UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE
ÁREA DE CIÊNCIA DA COMPUTAÇÃO

Adriana Leite Costa

MADAE-Pro: Um processo baseado no conhecimento para Engenharia de Domínio e de Aplicações Multiagente

São Luís
2009

ADRIANA LEITE COSTA

MADAE-PRO: UM PROCESSO BASEADO NO CONHECIMENTO PARA ENGENHARIA DE DOMÍNIO E DE APLICAÇÕES MULTIAGENTE

Dissertação de Mestrado apresentada ao curso de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão, como parte dos requisitos para a obtenção do título de Mestre em Engenharia de Eletricidade, na área de Ciência da Computação.

Orientadora*:* Profª. Dra. Rosario Girardi

São Luís
2009

Costa, Adriana Leite
MADAE-Pro: um processo baseado no conhecimento para engenharia de domínio e de aplicações multiagente / Adriana Leite Costa. – São Luís, 2009.

157f.

Orientador: María del Rosario Girardi Gutiérrez.
Impresso por computador (fotocópia).
Dissertação (Mestrado) – Universidade Federal do Maranhão, Curso de Pós-graduação em Engenharia de Eletricidade. São Luís, 2009.

1. Software – desenvolvimento. 2. Software – reuso. 3. Sistemas multiagente. I. Gutiérrez, María del Rosario Girardi, orientadora. II Título.

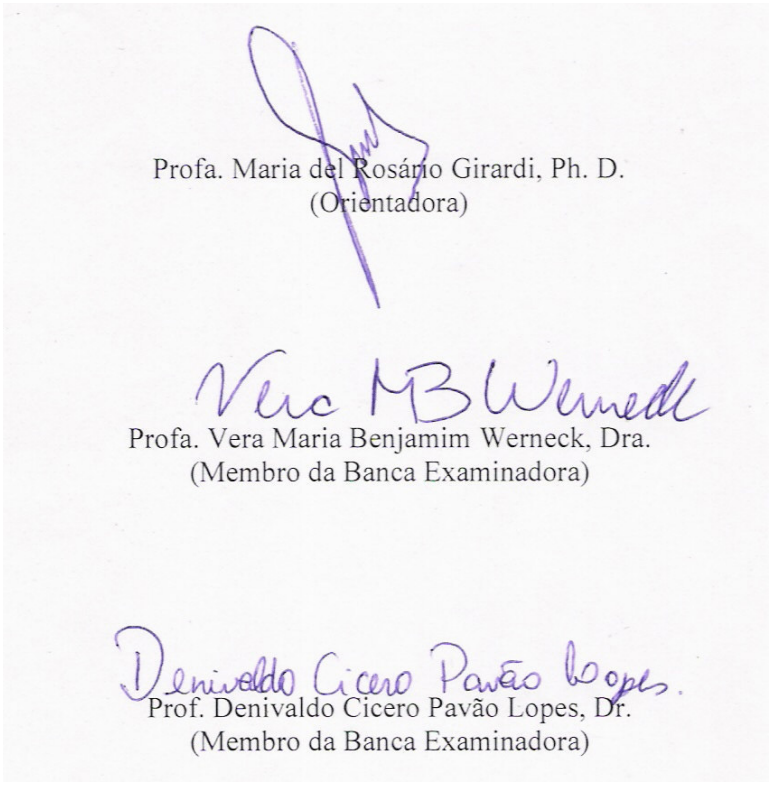
CDU 004.4'2

MADAE-PRO: UM PROCESSO BASEADO NO CONHECIMENTO PARA ENGENHARIA DE DOMÍNIO E DE APLICAÇÕES MULTIAGENTE

ADRIANA LEITE COSTA

Dissertação aprovada em 17 de fevereiro de 2009.

BANCA EXAMINADORA:



Profa. Maria del Rosário Girardi, Ph. D.
(Orientadora)

Profa. Vera Maria Benjamim Werneck, Dra.
(Membro da Banca Examinadora)

Prof. Denivaldo Cicero Pavão Lopes, Dr.
(Membro da Banca Examinadora)

À minha avó, Ana Pinheiro Costa (in memoriam).

AGRADECIMENTOS

A Deus a quem primeiro recorro e encontro conforto nos momentos de dificuldade.

À minha família pelo apoio. Em especial, a minha mãe que sempre tem me ajudado em tudo o que pode. Ao meu tio João e a minha avó Dalva pela contribuição que tiveram na minha vida acadêmica.

À professora Dra. Rosario Girardi, cujo apoio intelectual, dedicação e estímulo foram fundamentais para realização desta dissertação e pela amizade e paciência.

Ao professor Dr. Denivaldo Lopes e a professora Dra. Vera Werneck por suas sugestões e correções que contribuíram para melhorar a qualidade deste trabalho.

Aos colegas do mestrado. Em especial, a Helaine, Alex, Pedriana e Luciana pela ajuda mútua durante a realização das disciplinas. Ao Fábio pelo companheirismo nos trabalhos em equipe que fizemos juntos. Ao Uiratan, pelas sugestões e ajuda na revisão de alguns dos capítulos do manuscrito. Ao Lucas e ao Mariano, cujos trabalhos têm servido para avaliar e melhorar o processo de desenvolvimento aqui apresentado.

Ao Jaclason e a Carla pela ajuda na preparação da apresentação. Ao meu amigo Pedro pelo incentivo.

Aos colegas do Departamento de Informática do Cefet. Em especial, ao “Seu Lima” e a “Lurdinha” pela amizade e apoio. Ao Prof. João Carlos, que primeiro me incentivou a ingressar no mestrado. Ao Prof. André por ter me concedido horário especial. Aos professores Cícero, Rafael, Gentil, Omar, Hélder, Santiago, Carla, Luna, Pedro Júnior, Raimundo, David, Salete, Karla, Jeane, Evaldinólia e Eveline pelo apoio e incentivo.

Aos integrantes e ex-integrantes do grupo GESEC que contribuíram direta ou indiretamente para realização desse trabalho.

Ao programa de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão, pela oportunidade de realização do curso.

“O caminho mais seguro para chegar ao verdadeiro futuro
(pois existe também um falso futuro) é ir na direção
em que teu medo cresce.”

Milorad Pávitch

RESUMO

O interesse pelo paradigma de desenvolvimento orientado a agentes tem aumentado nos últimos anos. Isso se deve principalmente ao crescente aumento da complexidade dos produtos de software atuais que requerem novas características como comportamento autônomo. No paradigma orientado a agentes, o software deixa de ter comportamento estritamente previsível e passa a ter controle sobre seu próprio comportamento, podendo tomar decisões a partir de observações do ambiente e de inferências realizada em sua base de conhecimento. Para guiar o desenvolvimento orientado a agentes tem sido proposto um conjunto de metodologias e processos pela comunidade da Engenharia de Software. Nesse trabalho, apresenta-se MADAE-Pro, um processo para o desenvolvimento de sistemas multiagente com alguns diferenciais em relação aos já propostos pela comunidade. A Engenharia de Domínio é um processo para criação de abstrações de software reusáveis no desenvolvimento de uma família de aplicações em um domínio particular de problema. A Engenharia de Aplicações é um processo para construção de aplicações baseadas no reúso de artefatos de software previamente produzidos no processo da Engenharia de Domínio. O MADAE-Pro é um processo dirigido por ontologias para a Engenharia de Domínio e de Aplicações Multiagente, o qual promove a construção e o reúso de famílias de aplicações. O processo é especificado em uma linguagem de representação de processos formal, evitando assim interpretações ambíguas. Outro diferencial do MADAE-Pro é o suporte ao reúso de software em todos os níveis de abstração, desde os requisitos até a implementação.

Palavras-chave: Processo de Desenvolvimento de Software. Ontologias. Reúso de Software. Sistemas Multiagente.

ABSTRACT

The interest in the agent-oriented paradigm development has increased in recent years. This is due mainly to the increasing complexity of current software that requires new characteristics as autonomy behavior. In the agent-oriented paradigm, the software has no longer a strictly predictable behavior, has from the control over their own behavior and can make decisions based on observations the environment and inferences upon its knowledge base. A set of meth and process have been already proposed for agent-oriented software engineering. Domain Engineering is a process for the development of a reusable application family in a particular domain problem, and Application Engineering, the one for the construction of a specific application in a family based on the reuse of software artifacts in the application family previously produced in the Domain Engineering process. MADAE-Pro is an ontology-driven process for multi-agent domain and application engineering which promotes the construction and reuse of agent-oriented applications families. The process is specified in a formal representation language, thus avoiding ambiguous interpretations. Another differential of MADAE-Pro is the reuse of software support in all levels of abstraction, from the requirements to the deployment.

