body of Knowledge* é um termo abrangente que descreve a soma do conhecimento dentro da profissão de Gerenciamento de Projetos (PM). *Project Management Body of Knowledge* (PMBOK) inclui o conhecimento de práticas tradicionais comprovadas amplamente aplicadas, bem como o conhecimento de empresas inovadoras e práticas avançadas que viram um uso mais limitado, e inclui materiais publicados e não publicados (HUIJBERS et al., 2004).

O PMBOK divide o projeto de processos em grupos de processos distintos: iniciação, planejamento, execução, controle e encerramento. Nota-se que estes grupos não implica que o projeto tem que passar por cada uma nessa ordem; eles só são fornecidos a fim de ser capazes de estruturar e categorizar os diferentes processos de projetos (HUIJBERS et al., 2004).

PMBOK também identifica es diversas áreas de projeto do conhecimento: gestão da integração, gerenciamento de escopo, tempo de gestão, gestão de custos, gestão da qualidade, gestão de recursos humanos, gestão da comunicação, gestão de riscos e gestão de contratos (HUIJBERS et al., 2004).

A primeira versão do PMBOK foi criada em 1986 e a versão atual é de 2008, 4ª edição. Ela foi gerada pelo PMI - *Project Management Institute*. O guia PMBOK fornece a "boa prática" que significa que existe um consenso geral de que a aplicação correta das habilidades, ferramentas e técnicas pode aumentar as chances de sucesso em uma ampla gama de projetos (SQUARE, 2008).

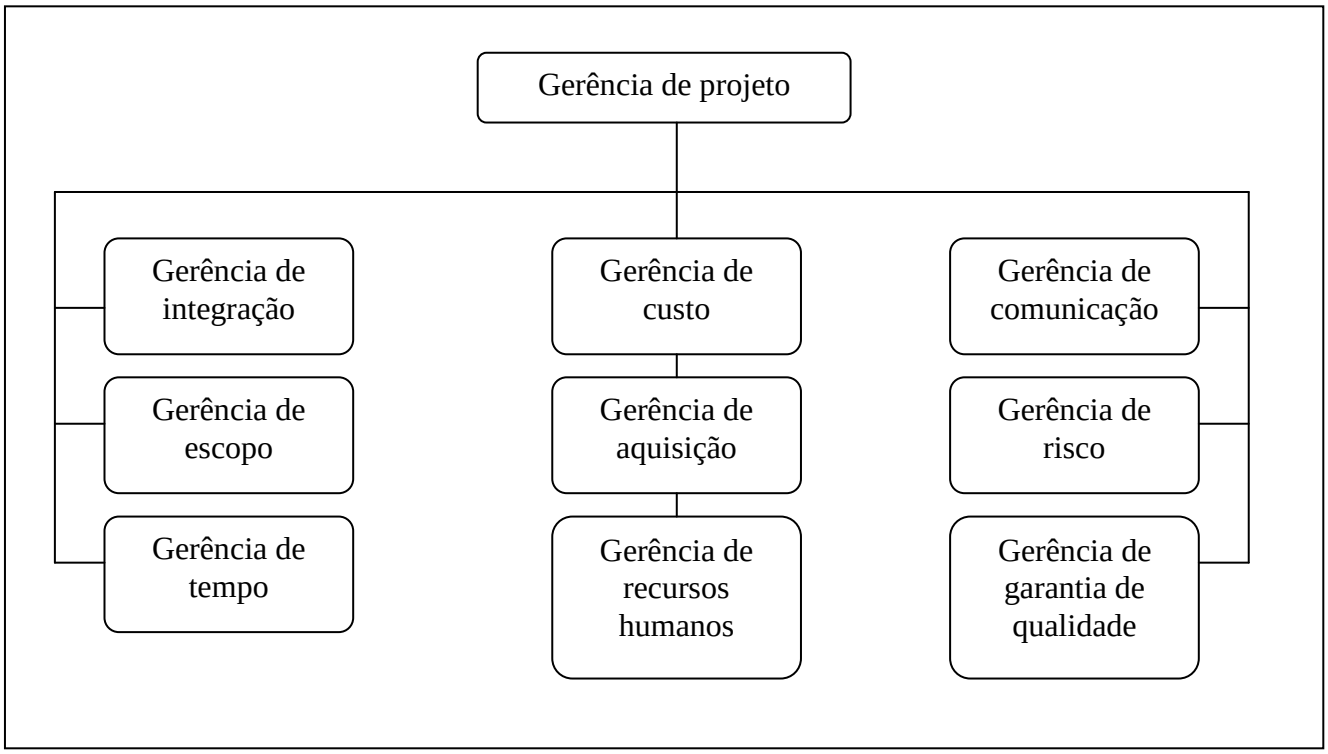
A gerência de projetos é a aplicação de conhecimento, habilidades, ferramentas e técnicas para projetar atividades, de maneira a satisfazer ou exceder as necessidades e expectativas dos *stakeholders*. Mas, satisfazer ou exceder as necessidades envolve um balanceamento entre as várias demandas concorrentes em relação ao (MACHADO et al., 2002):

- Escopo, tempo, custo e qualidade;
- *Stakeholders* com necessidades e expectativas diferenciadas; e
- Requisitos identificados (necessidades) e requisitos não identificados (expectativas).

Para cobrir todas as áreas que fazem parte da gerência de projetos o PMBOK se subdividiu em processos, conforme Figura 11. Cada processo se refere a um aspecto a ser considerado dentro da gerência de projetos e, todos os processos devem estar presentes quando da execução do projeto para que esse tenha sucesso. Esses processos são (MACHADO et al., 2002):

- Gerência de integração: O objetivo principal é realizar as negociações dos conflitos entre objetivos e alternativas do projeto, com a finalidade de atingir ou exceder as necessidades e expectativas de todas as partes interessadas. Envolve o desenvolvimento e a execução do plano do projeto, e o controle geral de mudanças.
- Gerência de Escopo: O objetivo principal é definir e controlar o que deve e o que não deve estar incluído no projeto. Consistem da iniciação, planejamento, definição, verificação e controle de mudanças do escopo.
- Gerência de Tempo do Projeto: O objetivo principal é garantir o término do projeto no tempo certo. Consiste da definição, ordenação e estimativa de duração das atividades, e de elaboração e controle de cronogramas.
- Gerência de Custo: O objetivo principal é garantir que o projeto seja executado dentro dos orçamentos aprovado. Consiste de planejamento de recursos, e estimativa, orçamento e controle de custos.
- Gerência de Qualidade do Projeto: O objetivo principal é garantir que o projeto satisfará as exigências para as quais foi contratado. Consiste de planejamento, garantia e controle de qualidade.

Figura 11 - Processos que compõem a Gerência de Projetos do PMBOK.



Fonte: MACHADO et al., 2002.

- Gerência de Recursos Humanos: O objetivo principal é garantir o melhor aproveitamento das pessoas envolvidas no projeto. Consiste de planejamento organizacional, alocação de pessoal e desenvolvimento de equipe.
- Gerência de Comunicação: O objetivo principal é garantir a geração adequada e apropriada, coleta, disseminação, armazenamento e disposição final das informações do projeto. Consiste do planejamento da comunicação, distribuição da informação, relatório de acompanhamento e encerramento administrativo.
- Gerência de Risco: O objetivo principal é maximizar os resultados de ocorrências positivas e minimizar as consequências de ocorrências negativas. Consistem de identificação, quantificação, tratamento e controle de tratamento de riscos.
- Gerência de Aquisição: O objetivo principal é obter bens e serviços externos à organização executora. Consiste do planejamento de aquisição, planejamento de solicitação, solicitação de propostas, seleção de fornecedores, e administração e encerramento de contratos.

2.3.3 Gerência de *Software* em Metodologias Ágeis

Como o PMBOK foi concebido para ser um guia genérico e completo para a gerência tradicional de projetos, ele desconsidera as peculiaridades da execução de produtos e serviços de *software*. Sua implantação prática na área de *software* exige um investimento importante

na concepção de um processo adequado, que venha realmente alavancar o negócio e facilitar a vida dos participantes, sem requerer trabalho adicional somente para atender aos seus requisitos (ARAUJO, 2009).

O ciclo de vida ágil é semelhante a qualquer outra abordagem de desenvolvimento. O que muda na abordagem ágil, porém, é que o ciclo fica reduzido em algumas semanas para cada componente funcional desenvolvido, tornando-se muito mais rápido e sendo executado diversas vezes. Considera-se que as áreas de PMBOK aplicam-se ao desenvolvimento ágil por meio da “elaboração progressiva”. Em contraposição ao gerenciamento tradicional de projetos, que parte do princípio de que o planejamento direciona os resultados e que entregar o planejado é sinônimo de sucesso em um projeto, a abordagem ágil considera que os resultados é que devem direcionar o planejamento, e que o sucesso advém de entregar o resultado desejado e não necessariamente o resultado planejado. A abordagem ágil esforça-se para reduzir o custo de mudança ao longo do processo de desenvolvimento de *software* (ARAUJO, 2009).

2.4 ONTOLOGIAS

Esta seção apresenta os conceitos de ontologias, abordando a taxonomia e as linguagens para construção de ontologias.

2.4.1 Definição

Nas aplicações que demandam representação e gerenciamento do conhecimento, como nas áreas de inteligência artificial, ciência da computação e engenharia do conhecimento, as ontologias são largamente utilizadas (DICKINSON; 2009).

Para uso das ontologias é necessária a adoção de linguagens específicas para escrevê-las, o que permite a descrição semântica de classes, termos e propriedades de conteúdos de determinada área de conhecimento. Portanto, podem-se definir ontologias como informações formais, especificações explícitas de conceitualização divididas, entrelaçadas, representando conceitos e suas relações para determinado domínio de conhecimento (ELUAN et al., 2007).

Segundo Neches et al. (1991), “uma ontologia define os termos básicos e as relações que compreendem o vocabulário de uma área tópico, bem como as regras para combinar termos e relações para definir as extensões deste vocabulário”.

Para Gruber (1993), “uma ontologia é a especificação explícita de uma conceitualização”. Então Borst (1997 apud PÉREZ-GÓMEZ; FERNÁNDEZ-LÓPEZ; CORCHO, 2004), estendeu esta definição para “ontologias são definidas como uma especificação formal de uma conceitualização compartilhada”.

Conforme Studer et al. (1998):

> Uma ontologia consiste em uma especificação formal e explícita de uma conceitualização compartilhada. Conceitualização refere-se a um modelo abstrato de alguns fenômenos no mundo através da identificação dos conceitos relevantes destes fenômenos. Significado explícito que o tipo de conceitos utilizados, e as restrições na sua utilização são definidos explicitamente. O termo formal refere-se ao fato de que a ontologia deveria ser entendível para a máquina. O termo compartilhado reflete a

> noção de que uma ontologia captura conhecimento consensual, ou seja, que não privado por alguns indivíduos, mas aceitado por todo um grupo (tradução nossa).

Portanto uma ontologia fornece um vocabulário para o compartilhamento de conhecimento sobre o domínio em questão, incluindo definições legíveis dos conceitos e relacionamento para a máquina. Sua utilização é bem aplicada ao compartilhamento da representação de um domínio entre agentes de *software*, reutilização de conhecimento em projetos de *software*, na separação entre conhecimento do domínio e conhecimento operacional, e na análise de um domínio em específico (NOY e MCGUINNESS, 2001).

2.4.2 Taxonomia

A taxonomia é um sistema pelo qual as categorias são relacionadas entre si por meio da inclusão de classes, A maior inclusão de uma categoria dentro de uma taxonomia, quanto maior o nível de abstração, cada categoria dentro de uma taxonomia inteiramente incluída dentro de uma outra categoria (a menos que seja a categoria de mais alto nível), mas não é exaustiva dessa categoria mais abrangente. Assim, o termo nível de abstração dentro de uma taxonomia refere-se a um determinado nível de inclusão (ROSCH, 1978).

Recentemente, o uso de taxonomias tem sido adotado por permitir acesso através de uma navegação em que os termos se apresentam de forma lógica, ou seja, em classes, subclasses, sub subclasses, e assim por diante, em quantos níveis de especificidade sejam necessários, cada um deles agregando informação sobre os documentos existentes na base. Uma vantagem desta forma de acesso é a garantia, para o usuário, da melhor seleção do termo de busca, uma vez que as classes contêm tópicos mutuamente exclusivos (ROSCH, 1978).

A caracterização de taxonomias consiste em (ROSCH, 1978):

- Possuir uma estruturada lista de conceitos de um domínio;
- Incluir termos com relações hierárquicas;
- Organizar e recuperar informações através de navegação;
- Permitir agregação de dados e evidenciar um modelo conceitual do domínio;
- Atuar como um mapa conceitual de tópicos explorado em um Sistema de Recuperação de Informação.

A categorização é um princípio básico de taxonomia, onde é organizado o pensamento, o raciocínio. Serve para reunir classes e fornecer uma ordem para disposição dos tópicos (CAMPOS, 2008).

2.4.3 Linguagens para construção de Ontologias

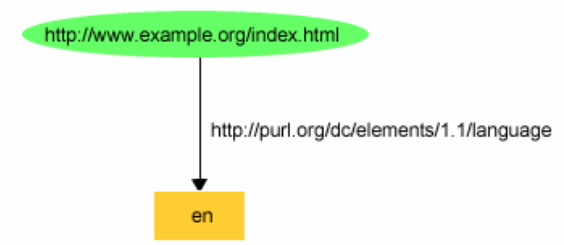
As ontologias precisam ser implementadas através de linguagens que transmitem o conhecimento necessário aos sistemas. As duas linguagens mais utilizadas são RDF/XML e a OWL.

2.4.3.1 *Resource Description Framework*

A fundamentação no processamento de metadados é compreendida pela linguagem *Resource Description Framework* (RDF). RDF foi desenvolvida pelo *World Wide Web Consortium* (W3C) para fornecer interoperabilidade no compartilhamento de informações compreensíveis a máquina. Por utilizar *tags* XML, esta linguagem também é conhecida como RDF/XML. Umas séries de facilidades viabilizam o processamento automático de recursos na Web, provendo atributos necessários para se embarcar semântica a dados e componentes de *software*. (MANOLA e MILLER, 2004).

O modelo desta linguagem é composto por três tipos de objetos que dão formação às expressões em RDF, os recursos, propriedades e declarações. Os recursos (*Resources*) são todas as entidades, físicas ou abstratas descritas por expressões RDF. As propriedades (*Properties*) são aspectos, características, atributos, ou relacionamentos utilizados para descrever recursos. Cada propriedade tem um significado específico, define sua limitação de valores permitidos, e os tipos de recursos que esta pode descrever. As declarações (*Statements*), ou seja, um recurso específico, organizado juntamente a uma propriedade nominativa onde esta propriedade carrega um valor para este recurso, caracteriza como uma declaração na linguagem RDF. Uma declaração tem por objetivo justamente dar significado semântico ao recurso, pois ela tem relação direta com uma *Uniform Resource Identifier* (URI), permitindo ao *software* conhecer o significado de tal recurso. A Figura 12 mostra um exemplo de uma declaração composta por um recurso, sua propriedade, e o valor desta propriedade (MANOLA e MILLER, 2004).

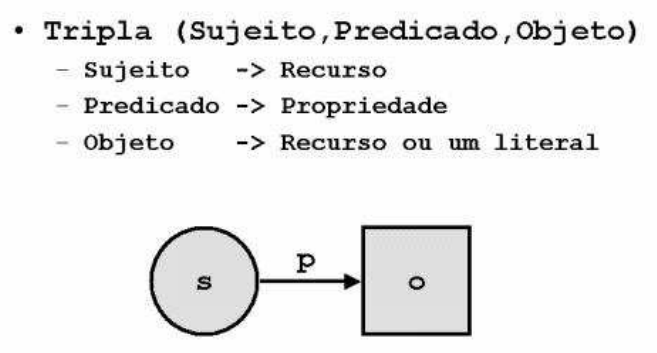
Figura 12 - Exemplo de um recurso.



Fonte: MANOLA e MILLER, 2004.

Uma afirmação em RDF é uma tripla (Sujeito, Predicado, Objeto), conforme Figura 13, entendida como "*Sujeito* possui *Predicado* com o valor *Objeto*". O sujeito e o predicado são recursos identificados pelo URI, já o objeto pode ser tanto um recurso como um literal numérico ou textual (THUSHAR e THILAGAM, 2009). O sujeito é o recurso descrito pelo predicado e o objeto. O predicado estabelece a relação entre o sujeito e o objeto. O objeto recebe a referência do predicado ou um valor literal.

Figura 13 - Representação de tripla



Fonte: THUSHAR e THILAGAM, 2009.

RDF possui um sistema de classes, semelhante aos utilizados em sistemas de modelagem e programação orientados a objetos. Existem coleções de classes, geralmente criadas para um determinado domínio ou propósito, denominadas *schemas*. Destacam-se em suas funcionalidades, a possibilidade de representar estruturas hierárquicas e estruturas de generalização/especialização. Contudo, recursos podem ser considerados instancias, ou podem ser consideradas subclasses de outras classes, permitindo ao desenvolvedor construir uma hierarquia de conceitos, e uma hierarquia de propriedades. A linguagem RDF, e o RDF *Schema* são largamente utilizados na representação de esquemas de bancos de dados através de ontologias (MANOLA e MILLER, 2004).

2.4.3.2 OWL

A linguagem OWL é destinada quando informações contidas em documentos precisam ser processadas por aplicações, ao contrário de situações em que o conteúdo só precisa ser apresentado aos seres humanos. A OWL pode ser utilizada para representar explicitamente o significado dos termos em vocabulários e as relações entre os termos. Esta representação dos termos e suas inter-relações são chamadas de ontologia. A OWL tem mais facilidades para expressar a semântica de que XML, RDF e RDFs, portanto, vai além dessas linguagens em sua habilidade de representar conteúdo interpretável por máquinas na Web (BECHHOFER,2004).

Utilizada pra descrever classes e relações entre classes, a OWL formaliza o domínio de conhecimento fazendo-se capaz a inferência sobre classes e indivíduos que compõem a ontologia, assim sendo utilizada na representação de esquemas de bancos de dados. Diversas vantagens para sua utilização em relação aos formatos clássicos como XML, pois tem capacidade de representar relacionamentos entre entidades como associação, hierarquia e especialização (BECHHOFER, 2004).

A linguagem OWL possui gradativamente um aumento no nível de expressividade em ordem crescente, onde subdivide-se em três tipos: OWL *Lite*, OWL DL e OWL *Full*. OWL *Lite* suporta restrições simples para aqueles usuários que necessitam principalmente de uma classificação hierárquica. Embora suporte restrições de cardinalidade, ela só permite valores de cardinalidade 0 ou 1. Também permite um caminho de migração mais rápido de tesauros e outras taxonomias. OWL *Lite* também tem uma menor complexidade formal que OWL DL.
A OWL DL suporta aqueles usuários que querem a máxima expressividade. OWL DL inclui todas as construções da linguagem OWL, porém elas somente podem ser usadas com algumas restrições. OWL DL é assim chamada devido a sua correspondência com as lógicas de descrição, um campo de pesquisa que estudou a lógica que forma a base formal da OWL.

OWL *Full* é direcionada àqueles usuários que querem a máxima expressividade e a liberdade sintática do RDF sem nenhuma garantia computacional. OWL *Full* permite que uma ontologia aumente o vocabulário pré-definido de RDF ou OWL. É improvável que algum *software* de inferência venha a ser capaz de suportar completamente cada recurso da OWL *Full* (NOY e MCGUINNESS, 2001).

Após criar a ontologia representando uma descrição formal dos modelos de domínios de conhecimento, é possível que um componente de *software* aprenda sobre as entidades e seus relacionamentos em um domínio de conhecimento específico. A linguagem também permite que componentes de *software* possam entender a informação sem precisar se organizar para um formato específico para troca ou representação destas informações. Ela fornece um balanço adequado entre o poder da representação do conhecimento e o custo necessário para a inferência de uma grande quantidade de informações. Na medida em que sistemas heterogêneos precisam realizar operações entre si, procurar a representação do seu esquema através da OWL evita que estes precisem chegar a acordos sobre o formato na troca de dados ou informações (BECHHOFER, 2004).

Os autores Noy e Mcguinness (2001), propõem alguns passos pra projetar e construir ontologias:

- Determinação do domínio e escopo: neste passo se define o domínio de conhecimento a ser representado pela ontologia, considerando o escopo da aplicação à qual utilizará a ontologia como base de conhecimento;
- Considerar a reutilização da base de conhecimento: o planejamento da reutilização da base de conhecimento, portanto, a verificação da possibilidade reutilização de ontologias que já estão prontas é considerada uma boa prática de engenharia de *software*;
- Levantamento de termos importantes: consiste no levantamento dos termos importantes do domínio de conhecimento em questão, candidatos a constituírem classes, ou seja, conceitos da ontologia;
- Definir as classes e a hierarquia: com base nos termos levantados, o processo inicia uma análise para eliminar possível redundância de conceitos, e por seguinte distribuindo as classes em uma estrutura hierárquica, conforme o domínio estudado;
- Definir as propriedades: a descrição dos conceitos existentes na ontologia ocorre nesta fase, através das propriedades atribuídas a cada classe do domínio ou indivíduos. Uma propriedade relaciona classes ou indivíduos, conectando uma classe ou indivíduo *Domain* (domínio) a classes ou indivíduos de um *Range* (escopo). Um exemplo de *Domain* e *Range* é: uma ontologia de *Pizza*, a propriedade *hasTopping* (temRecheio) liga indivíduos pertencentes à classe *Pizza* a indivíduos pertencentes à classe *PizzaTopping* (RecheioDePizza). Neste caso, o *domain* (domínio) da propriedade *hasTopping* é *Pizza* e o *range* (*escopo*) é *PizzaTopping* (RecheioDePizza).
- Definir as restrições das classes e propriedades: as restrições das propriedades definem o domínio de uma propriedade, sua cardinalidade, e o tipo de valor suportado pela propriedade;
- Criar as instâncias das classes: as instâncias compreendem os indivíduos que representam a estrutura de classes e sua hierarquia;

2.4.4 Propriedades da Ontologia

As propriedades de uma ontologia se dividem em: Inversa, Funcional, Funcional Inversa, Transitiva e Simétrica (HORRIDGE, M., 2004).

- A propriedade Inversa basicamente descreve uma relação do tipo, se A é relacionada pela propriedade “p” com B, então a inversa também é válida, ou seja, B é relacionado pela propriedade “p” com A.
- A propriedade Funcional descreve que deve existir no máximo outro indivíduo ou classe relacionado a ele através desta propriedade. Por exemplo, se Pedro tem como avó materna Maria Helena e a propriedade é funcional, e se Pedro tem como avó materna "Vó Maria", então Maria Helena e "Vó Maria" são, necessariamente, a mesma pessoa.
- Na propriedade Funcional Inversa quando sua propriedade inversa é funcional, ou seja, para certo indivíduo ou classe, deve existir no máximo outro indivíduo relacionado a ele através desta propriedade. Ela se define como a inversa da propriedade Funcional.
- A propriedade Transitiva se define como: se A relacionada pela propriedade “p” com B, e B relacionado pela propriedade “p” com C, então A também se relaciona com C pela mesma propriedade “p”.
- A propriedade Simétrica relaciona A com B, então o B também se relaciona com o A via propriedade simétrica.

2.5 DESENVOLVIMENTO DE SISTEMAS ORIENTADOS A AGENTES

Esta seção apresenta uma definição a agentes de *software,* abordando características e metodologia para construção de um sistema orientados a agentes.

2.5.1 Agentes de *Software*

Os agentes possuem características como autonomia, são proativos e comunicativos. Um agente de *software* possui capacidades de perceber e agir sobre um ambiente. Como uma entidade inteligente, um agente opera de forma flexível e racional em uma variedade de condições ambientais, dado seu equipamento perceptual e eficaz. Comportamentos de flexibilidade e racionalidade são alcançados por um agente com base em processos tais como, a resolução de problemas, planejamento, tomada de decisão e de aprendizagem. Como uma entidade de interação, um agente pode ser afetado em suas atividades por outros agentes, e talvez por seres humanos (WEISS, 1999). Um agente é autônomo, pois opera sem intervenção direta de seres humanos ou outros sistemas, além de ter controle sobre suas próprias ações e estado interno.

Outra característica é de ser considerado social, por cooperar com outros agentes ou humanos com objetivo de realizar tarefas. Também é reativo, por que percebe o seu ambiente e responde prontamente as mudanças que ocorrem neste, e acima de tudo é pró-ativo, devido

ao fato de poder influenciar seu ambiente a partir de um comportamento orientado a objetivos (BELLIFEMINE et al. 2007).

Um agente de *software* situado em um ambiente é capaz de executar ações com autonomia para atingir seus objetivos para os quais foi desenvolvido Wooldridge (2002). Para modelar os agentes é fundamental a arquitetura que pode ser definida em quatro grupos (BELLIFEMINE et al., 2007):

- Agentes baseados em lógica: O ambiente é simbolicamente representado e manipulado por técnicas de inferência sobre este;
- Agentes reativos: Decisão a partir do mapeamento da situação que resulta em ação é totalmente baseado em estímulos, com mecanismo de resposta baseado em sensores de dados;
- Agentes BDI: O agente com uma atitude mental de crença, desejo e intenção, e geralmente plano. As crenças representam o ambiente, os desejos correspondem aos objetivos que devem ser atingidos, e as intenções representam os desejos que o agente esta comprometido em atingir, são as tarefas a serem realizadas;
- Agentes Híbridos: arquitetura híbrida que permitem ao agente ser reativo e deliberativo.

Existem algumas ferramentas para desenvolver agentes e sistemas baseados em agentes. Essas ferramentas fornecem uma estrutura que facilita a construção como, por exemplo, classes próprias para trabalhar-se com os mesmos, fornecendo funcionalidades para desenvolver o agente desejado. Com isso uma entidade atuou com importantes especificações para programação orientada a agentes, chamada FIPA (*Foundation for Intelligent Physical Agents*). Esta entidade é usada pela maioria de sistemas que usam agentes trazendo a padronização da comunicação entre os mesmos, pré-requisitos para plataformas e *frameworks* como ambientes de execução de sistemas baseados em agentes, o gerenciamento de agentes e serviços fornecidos pela plataforma e ambiente de execução, e uma arquitetura abstrata capturando as características mais importantes de sistemas baseados em agentes.

2.5.2 Metodologia para Construção de Sistemas baseados em agentes

A metodologia para construção de sistemas orientados a agentes resulta na definição de modelos de interação e cooperação que capturam os relacionamentos sociais e as dependências entre agentes e papéis que estes desempenham no sistema, descrevem os elementos de processos utilizados na abordagem do sistema e focam nos artefatos de projeto e sua documentação O projeto de grande escala de sistemas baseados em agentes requer um processo de engenharia sofisticada, que integra aspectos de conhecimento, bem como engenharia de *software*. Nos últimos anos, vários abordagens surgiram as quais diferem basicamente na arquitetura alvo dos agentes ou os estágios considerados no processo de engenharia, por exemplo, projeto, análise de requisitos, etc. (HENDERSON-SELLERS e GIORGINI, 2005).

Existem algumas propostas para modelar sistemas baseado em agentes. Estas propostas, chamadas de metodologias para construção de sistemas baseados em agentes, possuem diferentes tipos. Os tipos mais conhecidos são *Gaia*, MAS-*CommonKADS*, *Tropos*, *Prometheus*, *MaSE*. Neste estudo a metodologia optada foi a MaSE, então o embasamento teórico é focado nesta metodologia (BERNY et.al., 2006).

A metodologia Tropos fornece suporte as atividades de análise e de projeto no desenvolvimento do sistema, desde a análise até a implementação do mesmo. Está dividida em cinco fases: fases inicial e final de requisitos, projetos arquitetural e detalhado, e implementação (BERNY et.al., 2006).

A metodologia Gaia possui uma linguagem própria para a modelagem de agentes. O processo de desenvolvimento contém duas fases: análise e design. Esta metodologia tem início na fase de análise, visando coletar e organizar a especificação que servirá de base para a fase de design (BERNY et.al., 2006).

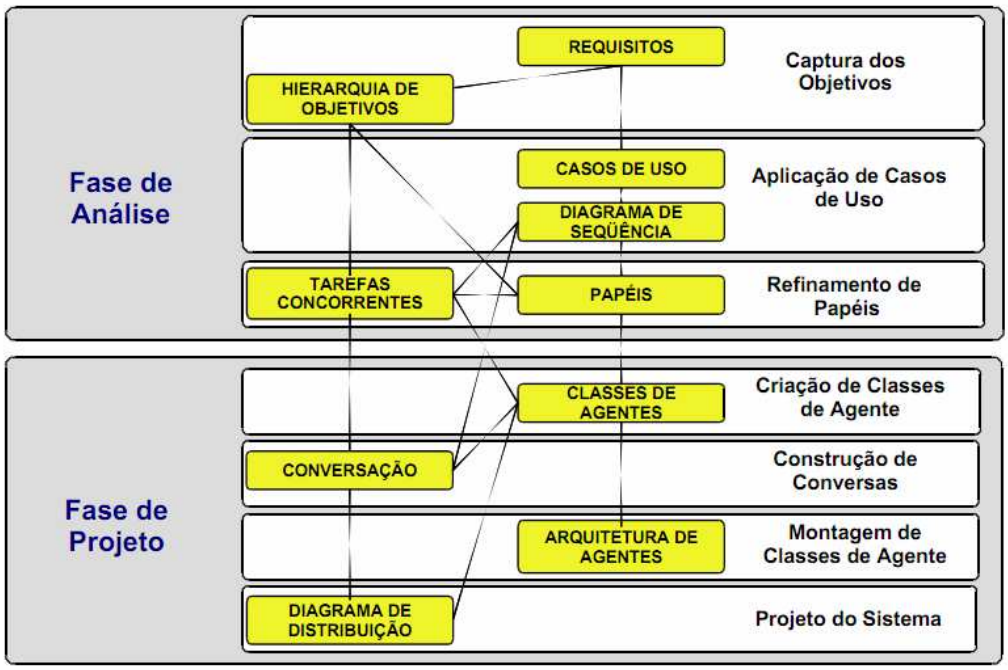
A metodologia Prometheus (PADGHAM, 2004) abrange desde a modelagem até a implementação de um sistema baseado em agentes baseado na plataforma para agentes BDI. Esta metodologia é composta por três fases, onde os componentes produzidos são utilizados tanto na geração de esqueleto do código, como também para realização de testes.

A metodologia MAS-CommonKADS tem o objetivo de modelar o conhecimento de resolução de problemas empregado por um agente para realizar uma tarefa. Este modelo divide o conhecimento da aplicação em três subníveis: nível do domínio (conhecimento declarativo sobre o domínio), nível de inferência (uma biblioteca de estruturas genéricas de inferência) e nível de tarefa (ordem das inferências) (WERNECK et.al., 2006).

No estudo será abordada a metodologia MaSE, onde os princípios e características com base na engenharia de *software* são divididos em duas fases, conforme Figura 14:

- Fase de Análise: Captura os objetivos do sistema, aplica os casos de uso e executa o refinamento dos papéis.
- Fase de Projeto: Cria as classes de agentes, construção da conversa entre os agentes, monta as classes de agente e projeta o sistema.

Figura 14 - Fases da MaSE



Fonte: DELOACH e WOOD, 2006.

MaSE é uma metodologia orientada a agentes, desenvolvida pela *Air Force Institute of Technology* (AFIT). Na metodologia MaSE, os agentes não são considerados como sendo necessariamente autônomos, pró-ativos, são processos de *software* simples que interagem uns com os outros para atingir o objetivo geral do sistema. MaSE assume a existência prévia da especificação de requisitos para o início do desenvolvimento da metodologia e segue de forma estruturada até a sua implementação, combinando vários modelos pré-existentes em uma única metodologia estruturada. O objetivo principal da MaSE é guiar o projetista através de todo o ciclo de vida do *software*, independentemente de qualquer arquitetura multiagentes, arquitetura de agente, linguagem de programação ou sistema de troca de mensagens (DELOACH e WOOD, 2006).

Os principais diagramas atualmente disponibilizados pela MaSE, e suportados pela *AgentToolIII* são:

- Modelo de Objetivos: diagrama destinado ao desenvolvimento da árvore de objetivos do sistema multiagentes que se pretende implementar.
- Modelo Organizacional: diagrama destinado a modelar as interações da organização.
- Modelo de Papéis: diagrama que representa os papéis a serem desempenhados pelo sistema multiagentes.
- Modelo de Agentes: diagrama que representa os agentes de *software* que irão popular a organização.
- Modelo de protocolos: diagrama que descreve a sequência de mensagens trocadas entre agentes, e entre agentes e atores externos.
- Modelo de plano de agentes: diagrama que representa o plano de agentes e o meio pelo qual cada agente atinge os objetivos sob sua responsabilidade na organização.

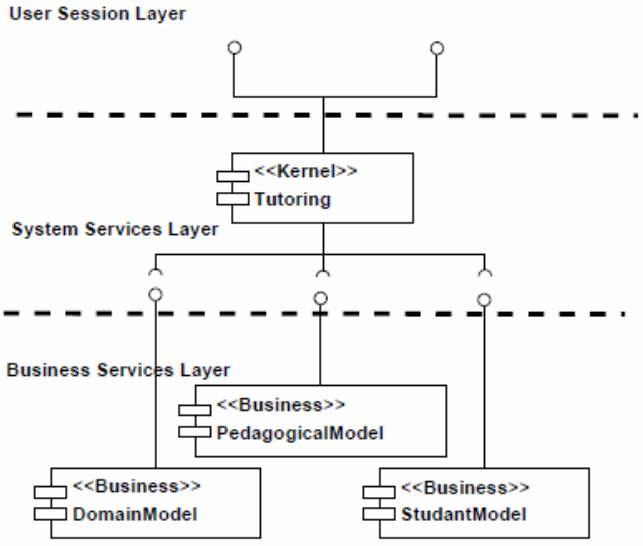
3 TRABALHOS RELACIONADOS

Este capítulo tem como objetivo apresentar trabalhos que possuam similaridade com o trabalho proposto. Foram encontrados cinco trabalhos, que abaixo foram detalhados e comparados com o trabalho proposto. A proposta de comparação dos trabalhos correlacionados foram os conceitos utilizados neste trabalho. Primeiramente foram pesquisados trabalhos em que se encaixavam com pelo menos um conceito utilizado no trabalho proposto, ou a combinação de dois ou mais conceitos. Os conceitos comparados foram: Linha de Produto de *Software*, Gerência de projeto de *Software*, Metodologias Ágeis, Representação de conceitos por Ontologias, LPS para gerência de projeto de *software* em metodologias ágeis, suporte a variabilidade de LPS por ontologias e criação de processo baseado em ontologia e LPS.

O primeiro trabalho pesquisado foi o trabalho de Silva (2011) que possui como domínio sistemas tutores inteligentes sob perspectiva da Web Semântica. Trata-se de sistemas tutores construídos através de uma linha de produto de *software* baseada na Web Semântica, onde aborda a problemática concepção de desenvolvimento em larga escala.

Foram definidos os principais artefatos que compõem a arquitetura da linha de produto de *software* conforme ilustrado na Figura 15.

Figura 15 - Artefatos da arquitetura da linha de produto.



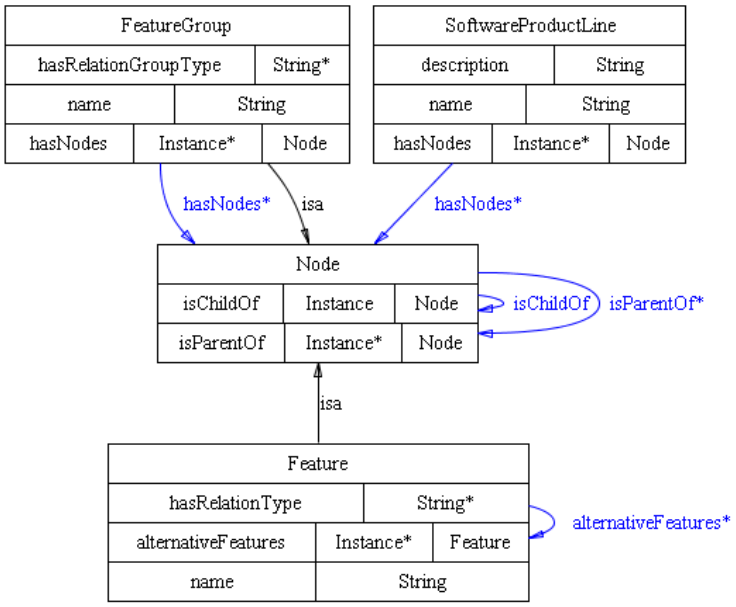
Fonte: SILVA, 2011.

O processo de modelagem de um sistema tutor inteligente foi relacionado aos modelos específicos de características gerais de Sistemas Tutores inteligentes juntamente com os modelos necessários para especificação de LPS. O autor apresenta quatro modelos reproduzidos em ontologias que foram combinados e compartilhados, para que o domínio fosse adaptado e projetado no sentido de fornecer uma LPS.

A primeira ontologia, chamada de LPS *Ontology*, foi projetada para ser utilizado como um metamodelo para especificação semântica de FODA (KANG et al., 1990), onde a partir

deste modelo, é possível modelar ontologias de qualquer domínio de aplicação, dentro das especificações do próprio modelo. A Figura 16 representa essa ontologia.

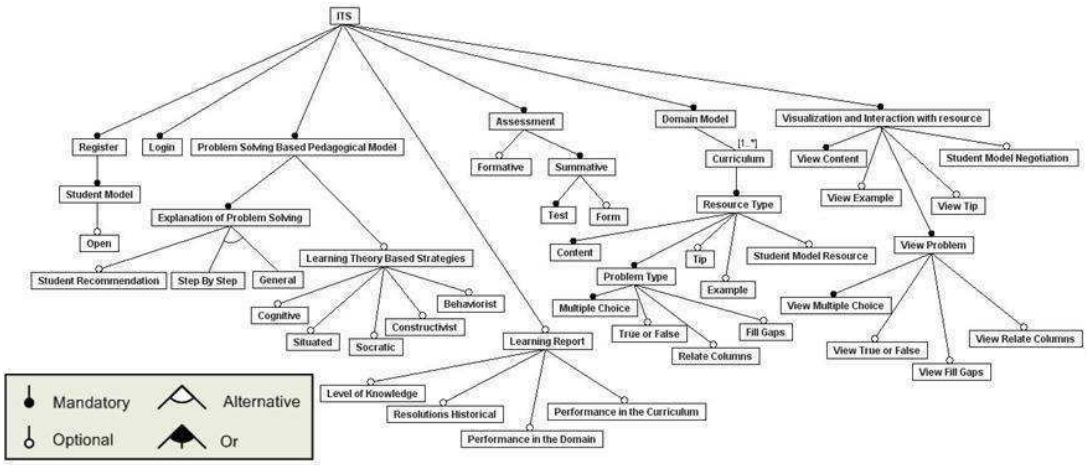
Figura 16 - Ontologia para Representação de Linha de Produto de *Software*.



Fonte: SILVA, 2011.

A segunda especificação, chamada de LPS-ITS *Ontology* fornece todas as possibilidades de um sistema Tutor, possibilitando até mudanças, demonstradas na Figura 17.

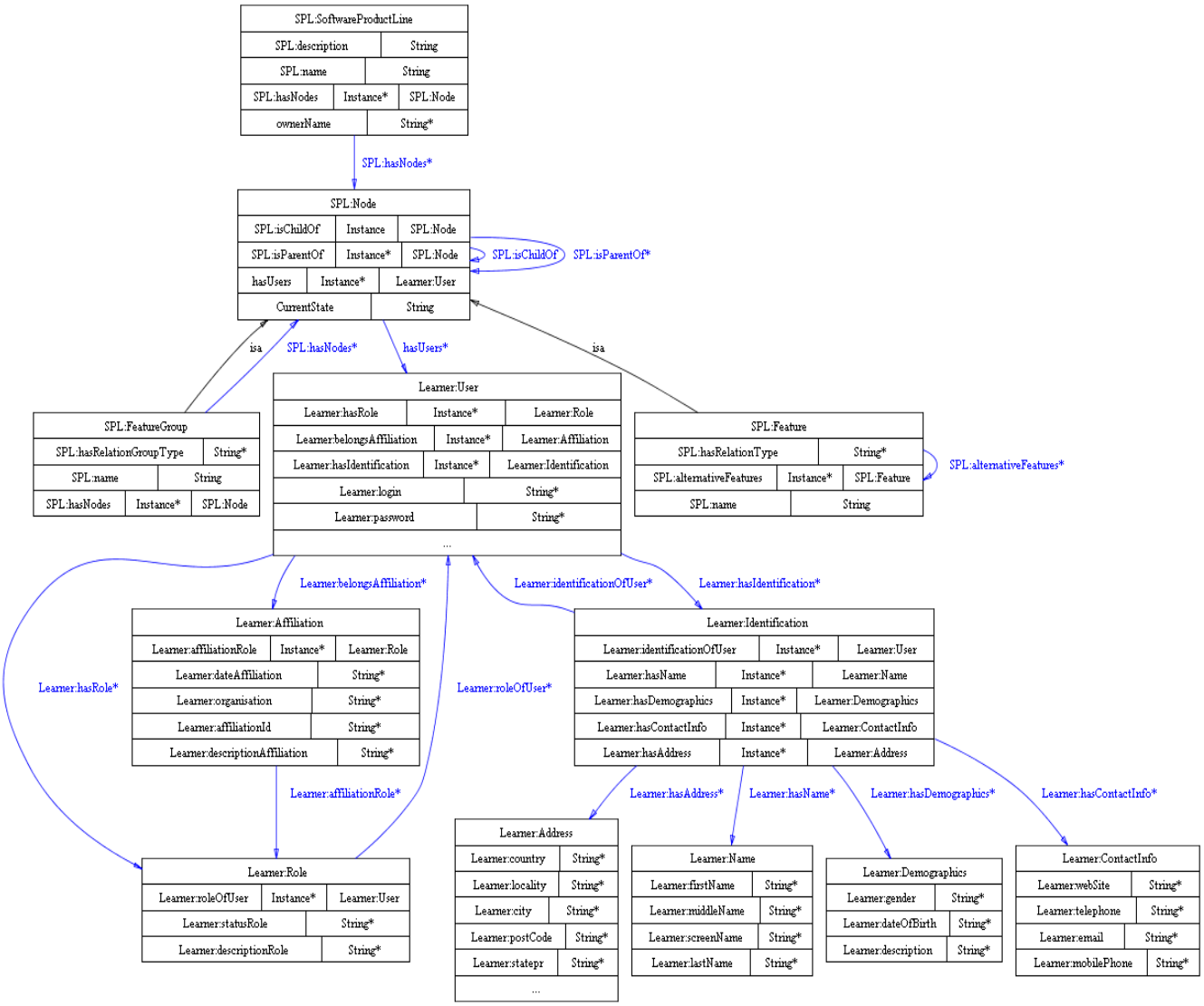
Figura 17 - Modelagem de *Features* representada na ontologia LPS-ITS.



Fonte: SILVA, 2011.

A terceira especificação adaptada de outro trabalho pelo autor, chamada *Forbile.Learner Ontology*, disponibiliza um mecanismo para representação de conhecimento de aprendizes. E a ultima (ver Figura 18), chamada *Decision Model Ontology*, representa a instanciação propriamente dita de um Sistema Tutor Inteligente, especificando cada produto de forma adequada.

Figura 18 - Ontologia para o Modelo de Decisão.



Fonte: SILVA, 2011.

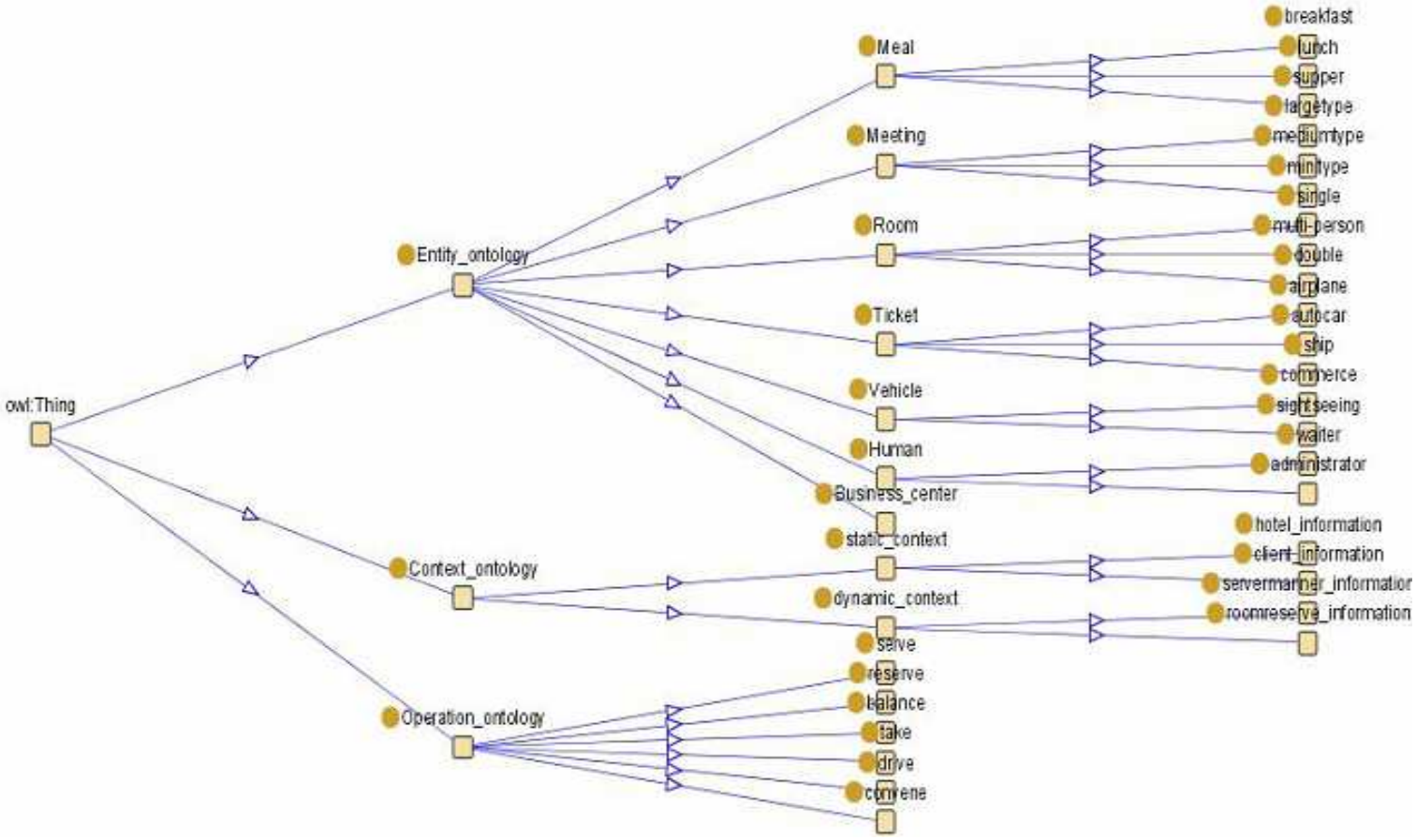
Os resultados mostraram-se favoráveis para o autor, onde seus experimentos foram analisados em dois domínios: programação e física. O uso de ontologias no trabalho se mostrou viável, pois a linha de produto pode ser flexível para diferentes domínios de conhecimento.

Outro trabalho relacionado com ontologias e linha de produto de *software* foi proposto por Bu-Qing et al. (2009). Este trabalho propõe uma abordagem orientada a serviços e

dirigida a processo, para a modelagem da variabilidade na linha de produto de *software*. Nesta abordagem, análise de variabilidade significa uma ontologia do modelo de domínio como ponto inicial, uma ontologia para o modelo de processo como o centro e uma ontologia para o modelo de serviços como a finalidade, as quais organizam a família de núcleo de ativos para identificar e modelar a variabilidade entre perspectivas do domínio, processos e serviços. A abordagem é discutida em um domínio de reserva de hotel.