Keywords: Software Process Development. Ontologies. Software Reuse. Multi-agent Systems.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AAUT	Aplication Agents Under Test
ADEMAS	Aplication Design technique for Multi-Agent Systems
AIMAS	Application Implementation Technique for Multi-Agent Systems
AUML	Agent Unified Modeling Language
CASE	Computer-Aided Software Engineering
CDB	Component-based Development
DDEMAS	Domain Design technique for Multi-Agent Systems
DIMAS	Domain Implementation technique for Multi-Agent Systems
DSL	Domain-Specific Language
FBC	Filtragem Baseada no Conteúdo
FC	Filtragem Colaborativa
FH	Filtragem Híbrida
FIPA	Foundation for Intelligent Physical Agents
GENMADEM	Generative Multi-agent Domain Engineering Methodology
GESEC	Grupo de pesquisa em Engenharia de Software e Engenharia do Conhecimento
GRAMO	Generic Requiremente Analysis Method based on Ontologies
HTML	Hypertext Marckup Language
JADE	Java Agent Development Framework
JESS	Java Expert System Shell
MAAEM	Multi-agent Aplication Engineering Methodology
MADAE-IDE	Multi-agent Domain and Application Engineering – Integrated Development Environment
MADAE-ML	Multi-agent Domain and Aplication Engineering – Modeling Language
MADAE-Pro	Multiagent Domain and Application Engineering Process
MADEM	Multi-agent Domain Engineering Methodology
MDA	Model Driven Architecure
MOF	Meta-Object Facility
MVP-L	MultiView Process Modeling Language
OCL	Object Constraint Language
OMG	Object Management Group

ONTOMADEM	Ontology for Multi-Agent Domain Engineering Methodology
ONTORMAS	Ontology for Reusing Multiagent Software
ONTOWUN-DM	Ontolology-driven Model of Recommender Systems based on Web Usage Mining
OPEN	Object-Oriented Process, Environment and Notation.
OPF	OPEN Process Framework
PASSI	Process for Agent Societies Specification and Implementation
PML	Process Modeling Language
PTK	PASSI Toolkit
RUP	Rational Unified Process
SAADAM	Software Agent Development An Agile Methodology
SPEM	Software Process Engineering Metamodel
SRAMO	Specific Requirement Analysis Method based on Ontologies
TA	Test Agents
TAOM4E	Tool for Agent Oriented visual Modeling for the Eclipse platform
UML	Unified Modeling Language
XP	Extreme Programming

SUMÁRIO

1 INTRODUÇÃO

O aumento da complexidade dos produtos de software atuais tem criado uma demanda não só por novos paradigmas de desenvolvimento capazes de lidar de forma mais eficiente com tal complexidade como também por processos que forneçam diretrizes aos desenvolvedores para guiá-los nas tarefas de desenvolvimento. Neste contexto, destaca-se o paradigma de desenvolvimento de software multiagente. Neste paradigma, as abstrações de software, os agentes, possuem um comportamento autônomo deixando de ter comportamento estritamente previsível para ter controle sobre seu próprio comportamento.

A produtividade e qualidade são dois fatores desejáveis no desenvolvimento de um produto de software e uma das formas de obtê-los é através do reúso. O reúso de software é um processo de criação de sistemas de software a partir de software existente, ao invés de desenvolvê-los a partir do zero (KRUEGER, 1992).

As ontologias (GRUBER, 1995) provêem uma terminologia não-ambígua que pode ser compartilhada por todos os envolvidos no processo de desenvolvimento de software. Elas podem ser tão genéricas quanto necessário permitindo o seu reúso e fácil extensão. Estas características tornam a ontologias úteis para representar o conhecimento das técnicas e metodologias para a Engenharia de Software e é um mecanismo de abstração apropriado para especificação de artefatos de software de alto nível como modelos de domínio, *frameworks* e padrões de software (GIRARDI et al., 2004).

Nesta dissertação de mestrado, propõe-se MADAE-Pro (*Multiagent Domain and Application Engineering Process*), um processo baseado no conhecimento para o desenvolvimento de famílias de aplicações multiagente. Este processo integra conceitos da Engenharia de Domínio e da Engenharia de Aplicações para possibilitar o reúso de artefatos de software em vários níveis de abstração. A Engenharia de Domínio é um processo para criação de abstrações de software reusáveis no desenvolvimento de uma família de aplicações em um domínio particular de problema. A Engenharia de Aplicações é um processo para construção de aplicações baseadas no reúso de artefatos de software previamente produzidos no processo da Engenharia de Domínio.

Os conceitos utilizados pelo MADAE-Pro são expressos em uma ontologia e os seus produtos são representados como instâncias desta. Isso permite incorporar ao MADAE-Pro as vantagens fornecidas pelas ontologias como a estruturação do conhecimento, compartilhamento de conhecimento e facilidade de reúso.

1.1 Objetivos

O objetivo geral desse trabalho é a especificação de um processo baseado no conhecimento para melhorar a qualidade e produtividade do desenvolvimento de famílias de aplicações multiagente através do suporte ao reúso de software.

Para alcançar esse objetivo geral, planeja-se atingir os seguintes objetivos específicos:

- Revisão e refinamento das metodologias MADEM (*Multi-agent Domain Engineering Methodology*) (GIRARDI e MARINHO, 2007) e MAAEM (*Multi-agent Domain Engineering Methodology*) (LINDOSO, 2006) (LEITE et al., 2008b) e da ferramenta ONTORMAS (*ONTOlogy driven tool for the Reuse of Multi-Agent Systems*) (LEITE et al., 2008a) e integrá-las no processo MADAE-Pro.
- Especificar o processo MADAE-Pro utilizando a linguagem SPEM.

1.2 Estruturação

Este trabalho, incluindo esta introdução, está estruturado em quatro capítulos. No capítulo 2, é apresentada uma visão geral dos processos para desenvolvimento de software e da sua classificação quanto ao ciclo de vida, formalidade e aplicabilidade. Alguns importantes processos de desenvolvimento, dos paradigmas orientado a objetos e a agentes são discutidos e é feito um comparativo entre eles. Este capítulo traz também um estudo sobre o processo de desenvolvimento baseado no reúso de software e das ontologias. Por fim, algumas linguagens para modelagem de processos de desenvolvimento de software e um breve comparativo entre elas são apresentados.

No capítulo 3, parte central da dissertação, é apresentada a especificação do processo MADAE-Pro utilizando a linguagem SPEM. Neste capítulo, é

apresentada também a ferramenta ONTORMAS que suporta as atividades de modelagem do MADAE-Pro e a linguagem de modelagem de sistemas multiagente MADAE-ML (*Multi-agent Domain and Aplication Engineering – Modeling Language*) que é utilizada para representar os modelos desenvolvidos através do MADAE-Pro. O processo é exemplificado através de exemplos extraídos do desenvolvimento da ONTOSERS (MARIANO, 2008), uma família de aplicações para recomendação de informações baseada na tecnologia da Web Semântica, e da INFOTRIB (MARIANO, 2008), uma aplicação para recomendação de informações para o domínio tributário.

No último capítulo, as conclusões do trabalho são apresentadas, incluindo as contribuições do processo MADAE-Pro para o grupo GESEC e para o desenvolvimento de software multiagente, as limitações que o mesmo ainda possui e que precisam ser superadas e as perspectivas de trabalhos relacionados que estão sendo ou que serão desenvolvidos no futuro.

2 PROCESSOS PARA DESENVOLVIMENTO DE SOFTWARE

2.1 Introdução

Um processo de desenvolvimento é referido dentro da comunidade da Engenharia de Software como a soma de todas as atividades da Engenharia de software, abrangendo todo o ciclo de vida, desde a captura de requisitos até a atividade de manutenção (MOHAMED, 2001).

Na literatura, existem divergências na conceitualização dos termos processo de software e metodologia, em vários casos estes são tratados como sinônimos e outros definem que uma metodologia faz parte de um processo. Segundo (GHEZZI et al., 1991), uma metodologia é um conjunto de métodos cobrindo e conectando diferentes estágios de um processo. Para (FUGGETTA, 2000), um processo de desenvolvimento de software é um conjunto coerente de políticas, estruturas organizacionais, tecnologias, procedimentos e artefatos que são necessários para conceber, desenvolver, implantar e manter um produto de software.

Tanto metodologia quanto processo enfatizam as diferentes atividades necessárias para construir um produto de software. Entretanto, enquanto o conceito de processo de desenvolvimento de software é mais geral englobando todas as atividades de desenvolvimento, incluindo suas fases e a ordem na qual são realizadas, a metodologia trata mais especificamente em como cada fase de desenvolvimento deverá ser realizada e dos artefatos a serem produzidos. A partir dessas definições, concluímos que um processo de desenvolvimento envolve a utilização, entre outros elementos, de uma metodologia. No processo, é definido “o que” é feito para desenvolver um produto de software através da definição de fases e a subdivisão destas em tarefas, também chamadas de passos ou atividades. Também, fornece as diretrizes de “como” desenvolver o software através de um conjunto de métodos ou de uma metodologia, quais insumos usa e quais os artefatos produzidos em cada fase de desenvolvimento. Especifica também “quem” é o responsável por cada tarefa de desenvolvimento e em que “ordem” através da definição de um ciclo de vida. Um processo pode ser ainda apoiado por ferramentas que auxiliem a realização de uma ou mais etapas de desenvolvimento, o que propicia maior produtividade, qualidade e controle do processo.

Um processo de desenvolvimento de software é dividido genericamente em três níveis de abstração básicos, como ilustrado na Figura 1 (ROLLAND, 1993). No primeiro nível, estão os conceitos fundamentais do processo, como fase e produto e os relacionamentos permitidos entre eles. Nesse nível, são definidos quais elementos fazem parte do processo, geralmente, através de uma linguagem de modelagem de processos. A representação do primeiro nível é chamada de meta-modelo. No segundo nível são definidos os modelos do processo, ou seja, uma descrição do processo sob várias perspectivas. Esse nível representa uma instanciação do primeiro nível, agrupando os conceitos e relacionamentos em modelos. Por exemplo, um modelo representando as fases que compõem o processo ou fluxo de realização das atividades do processo. Já o terceiro nível representa a instanciação do processo para desenvolvimento de um determinado produto. Alguns autores denominam esse último nível como de execução do processo.

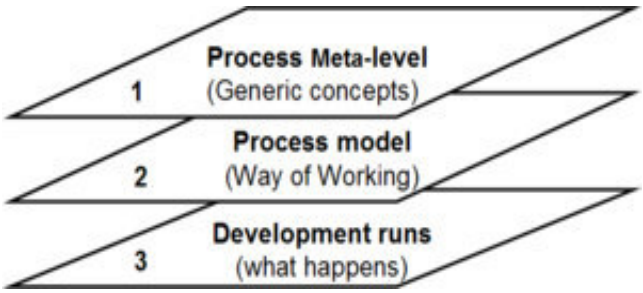


Figura 1 - Níveis de Abstração do Processo. Fonte: (ROLLAND, 1993)

Os processos de software trazem em sua especificação experiências de desenvolvimento bem-sucedidas que podem ser compartilhadas entre todos os envolvidos no desenvolvimento, aumentando assim a qualidade do produto desenvolvido. Com isso, o desenvolvimento de software realizado de maneira informal, ou seja, sem utilização de métodos ou técnicas tem perdido cada vez mais espaço devido à crescente exigência por qualidade e produtividade.