O domínio de reserva de hotel foi modelado através de uma ontologia. Esta ontologia contém todos os artefatos que definem uma reserva de hotel. A Figura 19 apresenta a ontologia de reserva de hotel.

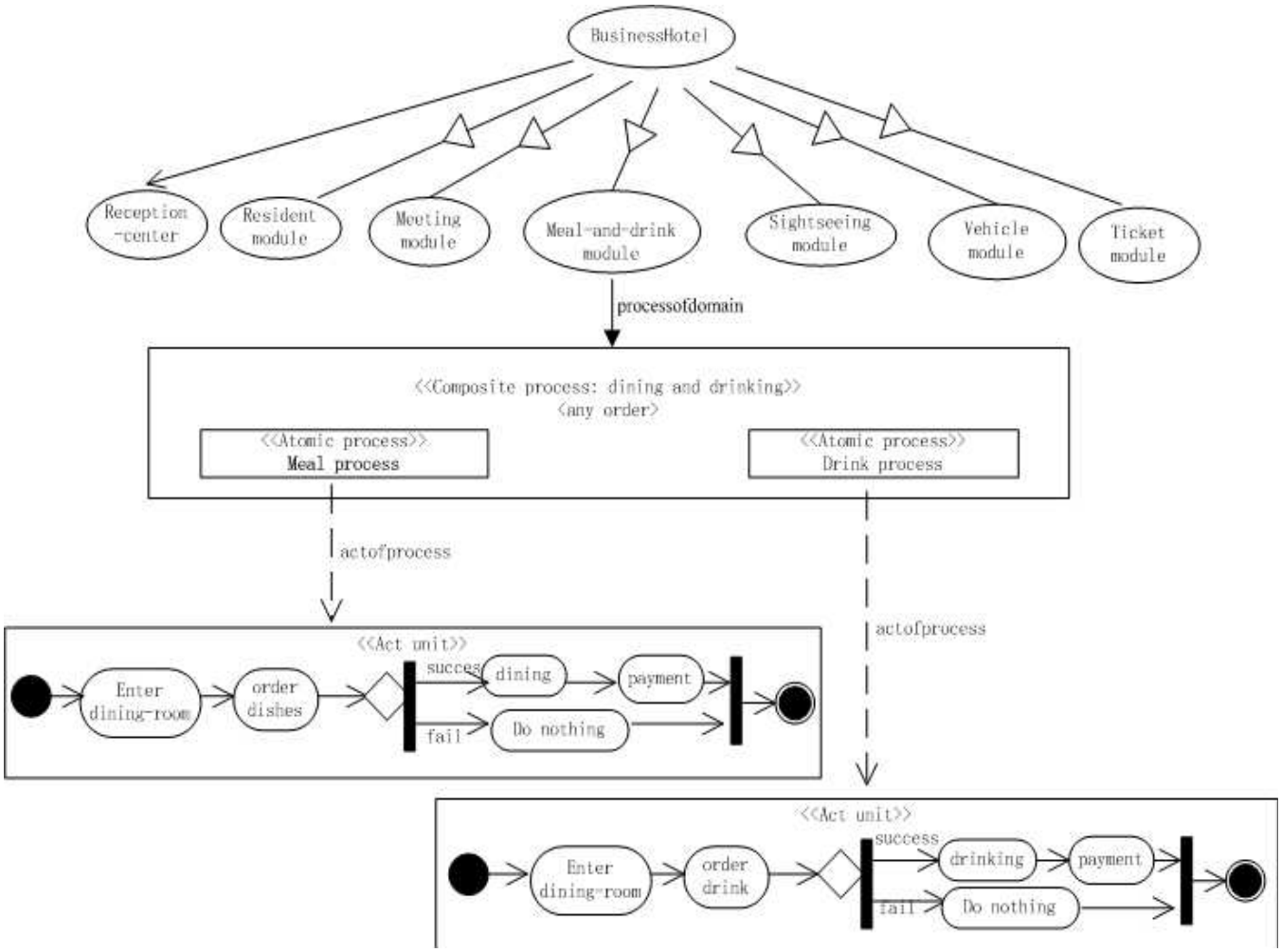
Figura 19 - Ontologia do modelo de domínio.



Fonte: BU-QING et al., 2009.

Depois de modelado o domínio, uma ontologia é desenvolvida para o modelo de processo que visa atender os requisitos comuns e individuais de um cliente. A Figura 20 representa esta ontologia.

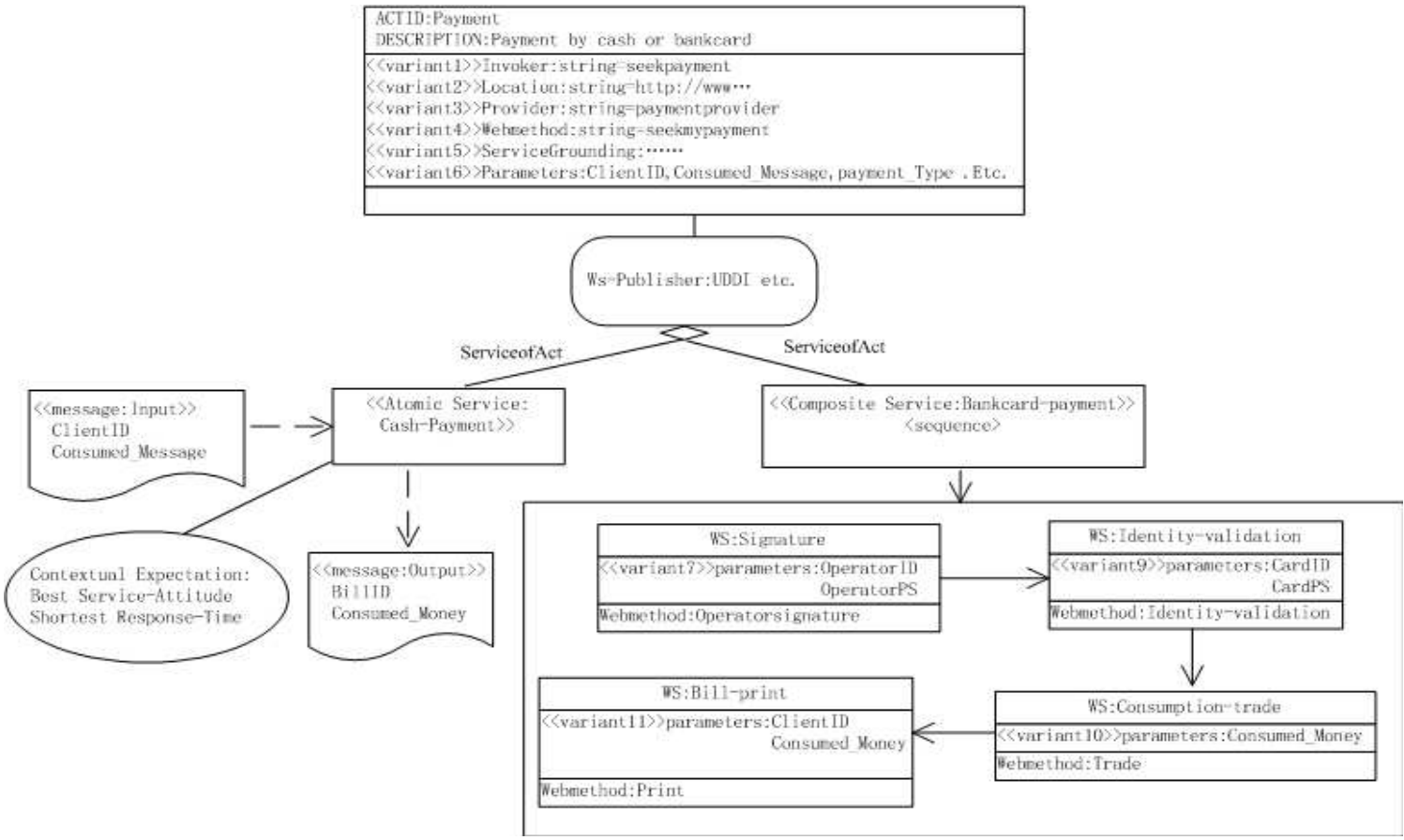
Figura 20 - Ontologia de modelo de processo.



Fonte: BU-QING et al., 2009.

Por fim, é modelada a ontologia de serviços onde são identificados os diferentes pontos de variação. Como ponto de partida, os requisitos de pagamento em dinheiro ou cartão foram desenvolvidos na ontologia de serviços. A Figura 21 representa esta ontologia.

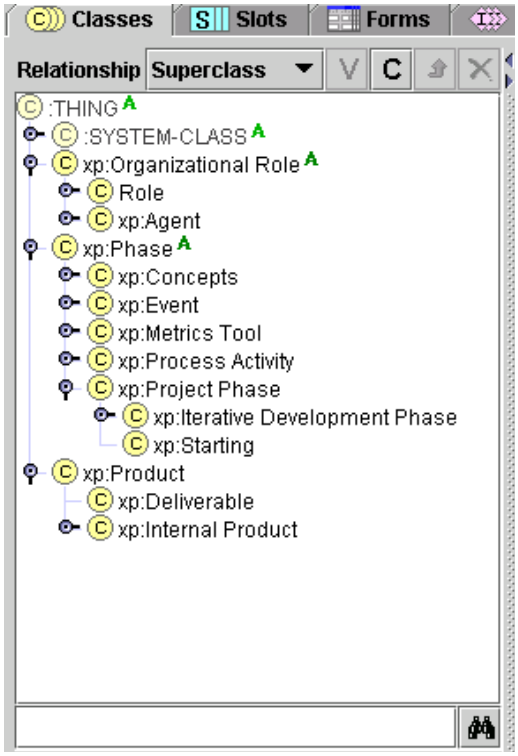
Figura 21 - Ontologia do modelo de serviços.



Fonte: BU-QING et al., 2009.

O trabalho de Ceravolo et al. (2003) descreve uma ontologia especificando os principais conceitos utilizados na metodologia *Extreme Programming*. A ontologia foi desenvolvida conforme estabelece a abordagem de modelagem de processos na engenharia de *software*. Três conceitos foram definidos como os principais na ontologia, chamando-os de *core class*: papel organizacional (*Oganization Role*), fase (*phase*) e produto (*product*). A Figura 22 representa as classes modeladas na XP *ontology* (XPO).

Figura 22 - Ontologia XPO.



Fonte: CERAVOLO et al., 2003.

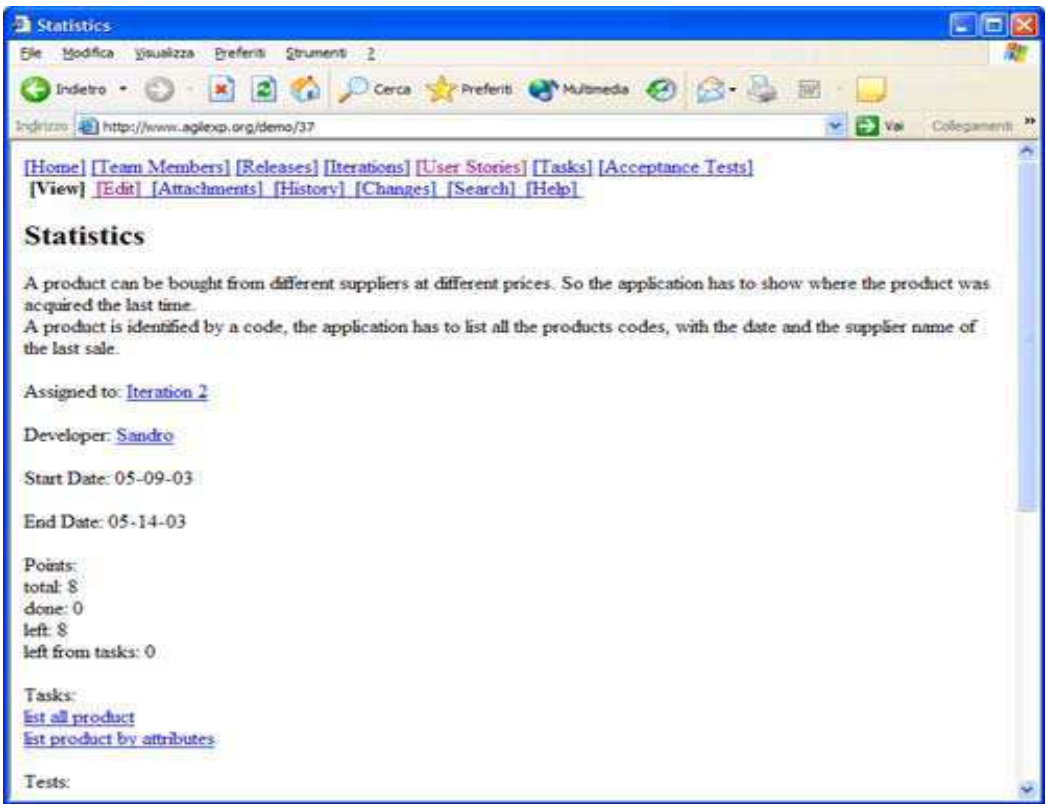
A classe papel organizacional tem duas subclasses diretas, chamado Papel e Agente. Um agente é uma pessoa ou uma organização que tem um papel. Um papel pode ser uma função geral, um papel da organização. A classe fase possui subclasses que representam conceitos gerais e comuns a todos os processos. A cada evento que ocorre, um processo é iniciado e diferentes atividades são executadas. Outras subclasses dessas subclasses estão modeladas na ontologia, definindo etapas do processo XP como *feedback* para o cliente, históricos dos usuários e modificações, codificação das atividades, testes, planejamento de requisitos, acompanhamento do projeto e ainda incluindo técnicas típicas XP como CRC *Session* e *Spike Solution* onde são usadas para fazer projetos mais simples. Por último a classe produto, que descreve produto de entrega e interno. O produto interno são as iterações e os documentos de planejamento de entrega.

O trabalho relata que esta ontologia pode ser útil para indexar documentos relevantes e artefatos XP, como históricos dos usuários e sites *Wiki*, além de análise de processos ágeis, processo de mineração de dados sobre as atividades dos programadores e repositórios de conteúdos. O exemplo de uso da XPO ilustrado no artigo foi um histórico dos usuários (*sites Wiki*) chamado de Estatística, conforme Figura 23, onde atributos como data de início e fim, a versão e o desenvolvedor responsável.

As informações são consultadas por agentes de *software* através de um Sistema de Representação do Conhecimento, que são compostos de uma linguagem e ambiente para

construção de ontologias. Assim quando escolhido algum conceito como o histórico, é possível recuperar todas as informações relacionadas ao projeto.

Figura 23 - Resultado de estatística do projeto.



Fonte: CERAVOLO et al., 2003.

No trabalho realizado por Simidchieva et. al. (2007), foi proposta uma representação de variações de processos como uma família de processos, através da ferramenta Little-JIL (Linguagem de programação para coordenação de processos). Verificou-se que muitas equipes realizavam processos de uma forma que se diferenciavam umas das outras, sendo que essas diferenças eram em forma de detalhes, porém esses detalhes de processo eram importantes no contexto como um todo. Foi utilizada uma abordagem de família de produtos de *software* para processos, que representavam todas as variações de um metaprocesso. Porém essas variações deviam ser suficientemente semelhantes para que pudessem pertencer à mesma família.

O trabalho realizado por Mallmann (2011) propõe um modelo abstrato de gerência do processo de *software* para metodologias ágeis. Ele apresenta os principais modelos de metodologias ágeis e gerência de projetos utilizados para a definição de seu metamodelo, e também um ambiente pró-ativo para gerência de *software* em metodologias ágeis. O autor selecionou três metodologias ágeis como alvo de seu estudo (XP, *Scrum* e FDD). A seleção destas metodologias se deu em função de existirem estudos indicando que elas se complementam. Segundo o autor, XP foca no desenvolvimento de *software*, *Scrum* é mais voltada para o gerenciamento do projeto e FDD é orientada a modelagem.

O metamodelo proposto contempla um conjunto de características e conceitos das três metodologias ágeis de desenvolvimento de *software* e do modelo de gerência de projetos PMBOK, que permitiu a criação de uma aplicação para melhor gerenciar projetos de *software* utilizando metodologias ágeis. O metamodelo foi projetado para ser modular o suficiente, permitindo que seja instanciado em um determinado projeto, gerando um modelo concreto de acordo com as características da metodologia estabelecidas pelo gerente.

A partir do modelo concreto, é gerado um modelo gerencial responsável por armazenar as informações coletadas, assim como dados resultantes do processo de rastreabilidade de artefatos de *software*, proposto por Dall'Oglio (Dall'Oglio apud, Mallmann, 2011).

O objetivo do ambiente é (Mallmann, 2011):

- Gerenciar recursos: gerenciar recursos necessários para um projeto, tais como recursos físicos, pessoas e equipes.
- Controlar atividades: controlar atividades gerenciais do projeto numa dada fase, como verificar produtos de trabalho, atribuição de papéis e processos gerenciais.
- Controlar itens de desenvolvimento: controlar andamento das tarefas específicas de determinada atividade, histórico de alterações e efetuar medição dos impactos gerados.
- Controlar os ciclos de desenvolvimento: controlar as funcionalidades concluídas num ciclo com relação às que tinham sido previstas, além da ocorrência de reuniões de planejamento e verificação se os itens dum ciclo estão sendo cumpridos dentro da prioridade para eles estabelecida.

3.1 QUADRO COMPARATIVO

Este quadro comparativo, ver Tabela 1, aborda as características existentes no trabalho proposto e as características dos trabalhos relacionados.

Tabela 1 - Quadro comparativo de trabalhos

	Trabalho de Silva (2011)	Trabalho de Bu-qing et al. (2009)	Trabalho de Ceravolo et.al. (2003)	Trabalho de Simidchieva et. Al.(2007)	Trabalho de Mallmann (2011)	Trabalho Proposto
Linha de produto de *software*	Sim	Sim	Não	Sim	Não	Sim
Gerência de projeto de *software*	Não	Não	Sim	Sim	Sim	Sim
Metodologias ágeis	Não	Não	Sim	Não	Sim	Sim
Representação por ontologia	Sim	Sim	Sim	Não	Não	Sim
LPS para gerência de projeto de *software* em metodologias ágeis	Não	Não	Não	Não	Não	Sim
Suporte a variabilidade da LPS por ontologia	Sim	Sim	Não	Não	Não	Sim
Criação de um novo processo de desenvolvimento de *software* baseado em ontologia e LPS	Não	Não	Não	Não	Não	Sim

Fonte: Desenvolvido pelo autor.

4 MODELO PROPOSTO

Este capítulo apresenta o modelo para o desenvolvimento da solução proposta neste trabalho. É proposto o modelo para o desenvolvimento de uma ontologia que define uma linha de produto de *software* para gerência de processo de *software* com metodologias ágeis. São abordados os conceitos descritos no embasamento teórico, aplicados ao modelo de solução.

Mallmann (2011) apresentou uma proposta de modelo abstrato e ambiente de gerência de *software* do qual podem ser instanciados modelos concretos específicos para determinados projetos, que torna mais eficaz a gerência de projetos de *software* em ambientes de desenvolvimento que utilizem metodologias ágeis.

O objetivo deste trabalho é propor um alinhamento do modelo abstrato proposto por Mallmann (2011) aos conceitos de ontologias e linhas de produtos de *software* de modo que, esta linha de produto seja definida através de uma ontologia, onde os produtos gerados também sejam modelos de gerência de projetos ágeis. A modelagem que o autor utilizou no seu ambiente proporciona ser facilmente adequado à abordagem de linha de produto de *software*. A partir daí, este trabalho se difere em duas técnicas por modelar um processo de desenvolvimento de *software* através de uma ontologia que representa a arquitetura de uma linha de produto de *software*.

A ontologia construída define uma linha de produto de *software* em que suas características constituem conceitos das metodologias ágeis XP, *Scrum* e FDD, e do método de gerência PMBOK. Através desta ontologia pode ser criados processos de desenvolvimento de *software* em que as características variam entre os conceitos a cima. O conceito de linha de produto através de sua arquitetura permite criar um processo fixo, que possui características comuns entre os conceitos mencionados, ou um processo que possui além das características comuns algumas variáveis, tornando flexível a arquitetura da linha de produto.

4.1 VISÃO GERAL

A temática proposta neste estudo contempla informações sobre o domínio de gerenciamento de processo de desenvolvimento de *software* baseado em metodologias ágeis. A ontologia tem por objetivo definir uma linha de produto de *software* (LPS), baseada no domínio em questão. O processo de desenvolvimento de *software* constitui o produto final desta linha, sendo gerado através de consultas realizadas por agentes de *software*. A flexibilidade da ontologia fornece a capacidade de variação no estabelecimento do processo.

O desenvolvimento de uma linha de produto para gerência de projeto com métodos ágeis visa melhorar o gerenciamento do desenvolvimento de *software*, fornecendo uma ferramenta automatizada para apoiar o planejamento, o monitoramento, e o controle do trabalho de desenvolvimento. A integração da linha de produto com métodos ágeis fornece suporte para a priorização do desenvolvimento de tarefas a serem executadas durante o decorrer do processo.

4.2 ONTOLOGIA PARA REPRESENTAÇÃO DE LPS

As ontologias são estruturas de representação de conhecimento particularmente úteis para especificação de abstrações de *software* de alto nível. Elas proveem uma terminologia

unívoca que pode ser compartilhada por todos os envolvidos em um processo de desenvolvimento. Além disso, são facilmente extensíveis e adaptáveis, possibilitando um efetivo reuso.

As ontologias tornam explícito o conhecimento do domínio para a linha de produto, permitindo a reutilização. Em Falbo et al. (2002), define ontologia como um modelo para domínio de aplicação, sendo usada basicamente para especificá-los e desenvolvê-los aumentando o reuso.

4.3 ENGENHARIA DA LINHA DE PRODUTO

A engenharia da LPS é responsável por definir a plataforma da LPS, onde todos os artefatos reutilizáveis que definem os aspectos comuns e variáveis devem estar presentes. Esta etapa inicia com a análise de domínio, onde os requisitos são levantados e modelados. Identificando o domínio e as características têm-se os *assets* reutilizáveis para instanciar produtos.

O domínio do trabalho proposto consiste no gerenciamento de projetos com metodologias ágeis, que foi extraído do trabalho feito por Mallmann (2011).

4.3.1 Análise do Domínio

A análise de domínio deve organizar o conhecimento, os relacionamentos entre os vários elementos no domínio e representar esse conhecimento. Esta etapa é definida também como o processo de identificação, coleta, organização e representação das informações relevantes em um domínio, é baseado no estudo de sistemas existentes e suas histórias de desenvolvimento, conhecimento capturado de especialistas de domínio, teoria de base e tecnologia emergente em um domínio.

A análise do domínio pode ser definida como o processo pelo qual a informação usada para o desenvolvimento de *software* é identificada, capturada e organizada para que seja reutilizável quando da criação de novos sistemas.

O objetivo é identificar e organizar os requisitos do domínio. O levantamento de requisitos está presente no início do ciclo de vida de *software*, numa fase em que os responsáveis pelo desenho do sistema estão a adquirir toda a informação necessária para garantir que o produto em questão satisfaça quem o vai consumir.

Algumas etapas podem ser seguidas na análise de domínio, como:

- O planejamento: Esta fase é marcada pela análise geral, onde se consistem verificar benefícios, custo e o conhecimento sobre o domínio;
- Aquisição dos dados: Aqui são identificadas as fontes de dados, os requisitos que são disponíveis. As informações são geralmente capturadas de fontes como literatura técnica;
- Análise dos dados e modelagem: Nesta etapa, o conhecimento capturado é inicialmente avaliado quanto à consistência, correção e completude e, então, modelado identificando entidades, relações, funções e axiomas comuns às diversas fontes analisadas.

O manifesto ágil possui várias métricas para o processo de desenvolvimento como abordadas na revisão bibliográfica feita neste trabalho. Para a solução, foram utilizadas apenas as três metodologias mais utilizadas segundo Prass e Puntel (2011) e Gonçalves e Filho (2008), que são a XP, *Scrum* e FDD. Da mesma forma, a gerência de projeto também possui alguns modelos. No estudo será abordado apenas o modelo PMBOK.

4.3.2 Requisitos de Domínio

Antes de construir um produto, coletar os requisitos é fundamental, pois os requisitos formam as características do produto. Os requisitos do domínio da LPS constituem todas as características que possuem as três metodologias ágeis, XP, *Scrum* e FDD, e o método de gerência PMBOK. Abaixo são especificados os requisitos de cada um.

4.3.3 Elementos Constitutivos do Domínio XP

O *Extreme Programming* (XP) é uma técnica utilizada para criar *software* flexível e de alta qualidade, empregando a equipe de desenvolvimento em atividades que produzam resultados na forma de *software* em curto espaço de tempo, e ainda customizando o produto de acordo com a necessidade do usuário.

Projetos XP dividem o tempo disponível para o projeto utilizando conceitos de *releases* e iterações. Um *release* representa um marco no tempo, no qual um conjunto coeso de funcionalidades é finalizado e lançado para consumo de seus usuários. No espaço de tempo de um *release,* a equipe implementa funcionalidades em iterações curtas e fixas, para fornecer cadência ao processo de desenvolvimento. Uma iteração é um incremento de *software* útil que é projetado, programado, testado, integrado e entregue durante um espaço de tempo (TELES 2005).

Outro conceito é o de histórias que representam funcionalidades que refletem necessidades do cliente, e são suficientemente pequenas para que os programadores possam implementar um pequeno conjunto delas a cada iteração. O desenvolvimento baseado em iterações curtas é uma forma de elevar as chances de convergir para um sistema que efetivamente atenda às necessidades de seus usuários dentro do tempo disponível para o projeto (TELES 2005).

Com o objetivo de assegurar que as partes trabalhem bem em conjunto, o XP utiliza uma breve reunião diária chamada de *stand up meeting*. O objetivo da reunião é alinhar os membros da equipe informando os resultados obtidos no dia anterior e permitindo que os participantes priorizem as atividades do dia que se inicia (TELES 2005).

Alguns elementos constituem as regras que definem o ambiente XP:

- Registro e descrição dos requisitos dos utilizadores (*User Stories*);
- Identificação e calendarização dos entregáveis do projeto (*Release planning*);
- Registro e descrição das tarefas a executar para a implementação de cada um dos requisitos (*Tasks*);
- Identificação dos testes de aceitação dos requisitos (*Test Cases*);
- Equipe de desenvolvimento;

- Cliente fazendo parte da equipe;
- Estruturar o plano de projeto num conjunto de iterações de curta duração (*Iteration planning*).

4.3.4 Elementos Constitutivos do Domínio *Scrum*

A metodologia *Scrum* assume-se como uma metodologia extremamente ágil e flexível. O desenvolvimento de processo *Scrum* se divide em iterações (*sprints*) de trinta dias. Equipes pequenas, de até dez pessoas, são formadas por projetistas, programadores, engenheiros e gerentes de qualidade. Estas equipes trabalham em cima de funcionalidades, ou seja, requisitos, definidas no início de cada *sprint*. A equipe é responsável pelo desenvolvimento desta funcionalidade.

Um *Sprint* inicia-se com uma reunião de planejamento (*Sprint Planning Meeting*), na qual o *Product Owner* e a equipe decidem em conjunto o que deverá ser implementado (*Selected Product Backlog*). A reunião é dividida em duas partes. Na primeira parte (*Sprint Planning* 1) o *Product Owner* apresenta os requisitos de maior valor e prioriza aqueles que devem ser implementados. A equipe então define colaborativamente, o que poderá entrar no desenvolvimento da próxima *Sprint*, considerando sua capacidade de produção. Na segunda parte (*Sprint Planning* 2) o time planeja seu trabalho, definindo o *Sprint Backlog* que são as tarefas necessárias para implementar as funcionalidades selecionadas no *Product Backlog*. Nas primeiras *Sprints* é realizada a maioria dos trabalhos de arquitetura e de infraestrutura.

4.3.5 Elementos Constitutivos do Domínio FDD

Os elementos que constituem a metodologia *Feature Driven Development* (FDD) chamam a atenção por algumas características peculiares. Essas características variam em resultados rápidos, blocos de funcionalidades pequenos, sem restrição do tamanho do projeto e da equipe, planejamento detalhado, monitoramento do projeto com resumos de alto nível. Estas características se definem ao executar os cinco processos da metodologia: Desenvolver um Modelo Abrangente, Construir uma Lista de Funcionalidades, Planejar através de Funcionalidades, Projetar através de Funcionalidades e Construir através de Funcionalidades (BUBLITZ, 2009).

O ciclo da FDD inicia com a criação de um modelo de objetos do domínio do problema, em colaboração com os especialistas no domínio. Usando a informação vinda da atividade de modelagem e de quaisquer outras atividades de coleta de requisitos que já possam ter ocorrido, os desenvolvedores prosseguem para a criação da lista de funcionalidades. A seguir, é elaborado um plano para cada funcionalidade e são atribuídas responsabilidades. Então, equipes pequenas e formadas dinamicamente desenvolvem as funcionalidades, realizando repetidamente iterações de projeto (*design*) e construção, que duram não mais do que duas semanas e, geralmente, são muito mais curtas (BUBLITZ, 2009).

4.3.6 Elementos Constitutivos do Domínio PMBOK

Os elementos constitutivos do PMBOK visam à boa prática de gerenciamento com qualidade. Caracterizado por agrupar melhores práticas de gerência, onde a base de conhecimento de gerência de projetos está estruturada em cinco grupos: Iniciação, Planejamento, Execução, Monitoramento e Controle, e Encerramento.

A iniciação consiste na definição de um novo projeto ou uma fase do projeto. As pessoas envolvidas são identificadas e o gerente do projeto é definido. No planejamento são refinados os objetivos e desenvolvido um plano de trabalho. A execução é o trabalho definido no plano de projeto para atender às especificações que é executado. Coordenação de pessoas e de recursos, integrando e executando as atividades do projeto de acordo com o plano. Pode haver a necessidade de mudanças de planejamento ou redefinições para obter o resultado. Estas mudanças incluem duração, recursos e controle de riscos. O monitoramento e controle identificam se é necessário realizar mudanças e notificam as alterações no plano do projeto. Monitorando os processos de execução, estas ações preventivas tornam-se necessárias para o sucesso do projeto. Por último, no encerramento são finalizadas as atividades dentro do grupo de processos de gerenciamento do projeto, para formalizar a conclusão do projeto, fase ou obrigações contratuais. A aceitação do projeto ao cliente é verificada, uma revisão do trabalho pode ser realizada e as melhores decisões e práticas são concretizadas para uso em futuros projetos, arquivando os documentos relevantes para servirem como um histórico de dados (HUIJBERS et.al., 2004).

A partir dessa estrutura podem-se definir algumas etapas como:

- Desenvolver o termo e de plano do projeto;
- Orientar, gerenciar, monitorar e controlar a execução do projeto;
- Encerrar o projeto ou a fase;
- Coletar requisitos e definir o escopo;
- Definir, sequenciar, estimar recursos e duração em atividades;
- Desenvolver o plano de recursos humanos;
- Contratar ou mobilizar a equipe do projeto;
- Identificar as partes interessadas;
- Planejar as comunicações.

4.4 MODELO DE DOMÍNIO

Após realizar a análise de domínio onde foram extraídas as características das metodologias XP, *Scrum* e FDD e do método de gerência PMBOK, é realizada a modelagem destes elementos.

Mallmann (2011) propõe um modelo abstrato de gerência do processo de *software* para metodologias ágeis. Ele apresenta os principais modelos de metodologias ágeis e gerência de projetos utilizados para a definição de seu metamodelo, e também um ambiente pró-ativo para gerência de *software* em metodologias ágeis. O autor selecionou três metodologias ágeis como alvo de seu estudo (XP, *Scrum* e FDD). A seleção destas metodologias se deu em função de existirem estudos indicando que elas se complementam.

Segundo o autor, XP foca no desenvolvimento de *software*, *Scrum* é mais voltada para o gerenciamento do projeto e FDD é orientada a modelagem.

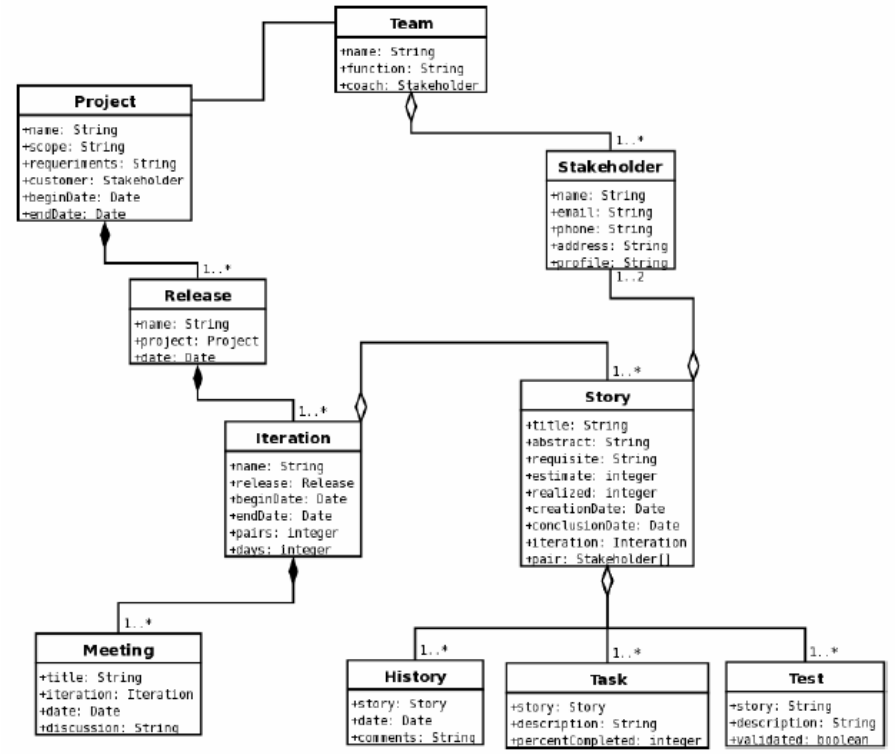
Para cada uma destas metodologias, Mallmann (2011) apresenta o modelo correspondente, além de fazer uma comparação entre os modelos estudados. Para o modelo de gerência de projetos estudado, o autor propôs e apresenta também um modelo simples do PMBOK.

Após realizar a análise de domínio onde foram extraídas as características das metodologias XP, *Scrum* e FDD e do método de gerência PMBOK, é realizada a modelagem destes elementos baseados no trabalho de Mallmann (2011).

4.4.1 Modelo de domínio XP

O modelo de domínio da XP consiste em modelar as características de um processo de desenvolvimento que utiliza esta metodologia. A Figura 24 representa um modelo de domínio proposto por Mallmann (2011), em que propôs um modelo de classes com os principais conceitos da XP.

Figura 24 - Modelo de domínio XP.



Fonte: MALLMANN, 2011.

As etapas modeladas foram:

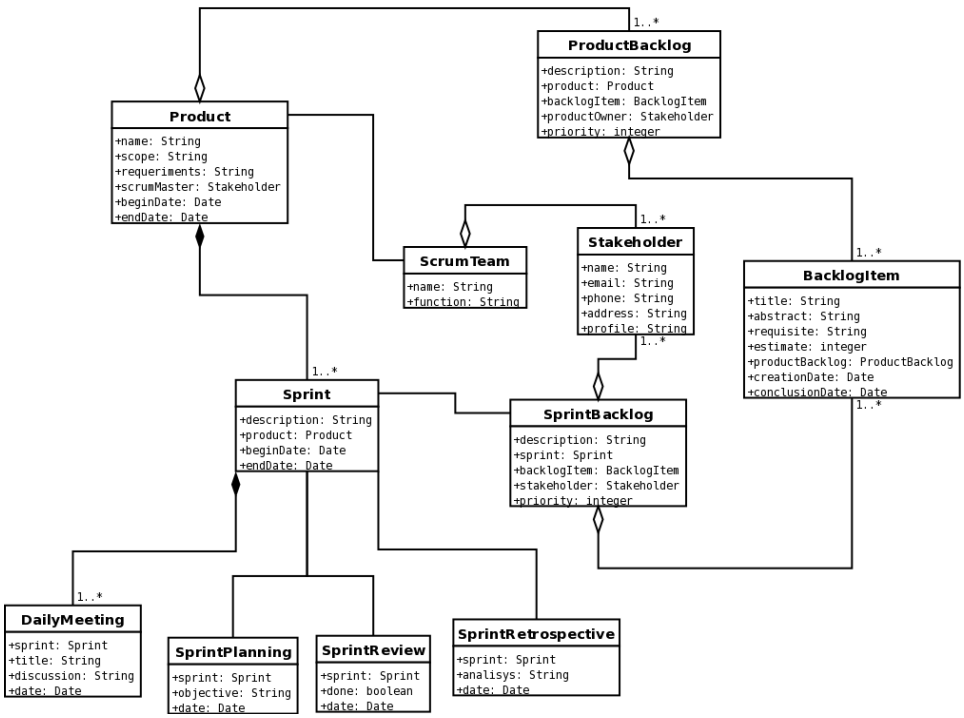
- ***Project:*** Projetos em que cada um contém informações de seu cliente, nome do projeto, escopo e requisitos;
- ***Stakeholder*****:** Responsável e detentor de uma parte do projeto;

- ***Team:*** Equipe do projeto contendo o líder do projeto;
- ***Release:*** São os diferentes *releases* de um projeto. Possuem informações essenciais como nome e data de lançamento.
- ***Iteration:*** São as iterações de um *release.*
- ***Story***: Cada *story* possui informações como seu título, resumo, estimativa, tempo realizado, data de criação, data de conclusão e par de programadores. Além disso, possui alguns objetos agregados como *History, Tasks e Tests*;
- ***History***: Armazena o histórico de decisões de uma *story*, através de informações como a data e um comentário;
- ***Tasks***: Tarefas nas quais uma *story* é dividida, possuindo uma descrição e o percentual de conclusão;
- ***Tests***: Provê uma descrição dos testes da *story*, verificando se são válidos;
- ***Meeting:*** Rápidas reuniões matinais realizadas antes de se iniciar um dia de trabalho.

4.4.2 Modelo de Domínio *Scrum*

As etapas da metodologia *Scrum* formam um gerenciamento flexível, adaptável e produtivo. Através dos princípios da *Scrum*, a Figura 25 apresenta o modelo de classes proposto por Mallmann (2011). As etapas modeladas estão descritas no arcabouço teórico mencionado anteriormente.

Figura 25 - Modelo de domínio da *Scrum.*

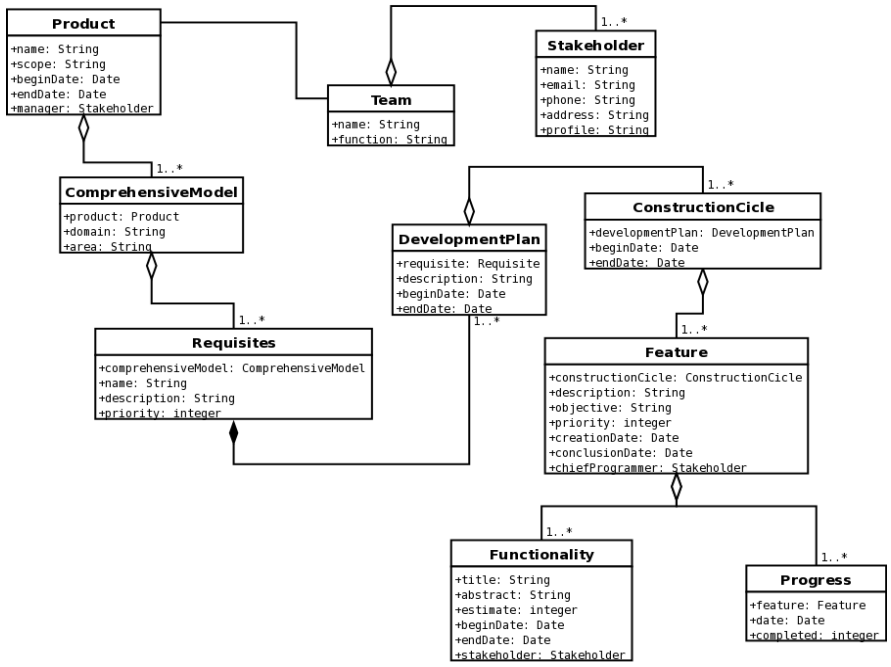


Fonte: MALLMANN, 2011.

4.4.3 Modelo de Domínio FDD

O modelo FDD, ver Figura 26, também proposto por Mallmann (2011) apresenta os principais conceitos e relacionamentos que definem um processo de desenvolvimento FDD.

Figura 26 - Modelo de domínio da FDD.



Fonte: MALLMANN, 2011.

As definições das classes são:

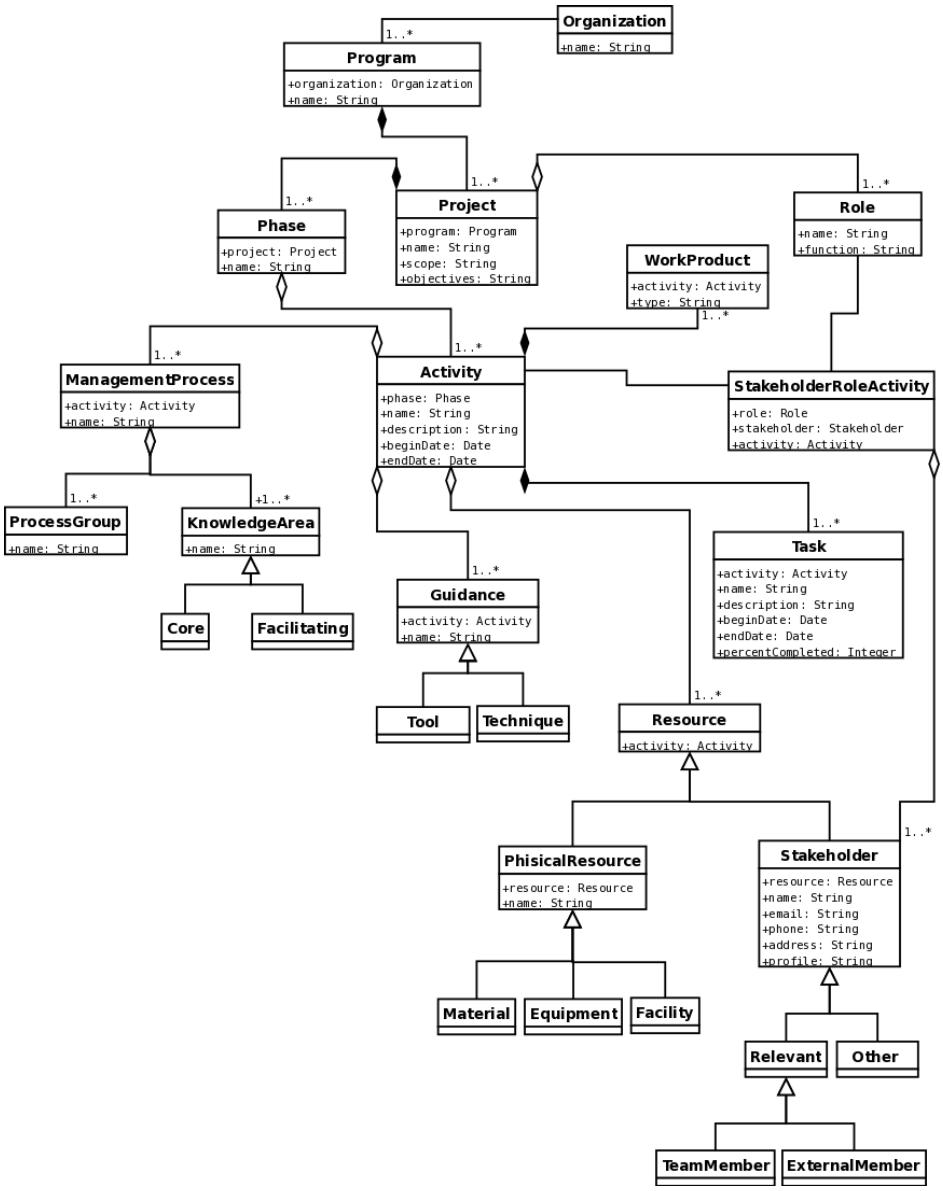
- ***Product:*** Armazena as informações do produto;
- ***Team:*** É a equipe do projeto;
- ***Stakeholder:*** Responsável e detentor de uma parte do projeto;
- ***ComprehensiveModel:*** Definição do que será feito, para guiar a equipe durante a fase de construção;
- ***Requisites:*** Informações/Características do domínio do negócio;
- ***DevelopmentPlan:*** Com a lista de requisitos e o modelo, deve-se planejar a ordem na qual as funcionalidades serão implementadas;
- ***ConstructionCicle:*** Ciclo de construção. Seguindo o planejamento, são eleitas as *Features* a serem desenvolvidas;
- ***Feature:*** Pacote de trabalho designado a um programador-chefe. Tem uma prioridade estabelecida assim como também uma data de criação e uma data de conclusão;
- ***Functionality:*** Trata-se de uma *Feature* decomposta em funcionalidades mais simples que possa ser desenvolvido o mais rápido possível;

- ***Progress:*** Armazena o progresso de uma *Feature.*

4.4.4 Modelo de Domínio PMBOK

O método de gerência PMBOK define que a iniciação, o planejamento, a execução, o monitoramento e o encerramento são os cinco grupos de processos de gerência de projetos. A Figura 27 ilustra esse modelo proposto por Mallmann (2011).

Figura 27 - Modelo de domínio do PMBOK.



Fonte: MALLMANN, 2011.

As classes são detalhadas como:

- ***Organization:*** Representa a empresa que se organiza por programas;
- ***Program:*** São grupos de projetos com a função de alcançar um objetivo;
- ***Project:*** Projeto onde são aplicados os processos e as áreas de conhecimento do PMBOK;
- ***Phase:*** São as fases do projeto;
- ***Activity:*** São as atividades gerenciais do projeto em determinada fase. Possui alguns objetos agregados como *ManagementProcess, Guidance, Task, Resource* e *WorkProduct;*
- ***ManagementProcess:*** Descrição dos processos gerenciais da atividade. Estes processos podem pertencer a grupos de processos (*ProcessGroup)* ou áreas de conhecimento (*KnowledgeArea*) que se subdividem em áreas centrais e de apoio;
- ***Guidance:*** São guias que podem ser utilizados como ferramentas ou apoio técnico;
- ***Task:*** Divisão das atividades em tarefas menores;
- ***Resource:*** Armazena os recursos necessários para um projeto. Estes recursos são divididos em recursos ativos (*Stakeholder)* e recursos inativos (*PhysicalResource*). A classe *Stakeholder* se divide em uma subclasse de pessoas relevantes (membros da equipe ou membros externos) e outra com as demais. A classe *PhysicalResource* se desmembra em materiais, equipamentos e facilitadores necessários para o projeto;
- ***Role:*** São os papéis designados ao projeto;
- ***StakeholderRoleActivity:*** Atividade para ser resolvida por um ou mais *stakeholders* assumindo determinado papel no projeto.

4.5 MODELO DE FEATURES

O modelo de *features* contempla todas as características dos quatro modelos de domínio desenvolvidos anteriormente, modelando possíveis variações que podem aparecer em um produto. As características estarão denominadas como, obrigatórias, alternativas e opcionais, assim definindo variações na linha de produto. Cada produto da linha é definido a partir de uma seleção destas características.

4.6 DEFINIÇÃO DA ONTOLOGIA PRA LINHA DE PRODUTO

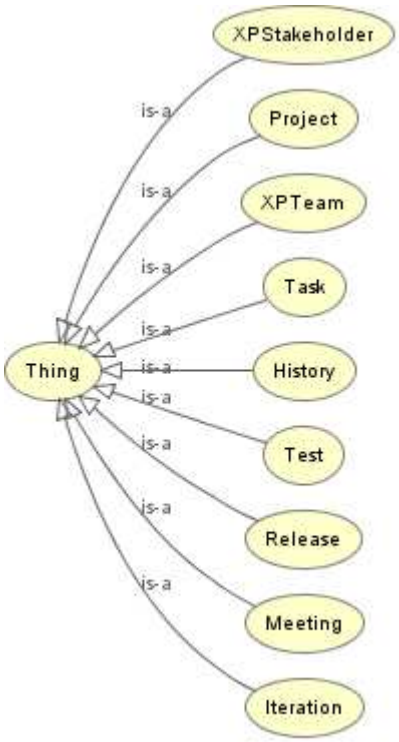
Após modelar as características que definem o domínio da LPS, foi construída uma ontologia para cada metodologia ágil e para o método de gerência, abordados aqui neste trabalho.

Contudo, nesta seção são abordadas as representações ontológicas de cada modelo de domínio, apresentando a ontologia de cada um, com o objetivo de evidenciar a solução e definir os resultados esperados. Após modelar o domínio é apresentada a sua estrutura, as possíveis variações e os objetivos funcionais.

4.6.1 Ontologia para o Modelo de Domínio XP

A ontologia construída para o modelo XP possui classes, relacionamentos entre classes e propriedades que determinam o comportamento de um processo de desenvolvimento XP. A ontologia foi desenvolvida de acordo com o modelo de domínio da subseção 4.4.2 que apresenta as principais características do domínio. A Figura 28 apresenta a ontologia XP.

Figura 28 - Ontologia XP.



Fonte: Desenvolvido pelo autor.

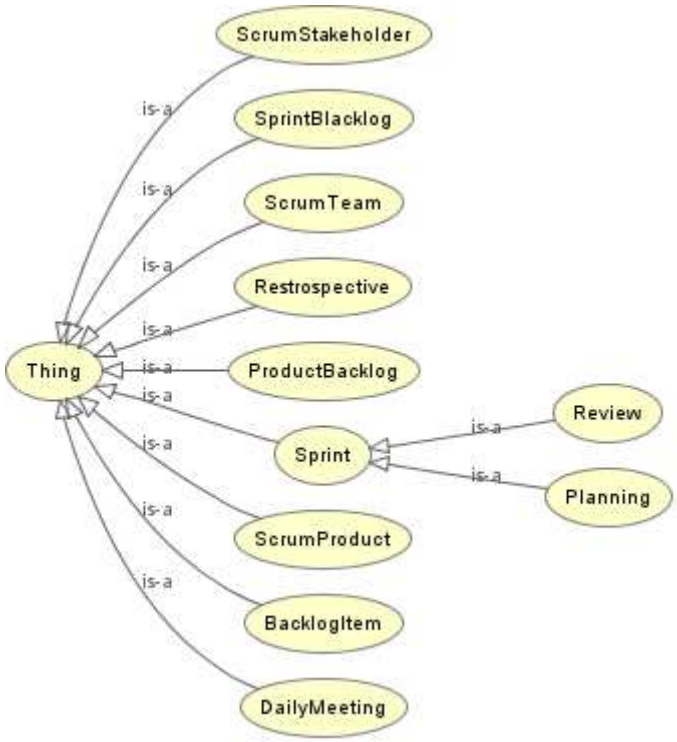
As classes da ontologia XP desenvolvidas como parte do domínio são:

- ***XPTeam***: representa a classe *Team* do modelo XP;
- ***XPStakeholder***: representa a classe *Stakeholder* do modelo XP;
- ***Iteration***: representa a classe *Iteration* do modelo XP;
- ***Release***: representa a classe *Release* do modelo XP;
- ***Project***: representa a classe *Project* do modelo XP;
- ***Meeting***: representa a classe *Meeting* do modelo XP.
- ***History***: representa a classe *History* do modelo XP.
- ***Test***: representa a classe *Test* do modelo XP.
- ***Task***: representa a classe *Task* do modelo XP.

4.6.2 Ontologia para o Modelo de Domínio *Scrum*

Da mesma forma que o modelo XP, o modelo *Scrum* foi representado através de uma ontologia. Esta ontologia aborda os conceitos e relacionamentos que definem o processo de desenvolvimento *Scrum*. Então, foi desenvolvida uma ontologia com as características *Scrum*, conforme ilustrada na Figura 29.

Figura 29 - Ontologia *Scrum.*



Fonte: Desenvolvido pelo autor.

As classes da ontologia *Scrum* desenvolvidas como parte do domínio são:

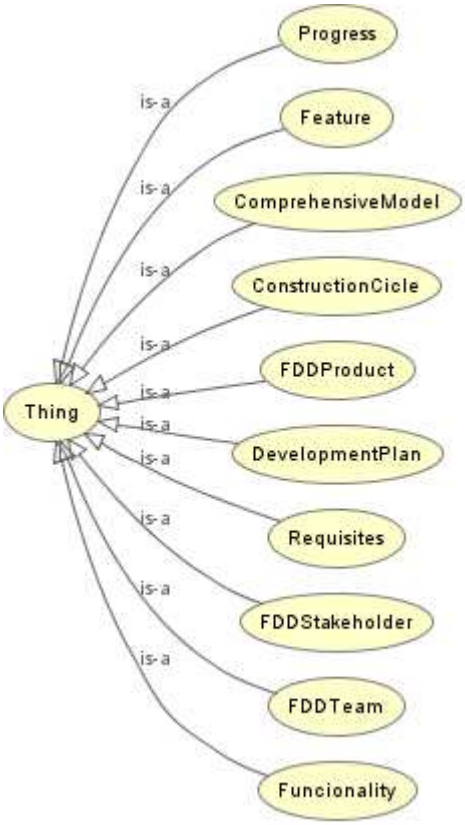
- ***ScrumProduct*** representa classe *Product* do modelo *Scrum*;
- ***ScrumStakeholder*** representa classe *Stakeholder* do modelo *Scrum*;
- ***DailyMeeting*** representa a classe *DailyMeeting* do modelo *Scrum*;
- ***Sprint*** representa a classe *Sprint* do modelo *Scrum*;
- ***SprintBacklog*** representa a classe *SprintBacklog* do modelo *Scrum*.
- ***BacklogItem*** representa a classe *BacklogItem* do modelo *Scrum*.
- ***Review*** representa a classe *Review* do modelo *Scrum*.
- ***Retrospective*** representa a classe *Retrospective* do modelo *Scrum*.
- ***Planning*** representa a classe *Planning* do modelo *Scrum*.
- ***ProductBacklog*** representa a classe *ProductBacklog* do modelo *Scrum*.

- ***ScrumTeam*** representa a classe *Team* do modelo *Scrum*.

4.6.3 Ontologia para o Modelo de Domínio FDD

Será construída uma ontologia para representar o modelo FDD. Esta ontologia através de classes e relacionamentos constituirá na formalização de um processo FDD. Para uma amostra da solução proposta, foi desenvolvida uma ontologia que possui algumas das características do modelo, de acordo com a Figura 30.

Figura 30 - Ontologia FDD.



Fonte: Desenvolvido pelo autor.

As classes da ontologia FDD desenvolvidas como parte do domínio são:

- ***FDDProduct:*** representa o conceito *Product* do modelo FDD;
- ***FDDTeam***: representa o conceito *Team* do modelo;
- ***ConstructionCicle***: representa a classe *ConstructionCicle* do modelo;
- ***Progress***: representa a classe *Progress* do modelo;
- ***Feature***: representa a classe *Feature* do modelo;
- ***Funcionality:*** representa a classe *Funcionality* do modelo;
- ***FDDStakeholder***: representa a classe *Stakeholder* do modelo.

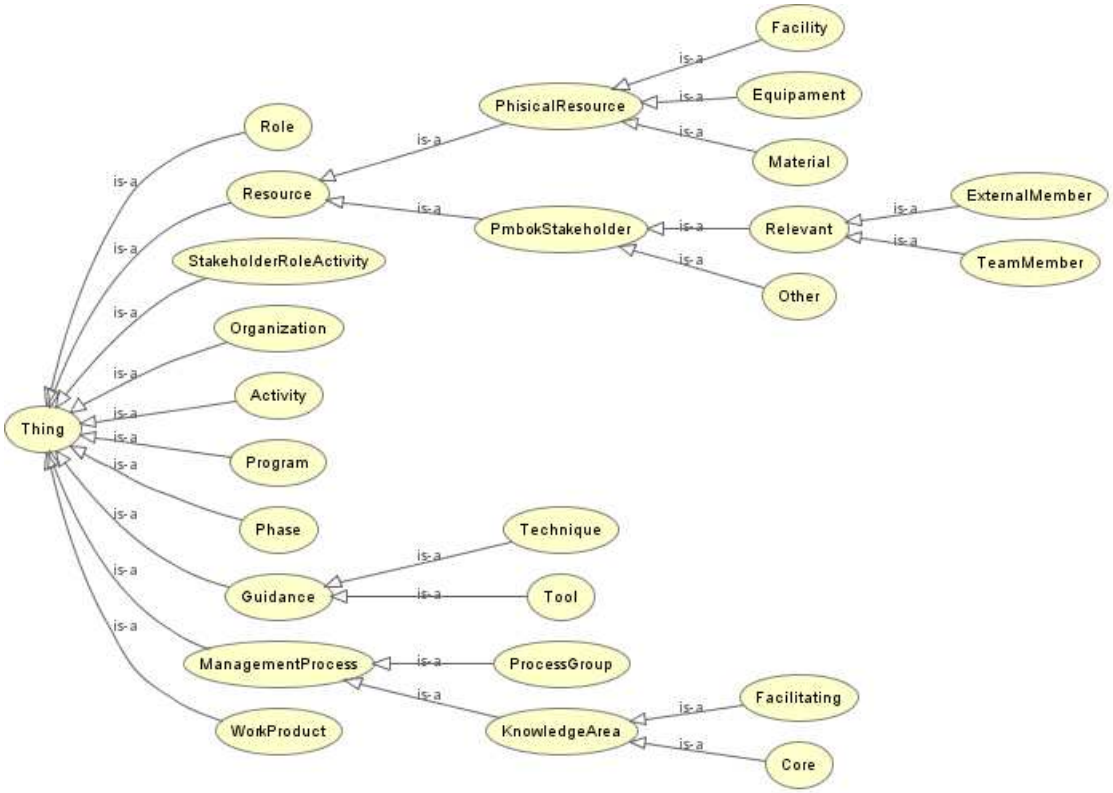
- ***DevelopmentPlan***: representa a classe *DevelopmentPlan* do modelo.
- ***ComprehensiveModel***: representa a classe *ComprehensiveModel* do modelo.
- ***Requisites***: representa a classe *Requisites* do modelo.

4.6.4 Representação do Modelo de Domínio PMBOK Através de uma Ontologia

Diante das características modeladas no domínio PMBOK, também foi criada uma ontologia para representar o método de gerência. Esta ontologia tem o mesmo objetivo das ontologias que representam as metodologias. Os conceitos se unem com os conceitos das outras ontologias proporcionando o objetivo esperado pelo trabalho: um gerenciamento de processo de desenvolvimento de *software* com metodologias ágeis.

Em síntese a ontologia possui características do modelo proposto pelo PMBOK. A Figura 31 representa a Ontologia do PMBOK.

Figura 31 - Ontologia PMBOK.



Fonte: Desenvolvido pelo autor.

As classes da ontologia PMBOK desenvolvidas como parte do domínio são:

- ***Organization:*** Representa a classe Organization do modelo PMBOK;
- ***Program:*** Representa a classe *Program* do modelo PMBOK;
- ***Project:*** Representa a classe *Project* do modelo PMBOK;
- ***Phase:*** Representa a classe *Phase* do modelo PMBOK;

- ***Role:*** Representa a classe *Role* do modelo PMBOK;
- ***Resource:*** Representa a classe *Resource* do modelo PMBOK;
- ***StakeholderRoleActivity:*** Representa a classe *StakeholderRoleActivity* do modelo PMBOK;
- ***Activity:*** Representa a classe *Activity* do modelo PMBOK;
- ***Guidance:*** Representa a classe *Guidance* do modelo PMBOK;
- ***ManagementProcess:*** Representa a classe *ManagementProcess* do modelo PMBOK;
- ***WorkProduct:*** Representa a classe *WorkProduct* do modelo PMBOK;
- ***PhisicalResource:*** Representa a classe *PhisicalResource* do modelo PMBOK;
- ***PmbokStakeholder:*** Representa a classe *Stakeholder* do modelo PMBOK;
- ***Technique:*** Representa a classe *Technique* do modelo PMBOK;
- ***Tool:*** Representa a classe *Tool* do modelo PMBOK;
- ***ProcessGroup:*** Representa a classe *ProcessGroup* do modelo PMBOK;
- ***KnowledgeArea:*** Representa a classe *KnowledgeArea* do modelo PMBOK;
- ***Facilitating:*** Representa a classe *Facilitating* do modelo PMBOK;
- ***Core:*** Representa a classe *Core* do modelo PMBOK;
- ***Other:*** Representa a classe *Other* do modelo PMBOK;
- ***Relevant:*** Representa a classe *Relevant* do modelo PMBOK;
- ***Material:*** Representa a classe *Material* do modelo PMBOK;
- ***Equipament:*** Representa a classe *Equipament* do modelo PMBOK;
- ***Facility:*** Representa a classe *Facility* do modelo PMBOK;
- ***ExternalMember:*** Representa a classe *ExternalMember* do modelo PMBOK;
- ***TeamMember:*** Representa a classe *TeamMember* do modelo PMBOK.

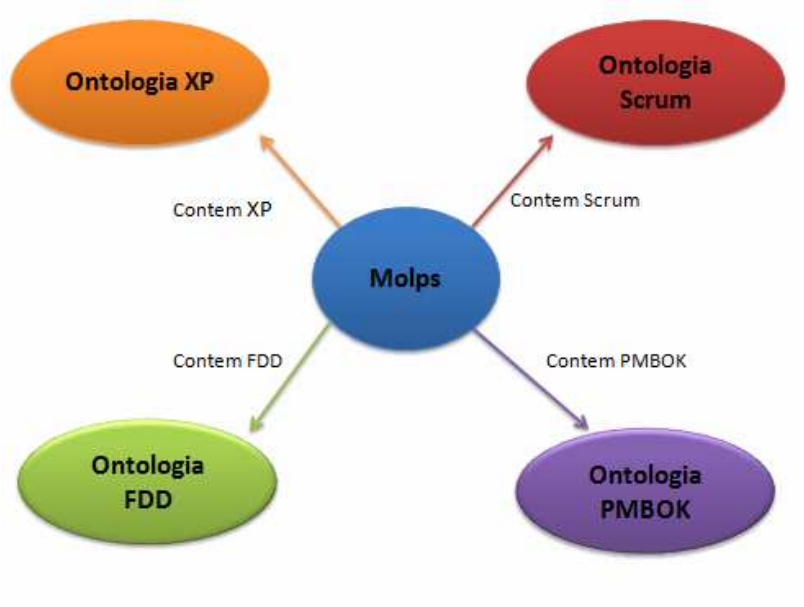
4.6.5 Molps: Representação da LPS através de Ontologia

A ontologia criada foi alinhada ao metamodelo desenvolvido por Mallmann (2011), onde o autor construiu seu metamodelo contemplando um conjunto de características e conceitos das três metodologias ágeis de desenvolvimento de *software* e do modelo de gerência de projetos PMBOK, que permitiu a criação de uma aplicação para melhor gerenciar projetos de *software* utilizando metodologias ágeis. Este metamodelo foi definido em banco de dados. Desta forma este trabalho apresenta uma forma diferente de representar estas características, onde foi definida uma linha de produto através de ontologia.

A Molps é uma ontologia que representa uma LPS para o domínio estudado. A LPS se define através do conjunto de características e conceitos das três metodologias ágeis de desenvolvimento de *software*, e do modelo de gerência de projeto PMBOK. A ontologia representa a arquitetura da LPS fornecendo uma arquitetura genérica, que possui as

características obrigatórias e ainda as características que definem os pontos de variações do processo de desenvolvimento de *software*. A Figura 32 representa a estrutura da Molps.

Figura 32 - Estrutura da Molps.



Fonte: Desenvolvido pelo autor.

As ontologias XP, *Scrum*, FDD e PMBOK compõem todas as classes e conceitos que definem o domínio de conhecimento da Molps. A Molps representa um metadomínio, pois une todas as informações das quatro ontologias.

Cada ontologia possui suas características, porém, elas podem não ser exclusivas e existir em outra ontologia. A Molps trata as classes comuns de forma que elas sejam descritas por apenas uma classe, que representa o mesmo conceito, mas de ontologias diferentes. Basicamente possui classes que são comuns às quatro ontologias e classes que são diferentes e que agregam mais informações ao domínio.

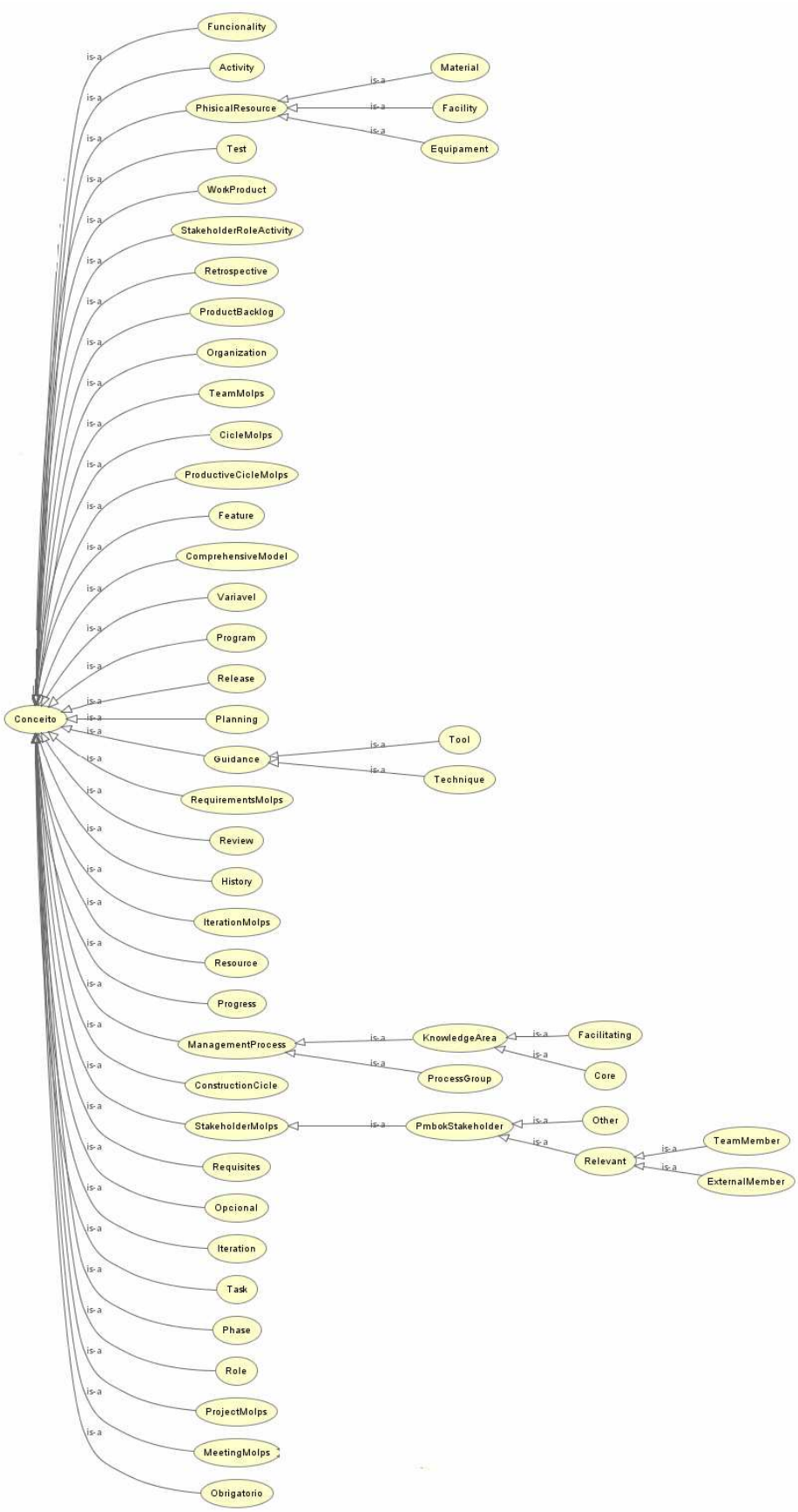
Os relacionamentos entre as classes da Molps especificam um processo de desenvolvimento de *software*. Abordando conceitos das três metodologias e do modelo de gerência de projeto, obtém-se um processo de desenvolvimento mais completo chegando ao objetivo estabelecido na engenharia de *software*: a qualidade no produto final.

As classes possuem propriedades que estabelecem as relações construídas entre as classes ou relações entre indivíduos. A Molps é constituída por propriedades que formam todos possíveis relacionamentos promovendo sentido semântico.

A Molps importa as quatro ontologias criadas, assim promovendo a união dos conceitos de cada domínio. Após importar foram criadas as classes que determinam as características comuns entre os conceitos. Estas classes possuem como parte da nomenclatura a palavra Molps. A ontologia possui alem das classes comuns, as classes que representam características diferentes agregando mais informação ao domínio. A união acontece através de propriedades ligando todas as classes importadas às classes Molps.

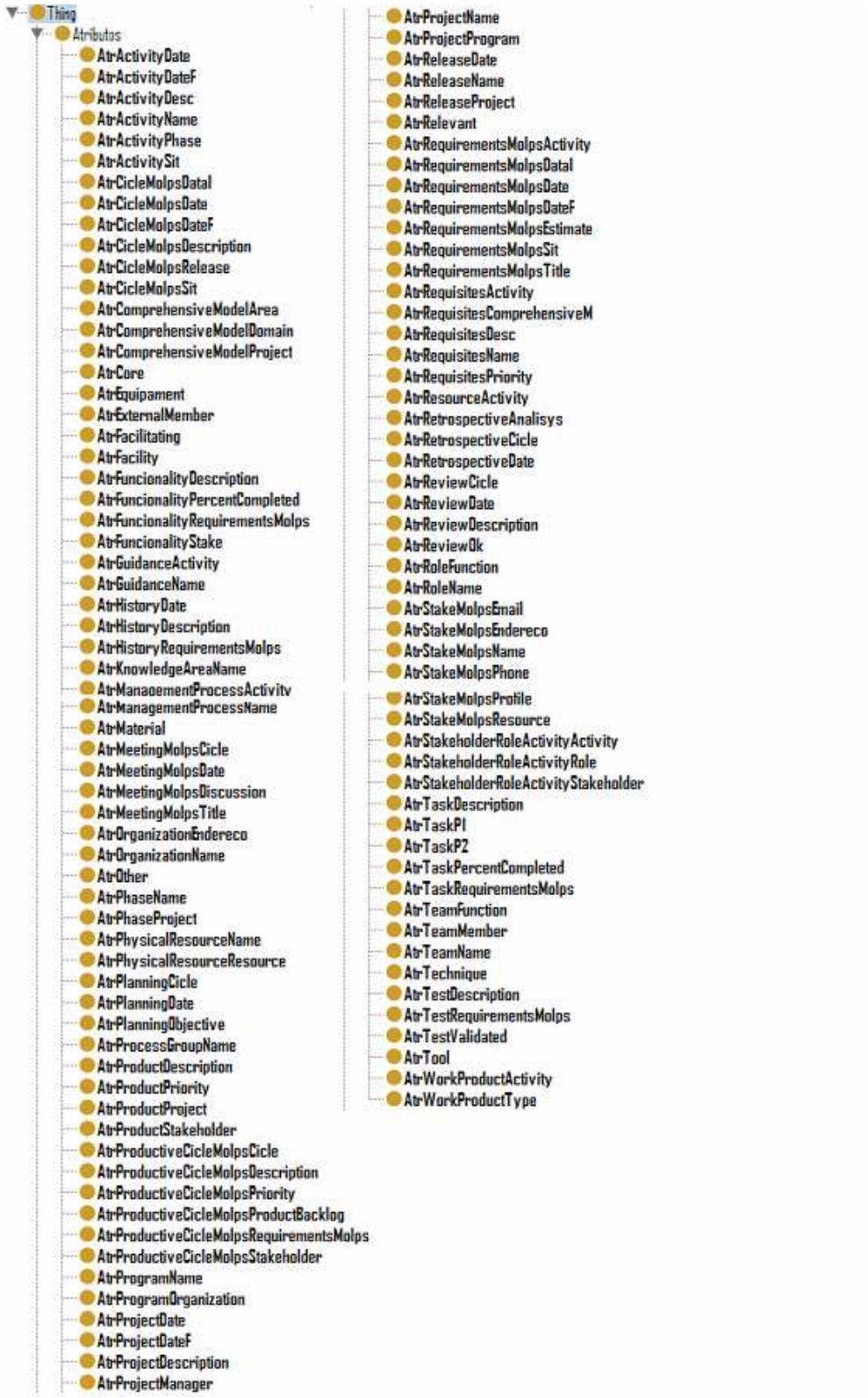
As classes criadas na ontologia possuem atributos para serem armazenadas as informações do processo de desenvolvimento de *software*. Nesta ontologia os atributos foram criados na forma de classes contendo na nomenclatura o nome da classe que ele representa e o nome do atributo. As informações são armazenadas em forma de indivíduos, cada individuo possui um relacionamento com a classe atributo que ele representa. A hierarquia da ontologia foi dividida em duas classes principais, a classe Atributos e a classe Conceito. A classe Conceito possui todas as classes que definem a arquitetura da LPS, ou seja, as características obrigatórias e variáveis para criar o processo de desenvolvimento de *software*. Já a classe Atributos possui todas as classes que representam os atributos das classes Conceito. Para ilustrar a ontologia foram criadas duas figuras, ver Figuras 33 e 34, que representam toda a Molps, mas por motivo de tamanho foram divididas em duas partes, a parte Atributos e a parte Conceito. A parte atributos está sendo representada através da ferramenta utilizada e a parte do Conceito está representada através do OWL Viz.

Figura 33 - Representação das classes Conceito da Ontologia Molps.



Fonte: Desenvolvido pelo autor.

Figura 34 - Representação das classes Atributos da Ontologia Molps.



Fonte: Desenvolvido pelo autor.

Sistematicamente a ontologia Molps tem por objetivo representar através de seus conceitos a unificação semântica das classes vindas dos métodos ágeis e do PMBOK. Para se

estabelecer esta declaração semântica, selecionaram-se as classes que tinham uma mesma conotação entre as diferentes metodologias de desenvolvimento de *software*. Após, cada grupo de classes que possuem a mesma conotação foi referenciado por uma classe única, que constitui a ontologia Molps. Esta única classe provê a unificação semântica, ou seja, o mesmo sentido é dado para duas classes provenientes de metodologias ágeis diferentes e com nomenclaturas diferentes.

As descrições das classes da Molps são:

- Classe *CicleMolps*: esta classe une conceitos como o *ConstructionCicle* do domínio FDD, o conceito de *Iteration* da XP e o conceito de *Sprint* da *Scrum*;
- Classe *MeetingMolps*: a classe une os conceitos de *Meeting* da XP e *DailyMeeting* da *Scrum;*
- Classe *ProjectMolps*: possui conceitos das classes *Project* do XP e PMBOK e *Product* do FDD e *Scrum;*
- Classe *TeamMolps*: une conceitos das classes *Team* das metodologias XP, FDD e *Scrum;*
- Classe *StakeholderMolps*: une os conceitos das classes *Stakeholder* dos conceitos XP, FDD, PMBOK e *Scrum*;
- Classe *RequirementsMolps*: une os conceitos das classes dos conceitos *Story* da XP, *Task* do PMBOK e *BacklogItem* do *Scrum*;
- Classe *ProductiveCicleMolps*: une conceitos das classes *SprintBacklog* da metodologia *Scrum.*

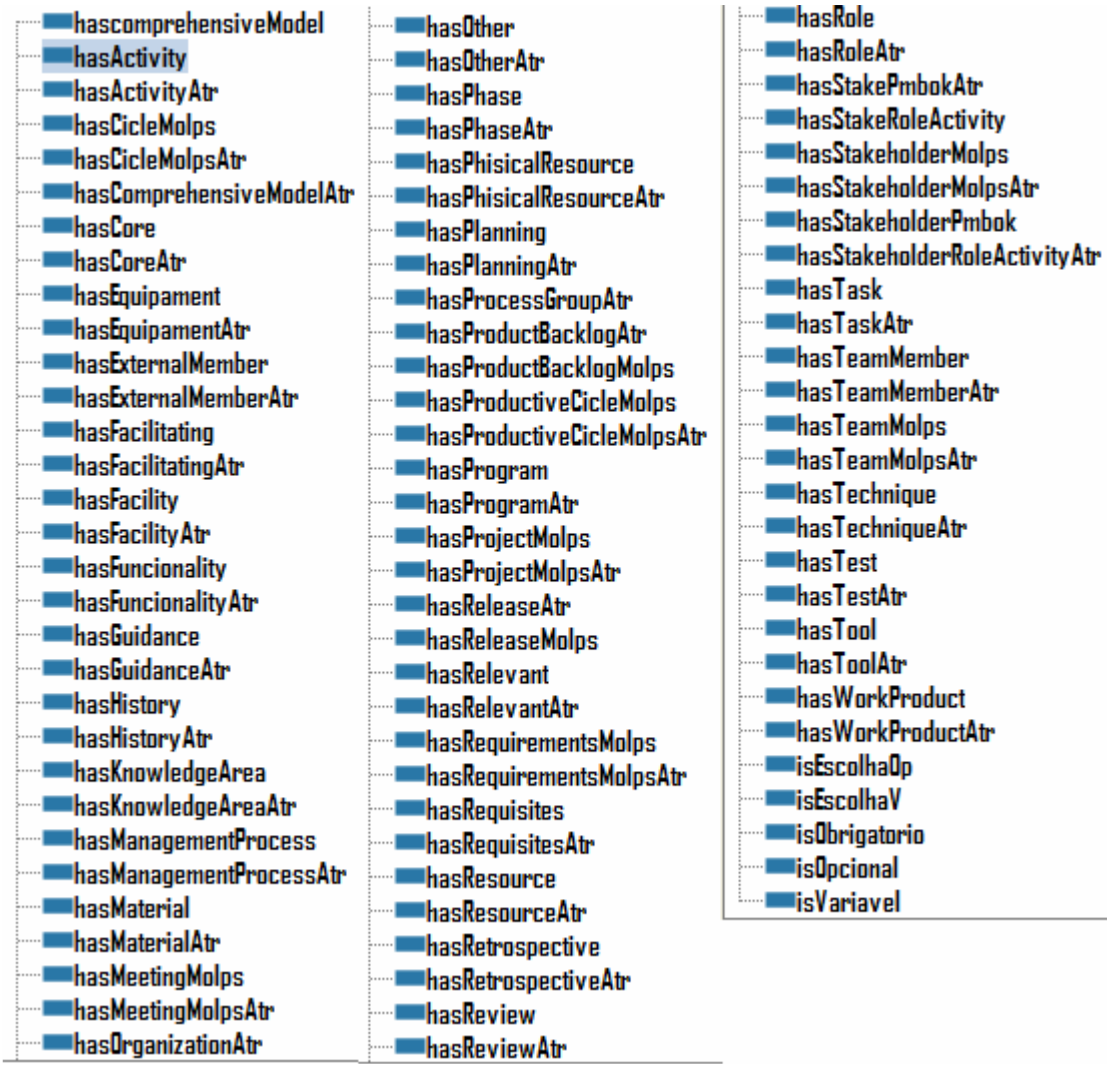
Foram excluídas todas as classes que são representadas pelas classes de nomenclatura Molps, ficando apenas as classes que representam características diferentes e que agregam informações ao domínio.

Notam-se na Figura 33 três classes chamadas Obrigatório, Opcional e Variável. Estas classes definem na arquitetura da LPS as características obrigatórias, que formam a arquitetura genérica do processo de desenvolvimento de *software* e as opcionais e variáveis que definem a variabilidade do processo. As classes da ontologia que se definem como obrigatórias opcionais ou variáveis estão relacionadas através de propriedades com as três classes.

4.6.5.1 Relacionamento entre as Classes

As propriedades de uma ontologia possuem a função de relacionar classes, promovendo a semântica do domínio. Cada classe da ontologia desenvolvida possui relacionamento/propriedade com outra classe ou individuo. A Figura 35 apresenta todos os relacionamentos da ontologia. Todos esses relacionamentos são transitivos.

Figura 35 - Relacionamentos da Molps.



Fonte: Desenvolvido pelo autor

A Figura 35 representa todos os relacionamentos entre as classes Conceito e as classes Atributos. Uma propriedade possui *domain* (*domínio*) e *range* (*escopo*). As propriedades conectam indivíduos ou classes de um *domain* (*domínio*) a indivíduos ou classes de um *range* (*escopo*). Abaixo são listados apenas os relacionamentos das classes Conceito, pois o sentido semântico da ontologia é formado por estas classes e relacionamentos.

- *hasStakeholderPmbok*: relaciona a classe *StakeholderMolps* que representa o *domain*, com a classe *PmbokStakeholder* que representa o *Range.*
- *hasTeamMolps*: relaciona a classe *ProjectMolps* que representa o *domain*, com a classe *TeamMolps* que representa o *Range.*
- *hascomprehensiveModel*: relaciona a classe *ProjectMolps* que representa o *domain*, com a classe *ComprehensiveModel* que representa o *Range.*
- *hasActivity*: relaciona a classe *Phase* que representa o *domain*, com a classe *Activity* que representa o *Range.*

- *hasCicleMolps*: relaciona a classe *Release* que representa o *domain*, com a classe *CicleMolps* que representa o *Range.*
- *hasHistory*: relaciona a classe *RequirementsMolps* que representa o *domain*, com a classe *History* que representa o *Range.*
- *hasMeetingMolps*: relaciona a classe *CicleMolps* que representa o *domain*, com a classe *MeetingMolps* que representa o *Range.*
- *hasPhase*: relaciona a classe *ProjectMolps* que representa o *domain*, com a classe *Phase* que representa o *Range.*
- *hasPlanning*: relaciona a classe *CicleMolps* que representa o *domain*, com a classe *Planning* que representa o *Range.*
- *hasProductBacklogMolps*: relaciona a classe *ProjectMolps* que representa o *domain*, com a classe *ProductBacklog* que representa o *Range.*
- *hasProductiveCicleMolps*: relaciona as classes *RequirementsMolps, ProductBacklog e CicleMolps* que representam o *domain*, com a classe *ProductiveCicleMolps* que representa o *Range.*
- *hasProjectMolps*: relaciona a classe *CicleMolps* que representa o *domain*, com a classe *Planning* que representa o *Range.*
- *hasReleaseMolps:* relaciona a classe *ProjectMolps* que representa o *domain*, com a classe *Release* que representa o *Range.*
- *hasRetrospective:* relaciona a classe *CicleMolps* que representa o *domain*, com a classe *Retrospective* que representa o *Range*.
- *hasReview:* relaciona a classe *CicleMolps* que representa o *domain*, com a classe *Review* que representa o *Range*.
- *hasRole:* relaciona a classe *StakeholderRoleActivity* que representa o *domain*, com a classe *Role* que representa o *Range*.
- *hasStakeholderMolps:* relaciona a classe *TeamMolps* que representa o *domain*, com a classe *StakeholderMolps* que representa o *Range*.
- *hasTask:* relaciona a classe *RequirementsMolps* que representa o *domain*, com a classe *Task* que representa o *Range*.
- *hasCore:* relaciona a classe *KnowledgeArea* que representa o *domain*, com a classe *Core* que representa o *Range*.
- *hasTest:* relaciona a classe *RequirementsMolps* que representa o *domain*, com a classe *Test* que representa o *Range*.
- *hasEquipament:* relaciona a classe *PhisicalResource* que representa o *domain*, com a classe *Equipament* que representa o *Range.*
- *hasExternalMember:* relaciona a classe *Relevant* que representa o *domain*, com a classe *ExternalMember* que representa o *Range*.
- *hasFacilitating:* relaciona a classe *knowledgeArea* que representa o *domain*, com a classe *Facilitating* que representa o *Range*.
- *hasFacility:* relaciona a classe *PhisicalResource* que representa o *domain*, com a classe *Facility* que representa o *Range.*

- *hasFuncionality:* relaciona a classe *RequirementsMolps* que representa o *domain*, com a classe *Funcionality* que representa o *Range*.
- *hasGuidance:* relaciona a classe *Activity* que representa o *domain*, com a classe *Guidance* que representa o *Range*.
- *hasKnowledgeArea:* relaciona a classe *ManagementProcess* que representa o *domain*, com a classe *KnowledgeArea* que representa o *Range*.
- *hasManagementProcess:* relaciona a classe *Activity* que representa o *domain*, com a classe *ManagementProcess* que representa o *Range*.
- *hasMaterial:* relaciona a classe *PhisicalResource* que representa o *domain*, com a classe *Material* que representa o *Range*.
- *hasOther:* relaciona a classe *PmbokStakeholder* que representa o *domain*, com a classe *Other* que representa o *Range*.
- *hasPhisicalResource:* relaciona a classe *Resource* que representa o *domain*, com a classe *PhisicalResource* que representa o *Range*.
- *hasProgram:* relaciona a classe *Organization* que representa o *domain*, com a classe *Program* que representa o *Range*.
- *hasRelevant:* relaciona a classe *PmbokStakeholder* que representa o *domain*, com a classe *Relevant* que representa o *Range*.
- *hasRequirementsMolps:* relaciona a classe *Activity* que representa o *domain*, com a classe *RequirementsMolps* que representa o *Range*.
- *hasRequisites:* relaciona a classe *Activity e ComprehensiveModel* que representam o *domain*, com a classe *Requisites* que representa o *Range*.
- *hasResource:* relaciona a classe *Activity* que representa o *domain*, com a classe *Resource* que representa o *Range*.
- *hasStakeRoleActivity:* relaciona a classe *Activity* que representa o *domain*, com a classe *StakeholderRoleActivity* que representa o *Range*.
- *hasTeamMember:* relaciona a classe *Relevant* que representa o *domain*, com a classe *TeamMember* que representa o *Range*.
- *hasTechnique:* relaciona a classe *Guidance* que representa o *domain*, com a classe *Technique* que representa o *Range*.
- *hasTool:* relaciona a classe *Guidance* que representa o *domain*, com a classe *Tool* que representa o *Range*.
- *hasWorkProduct:* relaciona a classe *Activity* que representa o *domain*, com a classe *WorkProduct* que representa o *Range*.

4.6.5.2 Criação de Indivíduos

Na ontologia criada os indivíduos retratam as informações de cada atributo. Toda e qualquer informação que será armazenada na ontologia, será descrita através de um indivíduo. Estes indivíduos são instâncias das classes Atributos, onde cada indivíduo criado define a instância da classe atributo que ele representar.

4.6.6 Variabilidade da Molps

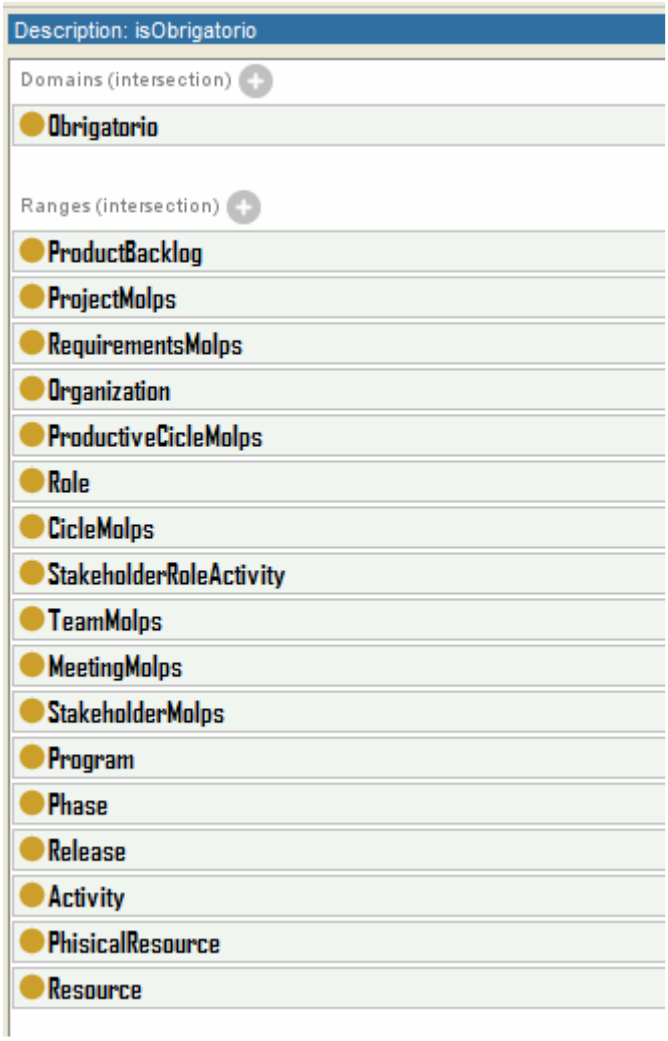
Como já mencionado na seção 4.6.5, algumas classes encontram-se com o mesmo significado nas ontologias XP, *Scrum*, FDD e PMBOK. Estas similaridades são tratadas pela Molps como um tipo de variabilidade. Algumas classes são iguais, porem são de origens diferentes, onde cada uma provem de uma ontologia diferente. Como já definido anteriormente, a Molps possui uma classe que representa essas similaridades. Esta classe representa as classes das ontologias que possuírem esta característica.

Foi realizado um estudo em literaturas e no trabalho realizado por Mallmann (2011) sobre quais seriam os pontos de variações do produto da LPS, assim pôde-se ter uma noção do que seria flexível na arquitetura da LPS. Além das características comuns entre as metodologias ágeis estudadas e o PMBOK, foram definidas como obrigatórias as características que agregavam informação e qualidade no processo de desenvolvimento de *software*.

4.6.6.1 Definição das Características Obrigatórias

As características obrigatórias estarão em todos os produtos derivados da linha de produto, não podendo ser modificadas. A definição das características obrigatórias foram definidas através de uma propriedade chamada de *isObrigatorio* que relaciona com a classe Obrigatório todas as classes que se designam obrigatórias na arquitetura da LPS, conforme Figura 36.

Figura 36 - Classes Obrigatórias.



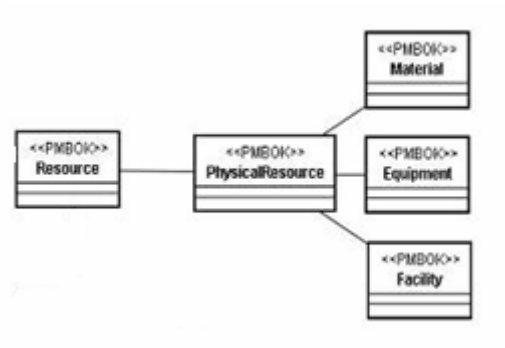
Fonte: Desenvolvido pelo autor.

De acordo com a Figura 36, a propriedade *isObrigatorio* possui como classe *domain* a classe *Obrigatorio*, e possui como classes r*anges* as classes: *ProductBacklog*, *ProjectMolps*, *RequirementsMolps*, *Organization*, *ProductiveCicleMolps*, *Role*, *CicleMolps*, *StakeholderRoleActivity*, *TeamMolps*, *MeetingMolps*, *StakeholderMolps*, *Program*, *Phase*, *Release*, *Activity*, *Resource* e *PhisicalResource*. Todas estas classes são obrigatórias na arquitetura LPS, estando estas presente nos processos de desenvolvimento de *software* que serão criados.

4.6.6.2 Definição das Características Opcionais

Na arquitetura da LPS existem características que possuem a definição de opcional. Estas características devem aparecer como opção de alguma classe obrigatória ou de alguma classe opcional. Um exemplo é ilustrado na Figura 37.

Figura 37 - Exemplo de classes opcionais.

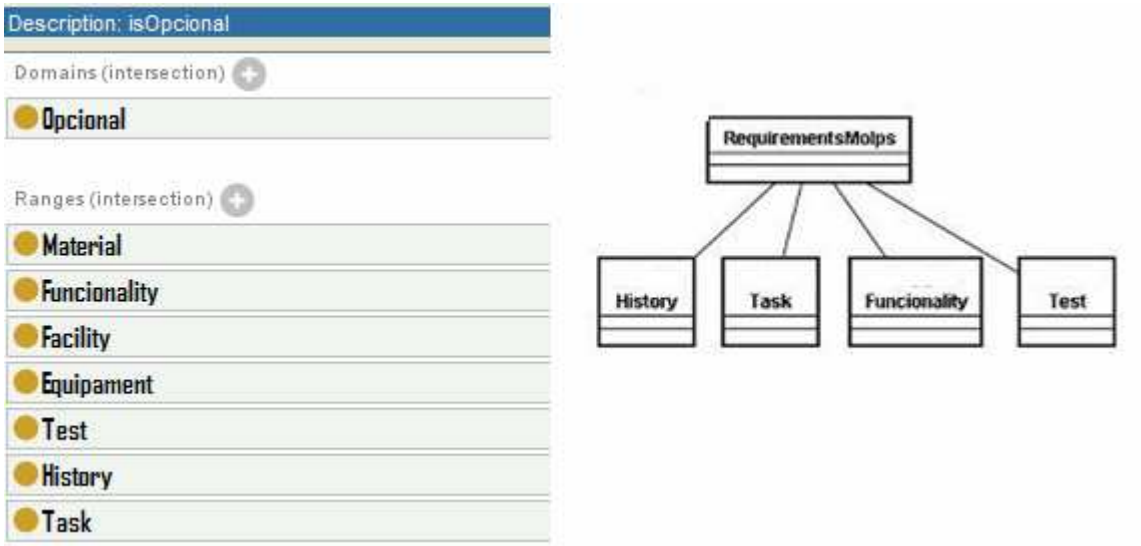


Fonte: Desenvolvido pelo autor

A Figura 37 apresenta as classes *Resource* e *PhisicalResource* que possuem característica obrigatória na arquitetura da LPS. As classes, *Facility*, *Equipament* e *Material* são subclasses de *PhisicalResource* e são opcionais na arquitetura, portanto pelo menos uma deve ser escolhida.

As classes opcionais da arquitetura da LPS estão relacionadas pela propriedade *isOpcional* com a classe *Opcional*. Pela Figura 38 podem observadas as classes opcionais.

Figura 38 - Classes Opcionais.



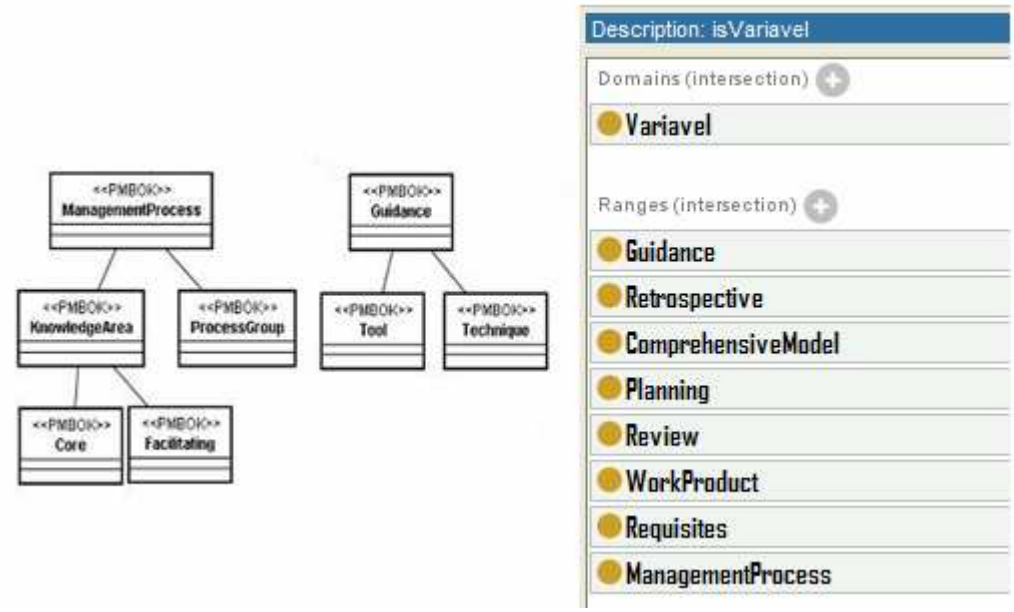
Fonte: Desenvolvido pelo autor.