Um processo de desenvolvimento de software é um modelo que especifica o ciclo de vida do software, descrevendo as fases pelas quais transita o produto de software ao longo de sua concepção e desenvolvimento e a metodologia que estabelece as técnicas a serem aplicadas em cada uma das fases de acordo a um determinado paradigma de desenvolvimento (LEITE e GIRARDI, 2009).

Neste capítulo são tratados diversos aspectos dos processos para o desenvolvimento de software, incluindo sua classificação quanto ao ciclo de vida, formalidade e aplicabilidade e algumas linguagens utilizadas para representá-los. Além disso, são discutidos aspectos importantes do reúso e da sua incorporação no processo de desenvolvimento de software. Ao final deste capítulo, um breve estudo comparativo entre os processos apresentados é feito.

2.2 Tipos de Processos de Software

Os processos de desenvolvimento de software diferem em vários aspectos como, por exemplo, no ciclo de vida utilizado, na metodologia, nas ferramentas e linguagens que auxiliam o desenvolvimento. Esses aspectos estão diretamente relacionados ao software a ser desenvolvido, por exemplo, um software complexo dificilmente poderá ser desenvolvido seguindo um processo que adote o ciclo de vida em cascata, onde o desenvolvimento é realizado de forma seqüencial. Com isso, a escolha do processo de software depende das características do produto de software a ser desenvolvido. Nas subseções seguintes são apresentados em maiores detalhes alguns aspectos em que os processos de software se diferenciam e que devem ser levados em conta na escolha do processo mais adequado ao software que se quer desenvolver.

2.2.1 Classificação quanto ao Ciclo de Vida

Um ciclo de vida define os diferentes estágios no tempo de vida de um produto de software. Normalmente, esses estágios são a análise e especificação de requisitos, projeto, desenvolvimento, verificação e validação, implantação e manutenção (FUGGETTA, 2000). Um processo está associado a um ciclo de vida ou a uma combinação destes. Entre os mais utilizados estão o ciclo de vida em cascata, o ciclo de vida baseado na prototipação, o ciclo de vida iterativo e incremental e o ciclo de vida em espiral.

O ciclo de vida em cascata

O ciclo de vida em cascata foi o primeiro a ser definido na Engenharia de Software e ainda é bastante utilizado por sua simplicidade. Ele consiste de um

conjunto de fases realizadas de forma seqüencial com poucas retroalimentações, por isso, é recomendado para sistemas simples, onde os requisitos estão claros e quase não há necessidade de iterações entre as fases.

Na Figura 2 é ilustrado o ciclo de vida em cascata. Na fase de *Definição de Requisitos* os requisitos do sistema são definidos através de técnicas como, por exemplo, “tormentas de idéias” e de entrevistas com o cliente. Na fase *Projeto de Sistema e de Software* é dada uma solução arquitetural aos requisitos identificados na fase anterior, através da definição de abstrações de software (módulos) e de como eles se comunicam entre si. Na fase *Implementação e Teste de Unidades,* cada módulo do software é codificado e testado a fim de verificar se atende a sua especificação*. Na fase Integração e Testes de Sistemas,* os módulos são integrados e testados em conjunto para verificar se todos os requisitos do sistema foram atendidos. Por fim, na fase *Operação e Manutenção,* o software é instalado no ambiente do usuário e são realizadas manutenções corretivas, para corrigir erros não identificados durante o desenvolvimento, e manutenções evolutivas, para adição de novas funcionalidades ao sistema ou alteração de alguma funcionalidade preexistente. Essas fases são genéricas e comumente seguidas no desenvolvimento de software. Entretanto, nem todos os processos de desenvolvimento incluem todas essas fases.

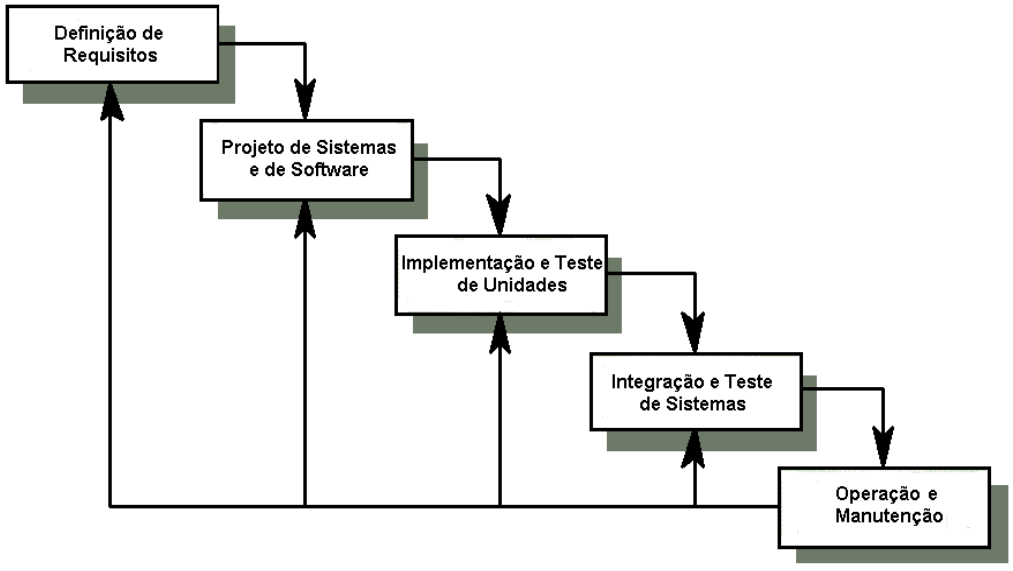


Figura 2 - Ciclo de Vida em Cascata. Fonte: (Sommerville, 2004)

O ciclo de vida em cascata apresenta algumas vantagens como:

- Simplicidade. A estrutura seqüencial desse ciclo de vida permite um simples entendimento do processo de software (PRESSMAN, 2002);
- A divisão das tarefas por fase é bem-definida, ou seja, é bem claro onde começa e termina cada fase, facilitando o processo de divisão de tarefas e o gerenciamento do projeto de software (PRESSMAN, 2002);
- Integração. Ao contrário do que ocorre em outros ciclos de vida, onde a aplicação é divida em várias partes e desenvolvida, muitas vezes, por diversas equipes e depois integrada, aqui o software é desenvolvido de forma integrada logo no início do desenvolvimento, diminuindo o esforço de integração.

Entretanto, este ciclo apresenta também desvantagens (PRESSMAN, 2002):

- Para alguns tipos de aplicação é difícil explicitar todos os requisitos logo no início do desenvolvimento, realizar todo o projeto e depois construir todo o software;
- É difícil modificar os requisitos depois que o sistema começa a ser desenvolvido;
- Erros na especificação dos requisitos podem ser detectados tardiamente, o que pode representar aumento nos custos, atraso na entrega do software e até a não realização do projeto;
- O cliente demora muito tempo para receber uma versão executável do software, uma vez que este só é entregue ao final do processo;
- Como o término de uma fase é pré-requisito para iniciar a seguinte, um atraso nas fases iniciais atrasa as demais fases de desenvolvimento.

O ciclo de vida baseado na prototipação

O ciclo de vida baseado na prototipação é indicado quando há dificuldade na identificação dos requisitos. Outra característica é que esse ciclo de vida é utilizado como parte de outros ciclos de vidas como, por exemplo, o iterativo e incremental e o em espiral para ajudar o cliente e o analista/desenvolvedor a definirem os requisitos da aplicação. Para realizar o desenvolvimento de uma

aplicação utilizando o ciclo de vida baseado na prototipação dois pré-requisitos devem ser observados. Primeiro, o protótipo tem que ser desenvolvido em pouco tempo para que não consuma muito do tempo destinado as outras tarefas do projeto de software, isso pode ser obtido, por exemplo, através da utilização de ferramentas CASE. Segundo, o protótipo deve ser de baixo custo, pelo menos deve ser mais barato construí-lo do que desenvolver o produto final, uma vez que o protótipo poderá ser descartado.

Na Figura 3 o ciclo de vida baseado na prototipação é ilustrado, onde o desenvolvimento inicia-se com a *Obtenção de Requisitos*, onde os requisitos iniciais da aplicação são definidos com a ajuda do usuário. A seguir, um *Projeto Rápido* é realizado, contendo as interfaces com o usuário. Depois é realizada a *Construção do Protótipo*, onde as interfaces são implementadas rapidamente, sem se preocupar com questões de desempenho, validação, etc. Em seguida, é realizada a *Avaliação pelo Cliente*, onde o cliente avalia o protótipo, fornecendo um "feedback" ao desenvolvedor. A partir das observações do cliente é realizado o *Refinamento de Requisitos*, onde os requisitos são refinados e o ciclo volta para o *Projeto Rápido* e se repete até que todos os requisitos tenham sido definidos. Quando os requisitos estiverem bem-definidos, o protótipo é descartado e é realizada a *Construção do Produto* utilizando outro ciclo de vida. Há também uma variante do ciclo de vida baseado na prototipação chamada de prototipação evolucionária, onde o protótipo não é descartado, mas sim refinado até se tornar uma aplicação funcional.