Nota-se na Figura 38 que a classe *RequirementsMolps* de característica obrigatória, possui quatro classes ligadas a ela de característica opcional que devem ser escolhida ao menos uma para a criação do processo de desenvolvimento de *software*.

4.6.6.3 Definição das Características Variáveis

Na arquitetura da LPS características variáveis também fazem parte dos pontos de variações do produto da LPS, portanto foram definidas na ontologia criada. As classes que se definem como variáveis possuem o relacionamento *isVariavel* entre a classe *Variavel* e as classes: *Guidance*, *Retrospective*, *ComprehnsiveModel*, *Planning*, *Review*, *WorkProduct*, *Requisites*, *ManagementProcess*. As classes *Guidance* e *ManagementProces* possuem subclasses que também são variáveis. Na Figura 39 podem ser observadas as classes e subclasses variáveis.

Figura 39 - Classes Variáveis.



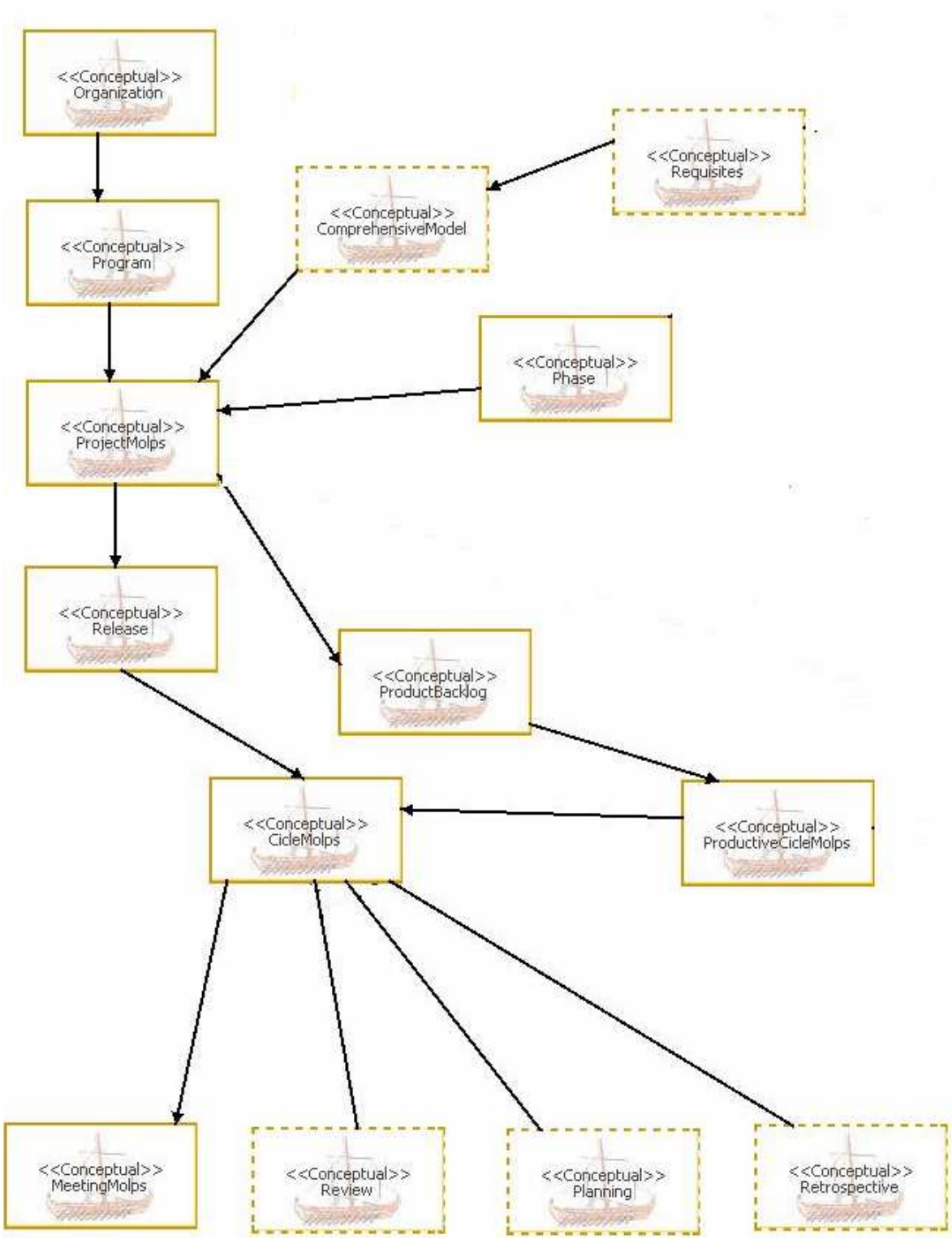
Fonte: Desenvolvido pelo autor.

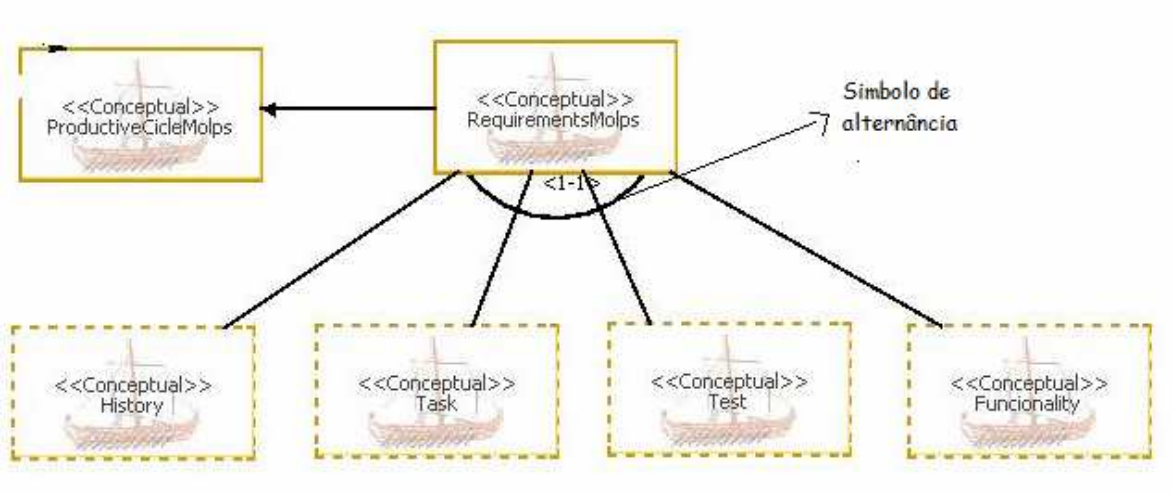
Conforme a Figura 39 nota-se as classes *ranges* que se determinam variáveis. Como as classes *Guidance* e *ManagementProcess* possuem subclasses que também são variáveis, não foi necessário acrescentá-las como *ranges* da propriedade *isVariavel*, pois se infere que elas fazem parte.

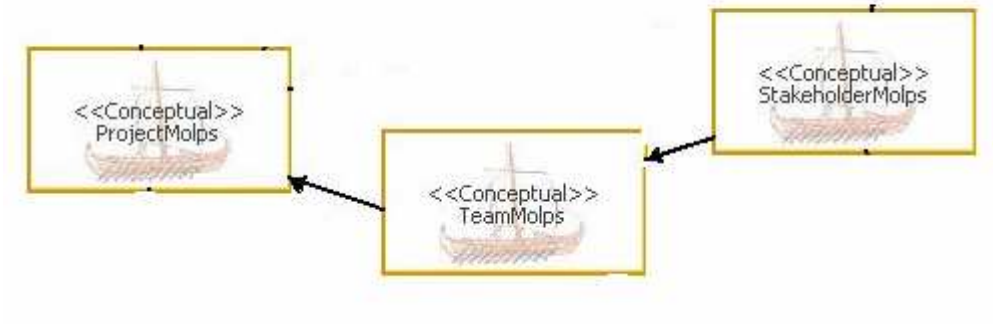
4.6.6.4 Modelo de *Features*

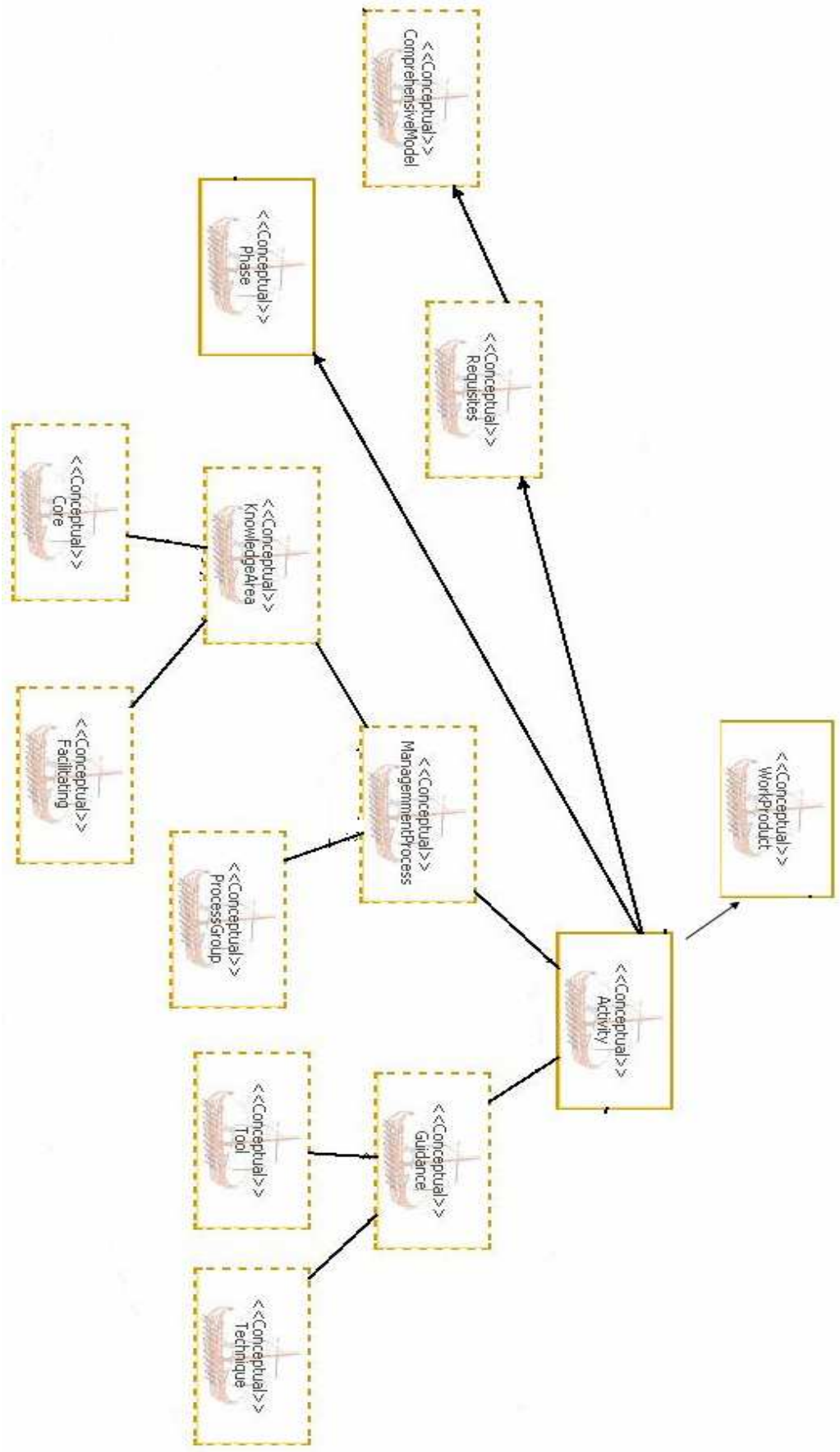
Foi representado um diagrama de *features* para serem observadas as possíveis variações na Molps. Para construção do diagrama foi utilizada a ferramenta *Odyssey Domain* SDE versão 1.6.0. No *Odyssey*, as *features* são utilizadas como um modelo semântico que serve para que o desenvolvedor tenha uma noção geral sobre as principais características do domínio antes de explorar uma série de diagramas detalhados sobre os diferentes aspectos modelados. As caixas pontilhadas representam características opcionais e alternativas, enquanto as caixas de traço continuo representam obrigatoriedade. A Figura 40 representa o diagrama de *features* da Molps. Foi separada em blocos a figura para melhor visualização.

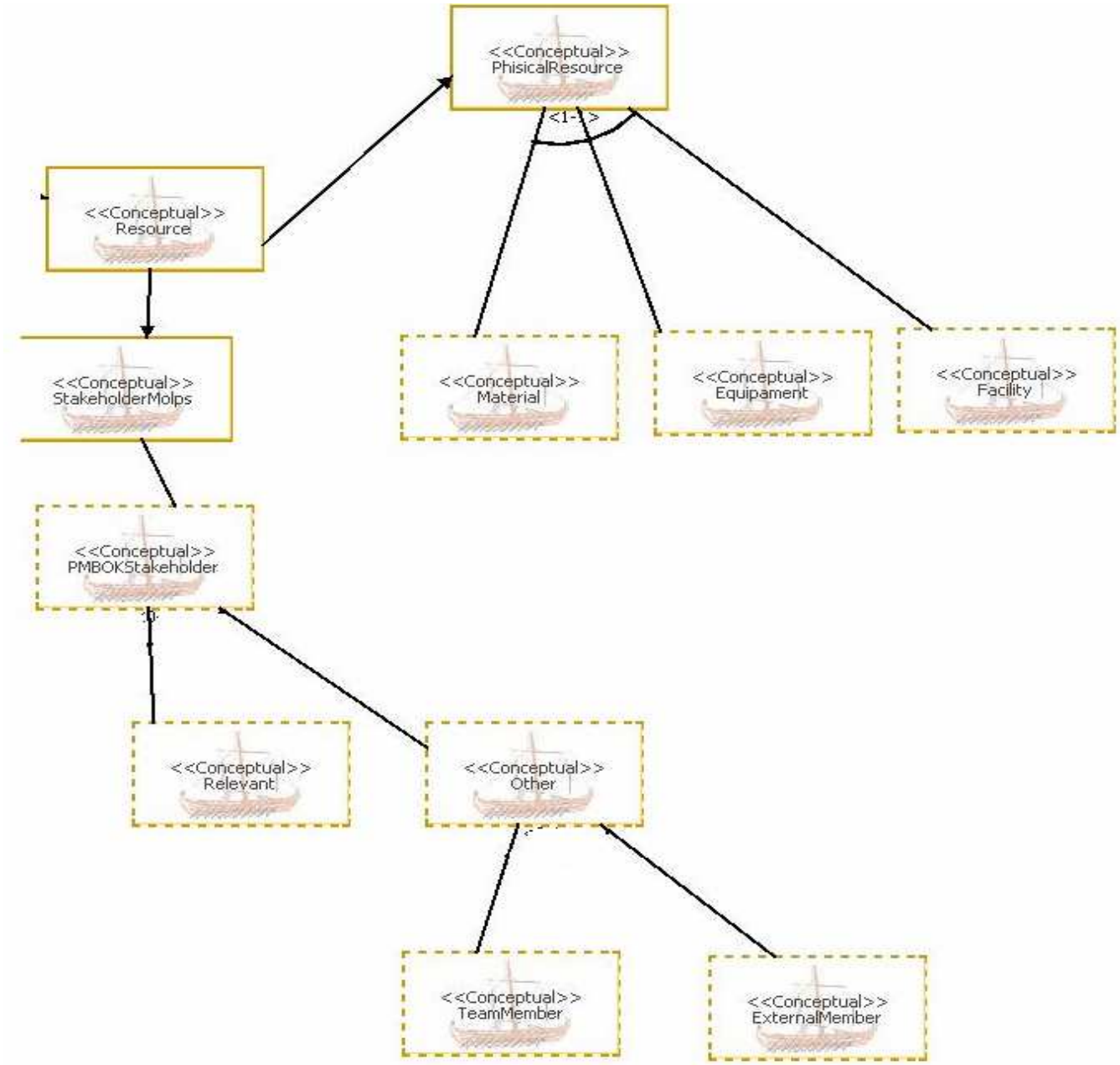
Figura 40 - Diagrama de *features* Molps.




<<Conceptual>>
ProductiveCicleMolps
<<Conceptual>>
RequirementsMolps
Símbolo de
alternância
<1-1>
<<Conceptual>>
History
<<Conceptual>>
Task
<<Conceptual>>
Test
<<Conceptual>>
Funcionality


<<Conceptual>>
ProjectMolps
<<Conceptual>>
StakeholderMolps
<<Conceptual>>
TeamMolps


<<Conceptual>>
ComprehensiveModel
<<Conceptual>>
Phase
<<Conceptual>>
Requisites
<<Conceptual>>
Core
<<Conceptual>>
KnowledgeArea
<<Conceptual>>
Facilitating
<<Conceptual>>
ManagemmentProcess
<<Conceptual>>
WorkProduct
<<Conceptual>>
ProcessGroup
<<Conceptual>>
Activity
<<Conceptual>>
Tool
<<Conceptual>>
Guidance
<<Conceptual>>
Technique



Fonte: Desenvolvido pelo autor.

Nesta ferramenta a variabilidade é tratada de forma que as características são representadas como obrigatórias, alternativas e opcionais. O símbolo de alternância representa que alguma classe precisa ser escolhida. Já a classe opcional pode ser escolhida como também não utilizada. Na Molps a variabilidade é tratada de forma diferente da ferramenta, as características são representadas como obrigatórias, opcional que representa a alternância na ferramenta e variáveis que representa as opcionais na ferramenta.

4.6.7 Objetivos Funcionais da Molps

A ontologia tem como objetivo gerar um produto. Este produto será um modelo de gerência de processo de desenvolvimento ágil reunindo as características da XP, *Scrum*, FDD e PMBOK. Gerando este processo de desenvolvimento, a pessoa responsável pelo projeto tem todo o planejamento que será desenvolvido no decorrer do projeto. Basicamente a ontologia permite:

- **Criar o processo de desenvolvimento de *software*:** Para gerar o produto, o usuário escolhe as características do produto. São realizadas consultas por agentes de *software* na ontologia definindo as características obrigatórias, variáveis e opcionais.
- **Armazenar todas as informações sobre o processo criado:** Tendo o produto gerado, todas as informações sobre o projeto, relativas às características escolhidas para o processo de desenvolvimento serão armazenadas na ontologia;
- **Permitir agentes de *software* consultar as informações armazenadas a qualquer momento:** O usuário solicita informações a serem consultadas, e com estas informações os agentes realizam as consultas. Sendo assim, o projeto estando no começo, meio ou fim, a pessoa responsável obtém as informações de andamento do projeto.

Os trabalhos correlatos que não utilizam ontologia como base de conhecimento, permitem consultas sobre o desenvolvimento do projeto. A ontologia tem um ganho em relação a estes trabalhos no processo de consultas, pois existem linguagens de consultas para ontologias com a capacidade de fazer inferências. A inferência torna informações implícitas em explícitas em uma consulta.

Nesta ontologia é possível aumentar o domínio de conhecimento. Para aumentar o domínio, metodologias ágeis diferentes podem ser inseridas na ontologia, bem como outro método de gerência de projeto de *software*. As formas de inserção podem ser de duas maneiras: importando para a Molps alguma ontologia que representa um método ágil ou criando novas classes direto na ontologia Molps. Nestas duas maneiras, bastam apenas relacionar as novas características com as já existentes na ontologia.

4.7 MODELAGEM DOS AGENTES DE *SOFTWARE*

Neste trabalho foram definidos três agentes de *software*: o Gerenciador de Interface, o Gerenciador da Ontologia e o Agente de Projeto.

- Gerenciador de Interface: este agente possui a função de montar a interface responsável pela construção do processo de desenvolvimento de *software* e a interface para editar algum processo existente. Ele recebe a requisição do usuário para criar um novo processo ou editar. Para a criação de um novo processo ele requisita ao agente Gerenciador da Ontologia todas as características que definem a arquitetura da LPS, para assim alimentar a interface. Para editar algum processo ele requisita ao Gerenciador da Ontologia que consulte na ontologia as características do processo escolhido para ser

editado. Após a criação ou editar algum processo o agente disponibiliza ao usuário o armazenamento de informações sobre o processo criado e/ou editado.

- Gerenciador da Ontologia: este agente possui a função de consultar a ontologia, ele é chamado pelo agente Gerenciador de Interface para realizar as consultas requisitadas na ontologia.
- Agente de Projeto: este agente é responsável pelas consultas na ontologia em relação às informações armazenadas. Este agente gera gráficos, relatórios e consultas sobre informações do projeto. Para as consultas e gráficos o agente alimenta interfaces para a visualização do usuário.

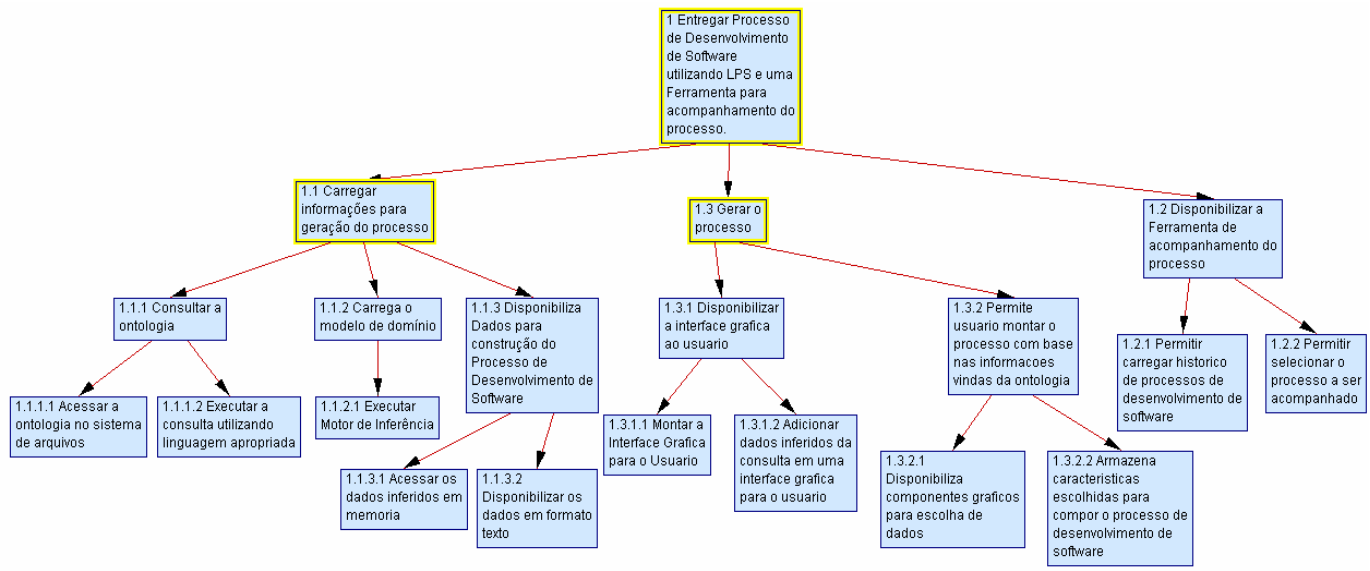
A engenharia de *software* para projetos orientados a agentes disponibiliza várias metodologias que permitem modelar os diversos componentes e artefatos de *software* envolvidos neste paradigma. A metodologia MaSE é utilizada no projeto e construção dos agentes deste trabalho. Esta metodologia permite descrever desde uma especificação inicial do sistema até a sua implementação através da interligação entre seus diagramas. O principal foco da MaSE é guiar o projetista através do ciclo de vida do *software* de uma especificação natural para um sistema de agente implementado. MaSE é mais focada, levando o desenvolvimento em etapas bem definidas, desde o levantamento de requisitos até a fase de implantação. Esta metodologia é apoiada através da ferramenta *AgentToolIII*, que disponibiliza um ambiente gráfico ao usuário.

Os três agentes de *software* modelados fazem o uso da ontologia como base de conhecimento para eles realizarem as tarefas designadas. As próximas seções apresentam os diagramas referentes aos agentes.

4.7.1 Diagrama de Objetivos

A identificação dos objetivos de um sistema orientado a agentes visa conhecer as especificações do sistema e transformá-las em uma estrutura de conjunto de objetivos. Conforme a metodologia é necessário conhecer qual é o objetivo principal do sistema, para então expandir em subobjetivos os quais permitirão atingir o primeiro. A Figura 41 apresenta o diagrama de objetivos.

Figura 41 - Diagrama de Objetivos dos agentes.



Fonte: Desenvolvido pelo autor.

Os objetivos descritos reúnem as atividades necessárias para obter o objetivo do sistema, que é entregar o processo de desenvolvimento de *software* utilizando LPS e uma ferramenta para acompanhamento do processo. A primeira etapa é carregar as informações para gerar o processo, ver objetivo 1.1, acessando a ontologia e consultando as informações que estão representados pelos objetivos 1.1.1, 1.1.1.1 e 1.1.1.2. Após consultar, o motor de inferência deve ser executado para carregar o modelo do domínio em memória, onde estão representados pelos objetivos 1.1.2 e 1.1.2.1. Por fim os dados são disponibilizados, ver objetivos 1.1.3, 1.1.3.1, 1.1.3.2, para que as informações sejam carregadas para a geração do processo.

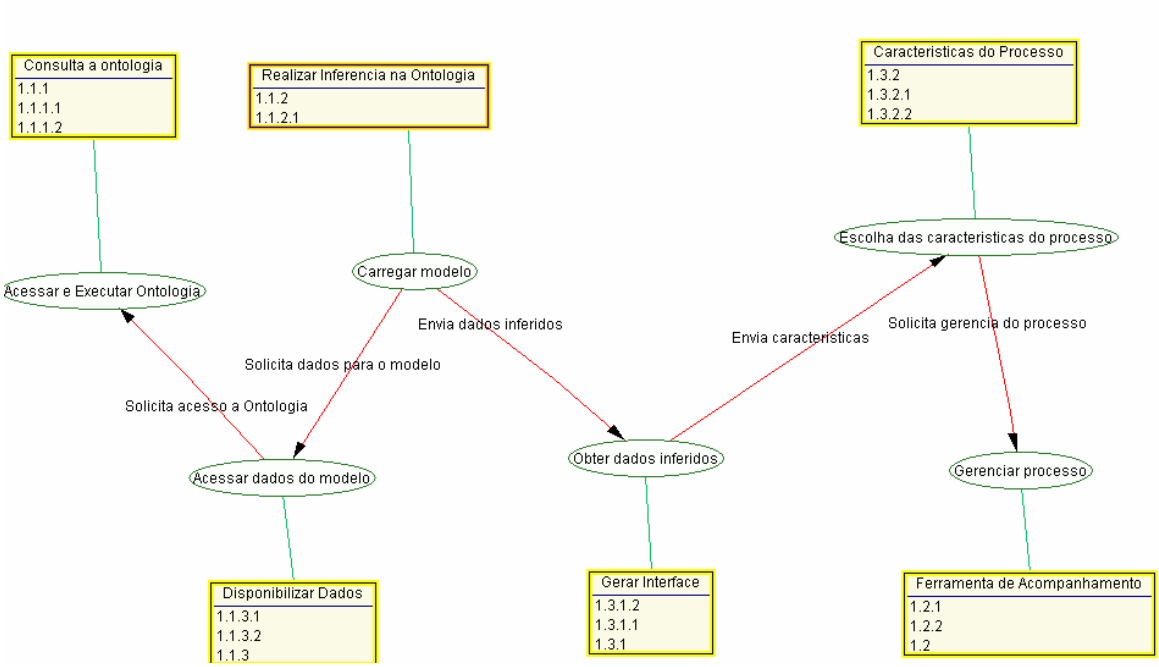
Depois de carregar as informações o processo é gerado. São disponibilizados os dados através de uma interface para realizar a criação do processo. Estas etapas estão descritas nos objetivos 1.3.1, 1.3.1.1 e 1.3.1.2. Para a criação do processo uma interface com dados para escolha e armazenamento de características é fornecida. Estas etapas estão descritas nos objetivos 1.3.2, 1.3.2.1, 1.3.2.2.

Para finalizar, uma ferramenta de acompanhamento do processo é disponibilizada. Para esta ferramenta é permitido carregar um processo criado e selecionar um acompanhamento. Esta ultima etapa esta representada pelos objetivos 1.2.1 e 1.2.2.

4.7.2 Diagrama de Papéis

Esta etapa permite definir através de um diagrama os papéis, tarefas e protocolos de comunicação que o sistema contém. No diagrama, ver Figura 42, todos os objetivos estabelecidos inicialmente devem ser vinculados a um papel.

Figura 42 - Diagrama de Papéis.



Fonte: Desenvolvido pelo autor.

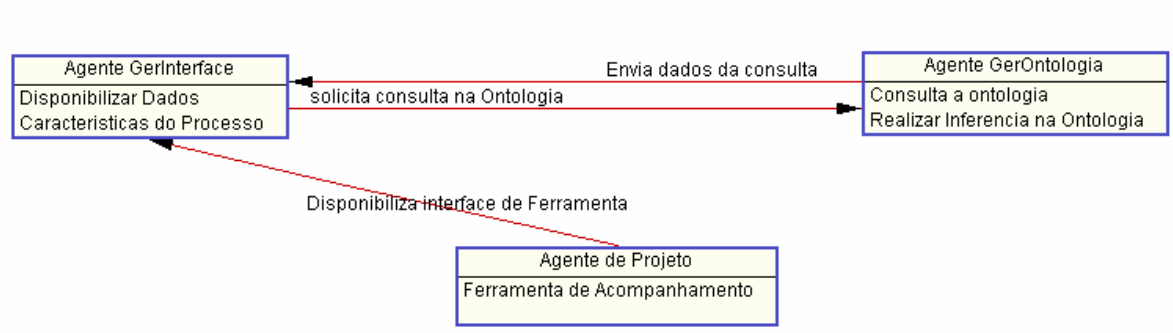
Este diagrama representa as *roles* que atendem os objetivos descritos no sistema. A *role* que consulta a ontologia, a *role* que disponibiliza dados e a *role* que realiza a inferência na ontologia são responsáveis pelos objetivos de consultar a ontologia e carregar o modelo do domínio em memória para posteriormente gerar o processo. As *roles* gerar interface e características do processo representam a etapa de criar um processo. E por último a *role* ferramenta de acompanhamento permite acompanhar um processo criado.

Os protocolos entre as *tasks* das *roles* representam o diálogo entre as etapas que os agentes atenderão. Os protocolos de solicitar acesso e solicitar dados representam a consulta na ontologia. Os protocolos de enviar dados e enviar características representam a criação do processo. E o protocolo que solicita gerencia permite que a ferramenta de acompanhamento do processo seja executada.

4.7.3 Diagrama de Agentes

Nesta fase as classes dos agentes são identificadas pelos papéis. O resultado desta fase é um diagrama das classes dos agentes, como mostra a Figura 43, na qual é descrito as classes dos agentes e a comunicação entre eles.

Figura 43 - Diagrama de agentes.



Fonte: Desenvolvido pelo autor.

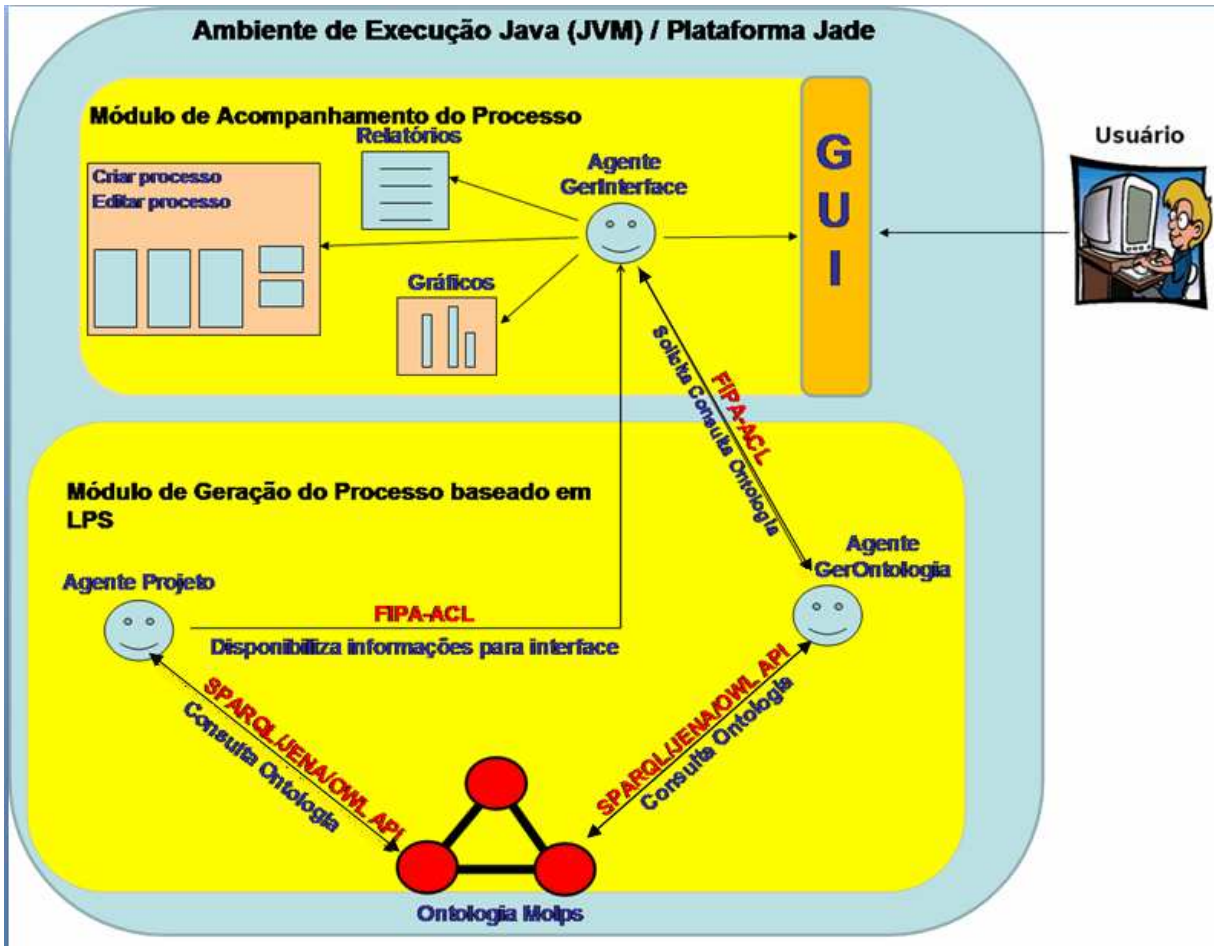
Neste diagrama os agentes são definidos com seus papéis. O agente *GerInterface* é responsável por disponibilizar através de uma interface os dados para criar o processo. Este agente envia uma mensagem para o gerenciador da ontologia, solicitando as características da arquitetura da LPS. O agente *GerOntologia* então consulta essas características realizando inferência na ontologia, e em seguida envia os dados da consulta para o *GerInterface*. Já o agente de projeto disponibiliza uma ferramenta de acompanhamento através de uma interface.

4.8 VISÃO GERAL DO SISTEMA

O sistema desenvolvido tem por objetivo construir um processo de desenvolvimento de *software* baseado em uma linha de produto de *software* definida através de ontologia. Para tal, disponibilizou-se uma camada baseada em ontologias e agentes de *software*. Para a criação de um processo e para acompanhar o processo, utiliza-se de uma ontologia que representa o esquema de banco de dados, bem como uma segmentação de informações do modelo de domínio que define a arquitetura da LPS. Através da ontologia busca-se preservar a igualdade semântica entre os diferentes domínios de informações. Este procedimento permite que o sistema construa um processo de desenvolvimento de *software* adquirindo o conhecimento necessário para buscar informações que precisar na ontologia.

A arquitetura do ambiente foi concebida para atender os objetivos da ontologia criada. Assim pode ser notado que o sistema permite não apenas a criação do processo, mas também uma ferramenta de acompanhamento para posteriores controles e consultas. A Figura 44 apresenta esta arquitetura.

Figura 44 - Arquitetura do ambiente.



Fonte: Desenvolvido pelo autor.

Esta estrutura compreende:

- A ontologia: define a arquitetura da LPS mapeando semanticamente todo o domínio da metodologia ágeis XP, *Scrum*, FDD e do método de gerencia PMBOK.
- Os agentes de *software*: os agentes têm por objetivo atender as funcionalidades de criar, editar e acompanhar um processo de desenvolvimento de *software*, responsáveis pelo sistema. A característica autônoma lhes confere a capacidade necessária para realizarem suas tarefas. Através do protocolo FIPA-ACL, eles se comunicam e interagem no ambiente para alcançar seus objetivos.
- Interfaces: abstraem todo o funcionamento do sistema para o usuário. São capazes de promover a comunicação e interação entre usuário e sistema. Na arquitetura através de interfaces o usuário consegue criar, editar e acompanhar um processo de desenvolvimento de *software*.

5 DOMÍNIO E CENÁRIO DE APLICAÇÃO

Para aplicação dos testes, o cenário escolhido foi uma instituição de ensino localizada na cidade de Três de Maio, Rio Grande do Sul. A instituição chamada de Sociedade Educacional Três de Maio (Setrem, www.setrem.com.br) possui um centro de tecnologia da informação (TI) onde os serviços na área de tecnologia são prestados. Qualquer demanda na área de tecnologia e suporte aos computadores, redes e desenvolvimento de *software*, são realizados no centro de TI.

Neste centro de TI existe uma equipe e papéis assumidos por cada integrante. Esta equipe possui uma pessoa responsável que encaminha todas as atividades a serem realizadas. Estas atividades se diferem em tipos como suporte a *hardware*, redes e *software*. Dependendo o tipo da atividade, podem ser realizadas correções, customizações e também pode ser classificada como um novo projeto a ser desenvolvido. No momento a instituição possui um portal onde todas as informações sobre a Setrem são encontradas. Neste portal existe um sistema de gerenciamento de ensino, chamado de Educar Web, em que alunos e professores possuem acesso. O centro de TI gerencia este portal juntamente com o Educar Web, onde correções e customizações são apoiadas pela equipe.

Para cenário aplicação foram analisadas todas as atividade e oportunidades em que os testes poderiam ser aplicados. Esta equipe não segue nenhuma metodologia de processo de desenvolvimento de *software* em suas demandas, então se observou uma oportunidade de aplicar o sistema desenvolvido e assim a equipe experimentar como seria seguir um método de processo de desenvolvimento de *software*. A cada demanda algumas etapas são gerenciadas pelo responsável da equipe, em que as atividades e o desenvolvimento da demanda são acompanhados. Como a equipe não segue uma metodologia, o gerenciamento do projeto é realizado de forma manual e de acordo com as necessidades da equipe.

6 DETALHES DA IMPLEMENTAÇÃO

Este capítulo aborda os detalhes da implementação da ontologia Molps e dos agentes de *software*. São apresentados os detalhes de implementação da ontologia que define a LPS, o ambiente que o usuário cria o processo de desenvolvimento, armazena e consulta as informações.

Neste capítulo todas as explicações sobre armazenamento e consultas ditas como feitas na ontologia, na verdade são realizadas no modelo em memória que representa a ontologia, mas no decorrer das explicações será descrito apenas como ontologia e não modelo em memória da ontologia.

6.1 ONTOLOGIA QUE DEFINE A LPS

Nesta seção será descrita a forma que foi utilizada para criar a ontologia que define a LPS. Será descrita a linguagem para criar ontologias escolhida, bem como a ferramenta de auxílio utilizada.

Para a construção da ontologia foi utilizada a ferramenta *Protégé 4.0*. No *Protégé* as ontologias podem ser editadas, vistas, e também fornece uma Java API para acessar as ontologias. A ferramenta possui uma arquitetura *plug-in* e permite a extensão da ferramenta com novas funcionalidades e capacidades.

Ontologias são utilizadas para capturar conhecimento sobre um domínio de interesse. Uma ontologia descreve os conceitos de um domínio e também as relações que existem entre esses conceitos. As diversas linguagens para construção de ontologias fornecem diferentes funcionalidades. O padrão mais recente de linguagens para ontologias é o OWL, desenvolvido no âmbito do W3C. Neste trabalho a linguagem escolhida foi o OWL. Esta linguagem está classificada em três sub-linguagens, OWL-*Lite*, OWL-DL e OWL-*Full*. A sublinguagem de OWL escolhida foi OWL-DL, pois é mais expressiva que a *OWL-Lite* e baseia-se em lógica descritiva, um fragmento de Lógica de Primeira Ordem, passível, portanto de raciocínio automático. É possível assim computar automaticamente a hierarquia de classes e verificar inconsistências na ontologia. Com a linguagem foi possível criar todas as classes da Molps, das ontologias que representam o domínio da Molps e as propriedades que relacionam as classes.

Após definir todas as classes e propriedades, através da ferramenta gerou-se a ontologia em formato OWL, sob o nome de *ontolps.owl*. O Apêndice A apresenta o código fonte da ontologia.

6.2 FERRAMENTA E LINGUAGENS UTILIZADAS PARA DESENVOLVER O AMBIENTE

Nesta seção são abordadas as linguagens utilizadas para o processo de construção do sistema, bem como a ferramenta utilizada para criação do mesmo.

6.2.1 IDE NetBeans

A ferramenta utilizada para construir o sistema foi a IDE *NetBeans*. Foi escolhida por ser um ambiente de desenvolvimento integrado de código aberto e gratuito que permite linguagens Java, C, C++, PHP entre outras. Sua característica multiplataforma permite criar aplicativos em plataformas *Windows*, *Linux*, *Solaris* e *MacOS*. Além dessas características auxilia programadores a escrever, compilar e debugar aplicações. Permite aplicações visuais *Swing* que é uma API (Interface de Programação de Aplicativos) Java para interfaces gráficas.

6.2.2 Java

A linguagem Java foi utilizada para escrever o sistema, pois é simples e de fácil aprendizado. Baseada no paradigma da Orientação a Objetos permite a modularização das aplicações, reuso e manutenção simples do código. Possui independência de plataforma, é distribuída com um vasto conjunto de bibliotecas ou APIs.

6.2.3 Jena e SPARQL

Jena é um *framework* em linguagem Java que originou no núcleo de pesquisa em Web Semântica. É um projeto de código aberto e gratuito, utilizado para desenvolvimento de aplicações que usem ontologias. Inclui suporte para manipulação de RDF, RDFS, OWL e DAML+OIL.

Neste trabalho a motivação partiu da possibilidade de criar, carregar, manipular e salvar ontologias em diversos formatos, entre eles OWL. As ontologias podem estar disponibilizadas localmente ou em um endereço na Web. Ontologias criadas podem ser salvas tanto em arquivos em disco como em bancos de dados, facilitando a persistência de ontologias extensas. Recursos para recuperação de ontologias embutidas em uma ontologia também estão disponíveis.

Através do Jena é possível fazer inferência sobre a ontologia com uso de *reasoners*. Um *reasoner* é uma entidade dentro do Jena capaz de responder a determinados questionamentos e afirmações feitos em relação a uma ontologia.

SPARQL é uma linguagem para consultas em bases de dados semânticos RDF. De característica baixo nível, é possível fazer uma analogia entre SPARQL e a linguagem de consultas a bases relacionais SQL. Através do *framework* Jena é possível realizar consultas em bases RDF utilizando SPARQL e também escrever e ler triplas RDF em diversos formatos, como RDF/XML, N3 e N-Triples. É uma linguagem totalmente orientada a dados, que recupera informações contidas em arquivos RDF, possibilitando inclusive a opção de combinar dados de arquivos de diferentes fontes. SPARQL possui a capacidade de fazer inferências.

A linguagem SPARQL segue a mesma estrutura de construção de arquivos RDF e é construída sobre *Triple Pattern* (Triplas), ou seja: *Subject* (Sujeito), *Predicate* (Predicado) e *Object* (Objeto). Algumas das principais cláusulas da linguagem SPARQL são:

- SELECT [DISTINCT]
- FROM (opcional)

- WHERE (opcional)
- ORDER BY (opcional)
- UNION (opcional – funcionamento diferente do SQL

6.3 AGENTES DE *SOFTWARE*

Existem algumas ferramentas para desenvolver agentes e sistemas baseado em agentes. Essas ferramentas fornecem uma estrutura que facilita a construção como, por exemplo, classes próprias para trabalhar-se com os mesmos, fornecendo funcionalidades para desenvolver o agente desejado. Entre as plataformas recomendadas pela FIPA (*Foundation for Intelligent Physical Agents*) que consiste em uma organização internacional sem fins lucrativos destinada ao desenvolvimento de padrões de *software* voltados à utilização em sistemas baseados em agentes, encontram-se as populares JADE, *Tryllian's Agent Development Kit*, Zeus, entre outras.

Neste trabalho foi utilizada a ferramenta JADE e suas bibliotecas: *http, iiop, jade, jadeTools e Commons-codec-1.3*. JADE (*Java Agent Development Framework*) é uma plataforma implementada em Java que simplifica a criação de sistemas baseados em agentes através de um *middleware* que cumpre as especificações da FIPA, através de ferramentas gráficas que suporta a depuração e fases da implantação. A plataforma pode ser distribuída através de máquinas que não precisam compartilhar o mesmo sistema operacional, e a configuração pode ser controlada através de uma interface remota. Através da FIPA que estabelece a padronização de ambientes para desenvolvimento e execução dos agentes, os agentes podem se comunicar, um solicitando algum serviço ou executando alguma ação e o outro possuindo opções de rejeitar ou aceitar a solicitação.

Os agentes especificados no modelo proposto possuem papel fundamental no sistema, pois é através deles que a criação e gerenciamento do processo são realizados. O agente Gerenciador de Interface se comunica com o Gerenciador da Ontologia que realiza todas as consultas na ontologia solicitada pelo Gerenciado de Interface. O Agente de Projeto realiza o gerenciamento do projeto após o processo ser criado.

O agente Gerenciado de Interface possui um comportamento chamado *ComportamentoMontaInterface* que envia uma mensagem ao agente Gerenciador da Ontologia. Este comportamento monta a interface quando o usuário solicitar criar um processo de desenvolvimento de *software*. Os dados mostrados na tela são consultados na ontologia pelo agente Gerenciador da Ontologia. A Figura 45 mostra a mensagem enviada.

Figura 45 - Troca de mensagens entre os agentes de *software*.

```
public void action() {

ACLMessage msgGerOntologia = new ACLMessage(ACLMessage.REQUEST);
msgGerOntologia.addReceiver(new AID("GerOntologia" ,AID.ISLOCALNAME));
msgGerOntologia.setProtocol("consulte-ontologia");
  try {
              msgGerOntologia.setContentObject(myAgent);
              myAgent.send(msgGerOntologia);
          } catch (IOException ex) {
              Logger.getLogger(GerInterface.class.getName()).log(Level.SEVERE, null, ex);
          }
```

Fonte: Desenvolvido pelo autor.

Como pode ser observado na Figura 45, o agente Gerenciador de Interface envia uma mensagem do tipo *ACLMessage.REQUEST* ao Gerenciador da Ontologia, *GerOntologia* no código, requisitando que consulte a ontologia para retornar as características obrigatórias e pontos de variação da arquitetura da LPS para montar a interface de criação de processo de desenvolvimento de *software*.

O agente Gerenciador de Ontologia possui o comportamento chamado *ComportamentoConsultaOntologi*a que recebe a mensagem, ver Figura 46, e consulta a ontologia para saber quais são as características que representam a arquitetura genérica da LPS e quais são os pontos de variações que o processo poderá obter.

Figura 46 - Recebimento de mensagem do comportamento do agente de *software*.

```
if(msg.getPerformative()==ACLMessage.REQUEST){
    msg.getContent();
    if(msg.getProtocol().equalsIgnoreCase("consulte-ontologia")){
```

Fonte: Desenvolvido pelo autor.

O agente consulta na ontologia quais são as características que devem estar obrigatórias no processo a ser criado pelo usuário. Primeiramente é carregado o modelo em memória da ontologia através do *framework* Jena, conforme Figura 47. A linha *ModelFactory.createMemModelMaker()* cria a representação do modelo em memória. Após para recuperar a ontologia o método *read* é chamado. A linha *Reasoner reasoner = ReasonerRegistry.getOWLReasoner()* representa o *reasoner* onde é possível fazer inferência sobre a ontologia. Por fim a linha *infmodel = ModelFactory.createInfModel(reasoner, model)* constrói um modelo inferido anexando ao modelo o *reasoner*.

Figura 47 - Jena carregando a ontologia.

```
try {
      InputStream in =
        new FileInputStream(new File("C:\\Users\\Michele\\Desktop\\OntoLPS3\\ontolps.owl"));
      Model model = ModelFactory.createMemModelMaker().createModel();
      model.read(in, null);
      Reasoner reasoner = ReasonerRegistry.getOWLReasoner();
      reasoner = reasoner.bindSchema(model);
      infmodel = ModelFactory.createInfModel(reasoner, model);
   } catch (Exception e) {
      System.out.println(e.toString());
   }
```

Fonte: Desenvolvido pelo autor.

Após carregar o modelo em memória é realizada a consulta para saber quais são as características obrigatórias na LPS. A Figura 48 ilustra o código SPARQL com Jena que consulta quem são as classes na ontologia que possuem relacionamento *isObrigatorio.*

Figura 48 - Código SPARQL para consulta das classes obrigatórias.

```
String consulta2 =
" SELECT " + " ?subject,?property,?object " +
" WHERE " +
" (?subject ?property ?object ) " +

"  using " +
" ont for <http://www.ontolps.com.br/ontolps.owl#> " ;

QueryResults resultadoQuery2 = preparaQuery(consulta2);
for (Iterator iter = resultadoQuery2; iter.hasNext(); ) {
   ResultBinding res = (ResultBinding)iter.next();
   Resource subject = (Resource)res.get("subject");
   Resource Property = (Resource)res.get("property");
   Resource Objeto = (Resource)res.get("object");

if (subject.getLocalName().equalsIgnoreCase("isObrigatorio")) {
        StmtIterator stmtitersubject1 = subject.listProperties();
          while (stmtitersubject1.hasNext()) {
             com.hp.hpl.jena.rdf.model.Statement stmtMolpsTeam =
            (com.hp.hpl.jena.rdf.model.Statement)stmtitersubject1.nextStatement();
             Property prop = stmtMolpsTeam.getPredicate();
             if(prop.getLocalName().equalsIgnoreCase("range")) {
                obRange = (Resource)stmtMolpsTeam.getObject();
                if(!obRange.getLocalName().equalsIgnoreCase("Thing") &&
                   !obRange.getLocalName().equalsIgnoreCase("Conceito")){
                   obrigatorio.add(obRange.getLocalName());
```

Fonte: Desenvolvido pelo autor.

Nesta figura é realizado com SPARQL, código onde é realizado o *Select*, um *Select* total na ontologia, todos os sujeitos, propriedades e objetos. Após, com o Jena é feita uma busca em todos os sujeitos do tipo *isObrigatorio*, primeiro círculo. A informação procurada é uma tripla do tipo "*subject =isObrigatorio - property=range - object*= classes que se definem obrigatórias", então através das linhas *subject.getLocalName().equalsIgnoreCase("isObrigatorio"), prop.getLocalName().equalsIgnoreCase("range")* e *obRange = (Resource)stmtMolpsTeam.getObject(),* é possível encontrar o objeto da tripla que nada mais é do que uma classe que se define obrigatória. Através desta consulta são retornadas para o Gerenciador de Interface todas as características obrigatórias.

Para encontrar as características variáveis e opcionais é feito o mesmo tipo de consulta, mas com os *subjects isOpcional* e *isVariavel* nas triplas. Após é enviado de volta mensagens do tipo *ACLMessage.INFORM*, ver Figura 49, para o agente Gerenciador de Interface contendo as informações solicitadas.

Figura 49 - Mensagem enviada como resposta da requisição solicitada.

```
ACLMessage msgInterface = new ACLMessage(ACLMessage.INFORM);
 msgInterface.addReceiver(new AID("GerInterface", AID.ISLOCALNAME));
 msgInterface.setProtocol("obrigatorio");
 try {
         msgInterface.setContentObject(obrigatorio);
       } catch (Exception ex) {
         System.out.println(ex.toString());
         ex.printStackTrace();
       }
        myAgent.send(msgInterface);

 ACLMessage msgInterface2 = new ACLMessage(ACLMessage.INFORM);
 msgInterface2.addReceiver(new AID("GerInterface", AID.ISLOCALNAME));
 msgInterface2.setProtocol("variavel");
 try {
         msgInterface2.setContentObject(variavel);
       } catch (Exception ex) {
         System.out.println(ex.toString());
         ex.printStackTrace();
       }
        myAgent.send(msgInterface2);

 ACLMessage msgInterface3 = new ACLMessage(ACLMessage.INFORM);
 msgInterface3.addReceiver(new AID("GerInterface", AID.ISLOCALNAME));
 msgInterface3.setProtocol("opcional");
 try {
         msgInterface3.setContentObject(opcional);
       } catch (Exception ex) {
         System.out.println(ex.toString());
         ex.printStackTrace();
       }
        myAgent.send(msgInterface3);
```

Fonte: Desenvolvido pelo autor.

Depois de receber as informações o agente Gerenciador de Interface monta a interface definindo as características obrigatórias, opcionais e variáveis para que o usuário crie seu processo de desenvolvimento de *software*.

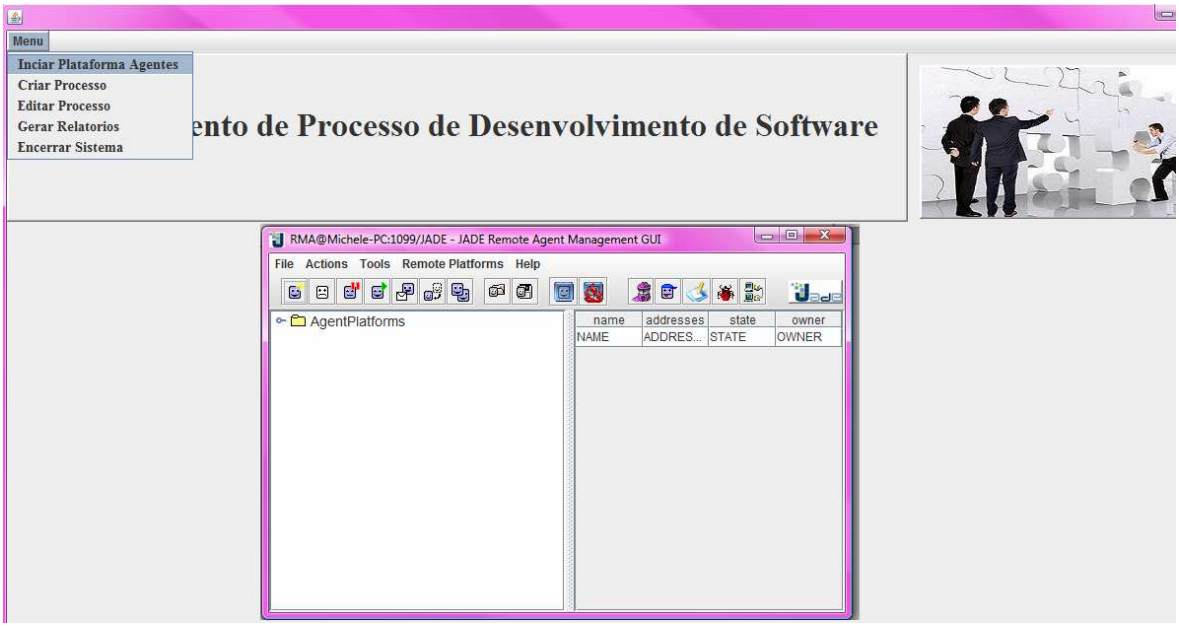
O agente de Projeto após o usuário criar e armazenar informações no processo referente ao projeto de *software* pode ser solicitado pelo usuário para realizar consultas sobre o andamento do projeto. Da mesma forma que o agente Gerenciador da Ontologia, o agente de Projeto possui um comportamento que pesquisa na ontologia as informações que o usuário solicitou através de uma interface.

6.4 CRIAÇÃO DO PROCESSO DE DESENVOLVIMENTO DE *SOFTWARE*

A criação do processo de desenvolvimento de *software* se da através da arquitetura da LPS definida pela ontologia. Esta arquitetura como já mencionado, possui as características genéricas que estarão presentes em todos os produtos derivados da LPS e as características que definem possíveis variações no processo. Estas características variáveis serão escolhidas pelo usuário de acordo com o processo de desenvolvimento que ele decidir criar.

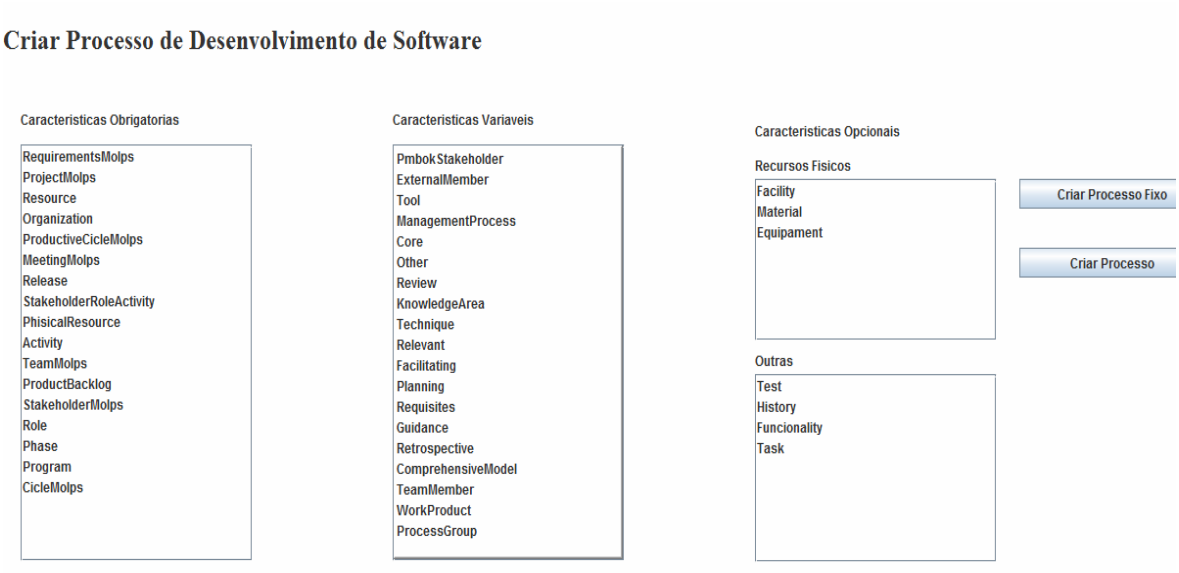
Primeiramente o usuário inicia a plataforma JADE para que os agentes de *software* sejam criados para executarem suas tarefas, ver Figura 50, e em seguida solicitar a criação do processo. Foi desenvolvida no sistema uma interface onde o usuário poderá criar seu processo de desenvolvimento de *software*. Esta interface é alimentada com dados vindos da ontologia, onde são feitas consultas para saber o que é obrigatório, opcional e variável para assim definir ao usuário quais serão as possíveis variações se assim ele desejar. Como já explicado na seção anterior, as consultas e quem alimenta a interface com os dados vindos da ontologia são agentes de *software*. Após o agente Gerenciador de Interface montar a interface para o usuário, ele poderá criar seu processo. A Figura 51 apresenta a interface de criação de processo de desenvolvimento de *software*.

Figura 50 - Inicialização da Plataforma JADE.



Fonte: Desenvolvido pelo autor.

Figura 51 - Interface de criação do processo de desenvolvimento de *software*.



Fonte: Desenvolvido pelo autor.

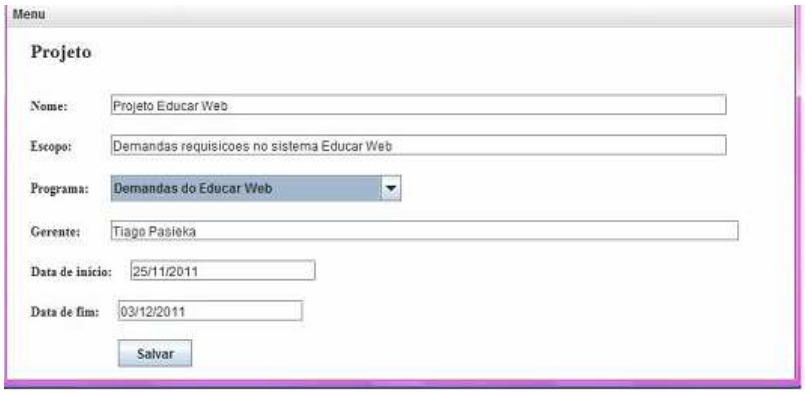
A Figura 51 possui três listas que definem o que é variável e obrigatório para o usuário escolher, e essas listas são alimentadas com as informações vindas da ontologia. Na lista do tipo variável o usuário poderá escolher como não escolher características. Nas listas opcionais ele precisa escolher ao menos uma de cada lista, e assim poderá criar seu processo com variabilidade. Além da opção de criar um processo com variabilidade, o usuário também poderá criar um processo com apenas a arquitetura genérica da LPS, escolhendo a opção “cria processo fixo”.

6.5 ARMAZENAMENTO DE INFORMAÇÕES

Assim que criado um processo de desenvolvimento de *software*, informações referentes ao projeto de *software* que este processo criado irá gerenciar poderão ser armazenadas na ontologia. Todas as informações serão armazenadas na forma de indivíduos que são instancias das classes que constituem o processo criado.

Cada característica do processo criado é uma classe na ontologia. Cada classe possui atributos para que informações sobre aquela classe sejam armazenadas. Como já explicado no capítulo do Modelo Proposto, os atributos foram definidos através de classes para que indivíduos/informações sejam instanciados. A Figura 52 apresenta uma das interfaces para armazenar as informações, e a parte que cria os indivíduos na ontologia.

Figura 52 - Interface de armazenar informações e código de criação dos indivíduos.




```
try{

OntClass nome =ontModel.getOntClass("http://www.ontolps.com.br/ontolps.owl#AtrProjectName");
OntClass desc =ontModel.getOntClass("http://www.ontolps.com.br/ontolps.owl#AtrProjectDescription");
OntClass ger =ontModel.getOntClass("http://www.ontolps.com.br/ontolps.owl#AtrProjectManager");
OntClass prog =ontModel.getOntClass("http://www.ontolps.com.br/ontolps.owl#AtrProjectProgram");
OntClass datal =ontModel.getOntClass("http://www.ontolps.com.br/ontolps.owl#AtrProjectDate");
OntClass dataF =ontModel.getOntClass("http://www.ontolps.com.br/ontolps.owl#AtrProjectDateF");
OntClass proj =ontModel.getOntClass("http://www.ontolps.com.br/ontolps.owl#ProjectMolps");
ObjectProperty ob= ontModel.getObjectProperty("http://www.ontolps.com.br/ontolps.owl#hasProjectMolpsAtr");


String valor1 = "Nome_".concat(jTextField1.getText().toString());
String valor2 = "Descricao_".concat(jTextField2.getText().toString());
String valor3 = jComboBox1.getSelectedItem().toString();
String valor4 = "Gerente_".concat(jTextField5.getText().toString());
FuncaoTrataEspaco fte= new FuncaoTrataEspaco();
valor1=fte.tiraEspaco(valor1, "_");
valor2=fte.tiraEspaco(valor2, "_");
valor3=fte.tiraEspaco(valor3, "_");
valor4=fte.tiraEspaco(valor4, "_");

String valor5 = "Inicio_".concat(jTextField4.getText().toString());
String valor6 = "Fim_".concat(jTextField3.getText().toString());
valor5 = valor5.replace("/", "_");
valor6 = valor6.replace("/", "_");

Individual ind1 = ontModel.createIndividual("http://www.ontolps.com.br/ontolps.owl#"+valor1 , nome );
Individual ind2 = ontModel.createIndividual("http://www.ontolps.com.br/ontolps.owl#"+valor2 , desc );
Individual ind3 = ontModel.createIndividual("http://www.ontolps.com.br/ontolps.owl#"+valor3 , prog );
Individual ind4 = ontModel.createIndividual("http://www.ontolps.com.br/ontolps.owl#"+valor4 , ger );
Individual ind5 = ontModel.createIndividual("http://www.ontolps.com.br/ontolps.owl#"+valor5 , datal );
Individual ind6 = ontModel.createIndividual("http://www.ontolps.com.br/ontolps.owl#"+valor6 , dataF );
Individual ind7 = ontModel.createIndividual("http://www.ontolps.com.br/ontolps.owl#"+valor1 , proj );
```

Fonte: Desenvolvido pelo autor.

Como pode ser observado o usuário dispõe de interfaces para armazenar as informações referentes às características escolhidas na criação do processo. A figura apresenta informações sobre o projeto, classe *ProjectMolps* da ontologia, e na parte do código é representada a criação dos indivíduos. Primeiramente são criadas variáveis do tipo Jena *OntClass* para obter as classes que serão instanciados os indivíduos. As classes são classes atributos da classe *ProjectMolps*, e essas classes atributos estão relacionadas com a classe *ProjectMolps* pela propriedade *hasProjectMolpsAtr*. As variáveis do tipo *OntClass* representam as classes atributos, e as informações descritas na interface são armazenadas em variáveis para serem utilizadas na criação dos indivíduos. Para criar indivíduos o Jena possui

o tipo *Individual* que permite criar o individuo com um valor que instancia determinada classe. Como pode ser observado na figura, o individuo é criado através do método *createIndividual* passando a URI com o valor e a classe que ele instancia. Após criar todos os indivíduos passando o valor e a classe que ele instancia, são descritos os relacionamentos entre eles, conforme Figura 53.

Figura 53 - Relacionamento entre indivíduos.

```
Iterator ef= dataF.listInstances();
 Individual indef = null;
 while(ef.hasNext()){
    indef = (Individual) ef.next();
}
   ob.addRange(indef);

     ind7.addProperty(ob, ind1);
```

Fonte: Desenvolvido pelo autor.

Na Figura 52 aparece uma linha de código em que a propriedade *hasProjectMolpsAtr* é atribuída a uma variável chamada "*ob*". Esta propriedade relaciona a classe *ProjectMolps* que é o *domain* do relacionamento, com as classes atributos que são os *ranges* do relacionamento, e a partir daí também irá relacionar os indivíduos instanciados. Isso é possível porque na implementação desenvolvida é criado um indivíduo do tipo *domain* que representa a instância da classe *ProjectMolps*, esse individuo terá o valor que representa o nome do projeto descrito pelo usuário na interface, e indivíduos *ranges* que representam instâncias das classes atributos de *ProjectMolps*. A Figura 53 ilustra uma parte do código que faz exatamente o relacionamento dos indivíduos, onde será atribuído ao relacionamento *hasProjectMolpsAtr* os indivíduos dos tipos *ranges* e *domain*. A variável *dataF* representa a classe atributo *AtrProjectDataF* que possui como instância o individuo *ind6*, ver Figura 52. A linha *dataF.listInstances()* retorna a instância criada que é o *ind6* e atribui a outra variável do tipo *Individual* chamada de *indef* conforme a linha "*indef = (Individual) ef.next()*" da Figura 53. Após, é adicionado como *range* da propriedade o indivíduo *indef* que representa a instância da classe *AtrProjectDataF*. Isso é feito para todos os indivíduos que são instâncias das classes atributos, sendo que todos eles serão *ranges* da propriedade. O individuo *domain* é relacionado com a propriedade da mesma forma que os indivíduos r*anges*, só muda o método *ob.addRange()* para o método *ob.addDomain()*. Como resultado de toda esta explicação, na Figura 52 a informação nome *Projeto Educar Web* será uma instância *domain*, e as informações Nome, Escopo, Programa, Gerente, Data inicio e fim, serão *ranges* da propriedade *hasProjectMolpsAtr*. Esse tipo de relacionamento é feito, pois quando o agente de *software* realizar uma consulta sobre a informação *Projeto Educar Web*, o resultado terá que ser as informações de escopo, gerente e datas que foram descritas referentes a informação *Projeto Educar Web*. Nota-se que a informação "Nome" é *domain* e *range* da propriedade, mas ela se difere na hora de instanciar onde a *domain* instância a classe *ProjectMolps* e a r*ange* instancia a classe atributo *AtrProjectName*. A Figura 54 ilustra como os indivíduos criados e relacionados através de uma propriedade são representados via OWL. Na Figura 54 os indivíduos representam informações sobre a versão do projeto.

Figura 54 - Versão OWL dos indivíduos.

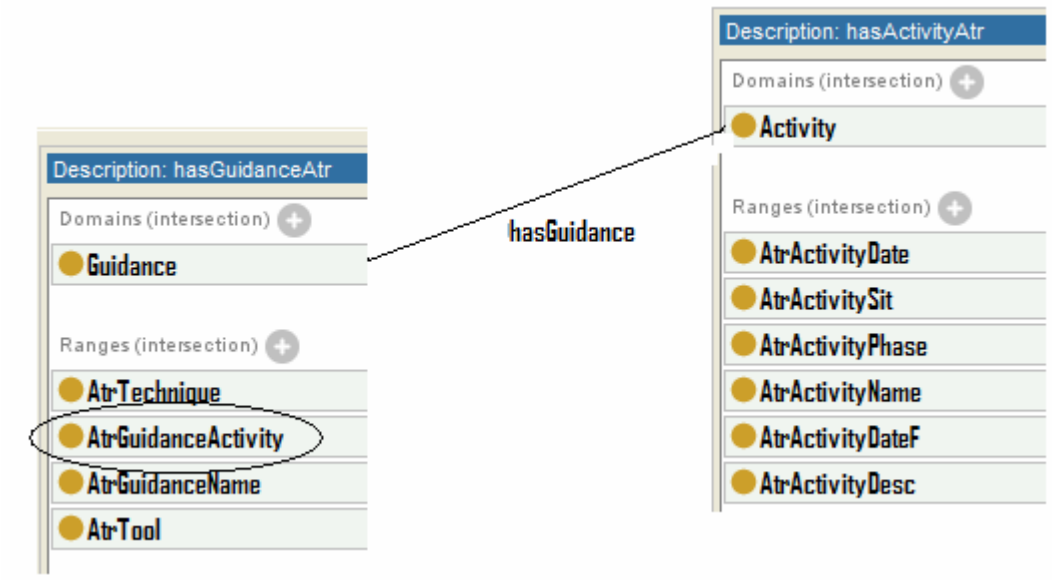
```
<xtremeprogramming:Release rdf:about="http://www.ontolps.com.br/xtremeprogramming.owl#Versao_1">
  <ontolps:hasReleaseAtr rdf:resource="http://www.ontolps.com.br/ontolops.owl#Versao_1"/>
  <ontolps:hasReleaseAtr rdf:resource="http://www.ontolps.com.br/ontolops.owl#day01_11_2011"/>
  <ontolps:hasReleaseAtr rdf:resource=
  "http://www.ontolps.com.br/ontolops.owl#Nome_Projeto_
  Desenvolvimento_de_um_novo_modelo_de_buscas_geracao_de_Historicos_Escolares"/>
 </xtremeprogramming:Release>
```

Fonte: Desenvolvido pelo autor.

Na figura pode ser notado que o indivíduo Versao_1 representa o *domain* e *range* do relacionamento *hasReleaseAtr*, mas representam instâncias de classes diferentes. Então quando um agente de *software* consultar as informações sobre a Versão_1, os resultados serão os indivíduos *ranges* que representam as informações *"Versao_1", "day01_11_2011" e "Nome_Projeto_Desenvolvimento_de_um_novo_modelo_de_busca_para_geracao_de_Historicos_Escolares"*.

Outro relacionamento realizado é entre indivíduos de classes diferentes, e este relacionamento é criado em tempo de execução incrementando a ontologia. Na ontologia algumas classes possuem atributos que representam outras classes, como visto na Figura 55.

Figura 55 - Classe *Guidance* possui atributo do tipo atividade.



Fonte: Desenvolvido pelo autor.

Como pode ser observada na figura, a classe *Guidance* (Guia de apoio) possui um atributo *AtrGuidanceActivity* que representa uma atividade. Isto significa que as informações armazenadas que representam os atributos da classe *Guidance* possuem uma atividade relacionada a eles. Da mesma forma que a *Guidance*, uma atividade armazenada com suas informações também possuirá um guia de apoio relacionado a ela. Como detalhe da implementação, para ocorrer este tipo de relacionamento foi criado um novo relacionamento entre a instância que representa um guia de apoio com a instância que representa a atividade

que este guia de apoio refere-se. Um exemplo seria uma atividade de adicionar uma tabela nova em um banco de dados. O guia de apoio seria informações sobre o banco de dados. Então quando houver uma consulta sobre esta atividade, as informações sobre o banco de dados também devem fazer parte da resposta da consulta.

Para acontecer este relacionamento foi codificado para que em tempo de execução a atividade que possui um guia de apoio tenha um relacionamento com este guia de apoio. A Figura 56 apresenta a parte do código que faz exatamente este relacionamento, nesta figura é realizado o relacionamento entre uma atividade e o guia de apoio.

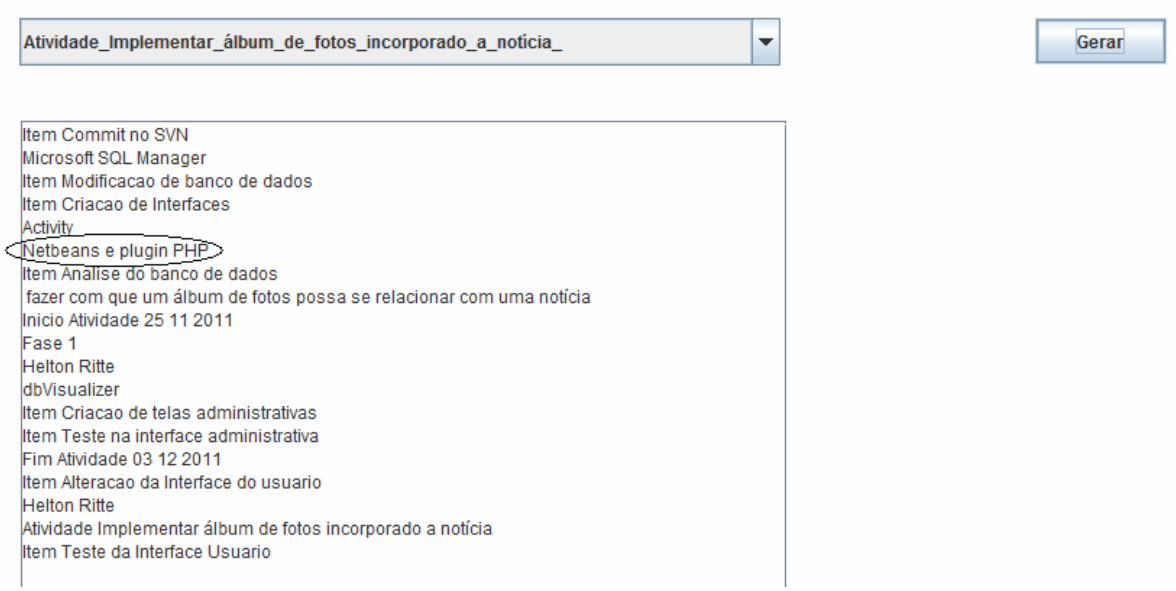
Figura 56 - Parte do código do relacionamento entre a atividade e o guia de apoio.

```
Iterator p= at.listInstances();
            Individual indp = null;
            while(p.hasNext()){
              indp = (Individual) p.next();
              String compara = indp.getLocalName();
              if(compara.equalsIgnoreCase(ind2.getLocalName())){
                 indp.addProperty(ob2, ind5);
```

Fonte: Desenvolvido pelo autor.

Quando o usuário armazenar uma informação de guia de apoio, ele precisa escolher uma atividade em que este guia de apoio refere-se. Assim que é armazenado o nome do guia de apoio e a atividade, é realizada uma busca nas instâncias da classe atividade onde estas instâncias são da classe *Activity* que quando consultadas trazem todas as instâncias das classes atributos relacionadas a ela, como explicadas nesta seção. A busca é para encontrar a instância da atividade que foi escolhida pelo usuário no guia de apoio, e quando encontrada esta instância é criado o relacionamento entre esta instância e a instância do guia de apoio que se refere a ela. O relacionamento criado é o mesmo relacionamento da classe *Activity* com seus atributos, o *hasActivityAtr*. Então pela Figura 56 a linha *indp.addProperty(ob2, ind5)* ilustra que o indivíduo *ind5* que representa a instância da classe guia de apoio é relacionado através da propriedade *ob2* que é *hasActivityAtr* com o indivíduo *indp* que representa a instância da classe atividade. Assim quando é realizada uma consulta sobre uma atividade, um guia de apoio relacionado a ela também é mostrado no resultado. O relacionamento *hasActivityAtr* é transitivo, entende-se que o guia de apoio se relaciona também com os atributos da atividade pois eles estão ligados pela mesma propriedade. Na Figura 57 uma consulta é realizada sobre uma atividade e a informação circulada é um guia de apoio que se refere a esta atividade.

Figura 57 - Consulta sobre uma atividade com um guia de apoio relacionado.

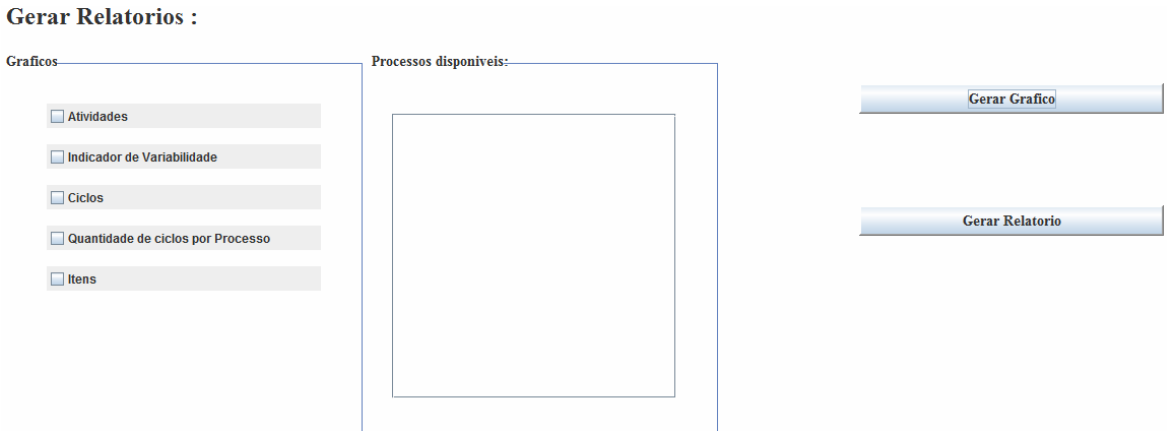


Fonte: Desenvolvido pelo autor.

6.6 CONSULTAS

Todas as informações armazenadas na ontologia poderão ser consultadas a qualquer momento. O usuário dispõe de uma interface em que poderá solicitar gráficos referentes ao andamento do projeto ou consultar informações e ainda gerar relatório. As solicitações são consultadas pelo agente de Projeto que retorna os resultados na forma de gráficos ou relatórios. A Figura 58 apresenta esta interface.

Figura 58 - Interface para gerar gráficos e relatórios.



Fonte: Desenvolvido pelo autor.

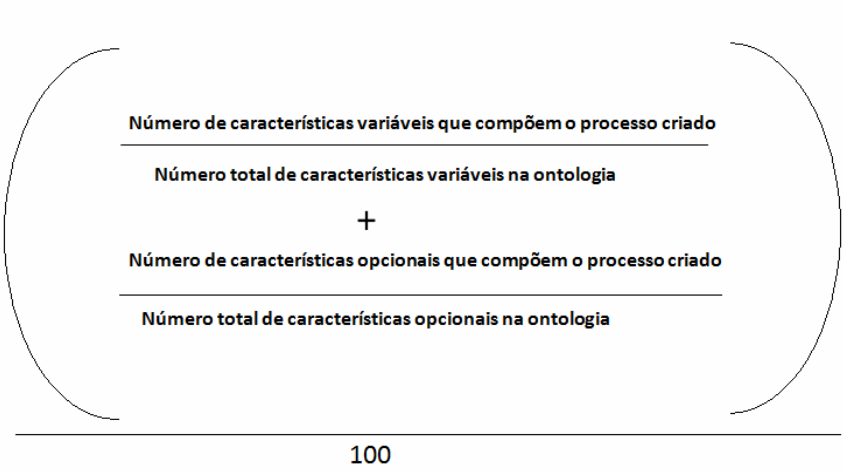
Nesta interface o usuário pode solicitar ao Agente de Projeto gráfico das atividades armazenadas, para assim saber se há algum atraso, da mesma forma com os itens de desenvolvimento que representam as tarefas que formam uma atividade e os ciclos de desenvolvimento das atividades. O gráfico Indicador de Variabilidade define em porcentagem

quanto o processo variou em suas características. Já o gráfico de Quantidades de Ciclos por Processo, ilustra quantos ciclos de desenvolvimento teve o projeto. Os gráficos Itens, Ciclos e Atividades, ilustram quais os atrasos e o que está completo no decorrer do projeto. .

Para criar o gráfico de atividade o Agente de Projeto consulta na ontologia todas as atividades armazenadas obtendo as datas finais de cada uma e a situação em que se encontra, podendo estar completa ou não. Com cada data final é feito um cálculo para saber se a atividades está atrasada. O cálculo é a subtração da data corrente do sistema com a data final da atividade, assim alimentando o gráfico com o resultado deste cálculo. No gráfico serão explicitadas as atividades em atrasos e as completas. Para os gráficos de ciclos e itens, as consultas e cálculos são da mesma forma que o gráfico da atividade.

No gráfico de Indicador de Variabilidade, o Agente de Projeto faz o seguinte cálculo, ilustrado na Figura 59.

Figura 59 - Cálculo para variabilidade do processo.

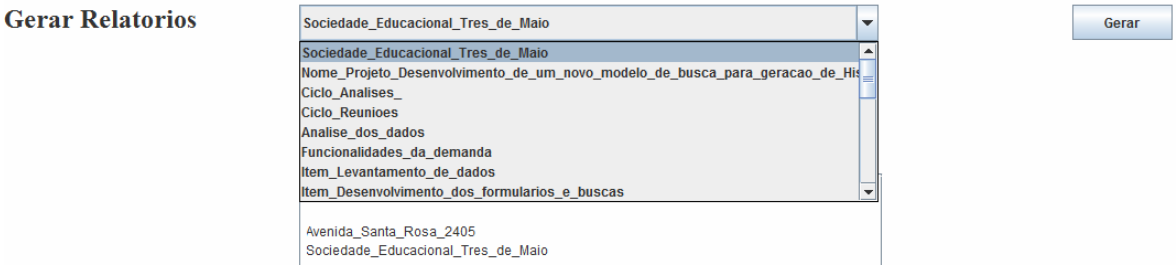


Fonte: Desenvolvido pelo autor.

Para saber o valor de características de variação no processo criado, foi definida em tempo de execução duas propriedades na ontologia, *isEscolhaV* e *isEscolhaOp*, adicionando como *ranges* destas propriedades as características variáveis e opcionais definidas no processo. Os números totais de características variáveis e opcionais são obtidos consultando as propriedades *isVariavel* e *isOpcional*, assim contando quantas classes *ranges* possuem essas propriedades.

Os relatórios contêm as informações sobre o projeto, como atividades, itens/tarefas, ciclos, entre outras. Para consultar e gerar relatório o usuário dispõe de uma interface de consulta, ver Figura 60.

Figura 60 - Interface para consulta e relatórios.



Fonte: Desenvolvido pelo autor.

A Figura 60 apresenta várias informações armazenadas na ontologia sobre um projeto. O usuário seleciona as informações que deseja consultar, em seguida o resultado aparece na tela e é criado um arquivo no diretório "C:\Temp\ do computador com as informações para ser utilizado pelo usuário como desejar.

A consulta para gerar o relatório é realizada pelo Agente de Projeto, que possui um comportamento chamado *ComportamentoProjeto*. Neste comportamento assim que o usuário selecionar a informação que desejar consultar, o agente busca esse indivíduo que representa esta informação. A Figura 61 apresenta a consulta referente a Figura 60, onde a informação *Sociedade Educacional Três de Maio* foi selecionada.

Figura 61 - Trecho de código para consulta.

```
 Individual indiv = ontModel.getIndividual(
     "http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#"+valor);
if(indiv!=null){
Iterator it2 = indiv.listProperties();

    while(it2.hasNext()){
       Statement stmt = (Statement) it2.next();
       Property pro = stmt.getPredicate();


          pega= (Resource) stmt.getObject();
          if(!pega.getLocalName().contains("Atr")){
          try{
          bw.append(pega.getLocalName());
          bw.newLine();
          }catch(Exception e){}
          jTextArea1.append(pega.getLocalName());
          jTextArea1.append("\n");
       }
```

Fonte: Desenvolvido pelo autor.

Na Figura 60 o usuário selecionou a informação que representa a classe organização da ontologia PMBOK importada na Molps. Como pode ser observado na Figura 61, se faz

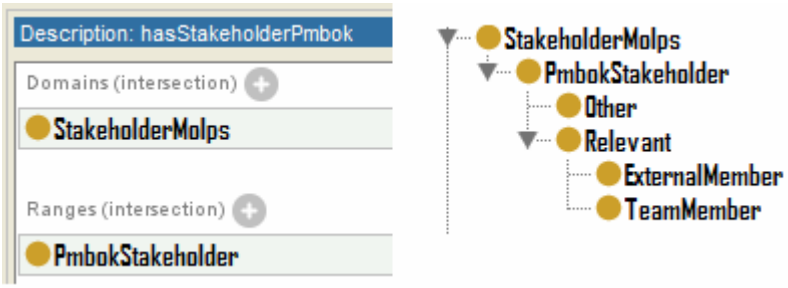
uma busca pelo indivíduo que representa a informação, mas o indivíduo buscado é aquele que foi armazenado como instância da classe *Organization* e de característica *domain.* Aqui pode ser notado o porquê da forma de armazenamento de indivíduos explicado na seção 5.5. Após achar o indivíduo, que é representado na variável *indiv,* são listadas todas as propriedades deste indivíduo. Em cada propriedade encontrada, neste caso será apenas uma propriedade, é adicionada na variável *pega* o valor do objeto que se relaciona, através desta propriedade, com o indivíduo da variável *indiv*. O resultado deste relacionamento são todos os indivíduos que neste relacionamento se encontram como *ranges*. Então as informações armazenadas sobre a organização do projeto são: *Sociedade Educacional Três de Maio* e *Avenida Santa Rosa 2405*. Para esclarecer, na ontologia essas informações ficaram armazenadas da seguinte forma:

- Instância da classe *Organization*: Sociedade Educacional Três de Maio;
- Instância da classe atributo *AtrOrganizationName*: Sociedade Educacional Três de Maio;
- Instância da classe atributo *AtrOrganizationEndereco*: Avenida Santa Rosa 2405;
- Propriedade: *hasOrganizationAtr*;
- *Domain* da propriedade: Sociedade Educacional Três de Maio;
- *Ranges* da propriedade: Sociedade Educacional Três de Maio e Avenida Santa Rosa 2405.

6.5.1 Inferência

A ontologia possui papel importante na inferência computacional. A ontologia Molps possui inferência em suas consultas. Um exemplo é ilustrado na Figura 62.

Figura 62 - Exemplo de inferência.



Fonte: Desenvolvido pelo autor.

Esta figura apresenta o relacionamento *hasStakeholderMolps* entre as classes *StakeholderMolps* e *PmbokStakeholder*. Nota-se que a classe *PmbokStakeholder* possui subclasses e essas possuem outras subclasses, que não possuem relacionamento direto com a classe *StakeholderMolps.* Ao fazer uma consulta sobre esta propriedade infere-se que a classe *StakeholderMolps* se relaciona com a as subclasses de *PmbokStakeholder* sem haver um relacionamento direto. Isto é possível porque existe uma transitividade em nível de herança entre as classes. Por estas classes possuírem subclasses a transitividade faz com que a classe

StakeholderMolps se relaciona com as subclasses de *PmbokStakeholder* e as subclasses das subclasses de *PmbokStakeholder*. A Figura 63 apresenta uma consulta que demonstra a inferência na ontologia.

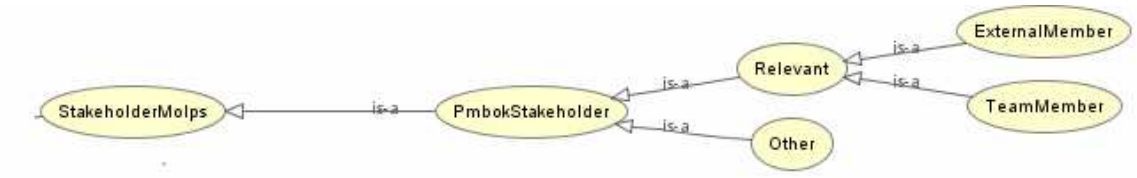
Figura 63 - Trecho de código representando inferência.

```
String consulta =
" SELECT " + " ?subject,?property,?object " +
" WHERE " +
" (?subject ?property ?object ) " +

"  using " +
" ont for <http://www.ontolps.com.br/ontolps.owl#> " ;

QueryResults resultadoQuery = preparaQuery(consulta);
for (Iterator iter = resultadoQuery; iter.hasNext(); ) {
   ResultBinding res = (ResultBinding)iter.next();
   Resource subject = (Resource)res.get("subject");
   Resource Property = (Resource)res.get("property");
   Resource Objeto = (Resource)res.get("object");

   if (subject.getLocalName().equalsIgnoreCase("hasStakeholderPmbok")) {

        StmtIterator stmtitersubject3 = subject.listProperties();

          while (stmtitersubject3.hasNext()) {
             com.hp.hpl.jena.rdf.model.Statement stmtMolpsTeam = (com.hp.hpl.jena.rdf.model.Statement)stmtitersubject3.nextStatement(
             Property prop = stmtMolpsTeam.getPredicate();

             if(prop.getLocalName().equalsIgnoreCase("range")) {

               obRangeOp = (Resource)stmtMolpsTeam.getObject();

             String getOntoLpsMolpsTeam =

                " SELECT " + " ?subject, ?property, ?object " +
                " WHERE " +
               " (?subject ?property <lps:"+obRangeOp.getLocalName()+"> ), " +

                " (?subject ?property ?object ) " +
                " using " +
                " lps for <http://www.ontolps.com.br/ontolps.owl#>, " +
                " fdd for <http://www.ontolps.com.br/fdd.owl#>, " +
                " pt for <http://www.w3.org/1999/02/22-rdf-syntax-ns#>, " +
                " rb for <http://www.w3.org/2000/01/rdf-schema#>, " +
                " sc for <http://www.ontolps.com.br/scrum.owl#> ";
              QueryResults resultadoQueryMolpsTeam = preparaQuery(getOntoLpsMolpsTeam);
              for (Iterator iterMolpsTeam2 = resultadoQueryMolpsTeam; iterMolpsTeam2.hasNext(); ) {
                 ResultBinding resMolpsTeam2 = (ResultBinding)iterMolpsTeam2.next();
                 Resource propMolpsTeam2 = (Resource)resMolpsTeam2.get("subject");
                 Resource propSSS = (Resource)resMolpsTeam2.get("property");
                 Resource objSSSS = (Resource)resMolpsTeam2.get("object");

                 if(propSSS.getLocalName().equalsIgnoreCase("subClassOf")) {

                    if(objSSSS.getLocalName().equalsIgnoreCase("StakeholderMolps")) {
                    System.out.println(propMolpsTeam2.getLocalName());
                    }
```

Fonte: Desenvolvido pelo autor.

A primeira parte circulada é a variável que recebe a classe que possui o relacionamento *hasStakeholderPmbok.* O segundo círculo apresenta a consulta sobre a classe encontrada, esta classe é a *StakeholderMolps*, que resulta nas classes relacionadas a ela. No terceiro círculo é “printado” na tela todas as classes que se relacionam com a *StakeholderMolps*. Pode-se notar que a classe *StakeholderMolps* se relaciona com a *PmbokStakeholder*, conforme Figura 64, mas infere-se que as subclasses de *PmbokStakeholder* também se relacionam com a *StakeholderMolps*.

Figura 64 - Relacionamento das classes StakeholderMolps e PmbokStakeholder.

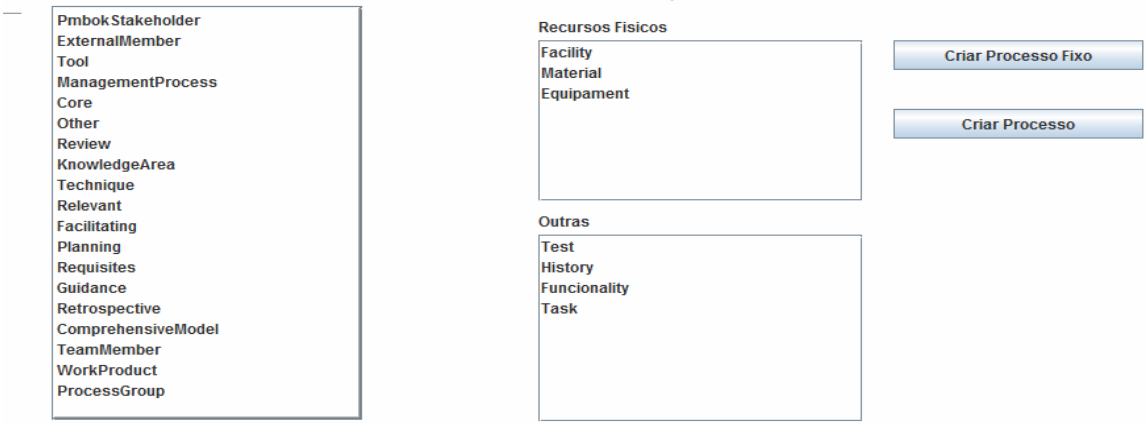


Fonte: Desenvolvido pelo autor.

Neste trabalho a capacidade de inferência mostrou-se explícita na maioria das consultas. Na interface que o processo é criado, as informações presentes nas listas foram consultas na ontologia, onde essa consulta possui inferência. A Figura 65 apresenta a parte do código que é possível observar a inferência.

Figura 65 - Trecho de código representando resultado na interface.

```
if (subject.getLocalName().equalsIgnoreCase("isVariavel")) {

        StmtIterator stmtitersubject3 = subject.listProperties();

          while (stmtitersubject3.hasNext()) {
            com.hp.hpl.jena.rdf.model.Statement stmtMolpsTeam =
            (com.hp.hpl.jena.rdf.model.Statement)stmtitersubject3.nextStatement();

            Property prop = stmtMolpsTeam.getPredicate();

            if(prop.getLocalName().equalsIgnoreCase("range")) {

                obRangeV = (Resource)stmtMolpsTeam.getObject();

            String getOntoLpsMolpsTeam =

              " SELECT " + " ?subject, ?property, ?object " +
              " WHERE " +
              " (?subject ?property <lps:"+obRangeV.getLocalName()+"> ), " +
              " (?subject ?property ?object ) " +
              " using " +
              " lps for <http://www.ontolps.com.br/ontolps.owl#>, " +
              " fdd for <http://www.ontolps.com.br/fdd.owl#>, " +
              " pt for <http://www.w3.org/1999/02/22-rdf-syntax-ns#>, " +
              " rb for <http://www.w3.org/2000/01/rdf-schema#>, " +
              " sc for <http://www.ontolps.com.br/scrum.owl#> ";

           QueryResults resultadoQueryMolpsTeam = preparaQuery(getOntoLpsMolpsTeam);
           for (Iterator iterMolpsTeam2 = resultadoQueryMolpsTeam; iterMolpsTeam2.hasNext(); ) {
             ResultBinding resMolpsTeam2 = (ResultBinding)iterMolpsTeam2.next();
             Resource propMolpsTeam2 = (Resource)resMolpsTeam2.get("subject");
             Resource propSSS = (Resource)resMolpsTeam2.get("property");
             Resource objSSSS = (Resource)resMolpsTeam2.get("object");
```



Fonte: Desenvolvido pelo autor.