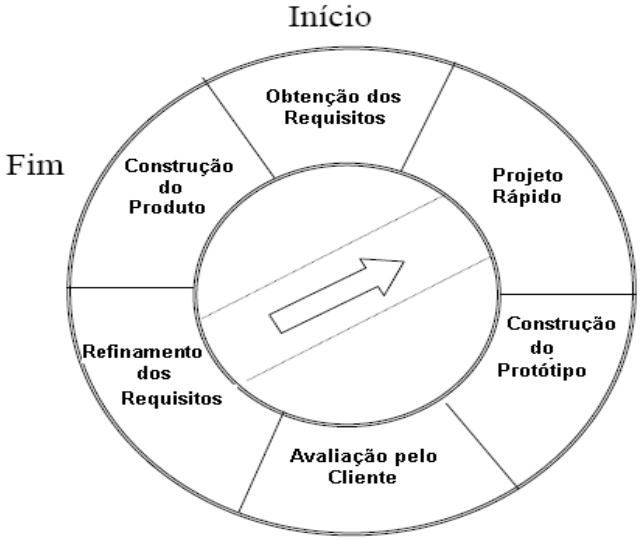


Figura 3 – Ciclo de Vida Baseado na Prototipação. Fonte: (Sommerville 2004)

O ciclo de vida baseado na prototipação apresenta algumas vantagens como:

- Facilidade para identificar os requisitos iniciais;
- Facilita a comunicação entre o usuário e o desenvolvedor;
- Requisitos bem-definidos evitam a propagação de erros para as fases posteriores;
- Garantia de o software atender as necessidades do cliente,

Entretanto, este ciclo apresenta também desvantagens (PRESSMAN, 2002):

- O cliente pode pressionar para utilizar o protótipo como produto final;
- É difícil prever o tempo em que o protótipo estará pronto e o seu custo.

O ciclo de vida iterativo e incremental

No ciclo de vida iterativo e incremental, o desenvolvimento do software é dividido em partes, o que facilita o entendimento de sistemas complexos. Cada parte representa um ciclo completo, incluindo captura de requisitos, projeto e implementação, exatamente como no modelo em cascata; a diferença é que aqui não é desenvolvido o software como um todo, mas dividido em partes. Em cada ciclo pode haver refinamentos com o intuito de melhorar ou corrigir etapas de desenvolvimento, chamadas de iterações. Na Figura 4, apresenta uma analogia ao desenvolvimento utilizando incrementos e iterações. São realizados incrementos, adicionando-se continuamente novas funcionalidades ao software até que se obtenha o software completo; já as iterações são refinamentos com o intuito de corrigir/melhorar partes do software até que este seja considerado adequado.

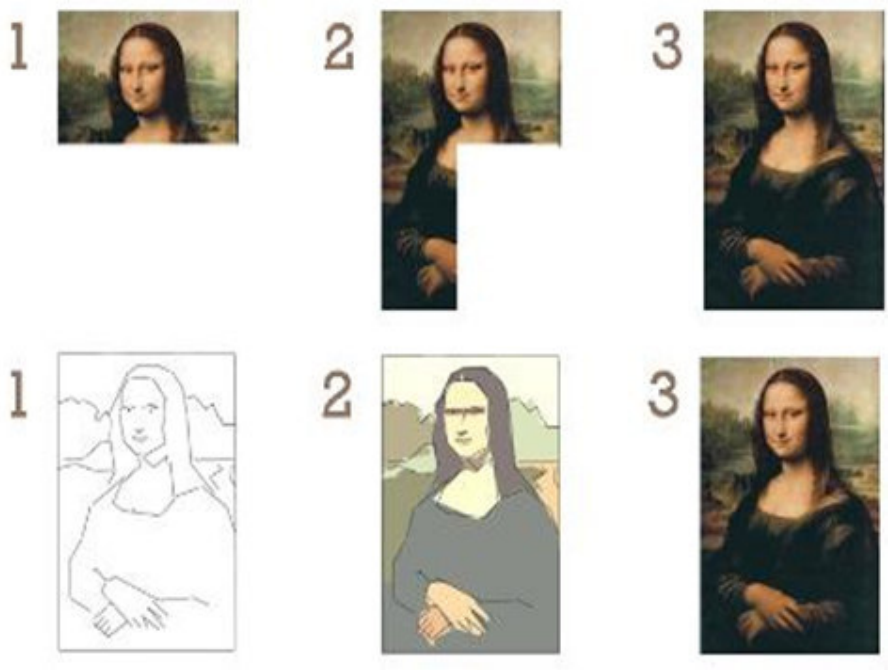


Figura 4 – Sucessivos Incrementos e Iterações. Fonte: (PATTON, 2008)

O ciclo de vida iterativo e incremental apresenta algumas vantagens como (Sommerville 2004):

- Disponibilidade de partes prontas do sistema mais cedo;
- Várias equipes de desenvolvimento podem trabalhar paralelamente;
- Facilidade nos testes. Geralmente, testar cada incremento é mais fácil do que testar o software pronto e em uma única vez;
- *Feedback* do cliente a cada incremento feito;
- Se houver erros nos requisitos o custo será menor, uma vez que não precisa corrigir o sistema todo, somente uma parte;
- Partes “urgentes” para o cliente podem ser desenvolvidas e implantadas enquanto outras menos importantes podem ainda estar sendo desenvolvidas.

Entretanto, este ciclo apresenta também desvantagens (Sommerville 2004):

- Alguns sistemas têm suas funcionalidades muito relacionadas, dificultando a divisão do desenvolvimento em partes.
- Dificuldade na integração das partes desenvolvidas.

O ciclo de vida em espiral

No ciclo de vida em espiral, o desenvolvimento é divido em um conjunto de ciclos, onde cada ciclo é subdivido em quatro partes (quadrantes). Cada ciclo representa uma fase de desenvolvimento e os quadrantes representam as atividades realizadas nessas fases. Uma característica importante do ciclo de vida em espiral é a análise de riscos em cada fase de desenvolvimento, onde os riscos são identificados e são propostas alternativas que tentem eliminar ou minimizar os riscos. Outra característica é a incorporação da prototipação para melhor identificar os requisitos e riscos. As fases de Análise de Viabilidade, Análise de Requisitos, Projeto do Produto, Projeto Detalhado, Implementação e Teste são realizadas no ciclo de vida em espiral. Em cada fase é realizada a atividade "*determinação de objetivos, alternativas e restrições"* (1º quadrante), onde são definidos os objetivos específicos do sistema para a fase, suas restrições e o planejamento do projeto. É realizada também a identificação dos riscos e são propostas estratégias com intuito de sanar ou amenizar esses riscos. A seguir é realizada a "*avaliação de alternativas, e riscos"* (2º quadrante), onde estratégias alternativas para evitar riscos definidas anteriormente são selecionadas e colocadas em prática. Por exemplo, se o risco for a possibilidade de erros na identificação dos requisitos pode ser construído um protótipo a fim de melhorar a comunicação com o cliente. Se o risco for considerado for muito grave, pode inviabilizar todo o projeto de software. No terceiro quadrante, "*desenvolve e verifica o produto no nível seguinte",* é realizado o desenvolvimento do software, para esta tarefa pode-se utilizar outro ciclo de vida como, por exemplo, a prototipação evolutiva. No último quadrante, o software é avaliado e um novo ciclo se inicia.

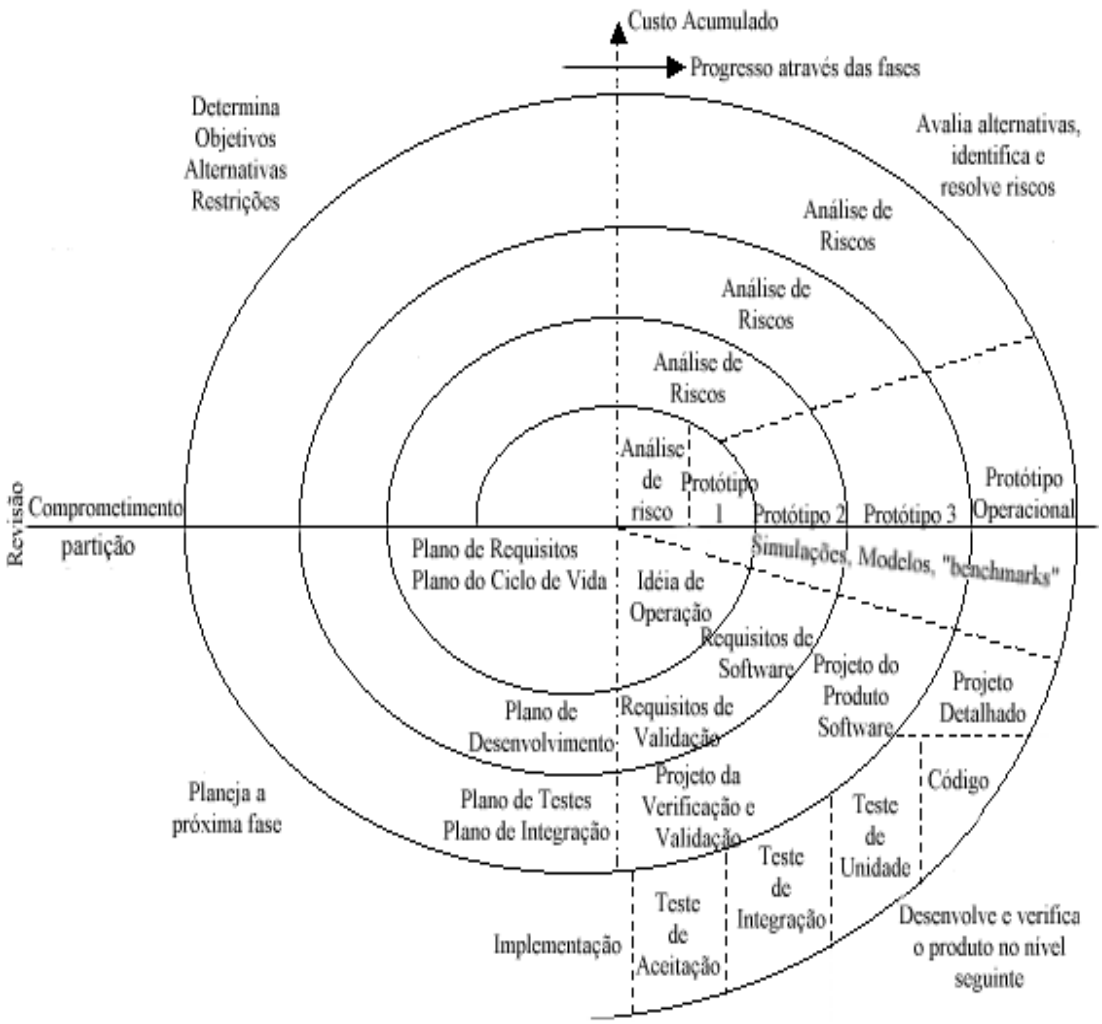


Figura 5 - Ciclo de Vida em Espiral. Fonte: (Sommerville, 2004)

O ciclo de vida em espiral apresenta algumas vantagens como:

- Análise de riscos em todas as fases de desenvolvimento, evitando ou minimizando que possíveis riscos inviabilizem o projeto de software;
- Uso de protótipos, o que facilita a comunicação com os clientes.

Entretanto, este ciclo apresenta também desvantagens:

- Complexidade para gerenciar e usar;
- Como o sistema é desenvolvido de forma seqüencial, erros nas fases iniciais podem afetar todo o projeto de software.

2.2.2 Classificação quanto à Formalidade

De acordo com a linguagem em que for especificado, um modelo de processo de software pode ser formal, semi-formal e informal. Um modelo de processo de software especificado textualmente é considerado *informal* devido à possibilidade de existirem ambigüidades e imprecisões, próprias da linguagem natural. Um processo de software é dito *formal*, quando ele é especificado em uma linguagem formal, processável por máquina. Já um processo especificado somente com uma linguagem visual é considerado como *semi-formal* (LEMOINE e GAUDI 2004), uma vez que apesar de as linguagens visuais fornecerem uma padronização da terminologia e representação gráfica dos elementos do processo e uma sintaxe precisa, essas em geral não são executáveis e não permitem validação de modelos.

Representar um processo utilizando uma linguagem formal apresenta diversas vantagens em relação às linguagens textual e visual como, por exemplo, validação de modelos, maior expressividade semântica, inferências mais precisas e a geração automática de código. Entretanto, as linguagens formais não são expressivas visualmente e requerem um conhecimento matemático avançado. A combinação de uma linguagem visual com uma linguagem formal tem se mostrado adequado, assim, o processo é representado em parte por alguma linguagem visual como, por exemplo, o SPEM e também através de uma linguagem formal, como OCL (BOOCH et al. 1999), podendo ser ainda complementada por uma descrição em linguagem natural. Dessa forma, os usuários do processo, podem ter um maior contato com a linguagem visual, enquanto que as definições formais podem ser embutidas em ferramentas de auxilio ao processo, tornando-se transparente ao usuário.

2.2.3 Classificação quanto à Aplicabilidade

A aplicabilidade do processo está associada ao paradigma de desenvolvimento, podendo também estar relacionada ao tamanho do projeto de software e as características do software a ser desenvolvido. Por exemplo, para projetos de software de pequeno porte se recomenda a utilização de processos ágeis que não geram muita documentação e são mais centrados na implementação,

enquanto que no caso de projetos de software complexos um processo tradicional é mais recomendado. Existem também algumas aplicações com características e necessidades especiais como é o caso de aplicações móveis e de tempo-real.

Nos últimos anos, surgiram diversos paradigmas de desenvolvimento como, por exemplo, o desenvolvimento estruturado e a orientação a objetos. Normalmente, à medida que surge um novo paradigma de desenvolvimento são propostas diversas metodologias com o intuito de guiar e aumentar a qualidade do desenvolvimento de um produto de software seguindo esse paradigma. Com o passar do tempo muitas dessas metodologias desaparecem e outras se consolidam.

O desenvolvimento orientado a agentes é um novo paradigma de desenvolvimento que tem começado a ser utilizado, principalmente no meio acadêmico e em grandes organizações de tecnologia como o Google e a Microsoft. O que diferencia esse paradigma dos demais é a abstração de software utilizado: o agente. Um agente representa um subsistema que possui características próprias como autonomia, pró-atividade e comportamento dirigido por objetivos. Os agentes podem trabalhar de forma conjunta para atingir os objetivos da organização constituindo um sistema multiagente. Essas características dos agentes os tornam apropriados para abordar problemas e soluções complexos, aumentando assim a necessidade de se utilizar algum processo que guiem os desenvolvedores durante a realização das tarefas de desenvolvimento.

Nas próximas subseções, aborda-se alguns dos processos de desenvolvimento de software mais utilizados, seguindo os paradigmas da orientação a objetos e da orientação a agentes e, ao final, faremos um estudo comparativo das características entre esses processos. A análise dos processos é importante, pois fornece subsídios tanto para melhorar o processo aqui definido, quanto para avaliá-lo.

2.2.3.1 Processos Aplicados ao Desenvolvimento Orientado a Objetos

A Orientação a Objetos é um dos paradigmas de desenvolvimento mais utilizado atualmente. Vantagens como facilidade de manutenção através de mecanismos de abstração e reúso de código com a utilização de conceitos de herança e de instanciação, têm feito com que a programação estruturada venha gradativamente sendo substituída pelo paradigma da orientação a objetos.

Linguagens de modelagem como a UML (Unified Modeling Language) (BOOCH et al., 1999) e de programação como Java, padrões no mercado de desenvolvimento, também tem contribuído para isso.

O Processo Unificado da Rational

O Processo Unificado da Rational (RUP) é um dos processos mais utilizados atualmente no desenvolvimento de software. Ele adota a UML como linguagem de modelagem visual. O RUP também possui uma grande quantidade de ferramentas desenvolvidas pela IBM que auxiliam grande parte das atividades realizadas durante o seu ciclo de vida, como o Rational Rose para modelagem visual com UML e o *Rational TestManager* para gerenciar as atividades de teste.

O RUP é um processo de software flexível e configurável (KROLL e KRUCHTEN, 2003). Isso quer dizer que ele permite desenvolver tanto software de pequeno quanto de médio e grande porte. Dependendo do software que estiver sendo desenvolvido, somente uma parte do processo será realizada, concentrando assim mais tempo nas atividades de programação e gerando menos documentação.

O RUP é dirigido por casos de uso. Isso significa que os casos de uso (um caso de uso representa uma funcionalidade do sistema do ponto de vista de um ator) são utilizados durante todo o processo de desenvolvimento. Eles representam os requisitos da aplicação, são utilizados para verificar e validar a arquitetura, realizar de testes, criar a documentação dos usuários e implantar o sistema.

O RUP segue práticas já consagradas na Engenharia de Software, entre elas:

- **Desenvolvimento Iterativo.** O desenvolvimento de software de maneira seqüencial não tem se mostrado eficiente para sistemas complexos, uma vez que é difícil definir o escopo do sistema logo no início do desenvolvimento, depois realizar o projeto, a implementação e depois testar o software, entretanto, para softwares simples a abordagem seqüencial é perfeitamente viável. O RUP recomenda que o desenvolvimento de software complexo, seja realizado através de sucessivas iterações, seguindo a bem conhecida técnica de “dividir para conquistar”. O desenvolvimento iterativo permite uma compreensão crescente do problema através de sucessivos incrementos e

refinamentos. Cada iteração corresponde a um ciclo de vida completo e resulta em código executável.

- **Uso de Arquiteturas baseadas em componentes.** O uso de arquitetura baseada em componentes flexibiliza o desenvolvimento do software, uma vez que componentes, podem ser adicionados ou modificados sem que haja grandes alterações na arquitetura global do software.

A Figura 6 ilustra as fases e as disciplinas do RUP, sendo que o eixo horizontal mostra as fases, enquanto que o eixo vertical representa um conjunto de atividades organizadas em disciplinas. Uma fase representa o fluxo de desenvolvimento de software de maneira temporal através de marcos bem definidos e as disciplinas representam as atividades organizadas de maneira lógica, por natureza. Cada fase pode passar por uma ou mais disciplinas e pode existir uma ou mais interações a fim de corrigir ou melhorar o desenvolvimento. Na Figura 6 é ilustrado como as atividades (organizadas em disciplinas) vão variando através do tempo (fases). Nas primeiras iterações, vemos que a ênfase do desenvolvimento é a analise de requisitos, depois no projeto, na implementação, e assim por diante.

O RUP é estruturado nas fases de *Iniciação*, *Elaboração*, *Construção* e *Transição* e nas disciplinas de *Modelagem de Negócios*, *Requisitos*, *Análise e Design*, *Implementação*, *Teste*, *Implantação*, *Gerenciamento de Configuração e Mudança* e *Gerenciamento de Projeto Ambiente*.