Esta figura consulta na ontologia o *subject isVariavel*, quando encontrado são listadas as propriedades existente entre esse *subject* e o *object.* Quando encontrada a propriedade que na tripla é do tipo *range* atribui-se à variável *obRange* o objeto que está relacionado como *range* do *subject isVariavel.* Após é realizado outra consulta SPARQL, mas com a variável *obRange* na cláusula *Where*, que retorna o objeto a ser adicionado na lista de características variáveis. Algumas classes não estão definidas explicitamente como variáveis na ontologia, como é o caso das classes *ExternalMember*, *TeamMember*, *Other* e *Relevant* vindas da classe *PMBOKStakeholder*. Mas como elas foram modeladas sendo subclasses de *PMBOKStakeholder*, infere-se que elas sejam do tipo variável e por isso pode ser notado na Figura 65 que elas se encontram na lista de características variáveis. A classe *Guidance* também possui subclasses, *Tool* e *Technique*, que não estão explicitamente relacionadas pela propriedade *isVariavel*, mas infere-se que elas também são variáveis. Esses relacionamentos podem ser vistos no capítulo Modelo Proposto na seção de Variabilidade da Molps.

Da mesma forma que acontece na lista de características variáveis, a lista de características opcionais também possui inferência na hora de consultar a ontologia. É realizada a consulta sobre a propriedade *isOpcional*, dentre as classes definidas como opcional a classe *PhisicalResource* possui subclasses que implicitamente são opcionais, não sendo relacionadas diretamente com a propriedade *isOpcional.*

No gráfico que indica a variabilidade de um processo, também há inferência na hora de calcular o grau de variação do processo. A Figura 65 apresenta a mesma parte de código implementada para o gráfico de variabilidade.

7 EXPERIMENTOS

Este capítulo apresenta o roteiro de testes responsável pela validação do trabalho proposto. As fases dos testes que compõem o experimento do sistema compreendem a criação de processos de desenvolvimento de *software*, focando na arquitetura da LPS definida pela ontologia, bem como a ferramenta de acompanhar o processo.

7.1 ROTEIRO DE TESTES

Neste capítulo são apresentados os testes realizados para atingir os objetivos esperados. Para os testes percebeu-se uma oportunidade no cenário de aplicação, pois com a ontologia desenvolvida o método de processo não precisará ser o mesmo em todos os projetos, fornecendo a flexibilidade em criar um processo unindo os melhores conceitos das metodologias ágeis e de gerência de projeto.

Para os testes desenvolvidos neste trabalho foi analisado o trabalho desta equipe, observando onde mais se concentravam os esforços e que tipos de casos de testes poderiam ser utilizados neste trabalho. Como resultado desta análise, optou-se em realizar os testes de acordo com as demandas rotineiras da equipe. Estas demandas consistem em customizações, requisições, mudanças, entre outras, em relação ao portal Setrem e o sistema de gerenciamento de ensino Educar Web. Cada membro da equipe recebe demandas para serem realizadas, e estas demandas possuem pessoas envolvidas e responsáveis por ela, tarefas a serem desenvolvidas para a conclusão da mesma, ciclos de reuniões e planejamento, testes, ferramentas, etc. Foi observado o que estas demandas implicam na equipe para serem concluídas, assim chegando à criação dos testes deste trabalho.

Cada demanda possui um fluxo de trabalho a ser cumprido para sua conclusão. Os testes consistiram em criar processos de desenvolvimento de *software* no sistema para cada demanda. Optou-se por acompanhar a equipe, e em cada nova demanda a pessoa responsável forneceu todas as informações referentes a ela e assim foi criado o processo de desenvolvimento de *software* levando em consideração as informações fornecidas. Cada processo foi criado junto com a pessoa responsável para que as características escolhidas fossem de acordo com ela e com a demanda.

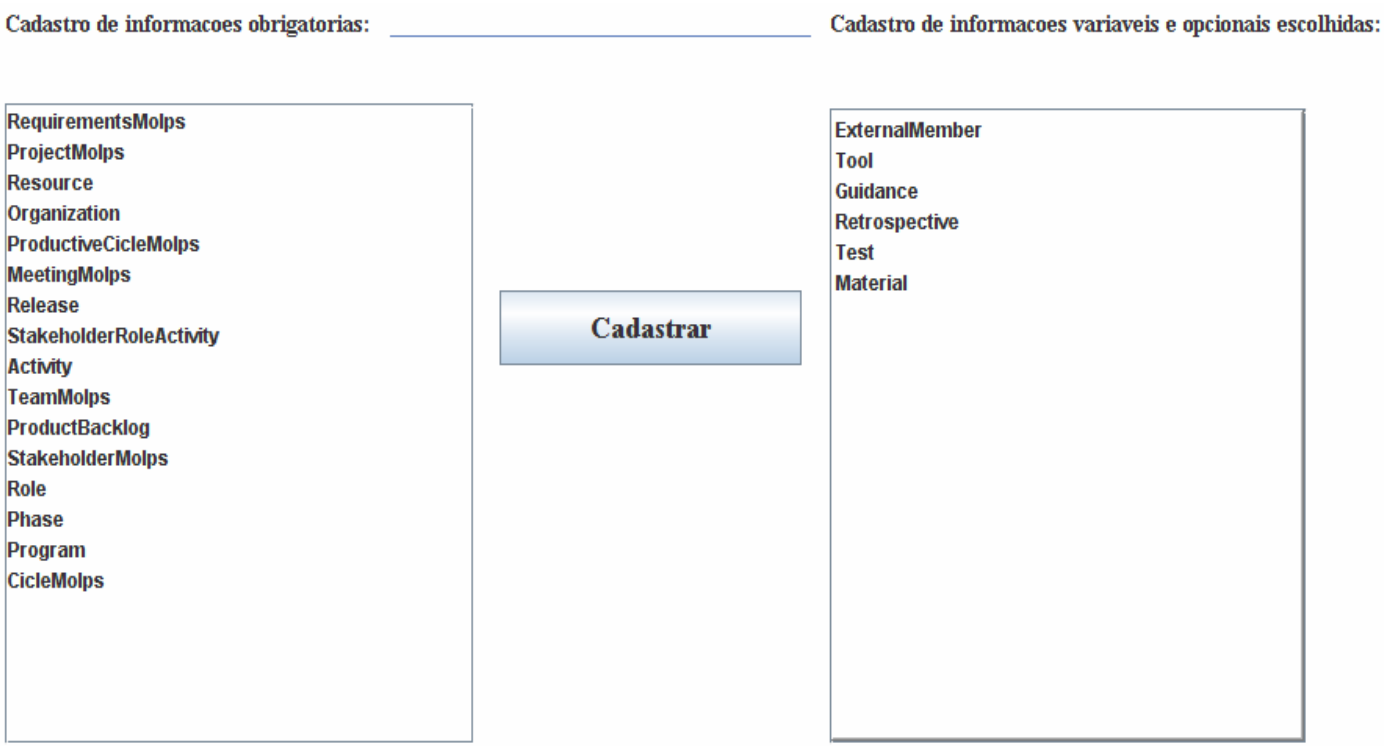
Outra opção de teste foi dada ao gerente de projeto para que ele criasse um processo em que a equipe utilizasse para realizar suas demandas. A equipe optou por criar um processo para cada demanda para fins de conhecimento e resultados que estes processos poderiam gerar. Desta forma vários processos foram gerados, analisando cada um em relação à variação entre eles e resultados no projeto. Foram utilizados quatro casos de testes, com três demandas diferentes. Para cada demanda um processo de desenvolvimento de *software* foi criado, sendo armazenadas todas as informações referentes à demanda.

7.2 PRIMEIRO EXPERIMENTO

O primeiro teste foi realizado no Educar Web com o analista e desenvolvedor e chefe da equipe. Esta pessoa foi designada a acrescentar mais uma utilidade para que históricos escolares pudessem ser gerados neste sistema. Esta demanda exigiu várias atividades, planejamento, ferramentas, ciclos de desenvolvimento, etc.

A primeira tarefa do usuário para criar o processo foi iniciar o sistema invocando a plataforma dos agentes de *software* e em seguida solicitando a criação de um novo processo de *software*. Assim os agentes de *software* entraram em ação para que o usuário pudesse criar o seu processo. Então se pode montar um processo de desenvolvimento de *software*, onde este processo representou os passos que a pessoa responsável tomaria para concluir a demanda. O processo criado possui todas as características obrigatórias da arquitetura da LPS e algumas características representando pontos de variação, conforme ilustrado na Figura 66.

Figura 66 - Características do processo desenvolvido no primeiro experimento.



Fonte: Desenvolvido pelo autor.

A Figura 66 apresenta o processo criado para este teste, e após esta etapa o usuário pode armazenar todas as informações sobre esta demanda na ontologia. Esta tela representa no sistema a parte em que o usuário armazena as informações referentes a essas características escolhidas, bastando apenas selecionar uma característica por vez e clicar no botão cadastrar. Através deste processo criado o responsável pela demanda pode armazenar e monitorar as etapas para a conclusão da mesma. Alguns dados iniciais de entrada deste experimento estão representados na Tabela 2.

Tabela 2 - Dados de entrada.

	Descrição
Organização	Setrem
Programa	Portal Educar Web
Projeto	Demandas no Portal
Atividade	Estudo e análise de

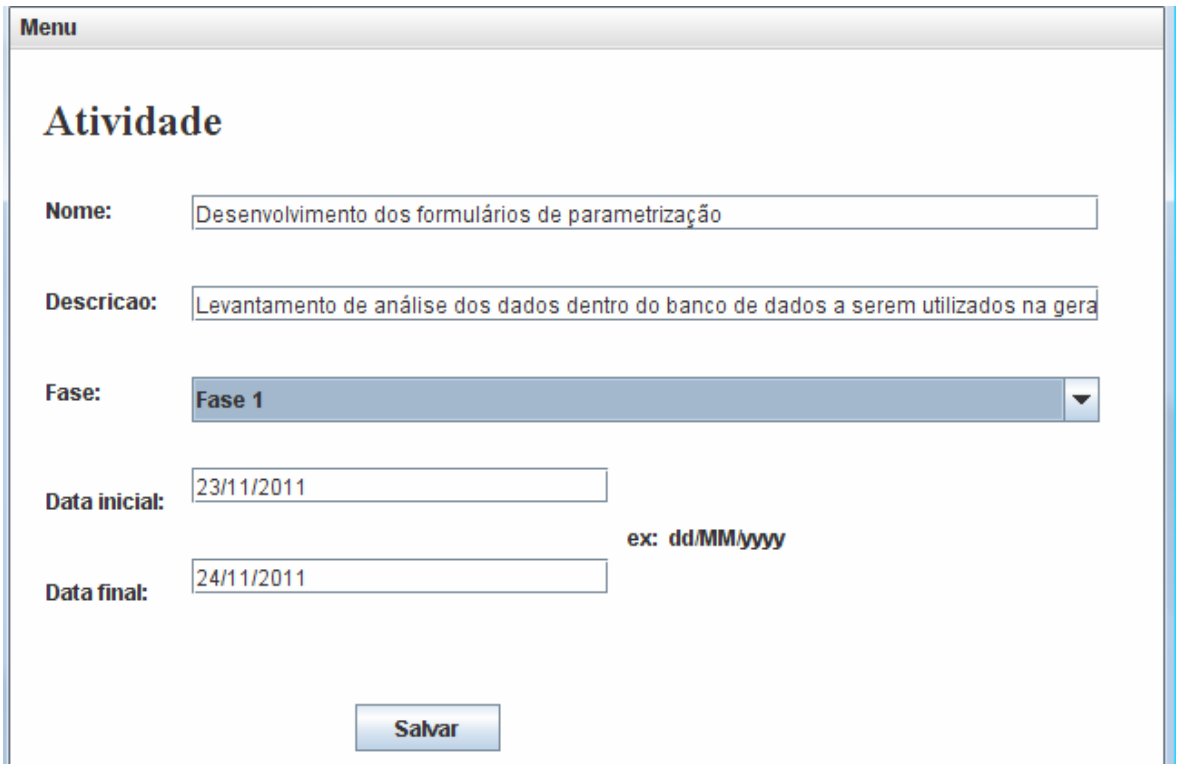
	todos os modelos de *layout* dos históricos escolares da Instituição.
Atividade	Levantamento de análise dos dados dentro do banco de dados a serem utilizados na geração dos históricos.
Atividade	Desenvolvimento dos formulários de parametrização e busca dos dados dentro do banco de dados
Atividade	Apresentação do primeiro *layout* de histórico escolar
Atividade	Testes de *Software*
Atividade	Primeira apresentação do sistema
Atividade	Disponibilização da solicitação para os usuários da secretaria
Atividade	Acompanhamento dos testes junto aos usuários
Atividade	Desenvolvimento de últimos ajustes e finalização da demanda
Ferramentas	*Delphi* 2007
Ferramentas	*Rave Report* 7.5
Ferramentas	SQL *Manager* 2010
Equipe	Edinei Steffen
Equipe	Márcia Wotrisch
Membro externo	Márcia Wotrisch
Recursos Físicos	Folhas impressas
Testes	Testes de análise

Testes	Testes de aceitação
Reuniões	Reuniões semanais
Itens de desenvolvimento	Estudo e análise
Itens de desenvolvimento	Levantamento de análise
Itens de desenvolvimento	Desenvolvimento dos formulários
Itens de desenvolvimento	Testes de *Software*
Ciclos de desenvolvimento	Reuniões referentes às atividades

Fonte: Desenvolvido pelo autor.

A Figura 67 apresenta as informações de uma das atividades referente à demanda.

Figura 67 - Interface de armazenar informações sobre atividades.

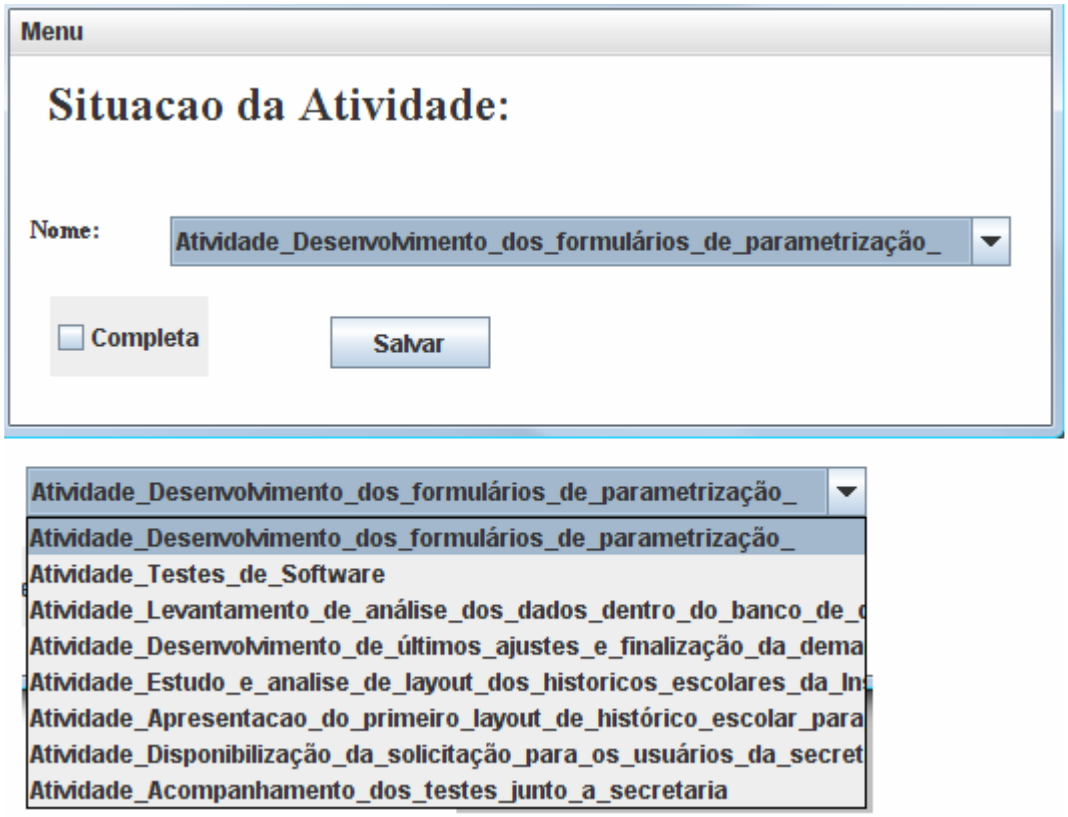


Fonte: Desenvolvido pelo autor.

A Figura 67 representa informações sobre uma das atividades desenvolvidas nesta demanda. Da mesma forma que esta interface, o sistema disponibiliza várias interfaces representando cada característica com seus devidos atributos.

Após armazenar as informações, no decorrer do trabalho de finalização desta demanda, a pessoa responsável pode monitorar as atividades, ciclos e itens de desenvolvimento, e consultar informações armazenadas. Cada atividade, ciclo e item de desenvolvimento finalizado o usuário pode marcar como completo, ver Figura 68, assim obtendo o controle e gerar gráficos para acompanhar os atrasos e finalizações.

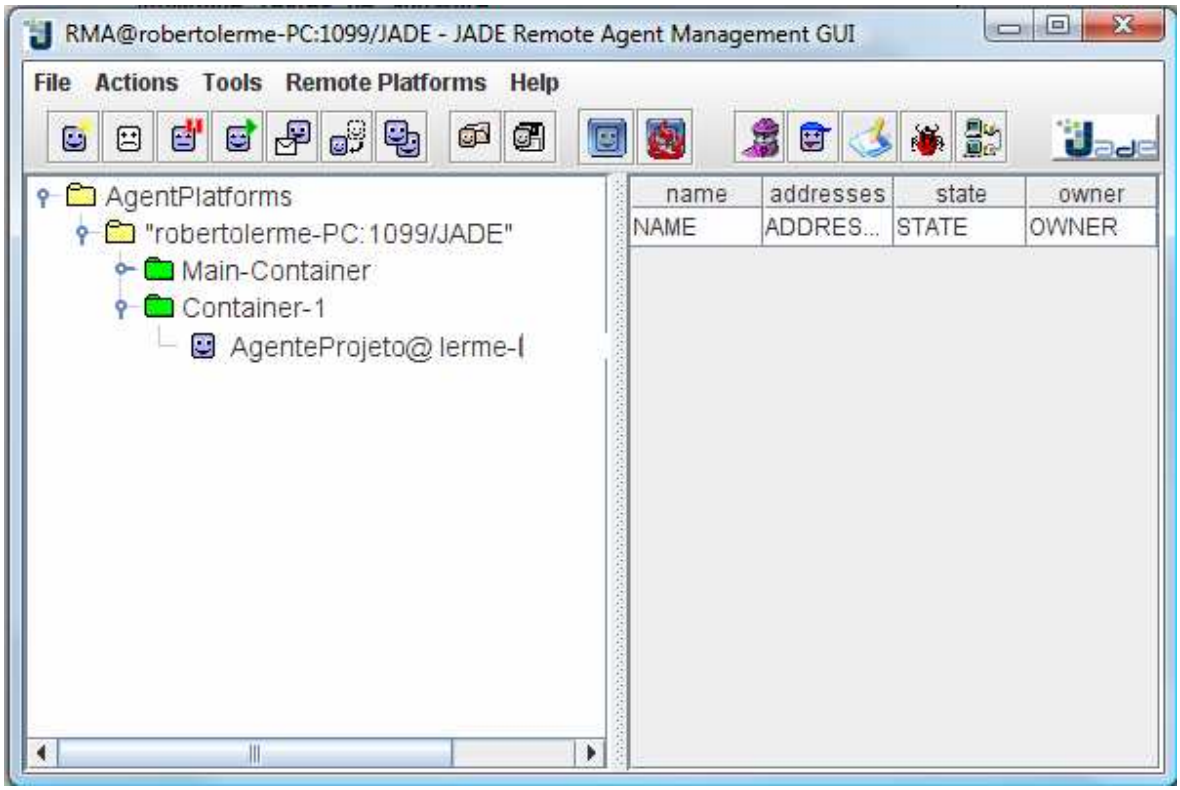
Figura 68 - Interface para marcar as atividades finalizadas.



Fonte: Desenvolvido pelo autor.

Neste teste foram gerados gráficos para controlar o desenvolvimento da demanda, conforme as Figuras 71, 72 e 73. Para a criação dos gráficos, é criado na plataforma Jade o Agente de Projeto, ilustrado na Figura 69.

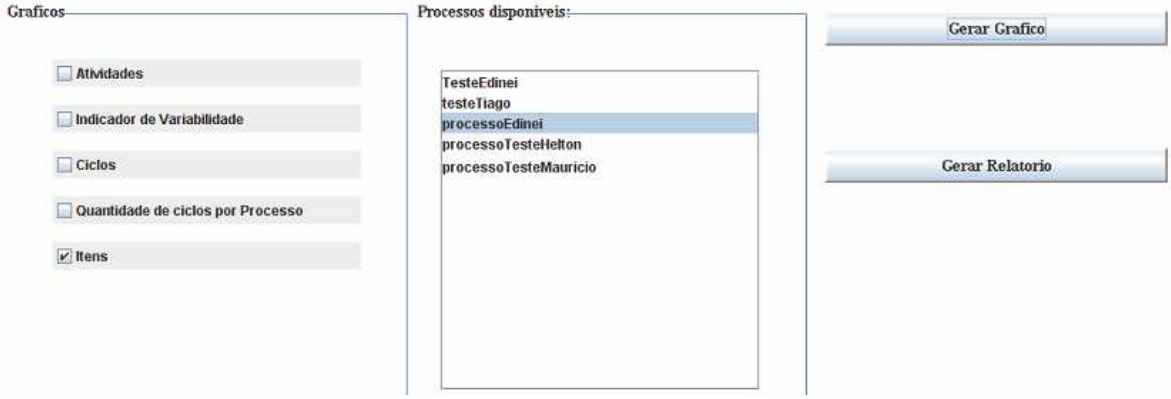
Figura 69 - Plataforma Jade, criação do Agente de Projeto.



Fonte: Desenvolvido pelo autor.

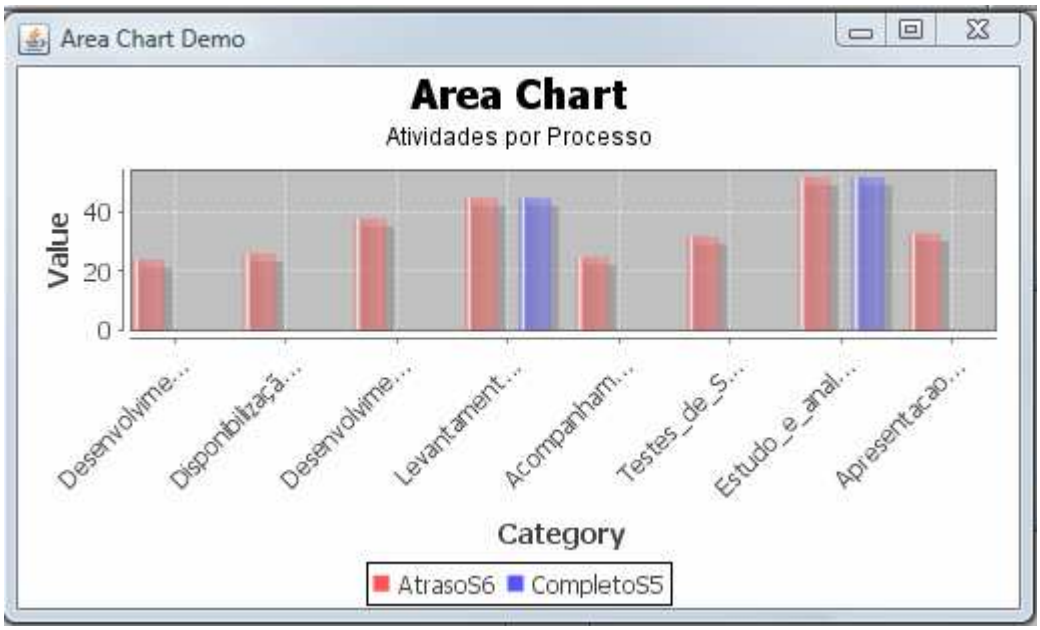
Após a criação do agente o usuário escolheu através de uma interface, ver Figura 70, os gráficos para que o Agente de Projeto realizasse a consulta na ontologia que representa o processo criado por ele.

Figura 70 - Interface para gerar os gráficos do processo.



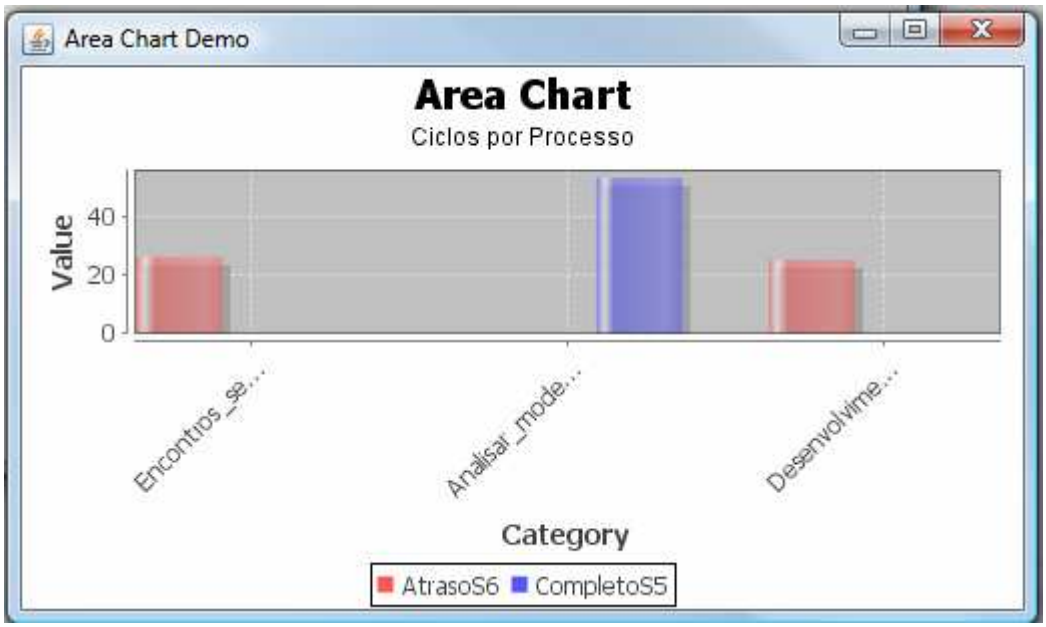
Fonte: Desenvolvido pelo autor.

Figura 71 - Gráfico de atividades.



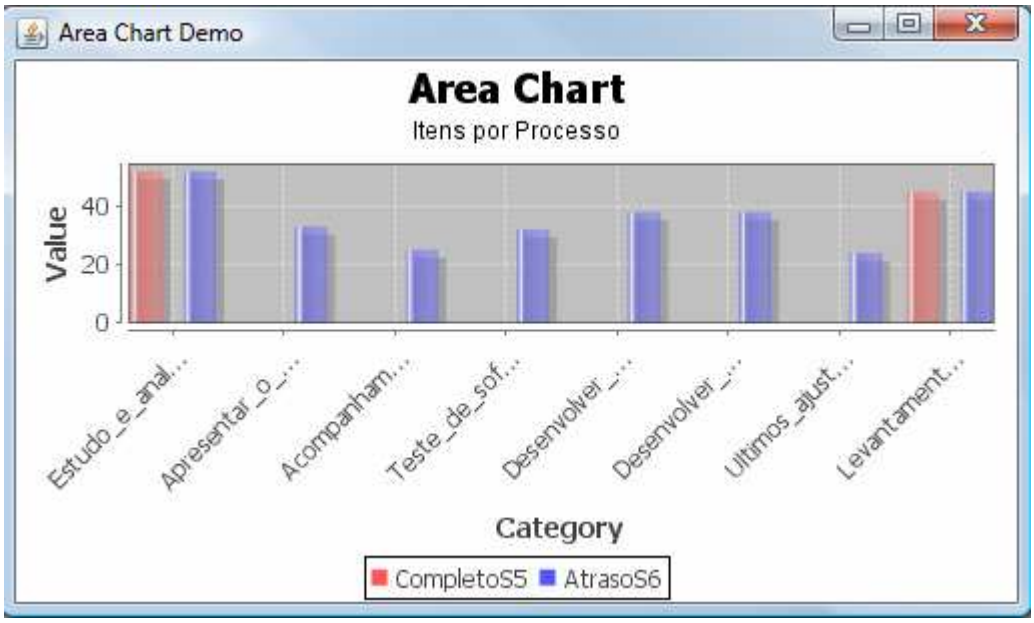
Fonte: Desenvolvido pelo autor.

Figura 72 - Gráfico de Ciclos de desenvolvimento.



Fonte: Desenvolvido pelo autor.

Figura 73 - Gráfico de Itens de desenvolvimento.



Fonte: Desenvolvido pelo autor.

De acordo com estes gráficos o responsável pela demanda de históricos escolares obteve o controle das etapas concluídas e atrasadas. O eixo Y dos gráficos representa a quantidades de dias do início da tarefa até o dia corrente em que o gráfico foi gerado. A consulta realizada pelo agente consiste em buscar todas as atividades, ciclos e itens com suas respectivas datas de fim, realizando o cálculo e alimentando o gráfico com as informações. A Figura 74 apresenta a consulta com o respectivo cálculo.

Figura 74 - Consulta e cálculo do gráfico da atividade.

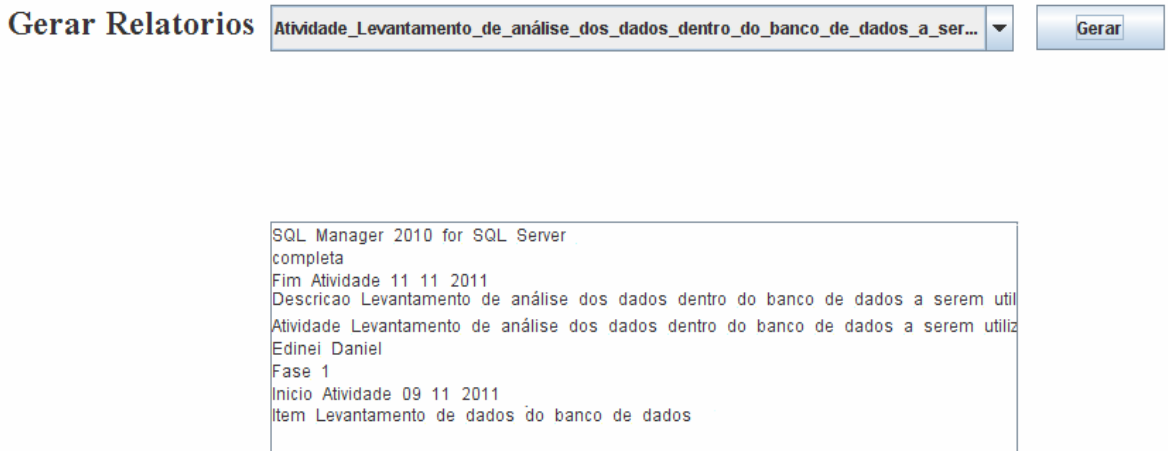
```
String consulta =
" SELECT " + " ?subject,?property,?object " +
" WHERE " +

" (?subject ?property ?object ) "+

"  using " +
" ont for <http://www.ontolps.com.br/ontolps.owl#> " ;

    QueryResults resultadoQuery = preparaQuery(consulta);
    for (Iterator iter = resultadoQuery; iter.hasNext(); ) {
        ResultBinding res = (ResultBinding)iter.next();
        Resource subject = (Resource)res.get("subject");
        Resource Property = (Resource)res.get("property");
        Resource Objeto = (Resource)res.get("object");

        if (Property.getLocalName().equalsIgnoreCase("hasActivityAtr")) {

            if(!Objeto.getLocalName().equalsIgnoreCase("")&&
                !Objeto.getLocalName().equalsIgnoreCase(subject.getLocalName()) ){

                    atividade =subject.getLocalName();
                    atividade=atividade.substring(10);

                if(Objeto.getLocalName().equalsIgnoreCase("completa")){
                        sitAtiv.put(atividade, Objeto.getLocalName());
                    }
                if(Objeto.getLocalName().contains("Fim_Atividade_")){
                    dateF= Objeto.getLocalName();
                }

        if(!dateF.isEmpty()){
        String data2= dateF.substring(14);
        data2=data2.replace("_","/");

        System.out.println("data2 "+data2);

        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
        Calendar cal = Calendar.getInstance();
        Calendar dataSistema = Calendar.getInstance();
           try {
               cal.setTime(sdf.parse(data2));

           } catch (ParseException ex) {
               Logger.getLogger(GraficoAtividade.class.getName()).log(Level.SEVERE, null, ex);
           }
           long diferenca = dataSistema.getTimeInMillis() - cal.getTimeInMillis();
           int tempoDia = 1000 * 60 * 60 * 24;
           long diasDiferenca = diferenca / tempoDia;
```

Fonte: Desenvolvido pelo autor.

Este código SPARQL consulta no processo criado as atividades armazenadas com suas respectivas datas, isto pode ser visto nos dois primeiros círculos. Em seguida é realizado o cálculo, obtendo a data corrente do sistema subtraindo com a data final da atividade resultando na quantidade de dias que está em atraso, onde pode ser notado no último círculo.

Neste teste também foram realizadas consultas sobre as informações armazenadas. Nestas consultas percebe-se um ganho de informações como resultado, conforme Figura 75.

Figura 75 - Consulta sobre uma atividade.



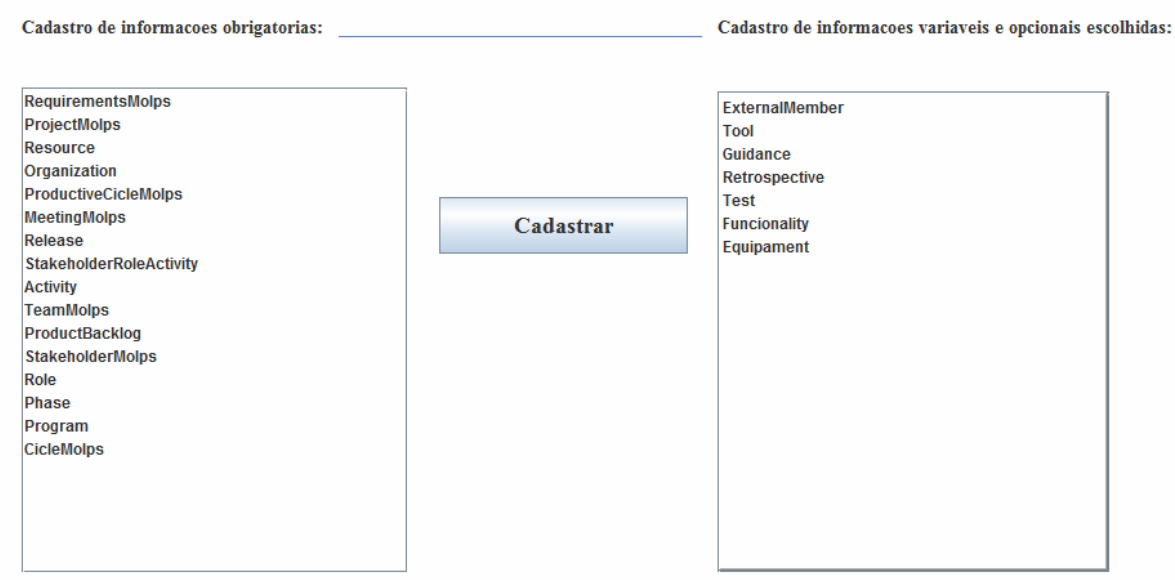
Fonte: Desenvolvido pelo autor.

Nesta figura a consulta foi realizada sobre uma atividade cadastrada na ontologia. Esta atividade possui atributos que a descrevem. Como resultado desta consulta, as informações ilustradas ali são as informações que representam seus atributos e também informações que representam atributos de outras características que possuem relacionamento com esta atividade. Então nesta figura as informações de *SQL Manager 2010 for SQL Server*, *Item levantamento de dados do banco de dados* e *Edinei Daniel* são informações provindas de outras classes que estão ligadas a esta atividade. A consulta realizada em uma ontologia permite trazer uma gama de informações por causa de seu conceito de metainformação.

7.3 SEGUNDO EXPERIMENTO

O segundo teste foi realizado com outra pessoa da equipe que foi designada a função de acrescentar ao Portal da Setrem a funcionalidade de incorporar um álbum de fotos nas notícias. Esta requisição será adicionada ao sistema do portal da Setrem, fazendo com que um álbum de fotos possa se relacionar com uma notícia, e quando isso ocorrer, mostrar o álbum de fotos incorporado na visualização das notícias em detalhes. Para este teste o método de desenvolvimento é o mesmo do primeiro teste. Algumas etapas para concluir esta requisição formam o processo criado para este teste. Para realizar este teste foi criado um novo processo, armazenado informações, foram gerados gráficos e consultadas informações. A Figura 76 apresenta a tela de cadastro das características que compõem o processo criado.

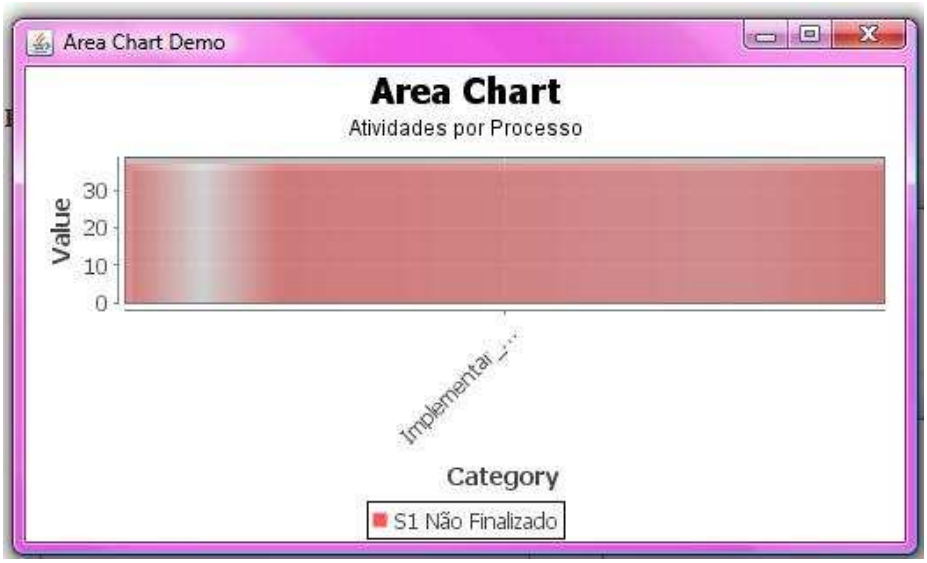
Figura 76 - Características que foram o processo criado para este teste.



Fonte: Desenvolvido pelo autor.

As informações armazenadas contemplam todas as etapas e descrição desta requisição. As pessoas envolvidas foram armazenadas, as ferramentas utilizadas, os testes realizados e assim por diante. Com base nessas informações consultas foram realizadas e gráficos foram gerados. Atividades, ciclos e itens de desenvolvimento foram armazenados, as Figuras 77, 78 e 79 apresentam os gráficos referentes às atividades, ciclos e itens de desenvolvimento.

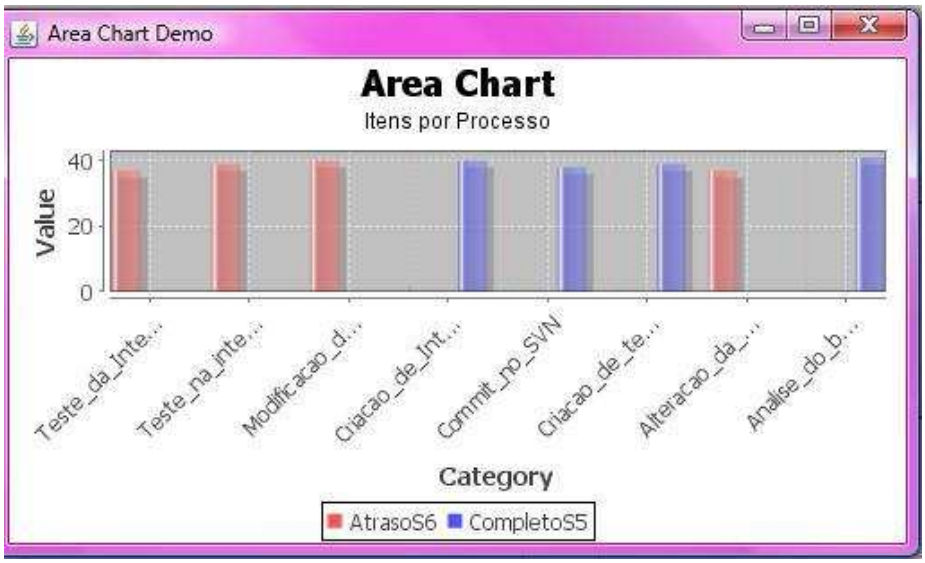
Figura 77 - Gráfico da atividade do processo.



Fonte: Desenvolvido pelo autor.

Neste processo a atividade armazenada foi *Implementar álbum de fotos incorporado a notícia*. No gráfico pode ser observado que a atividade ainda não foi concluída.

Figura 78 - Gráfico apresentando os itens de desenvolvimento.

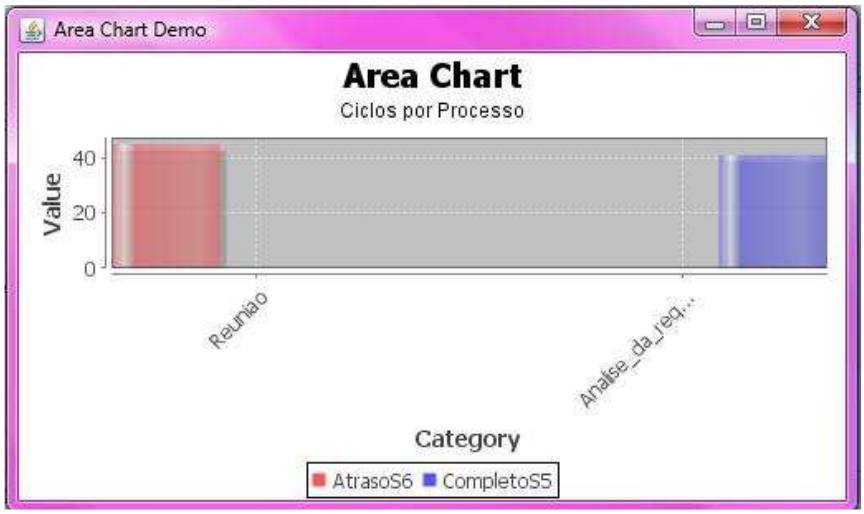


Fonte: Desenvolvido pelo autor.

Neste gráfico os itens em vermelho apresentam atrasos e os azuis já foram completados. Os valores do eixo y do gráfico apresentam a quantidade de dias em atrasos. As informações que representam os itens são:

- Análise de como implementar em banco de dados, em que impactaria;
- Quais telas na interface administrativa seriam necessárias criar, e na interface do usuário qual alterar;
- Modificações no banco de dados;
- Criação das telas administrativas para informar a relação de álbum com notícia;
- Testes na interface administrativa;
- *Commit* no SVN | *Update*;
- Alterações na interface do usuário;
- Testes na interface do usuário.

Figura 79 - Ciclos de desenvolvimento.



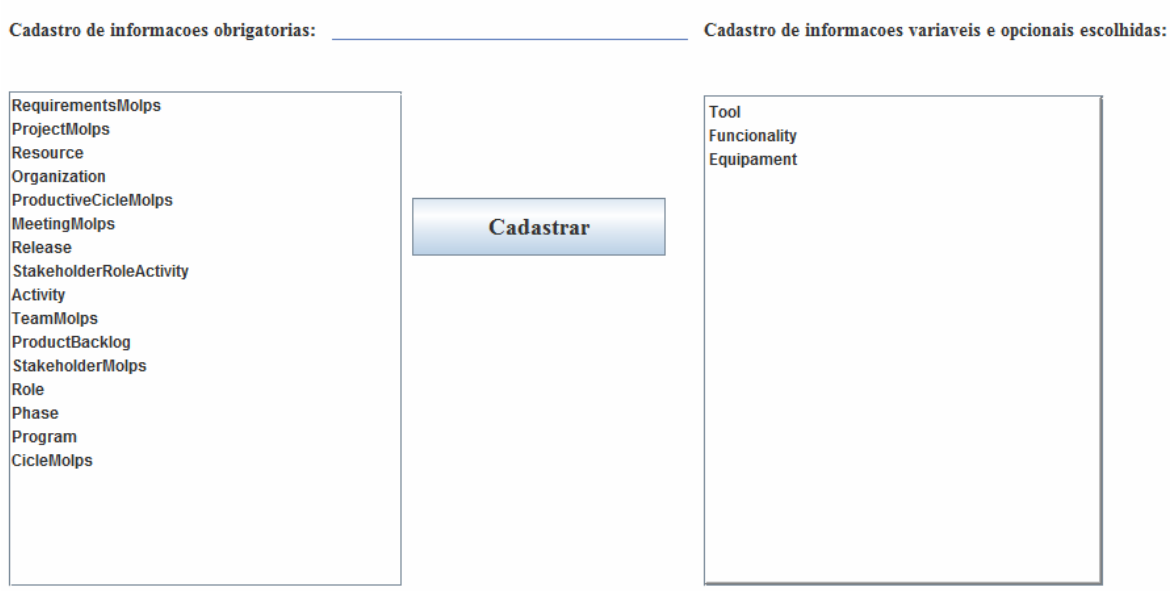
Fonte: Desenvolvido pelo autor.

Os dois ciclos representam as reuniões semanais e análise de impacto da nova requisição.

7.4 TERCEIRO EXPERIMENTO

O terceiro experimento foi realizado com um terceiro membro da equipe. Para esta pessoa foi designada a tarefa de adicionar ao Portal da Setrem a possibilidade de o usuário acompanhar os comentários de uma notícia. Para concluir esta requisição, novas customizações foram adicionadas ao sistema. Da mesma forma que os experimentos anteriores, foi criado um novo processo de desenvolvimento de *software*, foram armazenadas as informações sobre as pessoas envolvidas a esta requisição, foram gerados gráficos e consultadas algumas informações. A Figura 80 ilustra a tela de cadastro que contem as características do processo de desenvolvimento de *software* criado.

Figura 80 - Características do processo de desenvolvimento de *software* criado.



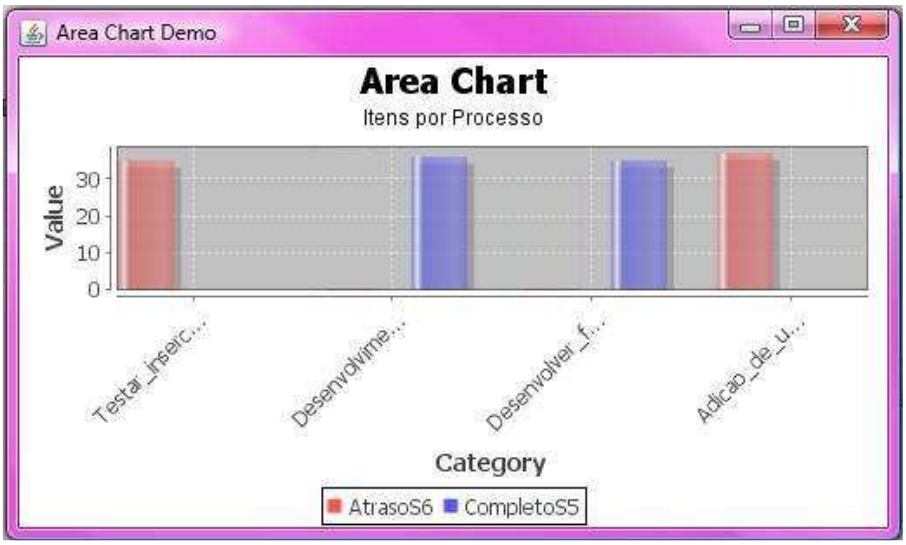
Fonte: Desenvolvido pelo autor.