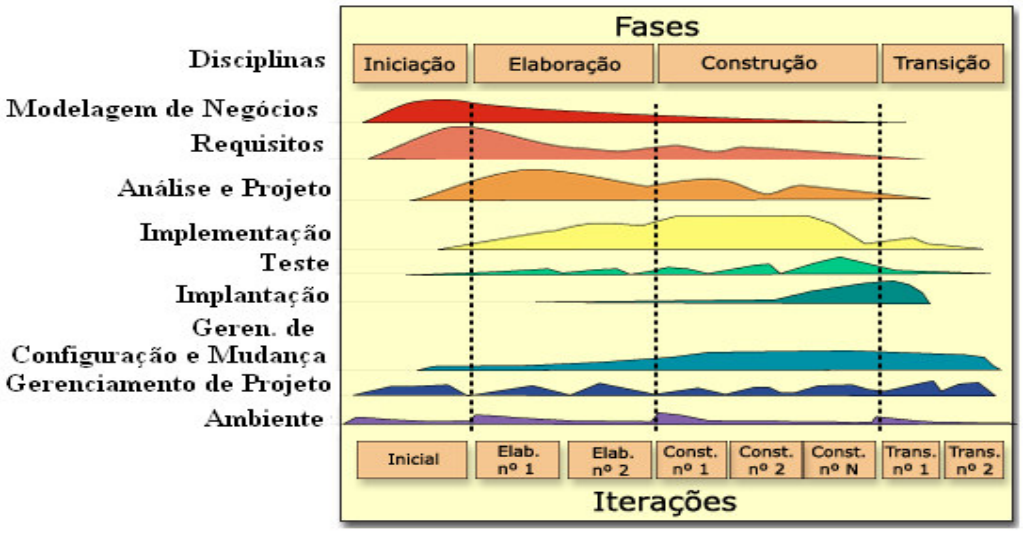


Figura 6 - Rational Unified Process

Na fase de *Iniciação* é dado, ênfase na definição ao escopo do software. O objetivo é identificar os requisitos, estimar recursos, prazos e riscos do projeto de software. Nessa fase também é construído um protótipo. Na fase de *Elaboração* é dado, ênfase na definição da arquitetura do software. Na fase de elaboração os requisitos são levantados em detalhes. Nessa fase também é realizada a análise de riscos e os elementos de alto risco são eliminados do projeto. Na fase de *Construção* é dado, ênfase na implementação do software. Nessa fase o software é implementado de forma iterativa e incremental e são também realizados testes. Na fase de *Transição* é dado, ênfase na implantação do software no ambiente do usuário final. São realizados testes finais, treinamento de usuários, etc.

O Processo OPEN

O OPEN (Object-oriented Process, Environment and Notation) (GRAHAM et al., 1997) é um processo para desenvolvimento de aplicações no paradigma orientado a objetos, no entanto, nos últimos anos tem sido feitos esforços com intuito de estendê-lo para o paradigma orientado a agentes.

O OPF (OPEN Process Framework) define um metamodelo do processo OPEN, o qual permite gerar processos específicos a uma organização, chamados de instâncias do processo. Cada instância é criada escolhendo-se as atividades, tarefas e técnicas específicas da organização.

O OPF define um conjunto de elementos que devem estar presentes no processo de desenvolvimento de software. Esses elementos são: “Work Products” que define os artefatos desenvolvidos no processo; “Producers” representa os profissionais que desenvolvem os artefatos; “Work Units” são as atividades realizadas pelos “producers” para desenvolver um “work product”; “Stages”, é um elemento mais genérico, que pode representar as fases do processo que envolvem uma ou mais “work units” quanto um ciclo de vida, que corresponde a um conjunto de fases do processo com uma ordem definida; “Languages”, representam o meio pelo qual o “work product” foi definido, por exemplo se o “work product” for um documento de requisitos o meio poderá ser a língua inglesa, já se for um código fonte poderá ser a linguagem Java. Na Figura 7 esses elementos do processo e como eles se relacionam entre si são ilustrados.

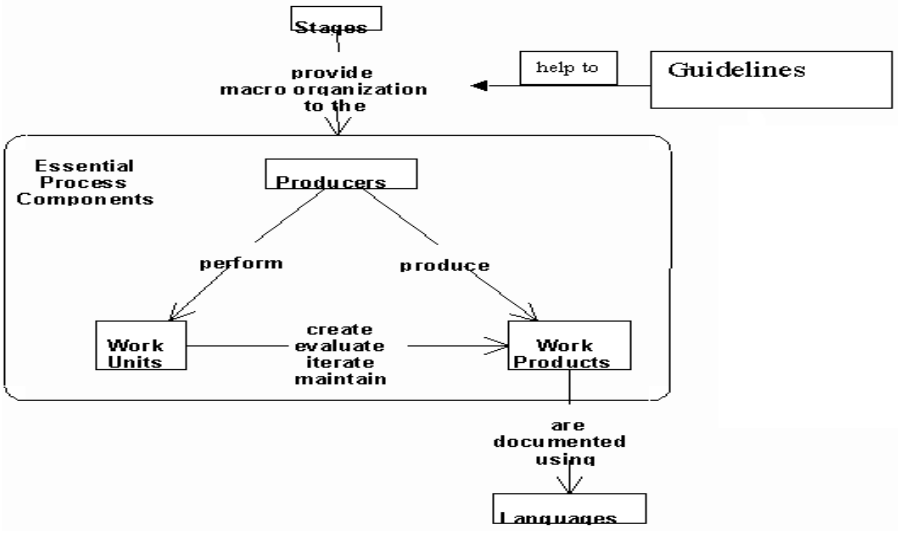


Figura 7 - Elementos do OPF. Fonte: (OPEN, 2008)

As fases definidas no OPEN são *Iniciação*, *Construção*, *Uso* e *Retirada*. Na fase de Iniciação, a captura de requisitos e o projeto da aplicação são realizados. Na fase de Construção, os produtos são desenvolvidos e preparados para serem entregues ao usuário. Na de fase Uso, o software é entregue ao usuário e colocado em funcionamento. Na fase de Retirada, o software é retirado de funcionamento.

Processos Orientados a Objetos Ágeis

Enquanto que nos projetos de software de médio e grande porte os processos tradicionais como o RUP e o OPEN são mais adequados, nos de pequenos portes eles representam custos e tempo de desenvolvimento desnecessário. Atualmente, tem ganho espaço os processos denominados ágeis, também conhecidos por “Métodos Ágeis”, nesse tipo de processo o enfoque é o código executável.

Extreme Programming (XP) (AMBLER, 2003) é um processo de desenvolvimento de software orientado a objetos ágil. Esse processo é para projetos de software de pequeno ou médio porte com equipes de até 12 desenvolvedores. Ele é recomendado também para projetos onde os requisitos mudam freqüentemente.

O ciclo de vida desse processo é incremental, onde o produto de cada iteração é validado pelo cliente. Normalmente, cada iteração dura de uma a duas

semanas. Na Figura 8, é ilustrado o ciclo de vida do XP. Primeiro, os requisitos são definidos junto ao cliente em "*User Stories*" (Figura 9), que são formas de representar os requisitos do sistema como casos de usos, escritas diretamente pelos usuários/clientes sem utilizar termos técnicos. A seguir, a fase de *"Release Planning"* é realizada, onde junto com o cliente são selecionadas quais das *"User Stories*" serão desenvolvidas primeiro e quando elas serão entregues para o cliente. Depois as *"User Stories"* são testadas e implementadas e, por fim, avaliadas pelo cliente. Como pode ser visualizado na figura, em cada fase de desenvolvimento do XP pode ser feitas iterações para refinar o produto, bem como para corrigir bugs.

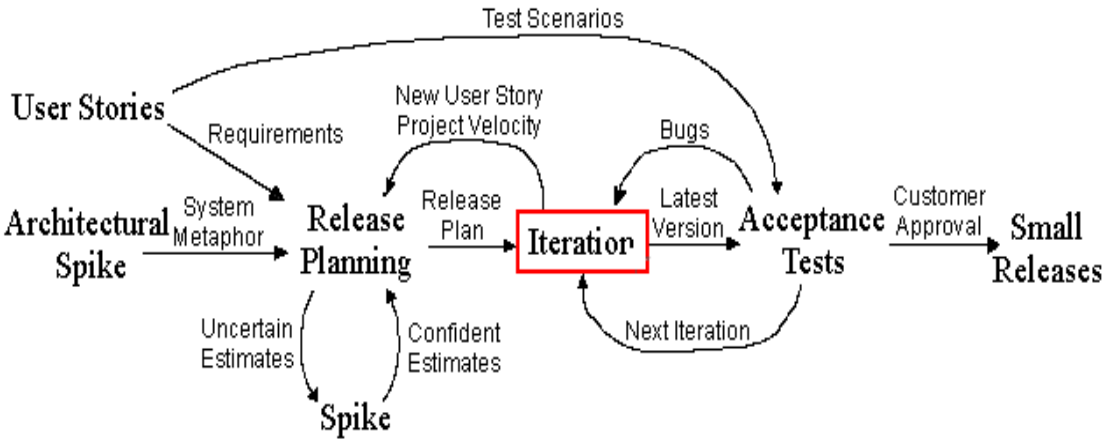


Figura 8 - Ciclo de vida do XP. Fonte: (XP, 2008)

ID# 005 Usuario Fulano de tal
<Story name> Correlação de atos
Source:
<Story texto>
Os atos após serem identificados e filtrados na consulta ao acervo do Diário Oficial devem ser correlacionados de forma que se identifique ...
Data início 01/02/2006 Data Fim 08/02/2006

Teste de Aceitação: ID# 011

Figura 9 - Exemplo de "User Stories"

O XP tem por objetivo produzir software em menos tempo e com menor custo. Para alcançar esses objetivos ele especifica um pequeno conjunto de valores, princípios e práticas. Valores indicam o propósito, aquilo em que se acredita em alguma coisa e a razão pela qual se age de determinada forma. Princípios, por sua vez, procuram clarificar os valores dentro do contexto de desenvolvimento de software.

O SCRUM (RISING e JANOFF, 2001) é um processo para desenvolvimento de software ágil. Ele é baseado em ciclos de trinta dias chamados de “Sprints”, nos quais não é permitido realizar modificações nos requisitos do software. No SCRUM existem três papéis definidos: “Product Owner” que define e prioriza as funcionalidades do produto e aceita ou não o produto final; “Scrum Master” representa o gerente do projeto e é responsável pela aplicação do SCRUM, como realização de reuniões; “Team” representa uma equipe de desenvolvimento de 5 a 6 pessoas. A equipe é mista, incluindo programadores, realizadores de teste, etc.

O ciclo de desenvolvimento SCRUM é ilustrado na Figura 10. Todas as funcionalidades do software com níveis de prioridade são definidas em uma lista chamada de “Spring Product”, já as funcionalidades a serem implementadas em um determinado Sprint são mantidas em uma lista chamada de “Spring Backlog”. No início de cada Sprint, faz-se um “Sprint Planning Meeting”, isto é, uma reunião de planejamento na qual o “Product Owner” prioriza os itens do “Product Backlog” e a equipe seleciona as atividades que ela será capaz de implementar durante o “Sprint”. Durante o “Sprint”, cada equipe se reúne diariamente em uma reunião de aproximadamente 15 minutos, dirigida pelo gerente do processo, onde os desenvolvedores respondem as seguintes perguntas: “O que eu fiz ontem?”, “O que eu vou fazer hoje?”, “Quais os problemas que eu estou enfrentando?”. Ao final de um “Sprint”, a equipe apresenta as funcionalidades implementadas em uma reunião de aproximadamente 4 horas chamada de “Sprint Review Meeting”. Por fim, faz-se uma “Sprint Retrospective”, onde o “Scrum Master” e o “Scrum Team” analisam o que pode se continuar fazendo e que precisa ser melhorado no próximo “Sprint”. Depois dessa reunião, inicia-se um novo “Spring”.

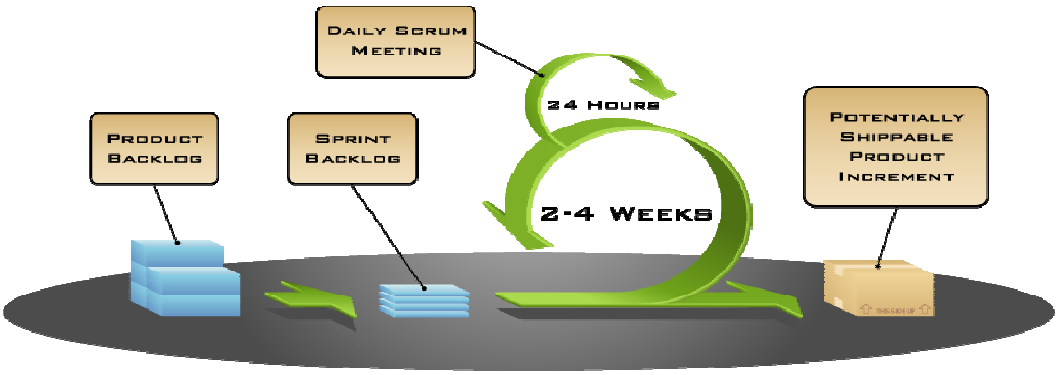


Figura 10 - Ciclo de Vida SCRUM. Fonte: (SOFTHOUSE, 2003)

2.2.3.2 Processos aplicados ao Desenvolvimento Orientado a Agentes

Diversos processos têm sido propostos para o desenvolvimento de sistemas multiagente. Alguns são específicos para uma determinada área de aplicação como o desenvolvimento de sistemas multiagente adaptativos. A grande maioria desses processos propostos é tradicional, ou seja, com um conjunto considerável de tarefas de desenvolvimento e de modelos gerados. No entanto, nos últimos anos têm sido apresentadas algumas propostas de processos ágeis. Algumas dessas propostas (WAUTELET et al., 2006) , (CHELLA et al., 2006) não são um processo para desenvolvimento de sistemas multiagente totalmente novo, mas um subconjunto de processos preexistentes com algumas adaptações próprias do desenvolvimento ágil, como diminuição da quantidade de modelos e de tarefas de desenvolvimento.

Processos Orientados a Agentes Tradicionais

Nesta subseção, apresenta-se duas das abordagens mais conhecidas e utilizadas no desenvolvimento de aplicações multiagente: o processo PASSI (COSSENTINO e POTTS, 2002) e a metodologia TROPOS (BRESCIANI et al., 2004). Aqui, é feita uma introdução a estas abordagens enfatizando suas principais características e, ao final da seção, um estudo comparativo entre elas e os outros processos e metodologias para desenvolvimento de software apresentados.

PASSI é um processo para especificação, projeto e implementação de sistemas multiagente que utiliza a linguagem AUML (BAUER et al., 2001), uma extensão da UML específica para o paradigma orientado a agentes.

No processo PASSI, um agente é uma entidade de software autônoma capaz de atingir um objetivo através de decisões autônomas, ações e relações sociais. Um agente pode ocupar alguns papéis funcionais durante suas interações com outros agentes para alcançar seus objetivos, onde um papel é uma coleção de tarefas executadas pelo agente perseguindo um sub-objetivo. Uma tarefa é definida como uma unidade intencional de um comportamento individual ou interativo (COSSENTINO e POTTS, 2002).

O processo PASSI é composto de cinco modelos: Modelo de Requisitos de Sistema, Modelo da Sociedade de Agentes, Modelo de Implementação de

Agentes, Modelo de Código e Modelo de Implantação, conforme ilustrado na Figura 11.

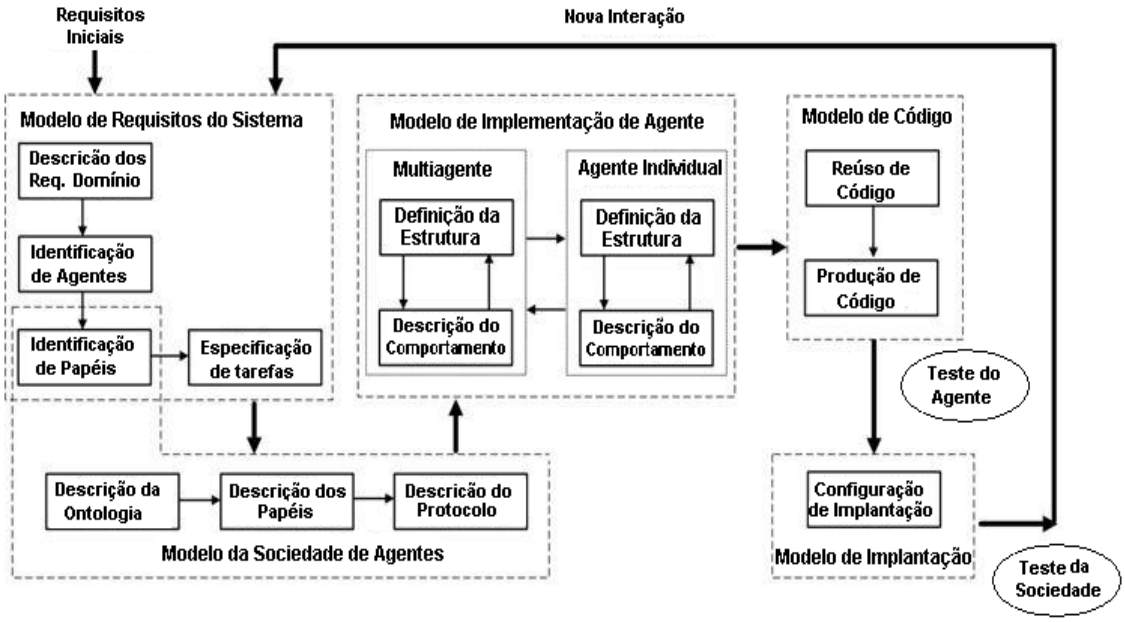


Figura 11 – A metodologia PASSI. Fonte: (COSSENTINO e POTTS, 2002)

O Modelo de Requisitos de Sistema é um modelo antropomórfico dos requisitos do sistema em termos de ação e propósito. O desenvolvimento desse modelo é realizado através dos seguintes passos:

- **Descrição do Domínio:** uma descrição funcional do sistema utilizando diagramas de caso de uso;
- **Identificação dos agentes:** associação de grupos de responsabilidades para agentes, representando-os como pacotes da UML;
- **Identificação de papéis:** uso de diagramas de seqüência para explorar cada responsabilidade do agente através de cenários específicos de papéis;
- **Especificação de tarefas:** Especificação através de diagramas de atividades das capacidades de cada agente.

O Modelo da Sociedade de Agentes é um modelo de interações sociais e dependências entre os agentes envolvidos na solução. O desenvolvimento deste modelo envolve os seguintes passos:

- **Identificação de papéis (descrito anteriormente);**
- **Descrição da ontologia do domínio:** uso de um diagrama de classes e restrições OCL para descrever o conhecimento relacionado a agentes individuais e as pragmáticas de suas interações;
- **Descrição dos papéis:** uso de diagramas de classes para mostrar papéis distintos executados pelos agentes, as tarefas de um papel, capacidade de comunicação e dependências entre agentes;
- **Descrição do protocolo:** uso de um diagrama de seqüência para especificar a gramática de cada pragmática do protocolo de comunicação em termos de performativas de atos de comunicação.

O Modelo de Implementação de Agentes é um modelo da solução arquitetural em termo de classes e métodos. O desenvolvimento deste modelo envolve os seguintes passos:

- **Definição da estrutura do agente:** uso de diagramas de classes para descrever a estrutura de solução em classes de agentes;
- **Descrição do comportamento do agente:** uso de diagramas de atividades ou de estados para descrever o comportamento de agentes individuais.

O Modelo de Código é um modelo da solução no nível de código. O desenvolvimento deste modelo envolve os seguintes passos:

- **Reúso de Código:** bibliotecas de diagramas de classes e de atividades associados com código fonte são reusadas;
- **Complementação de Código:** os esqueletos de código fonte anteriormente reusados são completados.

O Modelo de Implantação é um modelo da distribuição das partes do sistema através de unidades de processamento de hardware, e sua migração entre unidades de processamento. O desenvolvimento deste modelo envolve o seguinte passo:

- **Configuração de Implantação:** uso de diagramas de implantação para descrever a alocação de agentes para unidades de processamento disponíveis e quaisquer restrições sobre migração e mobilidade.

A atividade de teste é dividida em dois passos. No primeiro passo, teste do agente, cada agente é testado para verificar se seu comportamento cumpre os requisitos do sistema. No segundo passo, teste da sociedade, os agentes são testados em conjunto, para verificar se as interações se dão corretamente, ou seja, se contribuem na solução de problemas que necessitam de colaboração.