As informações armazenadas contemplam todas as etapas e descrição desta requisição. As informações sobre as pessoas envolvidas foram armazenadas, as ferramentas utilizadas, os testes realizados e assim por diante. Com base nessas informações consultas foram realizadas e gráficos foram gerados. As atividades, ciclos e itens de desenvolvimento foram armazenados, as Figuras 81, 82, 83 e 84 apresentam os gráficos referentes às atividades, ciclos e itens de desenvolvimento.

Figura 81 - Gráfico de atividade do processo criado.

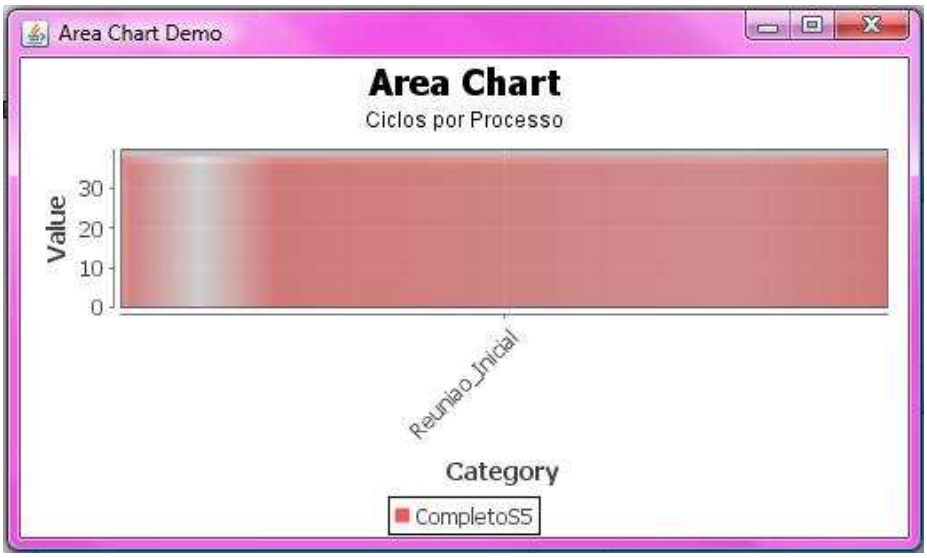
Fonte: Desenvolvido pelo autor.

Figura 82 - Gráfico de itens de desenvolvimento.



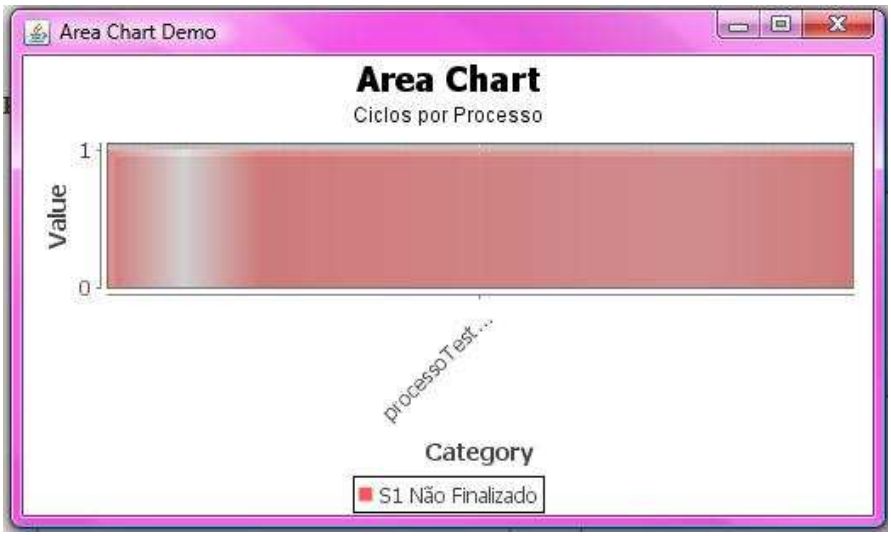
Fonte: Desenvolvido pelo autor.

Figura 83 - Gráfico de ciclos de desenvolvimento.



Fonte: Desenvolvido pelo autor.

Figura 84 - Gráfico de quantidades de ciclos do processo.



Fonte: Desenvolvido pelo autor.

No gráfico da Figura 81, a atividade armazenada foi *Possibilitar ao usuário o acompanhamento de notícias no Portal da Setrem*. A atividade ainda não está completa. No gráfico da Figura 82 as informações dos itens de desenvolvimento foram:

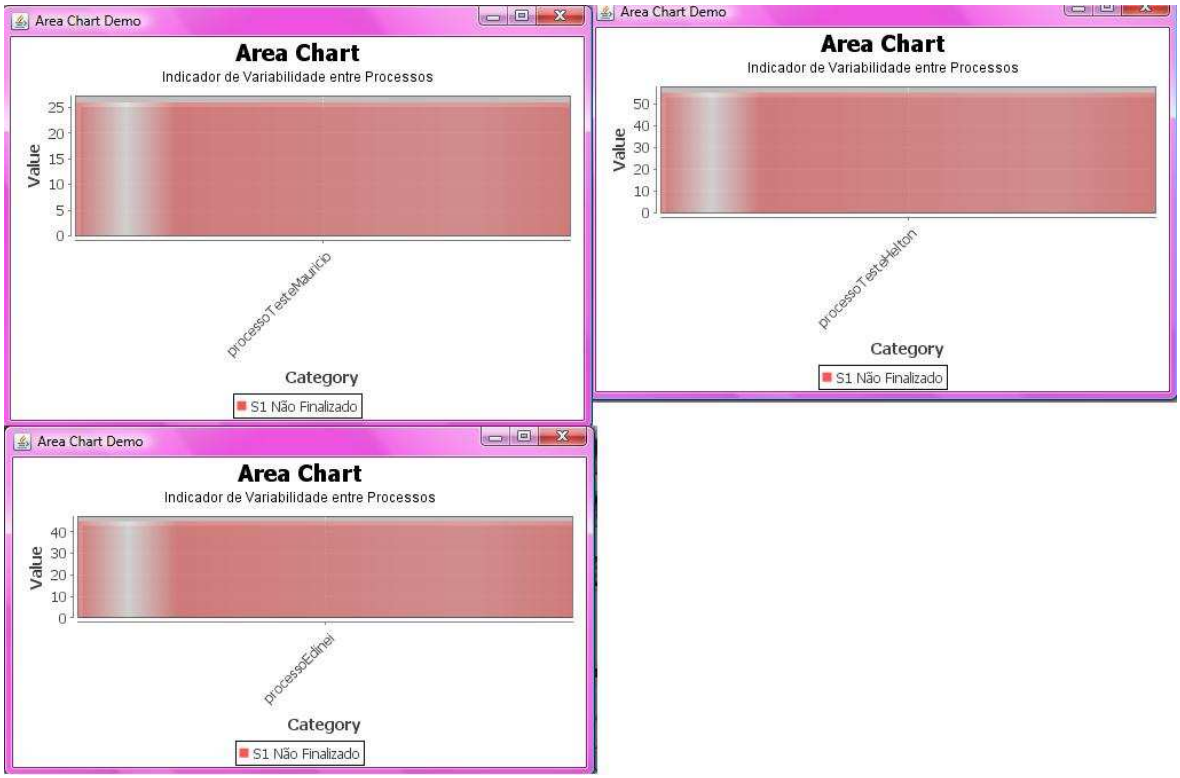
- Adição de um *checkbox* no formulário de comentário;
- Desenvolvimento da função que insere como acompanhante dos comentários;
- Função que envia um e-mail para os que estão acompanhando a notícia;
- Testar a inserção de comentários com acompanhamento da notícia.

No gráfico da Figura 83 a informação que representa o ciclo de desenvolvimento foi *Reunião inicial*. Esta informação é referente a uma reunião que foi realizada no inicio sobre a demanda e que já foi concluída. No último gráfico, Figura 84, apresenta quantos ciclos possui o processo criado.

7.5 QUARTO EXPERIMENTO

No quarto experimento foi realizada uma comparação referente ao grau de variabilidade entre os processos criados. Nesta comparação o responsável pela equipe pode analisar o quanto variou os processos criados. A Figura 85 apresenta o gráfico referente à variabilidade entre os processos.

Figura 85 - Variabilidade dos processos criados.



Fonte: Desenvolvido pelo autor.

Este experimento teve o propósito de demonstrar para equipe e responsável pela mesma, o grau que variou os três processos. Com base nos valores, a equipe juntamente com o responsável poderá optar por criar um processo de desenvolvimento único em que todos podem seguir. Uma opção foi o processo possuir todas as características variáveis e opcionais que os três processos abordam ou outras características de acordo com o que for relevante para a equipe.

7.6 RESULTADOS

Os experimentos realizados seguiram o roteiro de testes estabelecidos para validar e verificar o sistema desenvolvido. Os experimentos realizados tiveram como objetivo validar o sistema a partir das suas principais funcionalidades. Nesta seção, é discutido o resultado dos experimentos.

O sistema foi implantado na organização escolhida como cenário de aplicação. Sua implantação resultou na possibilidade de utilizá-lo para os experimentos. Os agentes de *software* trabalharam para viabilizar o mapeamento entre os domínios de informação do sistema. A cada solicitação do usuário os agentes de *software* trabalharam para atendê-las de maneira autônoma.

Conforme analisado, a arquitetura do sistema permitiu validar as funcionalidades definidas. Não foram necessárias alterações na arquitetura e sistema. Analisando a situação da organização escolhida como cenário de aplicação, pode-se obter uma ótima oportunidade de implantação do sistema.

7.6.1 Linha de Produto de *Software* Definida Através da Ontologia

Através dos experimentos a ontologia apresentou-se adequada para definir a LPS. Os conceitos de flexibilidade foram atingidos pela ontologia onde o domínio definiu uma arquitetura genérica dando a possibilidade de pontos de variações. Outra vantagem da ontologia foi em permitir a inferência na arquitetura da LPS para a construção do processo de desenvolvimento de *software*. Pois quando o usuário cria um processo, a consulta gerada pelos agentes de *software* possui a capacidade de inferir no domínio da ontologia.

Através do sentido semântico dado ao domínio da arquitetura da LPS, pode-se criar um modelo de características unidos os conceitos de métodos ágeis e gerencia de projeto para criação de processo de desenvolvimento de *software*. O sentido semântico foi alcançado pelos relacionamentos entre as características definidas no modelo de domínio da LPS. E através dos relacionamentos pode-se inferir conhecimento para as consultas realizadas.

A ontologia tornou-se uma base de dados para o sistema, onde informações foram armazenadas e acessadas. As informações armazenadas puderam ser relacionadas com outras informações assim promovendo o sentido semântico e um maior resultado de informações quando realizadas consultas.

Com o conceito de LPS pode-se criar uma arquitetura para a criação de processo de desenvolvimento de *software*. Nesta arquitetura é possível unir vários domínios assim tornando a flexibilidade presente em qualquer produto derivado desta LPS. Aderindo a este conceito, vantagens em criar um produto a partir de uma arquitetura pré-fabricada tornam-se explícitas.

7.6.2 Criar, Editar e Acompanhar Processo de Desenvolvimento de *Software*

Para qualquer funcionalidade escolhida neste sistema, os agentes de *software* consultam a ontologia. Quando se deseja criar um processo é realizada uma consulta pelo agente de *software* responsável por esta função, onde são buscadas as características da arquitetura da LPS. Nesta consulta a inferência está explícita, como já explicado na seção 5.5.1, pois quando o agente consulta informações outras informações que estão implícitas no modelo acabam tornando-se explícitas. Quando é criado um novo processo, o processo nada mais é do que um modelo em memória da ontologia Molps, mas com informações armazenadas. Então se pode editar qualquer processo, consultando a ontologia do processo criada.

A característica de inferência também é possível ser observada no acompanhamento do processo. Quando o usuário solicitar alguma consulta sobre alguma informação armazenada no processo, informações de outras características relacionadas com esta consulta serão explicitadas para o usuário, promovendo um ganho de informações nos resultados.

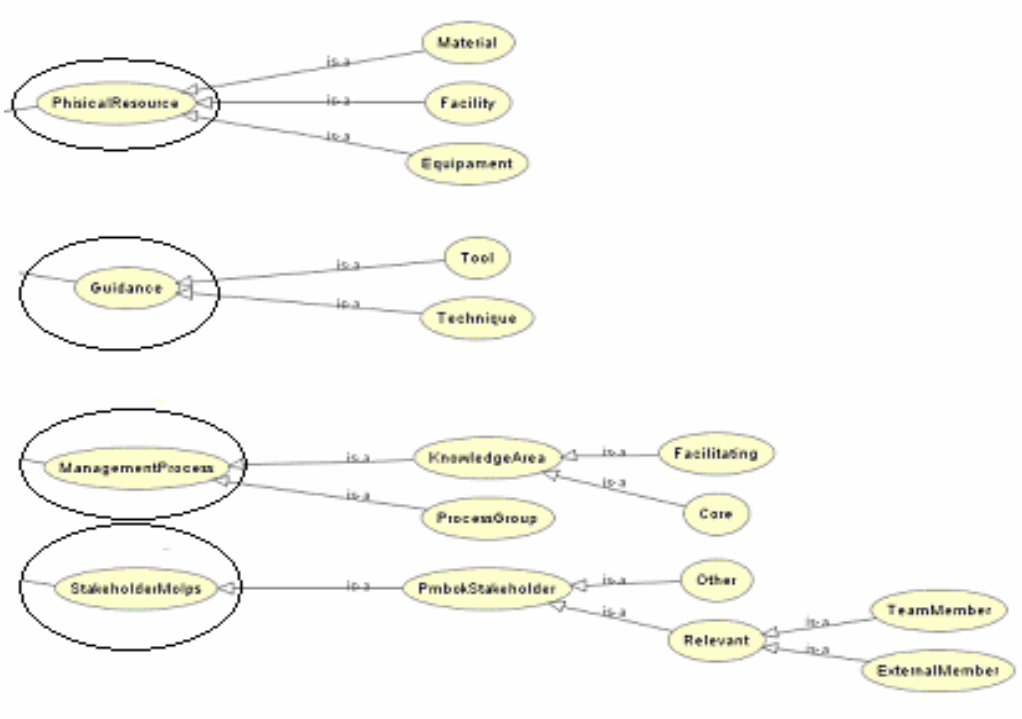
Nos experimentos o acompanhamento do processo criado foi através de gráficos onde se pode ter um controle do desenvolvimento das etapas do processo. Nestes gráficos o usuário pode ter o controle de atrasos e finalizações de atividades, ciclos e itens de desenvolvimento. Como estas etapas são as principais do projeto, o que define os passos para a conclusão do mesmo, os gráficos ajudam a monitorar se possíveis atrasos possam acontecer dentro do prazo estimado.

7.6.3 Destaque de Inferências nos Experimentos

Como já mencionado, a inferência é a capacidade de tornar informações que não estão explícitas em alguma consulta em informações explícitas. A inferência esteve presente no sistema desenvolvido e se destacou na criação de um processo de desenvolvimento, na construção de gráfico e na consulta de informações armazenadas.

Na criação do processo, a consulta realizada para alimentar a interface de criação de processo possui inferência, já explicado na seção 5.5.1. A Figura 86 apresenta as classes onde a inferência esta presente.

Figura 86 - Classes onde a inferência acontece.



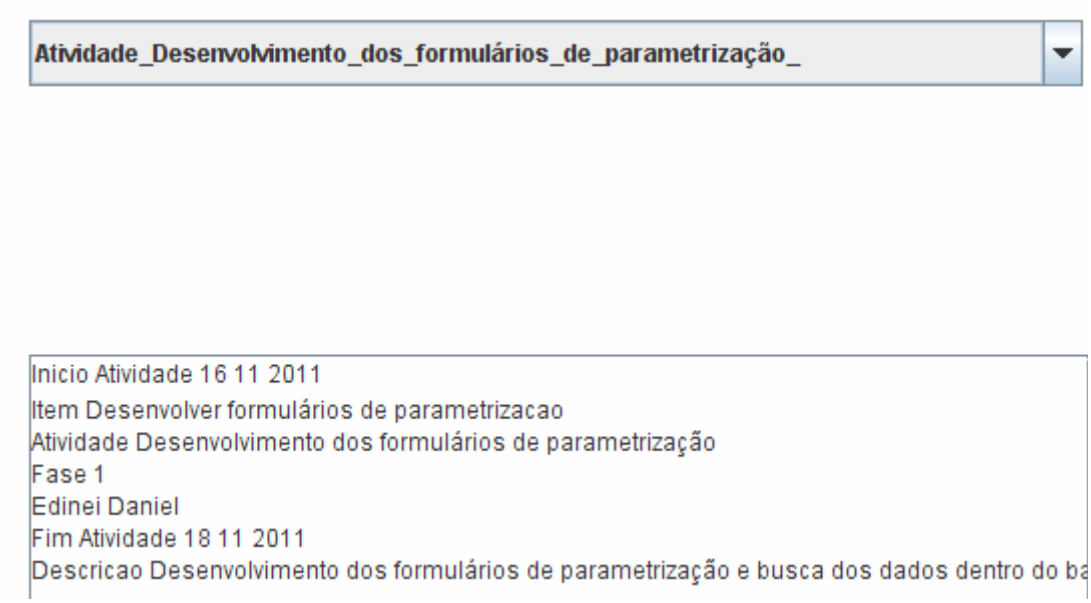
Fonte: Desenvolvido pelo autor.

As classes circuladas possuem subclasses em sua hierarquia. Quando são realizadas as consultas para saber quem são características variáveis e opcionais, as classes circuladas estão denominadas com alguma destas características. A consulta é apenas sobre elas, mas como elas possuem subclasses infere-se que estas classes também são de alguma destas características. Um exemplo é a classe *ManagementProcess* que é denominada variável no modelo de domínio da ontologia, e assim consultando-a alem dela as suas subclasses também são trazidas como resultado na consulta. E assim pode-se inferir na ontologia através das consultas.

O gráfico que indica a variabilidade também possui inferência na ontologia. Esta inferência é a mesma da criação do processo, pois também são consultadas as características variáveis e opcionais da ontologia para realizar o cálculo de variação no processo.

Nas consultas de informações também a inferência apareceu. A Figura 87 apresenta uma consulta em um processo criado, onde informações foram armazenadas sobre uma atividade de desenvolvimento.

Figura 87 - Atividade de desenvolvimento consultada.



Fonte: Desenvolvido pelo autor.

As informações consultadas apresentam informações sobre uma atividade de desenvolvimento do processo, onde as informações *Item Desenvolvimento de formulários de parametrização* e *Edinei Daniel* são informações vindas de outras classes. Isto acontece porque existe relacionamento transitivo entre a classe atividade e as classes de onde estas duas informações foram obtidas.

8 CONSIDERAÇÕES FINAIS

8.1 CONTRIBUIÇÕES

Esta seção aborda as contribuições deste estudo, destacando o porquê da escolha dos conceitos de linha de produto de *software*, ontologias, agentes de *software*, metodologias ágeis e PMBOK.

8.1.1 Linha de Produto de *Software*

O conceito de linha de produto de *software* tem como objetivo oferecer a agilidade, padronização e reuso de artefatos pré-fabricados. Com esse paradigma, as organizações que anteriormente abordavam o desenvolvimento de *software* projeto a projeto, devem concentrar seus esforços na criação e manutenção de uma linha de produto. Esta linha de produto será a base para a produção de uma coleção de produtos pertencentes a uma família.

No estudo desenvolvido a linha de produto de *software* constituiu a arquitetura para serem criados os processos de desenvolvimento de *software*. Assim através desta arquitetura pode-se criar um processo já pré-fabricado e ainda lhes proporcionando a flexibilidade em suas características. Quanto maior a flexibilidade na linha de produto, menor o custo de fabricação, pois o reuso se torna maior. Neste trabalho a linha de produto de *software* proporcionou ao usuário que pudesse variar seus processos desenvolvidos de acordo com os requisitos do projeto. Assim ocasionou que o processo se modelou ao projeto, e não mais o projeto se modelar ao processo.

O objetivo de definir um domínio de conhecimento baseado nas metodologias ágeis em estudo e do PMBOK foi atingido pela linha de produto. Através da linha de produto pode-se definir uma arquitetura representando este domínio.

8.1.2 Ontologia

O uso da ontologia no estudo contribuiu como uma base de conhecimento para os agentes de *software*. Não havia outra maneira melhor de um agente de *software* obter conhecimento. A ontologia serviu como a base de dados, onde informações foram armazenadas e consultas.

O uso da ontologia em definir a LPS promoveu a flexibilidade, conceito presente na LPS. O domínio da LPS foi definido na ontologia, assim promovendo uma estrutura semântica. Nesta estrutura pode-se definir a arquitetura genérica da linha de produto bem como os pontos de variações. A ontologia contribui como uma forma diferente para estruturar uma LPS. Outra contribuição da ontologia foi a capacidade de inferência no domínio de conhecimento. A inferência esteve presente nas consultas realizadas pelos agentes de *software*.

Através da ontologia o objetivo de representar o domínio/arquitetura da LPS foi atingido, pois a estrutura da LPS tornou-se um modelo semântico, sendo possível gerar um processo de desenvolvimento de *software* flexível, armazenar informações sobre o processo e realizar consultas inferindo no domínio. O domínio pode ser facilmente aumentado, inserindo novas classes que podem representar outras metodologias ágeis na ontologia Molps ou importando novas ontologias para a Molps.

8.1.3 Agentes de *Software*

Os agentes de *software* com sua característica de autonomia proporcionam ao sistema uma abstração de nível mais alto. Os agentes de *software* mostram-se adaptativos e mais preparados para lidar em sistemas. Por possuir um comportamento em sua definição, os agentes reagem ao ambiente atingindo seus objetivos.

No trabalho realizado pode-se definir no comportamento dos agentes funções que basicamente formam todo o funcionamento do sistema. Estes agentes foram capazes de obter conhecimento na ontologia para construir o processo de desenvolvimento de *software* e realizar as consultas.

8.1.4 Métodos Ágeis e PMBOK

O foco dos métodos ágeis é na necessidade do cliente, a entrega rápida, reconhecimento da capacidade dos indivíduos e, principalmente, a adaptação a ambientes de negócio bastante dinâmicos. Os métodos ágeis utilizados nesse trabalho, XP, *Scrum* e FDD, possuem técnicas e habilidades para atender estes objetivos.

Segundo estudos, XP e *Scrum* são as mais adotadas e que mais agregam valor, podendo estar combinadas ou isoladas. Neste trabalho uniram-se estas duas e mais o FDD por ser um processo altamente adaptativo e produz resultados frequentes. A união tornou o domínio da ontologia mais completo assim as características que se diferenciavam entre os conceitos acabaram se completando tornando um processo mais rico.

Para aumentar o nível de conhecimento e técnicas a essas metodologias ágeis, o PMBOK contribuiu fornecendo gerência ao processo de desenvolvimento. Além das técnicas das metodologias ágeis, o PMBOK consistiu neste trabalho como uma técnica para projetar as atividades que visam atingir requisitos definidos. Então unindo estes conceitos, se pode chegar a um domínio em que processos derivados deste domínio pudessem ser mais completos e ricos em etapas do que seguindo apenas um deles.

8.2 CONCLUSÃO

Esta dissertação modelou, implementou e validou o sistema que cria um processo de desenvolvimento de *software*, que se utiliza de uma arquitetura dedicada com base nos conceitos de ontologia, linha de produto de *software*, métodos ágeis, gerência de projeto de *software* e agentes de *software*.

O estabelecimento do cenário de aplicação como sendo uma instituição real e ativa contribuiu para a validação do sistema. O levantamento dos dados e informações nos registros internos da instituição, e o posterior cadastramento destes dados na ontologia, mostraram que este modelo foi flexível o bastante para ser utilizado pela instituição. O setor de TI responsável pelas requisições de *software* da instituição não possuía um processo de desenvolvimento de *software* o que se tornou uma oportunidade de testar este sistema. Os experimentos realizados abrangeram informações suficientes para realizar os testes e validar o sistema.

A ontologia mostrou-se uma base de informações muito importante, pois seu domínio semântico garantiu que as consultas realizadas pelos agentes de *software* atingissem os

objetivos esperados. Com a capacidade de inferir no domínio da ontologia, informações implícitas nas consultas se tornaram explícitas em seus resultados.

Com a possibilidade de criar processo de desenvolvimento de *software*, armazenar informações para posteriores consultas e gerações de gráficos, o sistema construído procurou mostrar que a solução se mantém viável para atender a questão de pesquisa levantada, bem como seus objetivos estabelecidos.

8.3 TRABALHOS FUTUROS

A sugestão de trabalhos futuros tem por objetivo complementar o estudo realizado, e incrementar a solução proposta. Apresentam-se a seguir, algumas sugestões para continuidade deste trabalho.

8.3.1 Aumentar o Domínio de Conhecimento da Ontologia que Define a Arquitetura da LPS

Atualmente a ontologia possui apenas as metodologias XP, *Scrum*, FDD e o método de gerência PMBOK como domínio de conhecimento. Como melhorias deste trabalho, poderiam ser acrescentadas outras metodologias para criar um processo de desenvolvimento de *software* abordando todas as melhores características de várias metodologias. Para acrescentar outras metodologias, podem ser importadas novas ontologias que definem alguma metodologia ou aumentar a ontologia Molps criando classes novas no domínio de conhecimento.

8.3.2 Desenvolver uma Ferramenta de Gerência de Projeto Mais Completa

Como pode ser observado, o foco deste trabalho não foi uma ferramenta que gerencie todo o projeto. Neste trabalho foram desenvolvidos apenas consultas e gráficos para acompanhar partes principais do projeto. Então como trabalhos futuros poderiam ser desenvolvidos mais controles e melhorias em consultas sobre as informações armazenadas no projeto. Alguns controles sobre prazos de atividades, ciclos e itens de desenvolvimento poderiam ser criados, acrescentando uma função de avisos das datas que poderiam ser enviadas por email ao gerente do projeto. A criação de um relatório rico em informações sobre o projeto também pode ser acrescentado no sistema.

REFERÊNCIAS

ABRAHAMSSON, P., SALO, O., RONKAINEN, J., WARSTA, J., *Agile Software Development Methods: Review and Analysis*. VTT, *Publication, Finland*, 2002, p.1-107.

ARAUJO, Leonardo B. P. Estudo comparativo da compatibilidade entre as melhores práticas PMI e *Scrum*. São Paulo, 2009, p.1-88.

BECHHOFER, Sean; HORROCKS, Ian; PATEL-SCHNEIDER, Peter F. OWL *Web Ontology Language*. In: World Wide Web *Consortion* – W3C, 2004.

BELLIFEMINE, Fabio; CAIRE, Giovanni; GREENWOOD, Dominic. *Developing multi-agent systems with* JADE. Wiley: 2007, p1-300.

BERNERS-LEE, Tim; HENDLER, James; LASSILA, Ora. *The Semantic Web, Scientific American*, p.34-43, 2001.

BERNY, Vanessa; ADAMATTI, Diana F.; GOMES, Daniela F.; COSTA, Antonio C.R. Utilização de metodologias para desenvolvimento de agentes: um estudo de caso na microeconomia. Vetor, Rio Grande, v.16, n.2, p. 55-70, 2006.

BOSH, Jan. *Adopting Software Product Lines:Approaches, Artefacts and Organization. University of Groningen, Pearson Education* (*Addison-Wesley* & ACM *Press*), p.1-4, ISBN 0-201-67494-7,2000.

BUBLITZ, Jorge L. Desenvolvimento Ágil dirigido a funcionalidades. Trabalho de Conclusão de Curso de Pós-Graduação Latu Sensu em Engenharia de Sistemas. ESAB, Cuiabá 2009.

CAETANO, Rodrigo. Metodologias de desenvolvimento: qual a mais adequada?- Computerworld, 2009

CAMPOS, Maria L. A. GOMES, Hagar E. Taxonomia e Classificação: o princípio de categorização, v.9, n.4. 2008.

CERAVOLO, Paolo; DAMIANI, Ernesto; MARCHESI, Michele; PINNA, Sandro. *A Ontology-based Process Modelling for XP. Proceedings of the Tenth Asia-Pacific Software Engineering Conference*, 2003, p.236 - 242.

CLEMENTS, P., NORTHROP, L. *Software Product Lines: Practices and Patterns*, SEI *Series in Software Engineering, Addison-Wesley*. 563 p., 2001.

CLEMENTS, Paul; NORTHROP, Linda. *Software Product Lines: Practices and Patterns.* Boston: *Addison-Wesley*, 2002, 563 p.

COSTA, Gustavo H. C. Engenharia de Requisitos no desenvolvimento de *Software* Ágil. (Trabalho de Pós-Graduação). Universidade Federal de Pernambuco, 2010, p.1-45.

CZARNECKI, KRZYSZTOF. EISENECKER, ULRICH . *GENERATIVE PROGRAMMING: METHODS, TOOLS, AND APPLICATIONS. ADDISON WESLEY*, 2000, 864P.

DÁRIO, Cláudia F. B. Uma Metodologia Unificada para o Desenvolvimento de Sistemas Orientados a Agentes .Dissertação de Mestrado. São Paulo, 2005, 176p.

DeLOACH, Scott A. *Engineering Organization-Based Multiagent Systems*. In: GARCIA, Alessandro F. et al. SELMAS 2005. *Springer-Verlag Berlin Heidelberg*, p.109-125, 2006.

DeLOACH, Scott A.; WOOD, Mark F. *Developing Muiltiagent Systems with agentTool. In: Proceedings of Lecture Notes in Artificial Intelligence. Springer – Verlag. Berling*, 2001.

DIAS, D. Tatiane. SANTOS, Neide. Web Semântica: Conceitos Básicos e Tecnologias Associadas. Básicos e Tecnologias Associadas. Disponível em:<http://magnum.ime.uerj.br/cadernos/cadinf/vol14/7-neide.pdf>. Acesso em: 2011.

DICKINSON, Ian. *Jena Ontology* API. Disponível em: http://jena.sourceforge.net/ontology/index.html. Acesso em: Abril, 2010

DURSCKI, Roberto C.; SPINOLA, Mauro M. Linhas de Produto de *Software*: riscos e vantagens de sua Implantação. VI Simpósio Internacional de Melhoria de Processos de *Software*. São Paulo, 2004,p.155-166.

ELUAN, Andrenizia A.; FACHIN, Gleisy; GAUTHIER, Fernando; TEDESCO, José. Web Semântica no Ensino à Distância. Universidade Federal de Santa Catarina, 2007.

FALBO, Ricardo. A. GUIZZARDI, G. DUARTE, K. C. *An Ontological Approach to Domain Engineering . Proceedings of the 14th Internatio nal Conference on Software Engineering and Knowledge Engineering*, SEKE'2002, p. 351 - 358, Ischia, Italy, 2002.

FALBO, Ricardo A. Gerência de Projetos de *Software*. Disponivel em: http://www.inf.ufes.br/~falbo/files/GerenciaProjetos*Software*.pdf. Acesso em: 2011

FIPA, "*Java Agent Development Framework*". Disponível em <http://jade.tilab.com> Acesso em: 2011

FOWLER, Martin. A nova Metodologia. Disponível em: http://simplus.com.br/artigos/a-nova-metodologia/#N58. Acesso em: 2011.

FRAKES, W.B. KANG, K. *Software Reuse research: Status and future.* IEEE *Trans. Software Eng*., 31: 529-536, 2005.

GEBER, Ramalho. *Extreme Programming* (XP). Disponível em: http://www.cin.ufpe.br/~gamr/FAFICA/Desenvolvimento%20de%20sistemas/XP.pdf. Acesso em: 2011.

GIMENES, Itana M. S. TRAVASSOS, Guilherme H. O Enfoque de Linha de Produto para Desenvolvimento de *Software*, 2002.

GIULI, RICHARD. *TUTORIAL FOR USING SPARQL WITH OWL*. 2006. DISPONÍVEL EM: HTTPS://MAILMAN.STANFORD.EDU/PIPERMAIL/PROTEGE-OWL/2006-SEPTEMBER/000082.HTML. ACESSO EM: 2011.

GONÇALVES, Eduardo S. FILHO, Heitor B. R. Ferramenta para Gerenciamento de Requisitos em Metodologias Ágeis. V.32, n.63, 2008, p.148-155.

GOUVEIA, Vitorino, A. G. Aplicação de uma Linha De Produtos de *Software* (LPS) no Contexto de uma PME .Portugal, 2007.

HANSSEN, Geir K. FAEQRI, Tor E. *Process fusion: An industrial case study on agile software product line engineering. Journal of Systems and Software* - JSS , vol. 81, no. 6, p. 843-854, 2008.

HENDERSON-SELLERS, Brian; GIORGINI, Paolo. *Agent-Oriented Methodologies. Idea Group* Inc: 2005.

HEPTAGON, TI LDTA. FDD - *Feature Driven Development*. Disponivel em: http://www.heptagon.com.br/fdd. Acesso em: 2011.

HIGHSMITH, James A. Adaptive *Software* Development. Dorset House Publishing, 1996.

HORRIDGE, Matthew. A Practical Guide To Building OWL Ontologies using the Protege-OWL plugin and CO-ODE Tools, Edition 1.0 , 2004

HUIJBERS, Rico; LEMMENS, Funs; SENDERS, Bram; SIMONS, Sjoerd; SPANN, Bert; TILBURG, Paul; VOSSEN, Koen. *Software Project Management: Methodologies & Techniques*. SE Project, 2004.

JARDIM, André. . Aplicações de Modelos Semânticos em Redes Sociais, 2010.

JR LIMA, Rogério A. Comparação Entre Ferramentas para Linha de Produtos De *Software*. Recife, 2008.

JURISON, Jaak. *Software Project Management: The Manager's View*. *Comunications of the Association for Information System*, vol. 2, Article 17, 1999.

KANG, Kyo; COHEN, Sholom G.; NOVAK, Willian E; PETERSON, Spencer. *Feature-Oriented Domain Analysis* (FODA) *Feasibility Study*, (CMU/SEI-90TR-21, ADA 235785), *Pittsburgh*, PA:SEI CMU, 1990.

LARMAN, Craig. Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e ao Processo Unificado. Bookman, 2007.

MACHADO, Cristina A. F.; BURNETT, Robert. Gerência de Projetos na Engenharia de *Software* em Relação as práticas do PMBOK, 2002.

MALLMANN, Paulo R. Um modelo abstrato de Gerência de *Software* para Metodologias Ágeis. Dissertação de Mestrado. Unisinos, São Leopoldo 2011.

MANOLA, Frank.; MILLER, Eric. RDF *Primer*. In: World Wide Web *Consortim* – W3C, 2004.

MARÇAL , Ana C. S.; FREITAS, Bruno; SOARES, Felipe; MACIEL, Teresa; BELCHIOR, Arnaldo. Estendendo o *SCRUM* segundo as Áreas de Processo de Gerenciamento de Projetos do CMMI. CLEI: XXXIII Conferencia Latino Americana de Informática, San Jose, Costa Rica, 9–12, 2007.

MARTINS, J.C.C., Gerenciando Projetos de Desenvolvimento de *Software* com PMI, RUP e UML, 2a edição revisada, Rio de Janeiro: Brasport, 2005.

NOY, Natalya F.; MCGUINNESS, Deborah L. Ontology development 101: A guide to creating your first ontology. In: Semantic Web Working Symposium, 2001.

PADGHAM L.; WINIKOFF M. Prometheus: a pragmatic methodology for engineering intelligent agents. In workshop on agent-oriented methodologies, pages 97–108, 2002

PÉREZ-GÓMEZ, Asunción; FERNÁNDEZ-LÓPEZ, Mariano; CORCHO, Oscar. *Ontological Engineering. Springer-Verlag London Limited*: 2004.

PERIN, Alexandre; ERDTMANN, Mathias; DEITOS, Rafael. Inteligência Artificial e Web Semântica, 2007.

POHL, Klaus; BÖCKLE, Günter; FRANK, J. *Software Product Line Engineering: foundations, principles and techniques. Springer*, 2005

PRASS, Fernando S. PUNTEL, Márcio D. Metodologias Ágeis: Um *overview* sobre FDD, MSF, *Scrum* e XP. Disponível em: http://www.devmedia.com.br/websys.2/webreader.asp?cat=1&revista=netmagazine_68#a-2048. Acesso em: 2011.

PRESSMAN, Roger S. Engenharia de *Software*, McGraw-Hill, 6ª edição. São Pauo, 2006.

PRESSMAN, Roger S. *Software Engineering - A practitioner's Approach*. 5ª edição, McGraw-Hill, 2001.

QUEIROZ, Jefferson G. *Scrum* para Gerenciamento de Projetos de Tecnologia da Informação em pequenas empresas de tecnologia da informação.Trabalho de conclusão de curso. 2010.

RAMOS, Marcelo A. PENTEADO, Rosângela A. D. Princípios Ágeis Aplicados à Reengenharia de *Software*, 2007.

RAWSTHORNE, Dan. *Scrum in a Nutshell*. 2009. Disponível em:http://www.*scrum*alliance.org/articles/151-*scrum*-in-a-nutshell. Acesso em: 2011

ROSCH, Eleanor. Principles of Categorization, University of California, Berkeley 1978.

SANTOS, Jorge; QUINTILIANO, Leandro L.; BARBOSA, Lilian; SILVA, Paulo R. H.; GIRALDES, Wanderley Jr. Métodos Ágeis. Disponível em: http://www.devmedia.com.br/articles/viewcomp.asp?comp=9443. Acesso em 2011.

SCHNEIDER, Ricardo Luiz. Fundamentos da Engenharia de *Software*. Disponível em: http://www.dcc.ufrj.br/~schneide/es/2000/1/. Acesso em: 2011.

SILVA F. G. et al. Uma análise das Metodologias Ágeis FDD e *Scrum* sob a Perspectiva do Modelo de Qualidade MPS.BR. Scientia Plena, vol. 5., Nº 12, 2009.

SILVA, Alan P. Uma Linha de Produto de *Software* baseada na Web Semântica para Sistemas Tutores Inteligentes.Tese de Doutorado. Paraíba, 2011.

SIMÃO, Inês C. N. Adaptação da Abordagem *Theme* para Linhas de Produtos de *Software*. Dissertação de Mestrado em Engenharia Informática, 2010, 151p.

SIMIDCHIEVA, B. I. Clarke, L.A. Osterwiel, L. J. Representing Process Variation with a Process Family. Laboratory for Advanced *Software* Engineering Research (LASER), University of Massachusetts at Amherst, 2007.

SQUARE, Newtown. Um Guia do Conhecimento em Gerenciamento de Projetos (Guia PMBOK). 4ª edição. Pennsylvania, 2008.

SOARES, Michel S. Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de *Software*. Disponível em http://www.dcc.ufla.br/infocomp/artigos/v3.2/art02.pdf em 04/05/2011. Acesso em: 2011

SOUZA, Luciano M. Método Ágil XP (*Extreme Programming*). Revista Eletrônica da FIA, volume 3, nº 3, 2007.

THUSHAR, A.K. THILAGAM, Santhi P. *An RDF Approach for Discovering the Relevant Semantic Associations in a Social Network.* Índia, 2009.

VAZQUEZ, C.E., SIMÕES, G.S., ALBERT, R.M. Análise de Pontos de Função: Medição, Estimativas e Gerenciamento de Projetos de *Software*, 3a edição, São Paulo: Editora Érica, 2005, 232p.

VERNON, Roger. *Agile scrum process overview*. 2010. Disponível em: http://www.suite101.com/content/agile-*scrum*-process-basics-overview-a264765. Acesso em: 2011.

VIANA, Antônio G. G. Gerenciamento de projetos em processo ágil de desenvolvimento de *software*. Disponível em: http://www.techoje.com.br/site/techoje/categoria/impressao_artigo/393. Acesso em: 2011.

WEISS, G. *Multiagent Systems: A Modern Approach to Distributed Modern Approach to Artificial Intellingence*, The MIT *Press*, 1999, 609p.

WERNER, C. M. L. e BRAGA, R. M. M. "A Engenharia de Domínio e o Desenvolvimento Baseado em Componentes". In: Desenvolvimento Baseado em Componentes: Conceitos e Técnicas,Ciência Moderna, 2005.

WERNECK ,V.M. B.; PEREIRA L. F.; SILVA T. S.; ALMENTERO E. K.; CYSNEIROS L. M. Uma Avaliação da Metodologia MAS-CommonKADS. In: *Proceedings of the Second Workshop on Software Engineering for Agent-oriented Systems*, (SEAS´06), Florianópolis, 2006, pp. 13-24.

WOOLDRIDGE, Michal. *An Introduction to Multi-Agent Systems. John University of Liverpool, United Kingdom*. Wiley & Sons Ltda: 2002, 461p.

ZHI-GEN, Hu. et al. *Research on Agile Project Management with Scrum method. In: International Conference on Services Science, Management and Engineering*, p. 26-29. 2009.

APÊNDICE A – Ontologia Molps

1) Ontologia Molps

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [
   <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
   <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
   <!ENTITY fdd "http://www.ontolps.com.br/fdd.owl#" >
   <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
   <!ENTITY scrum "http://www.ontolps.com.br/scrum.owl#" >
   <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
   <!ENTITY ontolps "http://www.ontolps.com.br/ontolps.owl#" >
   <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
   <!ENTITY xtremeprogramming
"http://www.ontolps.com.br/xtremeprogramming.owl#" >
   <!ENTITY pmbok "http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#"
>
]>

<rdf:RDF xmlns="http://www.ontolps.com.br/ontolps.owl#"
     xml:base="http://www.ontolps.com.br/ontolps.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
     xmlns:scrum="http://www.ontolps.com.br/scrum.owl#"
     xmlns:ontolps="http://www.ontolps.com.br/ontolps.owl#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

xmlns:xtremeprogramming="http://www.ontolps.com.br/xtremeprogramming.owl#"
     xmlns:pmbok="http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#"
     xmlns:fdd="http://www.ontolps.com.br/fdd.owl#">
    <owl:Ontology rdf:about=""/>

    <!--
    ///////////////////////////////////////////////////////////////////////////////
    //
    // Object Properties
    //
    ///////////////////////////////////////////////////////////////////////////////
     -->



    <!-- http://www.ontolps.com.br/fdd.owl#hascomprehensiveModel -->
```

```
    <owl:ObjectProperty rdf:about="&fdd;hascomprehensiveModel">
        <rdf:type rdf:resource="&owl;TransitiveProperty"/>
        <rdfs:range rdf:resource="&fdd;ComprehensiveModel"/>
        <rdfs:domain rdf:resource="#ProjectMolps"/>
    </owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasActivity -->

    <owl:ObjectProperty rdf:about="#hasActivity">
        <rdf:type rdf:resource="&owl;TransitiveProperty"/>
        <rdfs:range rdf:resource="&pmbok;Activity"/>
        <rdfs:domain rdf:resource="&pmbok;Phase"/>
    </owl:ObjectProperty>

    <!-- http://www.ontolps.com.br/ontolps.owl#hasActivityAtr -->

    <owl:ObjectProperty rdf:about="#hasActivityAtr">
        <rdf:type rdf:resource="&owl;TransitiveProperty"/>
        <rdfs:range rdf:resource="#AtrActivityDate"/>
        <rdfs:range rdf:resource="#AtrActivityDateF"/>
        <rdfs:range rdf:resource="#AtrActivityDesc"/>
        <rdfs:range rdf:resource="#AtrActivityName"/>
        <rdfs:range rdf:resource="#AtrActivityPhase"/>
        <rdfs:range rdf:resource="#AtrActivitySit"/>
        <rdfs:domain rdf:resource="&pmbok;Activity"/>
    </owl:ObjectProperty>

    <!-- http://www.ontolps.com.br/ontolps.owl#hasCicleMolps -->

    <owl:ObjectProperty rdf:about="#hasCicleMolps">
        <rdf:type rdf:resource="&owl;TransitiveProperty"/>
        <rdfs:range rdf:resource="#CicleMolps"/>
        <rdfs:domain rdf:resource="&xtremeprogramming;Release"/>
    </owl:ObjectProperty>

    <!-- http://www.ontolps.com.br/ontolps.owl#hasCicleMolpsAtr -->

    <owl:ObjectProperty rdf:about="#hasCicleMolpsAtr">
        <rdf:type rdf:resource="&owl;TransitiveProperty"/>
        <rdfs:range rdf:resource="#AtrCicleMolpsDataI"/>
        <rdfs:range rdf:resource="#AtrCicleMolpsDateF"/>
        <rdfs:range rdf:resource="#AtrCicleMolpsDescription"/>
        <rdfs:range rdf:resource="#AtrCicleMolpsRelease"/>
        <rdfs:range rdf:resource="#AtrCicleMolpsSit"/>
        <rdfs:domain rdf:resource="#CicleMolps"/>
    </owl:ObjectProperty>

    <!-- http://www.ontolps.com.br/ontolps.owl#hasComprehensiveModelAtr -->

    <owl:ObjectProperty rdf:about="#hasComprehensiveModelAtr">
```

```
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:domain rdf:resource="&fdd;ComprehensiveModel"/>
    <rdfs:range rdf:resource="#AtrComprehensiveModelArea"/>
    <rdfs:range rdf:resource="#AtrComprehensiveModelDomain"/>
    <rdfs:range rdf:resource="#AtrComprehensiveModelProject"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasCore -->

<owl:ObjectProperty rdf:about="#hasCore">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="&pmbok;Core"/>
    <rdfs:domain rdf:resource="&pmbok;KnowledgeArea"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasCoreAtr -->

<owl:ObjectProperty rdf:about="#hasCoreAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrCore"/>
    <rdfs:domain rdf:resource="&pmbok;Core"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasEquipament -->

<owl:ObjectProperty rdf:about="#hasEquipament">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="&pmbok;Equipament"/>
    <rdfs:domain rdf:resource="&pmbok;PhisicalResource"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasEquipamentAtr -->

<owl:ObjectProperty rdf:about="#hasEquipamentAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrEquipament"/>
    <rdfs:domain rdf:resource="&pmbok;Equipament"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasExternalMember -->

<owl:ObjectProperty rdf:about="#hasExternalMember">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="&pmbok;ExternalMember"/>
    <rdfs:domain rdf:resource="&pmbok;Relevant"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasExternalMemberAtr -->

<owl:ObjectProperty rdf:about="#hasExternalMemberAtr">
```

```
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrExternalMember"/>
    <rdfs:range rdf:resource="#AtrExternalMemberFunc"/>
    <rdfs:domain rdf:resource="&pmbok;ExternalMember"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasFacilitating -->

<owl:ObjectProperty rdf:about="#hasFacilitating">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="&pmbok;Facilitating"/>
    <rdfs:domain rdf:resource="&pmbok;KnowledgeArea"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasFacilitatingAtr -->

<owl:ObjectProperty rdf:about="#hasFacilitatingAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrFacilitating"/>
    <rdfs:domain rdf:resource="&pmbok;Facilitating"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasFacility -->

<owl:ObjectProperty rdf:about="#hasFacility">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="&pmbok;Facility"/>
    <rdfs:domain rdf:resource="&pmbok;PhisicalResource"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasFacilityAtr -->

<owl:ObjectProperty rdf:about="#hasFacilityAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrFacility"/>
    <rdfs:domain rdf:resource="&pmbok;Facility"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasFuncionality -->

<owl:ObjectProperty rdf:about="#hasFuncionality">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="&fdd;Funcionality"/>
    <rdfs:domain rdf:resource="#RequirementsMolps"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasFuncionalityAtr -->

<owl:ObjectProperty rdf:about="#hasFuncionalityAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
```

```
    <rdfs:domain rdf:resource="&fdd;Funcionality"/>
    <rdfs:range rdf:resource="#AtrFuncionalityDescription"/>
    <rdfs:range rdf:resource="#AtrFuncionalityPercentCompleted"/>
    <rdfs:range rdf:resource="#AtrFuncionalityRequirementsMolps"/>
    <rdfs:range rdf:resource="#AtrFuncionalityStake"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasGuidance -->

<owl:ObjectProperty rdf:about="#hasGuidance">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:domain rdf:resource="&pmbok;Activity"/>
    <rdfs:range rdf:resource="&pmbok;Guidance"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasGuidanceAtr -->

<owl:ObjectProperty rdf:about="#hasGuidanceAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrGuidanceActivity"/>
    <rdfs:range rdf:resource="#AtrGuidanceName"/>
    <rdfs:range rdf:resource="#AtrTechnique"/>
    <rdfs:range rdf:resource="#AtrTool"/>
    <rdfs:domain rdf:resource="&pmbok;Guidance"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasHistory -->

<owl:ObjectProperty rdf:about="#hasHistory">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:domain rdf:resource="#RequirementsMolps"/>
    <rdfs:range rdf:resource="&xtremeprogramming;History"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasHistoryAtr -->

<owl:ObjectProperty rdf:about="#hasHistoryAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrHistoryDate"/>
    <rdfs:range rdf:resource="#AtrHistoryDescription"/>
    <rdfs:range rdf:resource="#AtrHistoryRequirementsMolps"/>
    <rdfs:domain rdf:resource="&xtremeprogramming;History"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasKnowledgeArea -->

<owl:ObjectProperty rdf:about="#hasKnowledgeArea">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="&pmbok;KnowledgeArea"/>
    <rdfs:domain rdf:resource="&pmbok;ManagementProcess"/>
```

```
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasKnowledgeAreaAtr -->

<owl:ObjectProperty rdf:about="#hasKnowledgeAreaAtr">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="#AtrCore"/>
   <rdfs:range rdf:resource="#AtrFacilitating"/>
   <rdfs:range rdf:resource="#AtrKnowledgeAreaName"/>
   <rdfs:domain rdf:resource="&pmbok;KnowledgeArea"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasManagementProcess -->

<owl:ObjectProperty rdf:about="#hasManagementProcess">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:domain rdf:resource="&pmbok;Activity"/>
   <rdfs:range rdf:resource="&pmbok;ManagementProcess"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasManagementProcessAtr -->

<owl:ObjectProperty rdf:about="#hasManagementProcessAtr">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="#AtrManagementProcessActivity"/>
   <rdfs:range rdf:resource="#AtrManagementProcessName"/>
   <rdfs:domain rdf:resource="&pmbok;ManagementProcess"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasMaterial -->

<owl:ObjectProperty rdf:about="#hasMaterial">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="&pmbok;Material"/>
   <rdfs:domain rdf:resource="&pmbok;PhisicalResource"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasMaterialAtr -->

<owl:ObjectProperty rdf:about="#hasMaterialAtr">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="#AtrMaterial"/>
   <rdfs:domain rdf:resource="&pmbok;Material"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasMeetingMolps -->

<owl:ObjectProperty rdf:about="#hasMeetingMolps">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:domain rdf:resource="#CicleMolps"/>
```

```
    <rdfs:range rdf:resource="#MeetingMolps"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasMeetingMolpsAtr -->

<owl:ObjectProperty rdf:about="#hasMeetingMolpsAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrMeetingMolpsCicle"/>
    <rdfs:range rdf:resource="#AtrMeetingMolpsDate"/>
    <rdfs:range rdf:resource="#AtrMeetingMolpsDiscussion"/>
    <rdfs:range rdf:resource="#AtrMeetingMolpsTitle"/>
    <rdfs:domain rdf:resource="#MeetingMolps"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasOrganizationAtr -->

<owl:ObjectProperty rdf:about="#hasOrganizationAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrOrganizationEndereco"/>
    <rdfs:range rdf:resource="#AtrOrganizationName"/>
    <rdfs:domain rdf:resource="&pmbok;Organization"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasOther -->

<owl:ObjectProperty rdf:about="#hasOther">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="&pmbok;Other"/>
    <rdfs:domain rdf:resource="&pmbok;PmbokStakeholder"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasOtherAtr -->

<owl:ObjectProperty rdf:about="#hasOtherAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrOther"/>
    <rdfs:domain rdf:resource="&pmbok;Other"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasPhase -->

<owl:ObjectProperty rdf:about="#hasPhase">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:domain rdf:resource="#ProjectMolps"/>
    <rdfs:range rdf:resource="&pmbok;Phase"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasPhaseAtr -->

<owl:ObjectProperty rdf:about="#hasPhaseAtr">
```

```
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrPhaseName"/>
    <rdfs:range rdf:resource="#AtrPhaseProject"/>
    <rdfs:domain rdf:resource="&pmbok;Phase"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasPhisicalResource -->

<owl:ObjectProperty rdf:about="#hasPhisicalResource">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="&pmbok;PhisicalResource"/>
    <rdfs:domain rdf:resource="&pmbok;Resource"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasPhisicalResourceAtr -->

<owl:ObjectProperty rdf:about="#hasPhisicalResourceAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrEquipament"/>
    <rdfs:range rdf:resource="#AtrFacility"/>
    <rdfs:range rdf:resource="#AtrMaterial"/>
    <rdfs:range rdf:resource="#AtrPhysicalResourceName"/>
    <rdfs:range rdf:resource="#AtrPhysicalResourceResource"/>
    <rdfs:domain rdf:resource="&pmbok;PhisicalResource"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasPlanning -->

<owl:ObjectProperty rdf:about="#hasPlanning">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:domain rdf:resource="#CicleMolps"/>
    <rdfs:range rdf:resource="#Planning"/>
</owl:ObjectProperty>
    <!-- http://www.ontolps.com.br/ontolps.owl#hasPlanningAtr -->

<owl:ObjectProperty rdf:about="#hasPlanningAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrPlanningCicle"/>
    <rdfs:range rdf:resource="#AtrPlanningDate"/>
    <rdfs:range rdf:resource="#AtrPlanningObjective"/>
    <rdfs:domain rdf:resource="#Planning"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasProcessGroupAtr -->

<owl:ObjectProperty rdf:about="#hasProcessGroupAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrProcessGroupName"/>
    <rdfs:domain rdf:resource="&pmbok;ProcessGroup"/>
</owl:ObjectProperty>
```

```
<!-- http://www.ontolps.com.br/ontolps.owl#hasProductBacklogAtr -->

<owl:ObjectProperty rdf:about="#hasProductBacklogAtr">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="#AtrProductDescription"/>
   <rdfs:range rdf:resource="#AtrProductPriority"/>
   <rdfs:range rdf:resource="#AtrProductProject"/>
   <rdfs:range rdf:resource="#AtrProductStakeholder"/>
   <rdfs:domain rdf:resource="&scrum;ProductBacklog"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasProductBacklogMolps -->

<owl:ObjectProperty rdf:about="#hasProductBacklogMolps">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:domain rdf:resource="#ProjectMolps"/>
   <rdfs:range rdf:resource="&scrum;ProductBacklog"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasProductiveCicleMolps -->

<owl:ObjectProperty rdf:about="#hasProductiveCicleMolps">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:domain rdf:resource="#CicleMolps"/>
   <rdfs:range rdf:resource="#ProductiveCicleMolps"/>
   <rdfs:domain rdf:resource="#RequirementsMolps"/>
   <rdfs:domain rdf:resource="&scrum;ProductBacklog"/>
</owl:ObjectProperty>


<!-- http://www.ontolps.com.br/ontolps.owl#hasProductiveCicleMolpsAtr -->

<owl:ObjectProperty rdf:about="#hasProductiveCicleMolpsAtr">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="#AtrProductiveCicleMolpsCicle"/>
   <rdfs:range rdf:resource="#AtrProductiveCicleMolpsDescription"/>
   <rdfs:range rdf:resource="#AtrProductiveCicleMolpsPriority"/>
   <rdfs:range rdf:resource="#AtrProductiveCicleMolpsProductBacklog"/>
   <rdfs:range rdf:resource="#AtrProductiveCicleMolpsRequirementsMolps"/>
   <rdfs:range rdf:resource="#AtrProductiveCicleMolpsStakeholder"/>
   <rdfs:domain rdf:resource="#ProductiveCicleMolps"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasProgram -->

<owl:ObjectProperty rdf:about="#hasProgram">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:domain rdf:resource="&pmbok;Organization"/>
   <rdfs:range rdf:resource="&pmbok;Program"/>
```

```
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasProgramAtr -->

<owl:ObjectProperty rdf:about="#hasProgramAtr">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="#AtrProgramName"/>
   <rdfs:range rdf:resource="#AtrProgramOrganization"/>
   <rdfs:domain rdf:resource="&pmbok;Program"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasProjectMolps -->

<owl:ObjectProperty rdf:about="#hasProjectMolps">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="#ProjectMolps"/>
   <rdfs:domain rdf:resource="&pmbok;Program"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasProjectMolpsAtr -->

<owl:ObjectProperty rdf:about="#hasProjectMolpsAtr">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="#AtrProjectDate"/>
   <rdfs:range rdf:resource="#AtrProjectDateF"/>
   <rdfs:range rdf:resource="#AtrProjectDescription"/>
   <rdfs:range rdf:resource="#AtrProjectManager"/>
   <rdfs:range rdf:resource="#AtrProjectName"/>
   <rdfs:range rdf:resource="#AtrProjectProgram"/>
   <rdfs:domain rdf:resource="#ProjectMolps"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#has Release Atr -->

<owl:ObjectProperty rdf:about="#has Release Atr">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="#Atr Release Date"/>
   <rdfs:range rdf:resource="#Atr Release Name"/>
   <rdfs:range rdf:resource="#Atr Release Project"/>
   <rdfs:domain rdf:resource="&xtremeprogramming;Release"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#has Release Molps -->

<owl:ObjectProperty rdf:about="#has Release Molps">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:domain rdf:resource="#ProjectMolps"/>
   <rdfs:range rdf:resource="&xtremeprogramming;Release"/>
</owl:ObjectProperty>
```

```
<!-- http://www.ontolps.com.br/ontolps.owl#hasRelevant -->

<owl:ObjectProperty rdf:about="#hasRelevant">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:domain rdf:resource="&pmbok;PmbokStakeholder"/>
   <rdfs:range rdf:resource="&pmbok;Relevant"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasRelevantAtr -->

<owl:ObjectProperty rdf:about="#hasRelevantAtr">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="#AtrRelevant"/>
   <rdfs:domain rdf:resource="&pmbok;Relevant"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasRequirementsMolps -->

<owl:ObjectProperty rdf:about="#hasRequirementsMolps">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="#RequirementsMolps"/>
   <rdfs:domain rdf:resource="&pmbok;Activity"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasRequirementsMolpsAtr -->

<owl:ObjectProperty rdf:about="#hasRequirementsMolpsAtr">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="#AtrRequirementsMolpsActivity"/>
   <rdfs:range rdf:resource="#AtrRequirementsMolpsDataI"/>
   <rdfs:range rdf:resource="#AtrRequirementsMolpsDateF"/>
   <rdfs:range rdf:resource="#AtrRequirementsMolpsEstimate"/>
   <rdfs:range rdf:resource="#AtrRequirementsMolpsSit"/>
   <rdfs:range rdf:resource="#AtrRequirementsMolpsTitle"/>
   <rdfs:domain rdf:resource="#RequirementsMolps"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasRequisites -->

<owl:ObjectProperty rdf:about="#hasRequisites">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:domain rdf:resource="&fdd;ComprehensiveModel"/>
   <rdfs:range rdf:resource="&fdd;Requisites"/>
   <rdfs:domain rdf:resource="&pmbok;Activity"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasRequisitesAtr -->

<owl:ObjectProperty rdf:about="#hasRequisitesAtr">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
```

```
    <rdfs:domain rdf:resource="&fdd;Requisites"/>
    <rdfs:range rdf:resource="#AtrRequisitesActivity"/>
    <rdfs:range rdf:resource="#AtrRequisitesComprehensiveM"/>
    <rdfs:range rdf:resource="#AtrRequisitesDesc"/>
    <rdfs:range rdf:resource="#AtrRequisitesName"/>
    <rdfs:range rdf:resource="#AtrRequisitesPriority"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasResource -->

<owl:ObjectProperty rdf:about="#hasResource">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:domain rdf:resource="&pmbok;Activity"/>
    <rdfs:range rdf:resource="&pmbok;Resource"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasResourceAtr -->

<owl:ObjectProperty rdf:about="#hasResourceAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrResourceActivity"/>
    <rdfs:domain rdf:resource="&pmbok;Resource"/>
</owl:ObjectProperty>


<!-- http://www.ontolps.com.br/ontolps.owl#hasRetrospective -->

<owl:ObjectProperty rdf:about="#hasRetrospective">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:domain rdf:resource="#CicleMolps"/>
    <rdfs:range rdf:resource="#Retrospective"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasRetrospectiveAtr -->

<owl:ObjectProperty rdf:about="#hasRetrospectiveAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrRetrospectiveAnalisys"/>
    <rdfs:range rdf:resource="#AtrRetrospectiveCicle"/>
    <rdfs:range rdf:resource="#AtrRetrospectiveDate"/>
    <rdfs:domain rdf:resource="#Retrospective"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasReview -->

<owl:ObjectProperty rdf:about="#hasReview">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:domain rdf:resource="#CicleMolps"/>
    <rdfs:range rdf:resource="#Review"/>
</owl:ObjectProperty>
```

```
<!-- http://www.ontolps.com.br/ontolps.owl#hasReviewAtr -->

<owl:ObjectProperty rdf:about="#hasReviewAtr">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="#AtrReviewCicle"/>
   <rdfs:range rdf:resource="#AtrReviewDate"/>
   <rdfs:range rdf:resource="#AtrReviewDescription"/>
   <rdfs:range rdf:resource="#AtrReviewOk"/>
   <rdfs:domain rdf:resource="#Review"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasRole -->

<owl:ObjectProperty rdf:about="#hasRole">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="&pmbok;Role"/>
   <rdfs:domain rdf:resource="&pmbok;StakeholderRoleActivity"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasRoleAtr -->

<owl:ObjectProperty rdf:about="#hasRoleAtr">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="#AtrRoleFunction"/>
   <rdfs:range rdf:resource="#AtrRoleName"/>
   <rdfs:domain rdf:resource="&pmbok;Role"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasStakePmbokAtr -->

<owl:ObjectProperty rdf:about="#hasStakePmbokAtr">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="#AtrExternalMember"/>
   <rdfs:range rdf:resource="#AtrOther"/>
   <rdfs:range rdf:resource="#AtrRelevant"/>
   <rdfs:range rdf:resource="#AtrTeamMember"/>
   <rdfs:domain rdf:resource="&pmbok;PmbokStakeholder"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasStakeRoleActivity -->

<owl:ObjectProperty rdf:about="#hasStakeRoleActivity">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:domain rdf:resource="&pmbok;Activity"/>
   <rdfs:range rdf:resource="&pmbok;StakeholderRoleActivity"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasStakeholderMolps -->
```

```
<owl:ObjectProperty rdf:about="#hasStakeholderMolps">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="#StakeholderMolps"/>
   <rdfs:domain rdf:resource="#TeamMolps"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasStakeholderMolpsAtr -->

<owl:ObjectProperty rdf:about="#hasStakeholderMolpsAtr">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="#AtrStakeMolpsEmail"/>
   <rdfs:range rdf:resource="#AtrStakeMolpsEndereco"/>
   <rdfs:range rdf:resource="#AtrStakeMolpsName"/>
   <rdfs:range rdf:resource="#AtrStakeMolpsPhone"/>
   <rdfs:range rdf:resource="#AtrStakeMolpsProfile"/>
   <rdfs:range rdf:resource="#AtrStakeMolpsResource"/>
   <rdfs:domain rdf:resource="#StakeholderMolps"/>
</owl:ObjectProperty>



<!-- http://www.ontolps.com.br/ontolps.owl#hasStakeholderPmbok -->

<owl:ObjectProperty rdf:about="#hasStakeholderPmbok">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:domain rdf:resource="#StakeholderMolps"/>
   <rdfs:range rdf:resource="&pmbok;PmbokStakeholder"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasStakeholderRoleActivityAtr -->

<owl:ObjectProperty rdf:about="#hasStakeholderRoleActivityAtr">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="#AtrStakeholderRoleActivityActivity"/>
   <rdfs:range rdf:resource="#AtrStakeholderRoleActivityRole"/>
   <rdfs:range rdf:resource="#AtrStakeholderRoleActivityStakeholder"/>
   <rdfs:domain rdf:resource="&pmbok;StakeholderRoleActivity"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasTask -->

<owl:ObjectProperty rdf:about="#hasTask">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:domain rdf:resource="#RequirementsMolps"/>
   <rdfs:range rdf:resource="&xtremeprogramming;Task"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasTaskAtr -->

<owl:ObjectProperty rdf:about="#hasTaskAtr">
```

```
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrTaskDescription"/>
    <rdfs:range rdf:resource="#AtrTaskP1"/>
    <rdfs:range rdf:resource="#AtrTaskP2"/>
    <rdfs:range rdf:resource="#AtrTaskPercentCompleted"/>
    <rdfs:range rdf:resource="#AtrTaskRequirementsMolps"/>
    <rdfs:domain rdf:resource="&xtremeprogramming;Task"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasTeamMember -->

<owl:ObjectProperty rdf:about="#hasTeamMember">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:domain rdf:resource="&pmbok;Relevant"/>
    <rdfs:range rdf:resource="&pmbok;TeamMember"/>
</owl:ObjectProperty>



<!-- http://www.ontolps.com.br/ontolps.owl#hasTeamMemberAtr -->

<owl:ObjectProperty rdf:about="#hasTeamMemberAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrTeamMember"/>
    <rdfs:range rdf:resource="#AtrTeamMemberFunc"/>
    <rdfs:domain rdf:resource="&pmbok;TeamMember"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasTeamMolps -->

<owl:ObjectProperty rdf:about="#hasTeamMolps">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:domain rdf:resource="#ProjectMolps"/>
    <rdfs:range rdf:resource="#TeamMolps"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasTeamMolpsAtr -->

<owl:ObjectProperty rdf:about="#hasTeamMolpsAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrTeamFunction"/>
    <rdfs:range rdf:resource="#AtrTeamName"/>
    <rdfs:domain rdf:resource="#TeamMolps"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasTechnique -->

<owl:ObjectProperty rdf:about="#hasTechnique">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:domain rdf:resource="&pmbok;Guidance"/>
```

```
    <rdfs:range rdf:resource="&pmbok;Technique"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasTechniqueAtr -->

<owl:ObjectProperty rdf:about="#hasTechniqueAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrTechnique"/>
    <rdfs:domain rdf:resource="&pmbok;Technique"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasTest -->

<owl:ObjectProperty rdf:about="#hasTest">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:domain rdf:resource="#RequirementsMolps"/>
    <rdfs:range rdf:resource="&xtremeprogramming;Test"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasTestAtr -->

<owl:ObjectProperty rdf:about="#hasTestAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrTestDescription"/>
    <rdfs:range rdf:resource="#AtrTestRequirementsMolps"/>
    <rdfs:range rdf:resource="#AtrTestValidated"/>
    <rdfs:domain rdf:resource="&xtremeprogramming;Test"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasTool -->

<owl:ObjectProperty rdf:about="#hasTool">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:domain rdf:resource="&pmbok;Guidance"/>
    <rdfs:range rdf:resource="&pmbok;Tool"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasToolAtr -->

<owl:ObjectProperty rdf:about="#hasToolAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrTool"/>
    <rdfs:domain rdf:resource="&pmbok;Tool"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasWorkProduct -->

<owl:ObjectProperty rdf:about="#hasWorkProduct">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:domain rdf:resource="&pmbok;Activity"/>
```

```
    <rdfs:range rdf:resource="&pmbok;WorkProduct"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#hasWorkProductAtr -->

<owl:ObjectProperty rdf:about="#hasWorkProductAtr">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#AtrWorkProductActivity"/>
    <rdfs:range rdf:resource="#AtrWorkProductType"/>
    <rdfs:domain rdf:resource="&pmbok;WorkProduct"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#isEscolhaOp -->

<owl:ObjectProperty rdf:about="#isEscolhaOp">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#isEscolhaV -->

<owl:ObjectProperty rdf:about="#isEscolhaV">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#isObrigatorio -->

<owl:ObjectProperty rdf:about="#isObrigatorio">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:range rdf:resource="#CicleMolps"/>
    <rdfs:range rdf:resource="#MeetingMolps"/>
    <rdfs:domain rdf:resource="#Obrigatorio"/>
    <rdfs:range rdf:resource="#ProductiveCicleMolps"/>
    <rdfs:range rdf:resource="#ProjectMolps"/>
    <rdfs:range rdf:resource="#RequirementsMolps"/>
    <rdfs:range rdf:resource="#StakeholderMolps"/>
    <rdfs:range rdf:resource="#TeamMolps"/>
    <rdfs:range rdf:resource="&scrum;ProductBacklog"/>
    <rdfs:range rdf:resource="&xtremeprogramming;Release"/>
    <rdfs:range rdf:resource="&pmbok;Activity"/>
    <rdfs:range rdf:resource="&pmbok;Organization"/>
    <rdfs:range rdf:resource="&pmbok;Phase"/>
    <rdfs:range rdf:resource="&pmbok;PhisicalResource"/>
    <rdfs:range rdf:resource="&pmbok;Program"/>
    <rdfs:range rdf:resource="&pmbok;Resource"/>
    <rdfs:range rdf:resource="&pmbok;Role"/>
    <rdfs:range rdf:resource="&pmbok;StakeholderRoleActivity"/>
</owl:ObjectProperty>

<!-- http://www.ontolps.com.br/ontolps.owl#isOpcional -->
```

```
<owl:ObjectProperty rdf:about="#isOpcional">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="&fdd;Funcionality"/>
   <rdfs:domain rdf:resource="#Opcional"/>
   <rdfs:range rdf:resource="&xtremeprogramming;History"/>
   <rdfs:range rdf:resource="&xtremeprogramming;Task"/>
   <rdfs:range rdf:resource="&xtremeprogramming;Test"/>
   <rdfs:range rdf:resource="&pmbok;Equipament"/>
   <rdfs:range rdf:resource="&pmbok;Facility"/>
   <rdfs:range rdf:resource="&pmbok;Material"/>
</owl:ObjectProperty>



<!-- http://www.ontolps.com.br/ontolps.owl#isVariavel -->

<owl:ObjectProperty rdf:about="#isVariavel">
   <rdf:type rdf:resource="&owl;TransitiveProperty"/>
   <rdfs:range rdf:resource="&fdd;ComprehensiveModel"/>
   <rdfs:range rdf:resource="&fdd;Requisites"/>
   <rdfs:range rdf:resource="#Planning"/>
   <rdfs:range rdf:resource="#Retrospective"/>
   <rdfs:range rdf:resource="#Review"/>
   <rdfs:domain rdf:resource="#Variavel"/>
   <rdfs:range rdf:resource="&pmbok;Guidance"/>
   <rdfs:range rdf:resource="&pmbok;ManagementProcess"/>
   <rdfs:range rdf:resource="&pmbok;PmbokStakeholder"/>
   <rdfs:range rdf:resource="&pmbok;WorkProduct"/>
</owl:ObjectProperty>
 <!--
///////////////////////////////////////////////////////////////////////////////////
//
// Classes
//
///////////////////////////////////////////////////////////////////////////////////
 -->
<!-- http://www.ontolps.com.br/fdd.owl#ComprehensiveModel -->

<owl:Class rdf:about="&fdd;ComprehensiveModel">
   <rdfs:subClassOf rdf:resource="#Conceito"/>
</owl:Class>
<!-- http://www.ontolps.com.br/fdd.owl#Funcionality -->

<owl:Class rdf:about="&fdd;Funcionality">
   <rdfs:subClassOf rdf:resource="#Conceito"/>
</owl:Class>
 <!-- http://www.ontolps.com.br/fdd.owl#Requisites -->

<owl:Class rdf:about="&fdd;Requisites">
   <rdfs:subClassOf rdf:resource="#Conceito"/>
```

```
</owl:Class>
  <!-- http://www.ontolps.com.br/ontolps.owl#AtrActivityDate -->

<owl:Class rdf:about="#AtrActivityDate">
    <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
 <!-- http://www.ontolps.com.br/ontolps.owl#AtrActivityDateF -->

<owl:Class rdf:about="#AtrActivityDateF">
    <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrActivityDesc -->

<owl:Class rdf:about="#AtrActivityDesc">
    <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
 <!-- http://www.ontolps.com.br/ontolps.owl#AtrActivityName -->

<owl:Class rdf:about="#AtrActivityName">
    <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrActivityPhase -->

<owl:Class rdf:about="#AtrActivityPhase">
    <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrActivitySit -->

<owl:Class rdf:about="#AtrActivitySit">
    <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrCicleMolpsDataI -->

<owl:Class rdf:about="#AtrCicleMolpsDataI">
    <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrCicleMolpsDate -->

<owl:Class rdf:about="#AtrCicleMolpsDate">
    <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
 <!-- http://www.ontolps.com.br/ontolps.owl#AtrCicleMolpsDateF -->

<owl:Class rdf:about="#AtrCicleMolpsDateF">
    <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrCicleMolpsDescription -->

<owl:Class rdf:about="#AtrCicleMolpsDescription">
    <rdfs:subClassOf rdf:resource="#Atributos"/>
```

```
</owl:Class>
 <!-- http://www.ontolps.com.br/ontolps.owl#AtrCicleMolps*Release* -->

<owl:Class rdf:about="#AtrCicleMolps*Release*">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>



<!-- http://www.ontolps.com.br/ontolps.owl#AtrCicleMolpsSit -->

<owl:Class rdf:about="#AtrCicleMolpsSit">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrComprehensiveModelArea -->

<owl:Class rdf:about="#AtrComprehensiveModelArea">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrComprehensiveModelDomain -->
<owl:Class rdf:about="#AtrComprehensiveModelDomain">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
 <!-- http://www.ontolps.com.br/ontolps.owl#AtrComprehensiveModelProject -->

<owl:Class rdf:about="#AtrComprehensiveModelProject">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrCore -->

<owl:Class rdf:about="#AtrCore">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrEquipament -->

<owl:Class rdf:about="#AtrEquipament">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
 <!-- http://www.ontolps.com.br/ontolps.owl#AtrExternalMember -->

<owl:Class rdf:about="#AtrExternalMember">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
 <!-- http://www.ontolps.com.br/ontolps.owl#AtrExternalMemberFunc -->

<owl:Class rdf:about="#AtrExternalMemberFunc">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
  <!-- http://www.ontolps.com.br/ontolps.owl#AtrFacilitating -->
```

```
    <owl:Class rdf:about="#AtrFacilitating">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>



    <!-- http://www.ontolps.com.br/ontolps.owl#AtrFacility -->

    <owl:Class rdf:about="#AtrFacility">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
     <!-- http://www.ontolps.com.br/ontolps.owl#AtrFuncionalityDescription -->

    <owl:Class rdf:about="#AtrFuncionalityDescription">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>

    <!-- http://www.ontolps.com.br/ontolps.owl#AtrFuncionalityPercentCompleted -->
    <owl:Class rdf:about="#AtrFuncionalityPercentCompleted">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
    <!-- http://www.ontolps.com.br/ontolps.owl#AtrFuncionalityRequirementsMolps --
>

    <owl:Class rdf:about="#AtrFuncionalityRequirementsMolps">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
      <!-- http://www.ontolps.com.br/ontolps.owl#AtrFuncionalityStake -->

    <owl:Class rdf:about="#AtrFuncionalityStake">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>

    <!-- http://www.ontolps.com.br/ontolps.owl#AtrGuidanceActivity -->

    <owl:Class rdf:about="#AtrGuidanceActivity">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
    <!-- http://www.ontolps.com.br/ontolps.owl#AtrGuidanceName -->

    <owl:Class rdf:about="#AtrGuidanceName">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
    <!-- http://www.ontolps.com.br/ontolps.owl#AtrHistoryDate -->

    <owl:Class rdf:about="#AtrHistoryDate">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
    <!-- http://www.ontolps.com.br/ontolps.owl#AtrHistoryDescription -->
```

```
<owl:Class rdf:about="#AtrHistoryDescription">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrHistoryRequirementsMolps -->

<owl:Class rdf:about="#AtrHistoryRequirementsMolps">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrKnowledgeAreaName -->

<owl:Class rdf:about="#AtrKnowledgeAreaName">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrManagementProcessActivity -->

<owl:Class rdf:about="#AtrManagementProcessActivity">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrManagementProcessName -->

<owl:Class rdf:about="#AtrManagementProcessName">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrMaterial -->

<owl:Class rdf:about="#AtrMaterial">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrMeetingMolpsCicle -->

<owl:Class rdf:about="#AtrMeetingMolpsCicle">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrMeetingMolpsDate -->

<owl:Class rdf:about="#AtrMeetingMolpsDate">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrMeetingMolpsDiscussion -->

<owl:Class rdf:about="#AtrMeetingMolpsDiscussion">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
```

```
<!-- http://www.ontolps.com.br/ontolps.owl#AtrMeetingMolpsTitle -->

<owl:Class rdf:about="#AtrMeetingMolpsTitle">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrOrganizationEndereco -->

<owl:Class rdf:about="#AtrOrganizationEndereco">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrOrganizationName -->

<owl:Class rdf:about="#AtrOrganizationName">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrOther -->

<owl:Class rdf:about="#AtrOther">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrPhaseName -->

<owl:Class rdf:about="#AtrPhaseName">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrPhaseProject -->

<owl:Class rdf:about="#AtrPhaseProject">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrPhysicalResourceName -->

<owl:Class rdf:about="#AtrPhysicalResourceName">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrPhysicalResourceResource -->

<owl:Class rdf:about="#AtrPhysicalResourceResource">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
```

```
<!-- http://www.ontolps.com.br/ontolps.owl#AtrPlanningCicle -->

<owl:Class rdf:about="#AtrPlanningCicle">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrPlanningDate -->

<owl:Class rdf:about="#AtrPlanningDate">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
 <!-- http://www.ontolps.com.br/ontolps.owl#AtrPlanningObjective -->

<owl:Class rdf:about="#AtrPlanningObjective">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrProcessGroupName -->

<owl:Class rdf:about="#AtrProcessGroupName">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrProductDescription -->

<owl:Class rdf:about="#AtrProductDescription">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
 <!-- http://www.ontolps.com.br/ontolps.owl#AtrProductPriority -->

<owl:Class rdf:about="#AtrProductPriority">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
 <!-- http://www.ontolps.com.br/ontolps.owl#AtrProductProject -->

<owl:Class rdf:about="#AtrProductProject">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrProductStakeholder -->

<owl:Class rdf:about="#AtrProductStakeholder">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>



<!-- http://www.ontolps.com.br/ontolps.owl#AtrProductiveCicleMolpsCicle -->

<owl:Class rdf:about="#AtrProductiveCicleMolpsCicle">
```

```
        <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
    <!-- http://www.ontolps.com.br/ontolps.owl#AtrProductiveCicleMolpsDescription -
->

    <owl:Class rdf:about="#AtrProductiveCicleMolpsDescription">
        <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>

    <!-- http://www.ontolps.com.br/ontolps.owl#AtrProductiveCicleMolpsPriority -->
    <owl:Class rdf:about="#AtrProductiveCicleMolpsPriority">
        <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>

    <!--
http://www.ontolps.com.br/ontolps.owl#AtrProductiveCicleMolpsProductBacklog -->
    <owl:Class rdf:about="#AtrProductiveCicleMolpsProductBacklog">
        <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>

    <!--
http://www.ontolps.com.br/ontolps.owl#AtrProductiveCicleMolpsRequirementsMolps
-->

    <owl:Class rdf:about="#AtrProductiveCicleMolpsRequirementsMolps">
        <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
   <!-- http://www.ontolps.com.br/ontolps.owl#AtrProductiveCicleMolpsStakeholder -
->

    <owl:Class rdf:about="#AtrProductiveCicleMolpsStakeholder">
        <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
   <!-- http://www.ontolps.com.br/ontolps.owl#AtrProgramName -->

    <owl:Class rdf:about="#AtrProgramName">
        <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
    <!-- http://www.ontolps.com.br/ontolps.owl#AtrProgramOrganization -->

    <owl:Class rdf:about="#AtrProgramOrganization">
        <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
    <!-- http://www.ontolps.com.br/ontolps.owl#AtrProjectDate -->

    <owl:Class rdf:about="#AtrProjectDate">
        <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
    <!-- http://www.ontolps.com.br/ontolps.owl#AtrProjectDateF -->
```

```
<owl:Class rdf:about="#AtrProjectDateF">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrProjectDescription -->

<owl:Class rdf:about="#AtrProjectDescription">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrProjectManager -->

<owl:Class rdf:about="#AtrProjectManager">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrProjectName -->

<owl:Class rdf:about="#AtrProjectName">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrProjectProgram -->

<owl:Class rdf:about="#AtrProjectProgram">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#Atr*Release*Date -->

<owl:Class rdf:about="#Atr*Release*Date">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#Atr*Release*Name -->

<owl:Class rdf:about="#Atr*Release*Name">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#Atr*Release*Project -->

<owl:Class rdf:about="#Atr*Release*Project">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>


<!-- http://www.ontolps.com.br/ontolps.owl#AtrRelevant -->

<owl:Class rdf:about="#AtrRelevant">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
 <!-- http://www.ontolps.com.br/ontolps.owl#AtrRequirementsMolpsActivity -->

<owl:Class rdf:about="#AtrRequirementsMolpsActivity">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
```

```
<!-- http://www.ontolps.com.br/ontolps.owl#AtrRequirementsMolpsDataI -->

<owl:Class rdf:about="#AtrRequirementsMolpsDataI">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrRequirementsMolpsDate -->

<owl:Class rdf:about="#AtrRequirementsMolpsDate">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrRequirementsMolpsDateF -->

<owl:Class rdf:about="#AtrRequirementsMolpsDateF">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
 <!-- http://www.ontolps.com.br/ontolps.owl#AtrRequirementsMolpsEstimate -->
<owl:Class rdf:about="#AtrRequirementsMolpsEstimate">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrRequirementsMolpsSit -->

<owl:Class rdf:about="#AtrRequirementsMolpsSit">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrRequirementsMolpsTitle -->

<owl:Class rdf:about="#AtrRequirementsMolpsTitle">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrRequisitesActivity -->

<owl:Class rdf:about="#AtrRequisitesActivity">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>


<!-- http://www.ontolps.com.br/ontolps.owl#AtrRequisitesComprehensiveM -->

<owl:Class rdf:about="#AtrRequisitesComprehensiveM">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrRequisitesDesc -->

<owl:Class rdf:about="#AtrRequisitesDesc">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrRequisitesName -->
```

```
<owl:Class rdf:about="#AtrRequisitesName">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrRequisitesPriority -->

<owl:Class rdf:about="#AtrRequisitesPriority">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrResourceActivity -->

<owl:Class rdf:about="#AtrResourceActivity">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrRetrospectiveAnalisys -->

<owl:Class rdf:about="#AtrRetrospectiveAnalisys">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrRetrospectiveCicle -->

<owl:Class rdf:about="#AtrRetrospectiveCicle">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrRetrospectiveDate -->

<owl:Class rdf:about="#AtrRetrospectiveDate">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrReviewCicle -->

<owl:Class rdf:about="#AtrReviewCicle">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrReviewDate -->

<owl:Class rdf:about="#AtrReviewDate">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrReviewDescription -->

<owl:Class rdf:about="#AtrReviewDescription">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrReviewOk -->

<owl:Class rdf:about="#AtrReviewOk">
   <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
```

```
    <!-- http://www.ontolps.com.br/ontolps.owl#AtrRoleFunction -->

    <owl:Class rdf:about="#AtrRoleFunction">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
    <!-- http://www.ontolps.com.br/ontolps.owl#AtrRoleName -->

    <owl:Class rdf:about="#AtrRoleName">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
    <!-- http://www.ontolps.com.br/ontolps.owl#AtrStakeMolpsEmail -->

    <owl:Class rdf:about="#AtrStakeMolpsEmail">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
     <!-- http://www.ontolps.com.br/ontolps.owl#AtrStakeMolpsEndereco -->

    <owl:Class rdf:about="#AtrStakeMolpsEndereco">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
     <!-- http://www.ontolps.com.br/ontolps.owl#AtrStakeMolpsName -->

    <owl:Class rdf:about="#AtrStakeMolpsName">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
    <!-- http://www.ontolps.com.br/ontolps.owl#AtrStakeMolpsPhone -->

    <owl:Class rdf:about="#AtrStakeMolpsPhone">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>



    <!-- http://www.ontolps.com.br/ontolps.owl#AtrStakeMolpsProfile -->

    <owl:Class rdf:about="#AtrStakeMolpsProfile">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
    <!-- http://www.ontolps.com.br/ontolps.owl#AtrStakeMolpsResource -->

    <owl:Class rdf:about="#AtrStakeMolpsResource">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
   <!-- http://www.ontolps.com.br/ontolps.owl#AtrStakeholderRoleActivityActivity --
>
    <owl:Class rdf:about="#AtrStakeholderRoleActivityActivity">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
   <!-- http://www.ontolps.com.br/ontolps.owl#AtrStakeholderRoleActivityRole -->
```

```
    <owl:Class rdf:about="#AtrStakeholderRoleActivityRole">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
     <!--
http://www.ontolps.com.br/ontolps.owl#AtrStakeholderRoleActivityStakeholder -->
    <owl:Class rdf:about="#AtrStakeholderRoleActivityStakeholder">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
    <!-- http://www.ontolps.com.br/ontolps.owl#AtrTaskDescription -->

    <owl:Class rdf:about="#AtrTaskDescription">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
     <!-- http://www.ontolps.com.br/ontolps.owl#AtrTaskP1 -->

    <owl:Class rdf:about="#AtrTaskP1">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
     <!-- http://www.ontolps.com.br/ontolps.owl#AtrTaskP2 -->

    <owl:Class rdf:about="#AtrTaskP2">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
    <!-- http://www.ontolps.com.br/ontolps.owl#AtrTaskPercentCompleted -->

    <owl:Class rdf:about="#AtrTaskPercentCompleted">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>



    <!-- http://www.ontolps.com.br/ontolps.owl#AtrTaskRequirementsMolps -->

    <owl:Class rdf:about="#AtrTaskRequirementsMolps">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
    <!-- http://www.ontolps.com.br/ontolps.owl#AtrTeamFunction -->

    <owl:Class rdf:about="#AtrTeamFunction">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>
    <!-- http://www.ontolps.com.br/ontolps.owl#AtrTeamMember -->

    <owl:Class rdf:about="#AtrTeamMember">
       <rdfs:subClassOf rdf:resource="#Atributos"/>
    </owl:Class>

    <!-- http://www.ontolps.com.br/ontolps.owl#AtrTeamMemberFunc -->

    <owl:Class rdf:about="#AtrTeamMemberFunc">
```

```
    <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrTeamName -->

<owl:Class rdf:about="#AtrTeamName">
    <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
<!-- http://www.ontolps.com.br/ontolps.owl#AtrTechnique -->

<owl:Class rdf:about="#AtrTechnique">
    <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrTestDescription -->

<owl:Class rdf:about="#AtrTestDescription">
    <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>
 <!-- http://www.ontolps.com.br/ontolps.owl#AtrTestRequirementsMolps -->

<owl:Class rdf:about="#AtrTestRequirementsMolps">
    <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrTestValidated -->

<owl:Class rdf:about="#AtrTestValidated">
    <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrTool -->

<owl:Class rdf:about="#AtrTool">
    <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrWorkProductActivity -->

<owl:Class rdf:about="#AtrWorkProductActivity">
    <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#AtrWorkProductType -->

<owl:Class rdf:about="#AtrWorkProductType">
    <rdfs:subClassOf rdf:resource="#Atributos"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#Atributos -->
```

```
<owl:Class rdf:about="#Atributos"/>
 <!-- http://www.ontolps.com.br/ontolps.owl#CicleMolps -->

<owl:Class rdf:about="#CicleMolps">
   <rdfs:subClassOf rdf:resource="#Conceito"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#Conceito -->

<owl:Class rdf:about="#Conceito"/>

<!-- http://www.ontolps.com.br/ontolps.owl#MeetingMolps -->

<owl:Class rdf:about="#MeetingMolps">
   <rdfs:subClassOf rdf:resource="#Conceito"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#Obrigatorio -->

<owl:Class rdf:about="#Obrigatorio">
   <rdfs:subClassOf rdf:resource="#Conceito"/>
</owl:Class>



<!-- http://www.ontolps.com.br/ontolps.owl#Opcional -->

<owl:Class rdf:about="#Opcional">
   <rdfs:subClassOf rdf:resource="#Conceito"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#Planning -->

<owl:Class rdf:about="#Planning">
   <rdfs:subClassOf rdf:resource="#Conceito"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#ProductiveCicleMolps -->

<owl:Class rdf:about="#ProductiveCicleMolps">
   <rdfs:subClassOf rdf:resource="#Conceito"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#ProjectMolps -->

<owl:Class rdf:about="#ProjectMolps">
   <rdfs:subClassOf rdf:resource="#Conceito"/>
</owl:Class>
```

```
<!-- http://www.ontolps.com.br/ontolps.owl#RequirementsMolps -->

<owl:Class rdf:about="#RequirementsMolps">
   <rdfs:subClassOf rdf:resource="#Conceito"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#Retrospective -->

<owl:Class rdf:about="#Retrospective">
   <rdfs:subClassOf rdf:resource="#Conceito"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#Review -->

<owl:Class rdf:about="#Review">
   <rdfs:subClassOf rdf:resource="#Conceito"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#StakeholderMolps -->

<owl:Class rdf:about="#StakeholderMolps">
   <rdfs:subClassOf rdf:resource="#Conceito"/>
</owl:Class>



<!-- http://www.ontolps.com.br/ontolps.owl#TeamMolps -->

<owl:Class rdf:about="#TeamMolps">
   <rdfs:subClassOf rdf:resource="#Conceito"/>
</owl:Class>

<!-- http://www.ontolps.com.br/ontolps.owl#Variavel -->

<owl:Class rdf:about="#Variavel">
   <rdfs:subClassOf rdf:resource="#Conceito"/>
</owl:Class>

<!-- http://www.ontolps.com.br/scrum.owl#ProductBacklog -->

<owl:Class rdf:about="&scrum;ProductBacklog">
   <rdfs:subClassOf rdf:resource="#Conceito"/>
</owl:Class>

<!-- http://www.ontolps.com.br/xtremeprogramming.owl#History -->

<owl:Class rdf:about="&xtremeprogramming;History">
   <rdfs:subClassOf rdf:resource="#Conceito"/>
</owl:Class>
```

```
    <!-- http://www.ontolps.com.br/xtremeprogramming.owl#Release -->

    <owl:Class rdf:about="&xtremeprogramming;Release">
        <rdfs:subClassOf rdf:resource="#Conceito"/>
    </owl:Class>

    <!-- http://www.ontolps.com.br/xtremeprogramming.owl#Task -->

    <owl:Class rdf:about="&xtremeprogramming;Task">
        <rdfs:subClassOf rdf:resource="#Conceito"/>
    </owl:Class>

    <!-- http://www.ontolps.com.br/xtremeprogramming.owl#Test -->

    <owl:Class rdf:about="&xtremeprogramming;Test">
        <rdfs:subClassOf rdf:resource="#Conceito"/>
    </owl:Class>

    <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#Activity -->

    <owl:Class rdf:about="&pmbok;Activity">
        <rdfs:subClassOf rdf:resource="#Conceito"/>
    </owl:Class>

    <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#Core -->

    <owl:Class rdf:about="&pmbok;Core">
        <rdfs:subClassOf rdf:resource="&pmbok;KnowledgeArea"/>
    </owl:Class>

    <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#Equipament -->

    <owl:Class rdf:about="&pmbok;Equipament">
        <rdfs:subClassOf rdf:resource="&pmbok;PhisicalResource"/>
    </owl:Class>
     <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#ExternalMember
-->

    <owl:Class rdf:about="&pmbok;ExternalMember">
        <rdfs:subClassOf rdf:resource="&pmbok;Relevant"/>
    </owl:Class>
     <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#Facilitating -->

    <owl:Class rdf:about="&pmbok;Facilitating">
        <rdfs:subClassOf rdf:resource="&pmbok;KnowledgeArea"/>
    </owl:Class>

    <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#Facility -->

    <owl:Class rdf:about="&pmbok;Facility">
```

```
        <rdfs:subClassOf rdf:resource="&pmbok;PhisicalResource"/>
    </owl:Class>
     <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#Guidance -->

    <owl:Class rdf:about="&pmbok;Guidance">
        <rdfs:subClassOf rdf:resource="#Conceito"/>
    </owl:Class>
    <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#KnowledgeArea --
>

    <owl:Class rdf:about="&pmbok;KnowledgeArea">
        <rdfs:subClassOf rdf:resource="&pmbok;ManagementProcess"/>
    </owl:Class>
    <!--
http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#ManagementProcess -->
    <owl:Class rdf:about="&pmbok;ManagementProcess">
        <rdfs:subClassOf rdf:resource="#Conceito"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#Material -->

    <owl:Class rdf:about="&pmbok;Material">
        <rdfs:subClassOf rdf:resource="&pmbok;PhisicalResource"/>
    </owl:Class>
   <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#Organization -->
    <owl:Class rdf:about="&pmbok;Organization">
        <rdfs:subClassOf rdf:resource="#Conceito"/>
    </owl:Class>

    <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#Other -->

    <owl:Class rdf:about="&pmbok;Other">
        <rdfs:subClassOf rdf:resource="&pmbok;PmbokStakeholder"/>
    </owl:Class>

    <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#Phase -->

    <owl:Class rdf:about="&pmbok;Phase">
        <rdfs:subClassOf rdf:resource="#Conceito"/>
    </owl:Class>

    <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#PhisicalResource -
->

    <owl:Class rdf:about="&pmbok;PhisicalResource">
        <rdfs:subClassOf rdf:resource="#Conceito"/>
    </owl:Class>
```

```
    <!--
http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#PmbokStakeholder -->

    <owl:Class rdf:about="&pmbok;PmbokStakeholder">
        <rdfs:subClassOf rdf:resource="#StakeholderMolps"/>
    </owl:Class>

    <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#ProcessGroup -->

    <owl:Class rdf:about="&pmbok;ProcessGroup">
        <rdfs:subClassOf rdf:resource="&pmbok;ManagementProcess"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#Program -->

    <owl:Class rdf:about="&pmbok;Program">
        <rdfs:subClassOf rdf:resource="#Conceito"/>
    </owl:Class>

    <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#Relevant -->

    <owl:Class rdf:about="&pmbok;Relevant">
        <rdfs:subClassOf rdf:resource="&pmbok;PmbokStakeholder"/>
    </owl:Class>

    <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#Resource -->

    <owl:Class rdf:about="&pmbok;Resource">
        <rdfs:subClassOf rdf:resource="#Conceito"/>
    </owl:Class>

    <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#Role -->

    <owl:Class rdf:about="&pmbok;Role">
        <rdfs:subClassOf rdf:resource="#Conceito"/>
    </owl:Class>

    <!--
http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#StakeholderRoleActivity
-->

    <owl:Class rdf:about="&pmbok;StakeholderRoleActivity">
        <rdfs:subClassOf rdf:resource="#Conceito"/>
    </owl:Class>

    <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#TeamMember -->

    <owl:Class rdf:about="&pmbok;TeamMember">
```

```
        <rdfs:subClassOf rdf:resource="&pmbok;Relevant"/>
    </owl:Class>

    <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#Technique -->

    <owl:Class rdf:about="&pmbok;Technique">
        <rdfs:subClassOf rdf:resource="&pmbok;Guidance"/>
    </owl:Class>

    <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#Tool -->

    <owl:Class rdf:about="&pmbok;Tool">
        <rdfs:subClassOf rdf:resource="&pmbok;Guidance"/>
    </owl:Class>

    <!-- http://www.semanticweb.org/ontologies/2011/6/pmbok.owl#WorkProduct -->
    <owl:Class rdf:about="&pmbok;WorkProduct">
        <rdfs:subClassOf rdf:resource="#Conceito"/>
    </owl:Class>
</rdf:RDF>
```