O ciclo de vida do processo PASSI é iterativo. As iterações se dão na elicitação de requisitos e na implementação. O primeiro tipo de iteração ocorre quando surgem novos requisitos e envolve dependência entre os modelos de análise de requisitos e da sociedade agentes e entre modelos de implementação (Figura 11). O segundo tipo de iteração ocorre no modelo de implementação (Figura 12). Neste modelo, as interações ocorrem entre a implementação de um agente individual e da sociedade multiagente. Na modelagem de um agente individual, um diagrama de classe para cada agente é criado. O comportamento de cada agente é especificado como um diagrama de seqüência ou colaboração. Na modelagem multiagente, a estrutura dos agentes é representada em um diagrama de classe em termo de tarefas e os comportamentos em um diagrama de atividades.

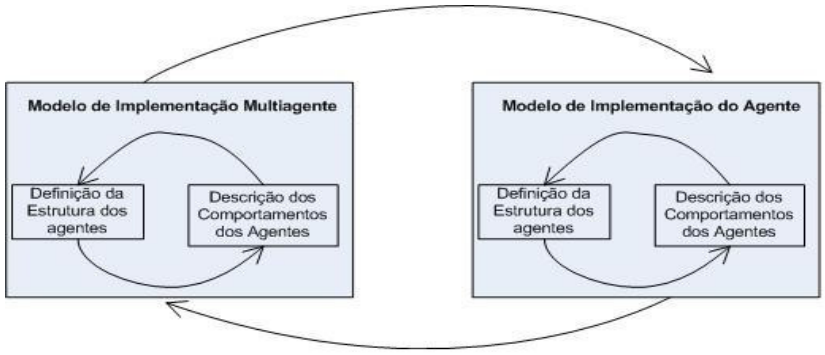


Figura 12 - Interações da Implementação dos agentes

As ferramentas PKT “PASSI Toolkit” e AgentFactory (CHELLA, 2004) apóiam as tarefas de desenvolvimento com o processo PASSI. A ferramenta PTK é um add-in para a ferramenta comercial de modelagem Rational Rose. Ela possibilita a geração parcial de alguns diagramas, permite checar a consistência de diagramas, gera documentação, acessa repositório de padrões, gera código na linguagem Java de acordo com o framework JADE (BELLIFEMINE et al., 2003) e permite engenharia reversa. A ferramenta AgentFactory (CHELLA, 2004) (Figura 13) suporta as fases de projeto e programação da metodologia PASSI. Essa ferramenta gerencia um repositório de agentes padrões que podem ser reusados e integrados para construir um sistema multiagente. Agente Factory também suporta a geração de código fonte conforme a plataforma de desenvolvimento de agentes JADE e com as especificações da FIPA (FIPA, 2008).

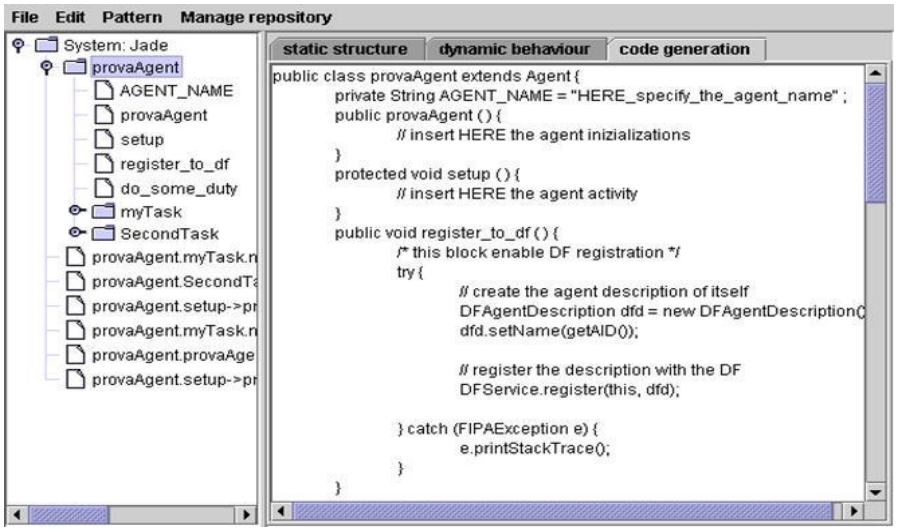


Figura 13 – Ferramenta Agent Factory

TROPOS é uma metodologia para desenvolvimento de sistemas multiagente. Ela utiliza conceitos de ator (agente, papel ou posição), objetivo e dependência entre atores como conceitos de modelagem de uma aplicação multiagente. Esses conceitos derivam do *i* framework* (CERNUZZI et al., 2005) (CERNUZZI e ZAMBONELLI, 2008). Esse framework fornece uma representação gráfica para os conceitos de modelagem de requisitos como atores, suas intenções (objetivos), as dependências entre eles, suas tarefas e responsabilidades. Alguns exemplos de elementos de modelagem presentes no framework i* estão ilustrados na Figura 14.

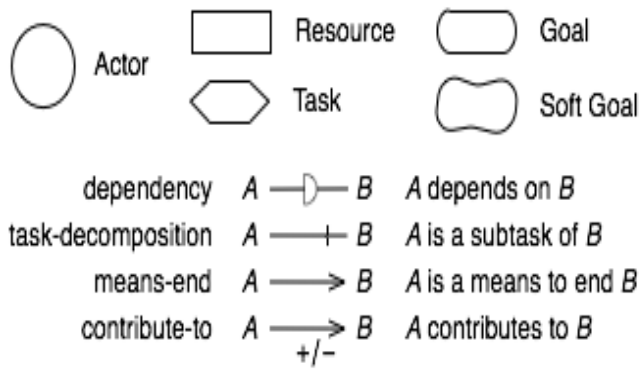


Figura 14 - Conceitos do Framework i*. Fonte: (BRESCIANI et al., 2004)

A TROPOS está organizada nas fases de Análise de Requisitos Iniciais, Análise de Requisitos Finais, Projeto Arquitetural, Projeto Detalhado e Implementação. A modelagem de requisitos é dividida em Análise de Requisitos Iniciais e Análise de Requisitos Finais. O objetivo da fase de Análise dos Requisitos Iniciais é entender o problema estudando o ambiente organizacional no qual a aplicação será implantada. Nessa fase, os atores do sistema são definidos, onde cada ator tem intenções de atingir objetivos, realizar tarefas e fornecer recursos. Na fase de Análise dos Requisitos Finais são identificados novos atores e também é definido as dependências entre os atores e quais atores são externos e internos (agentes). As dependências entre atores são de três tipos: “depender” é o ator que solicita auxilio de outro ator para realizar um plano ou para entregar um recurso; “dependee” é o ator que fornece auxílio ao ator “depender”; “dependum” é o recurso ou plano que liga os atores dependee e dependum.

Na fase de Projeto Arquitetural, a arquitetura global do sistema é definida em termo de subsistemas (atores), interconectados através de fluxos de controle (dependências). No Projeto Detalhado, o comportamento de cada ator e as interações entre eles são definidas. Na fase de Implementação é realizado um mapeamento das especificações de projeto detalhado para uma plataforma de implementação geralmente são utilizadas a plataforma JACK (BUSETTA et al., 1998) ou JADE.

A TAOM4E (*Tool for Agent Oriented Modeling*) é uma ferramenta para modelagem de agentes seguindo as diretrizes da metodologia TROPOS. Ela é integrada ao projeto Elipse através do sistema de plugins. TAOM4E é baseada nas recomendações da MDA (*Model Driven Architecture*) (MELLOR et al., 2004). A

arquitetura dessa ferramenta é flexível, permitindo a sua integração com outras ferramentas.

Processos Orientados a Agentes Ágeis

Agile PASSI é um processo para desenvolvimento de sistemas multiagente seguindo a filosofia de desenvolvimento ágil, inspirada mais especificamente na metodologia “eXtreme Programming”. Esse processo foi definido a partir de um subconjunto do processo PASSI.

As principais diferenças entre os dois processos é a quantidade de atividades realizadas e, conseqüentemente, de documentação produzida. Além dessas outro aspecto é o ciclo de vida que no processo PASSI original era somente iterativo, já o ciclo de vida de Agile PASSI é iterativo e incremental. As iterações e incrementos são realizados em função dos casos de uso definidos na fase de Requisitos. A cada iteração um conjunto de funcionalidades do sistema é desenvolvido. Conforme ilustrado na Figura 15 o processo Agile PASSI é dividido em quatro fases:

- **Requisitos**: Essa fase é composta das atividades de planejamento e descrição dos requisitos do domínio. No planejamento os requisitos são levantados textualmente utilizando “user stories”. Na descrição dos requisitos são utilizados diagramas de caso de uso da UML para representar as funcionalidades do sistema;
- **Sociedade de Agente**: Essa fase é divida em quatro atividades que produzem a descrição da ontologia do domínio e as interações dos agentes. Na descrição da ontologia do domínio os diagramas de casos de uso identificados na fase anterior são agrupados em pacotes estereotipados com o valor atribuído “agent” que representam os agentes da sociedade. As entidades do domínio da aplicação são representadas em um diagrama de classes da UML usando os relacionamentos de generalização, associação e agregação;
- **Codificação:** Essa fase é divida nas atividades de reúso de padrões e codificação. Na primeira atividade, tenta-se reusar padrões de projeto de agentes para obter partes de código fonte. Isso é realizado utilizando a ferramenta AgentFactory. Na atividade codificação, os programadores

complementam o código reusado na atividade anterior seguindo as práticas de desenvolvimento ágeis como, por exemplo, a programação em pares;

- **Teste**: A fase de teste é divida em duas partes, o plano de teste e o teste propriamente dito. Seguindo as regras da “eXtreme Programming” a atividade de plano de teste inicia-se antes da codificação. Depois da codificação é realizada a atividade de testes de acordo com o plano de teste realizado anteriormente.

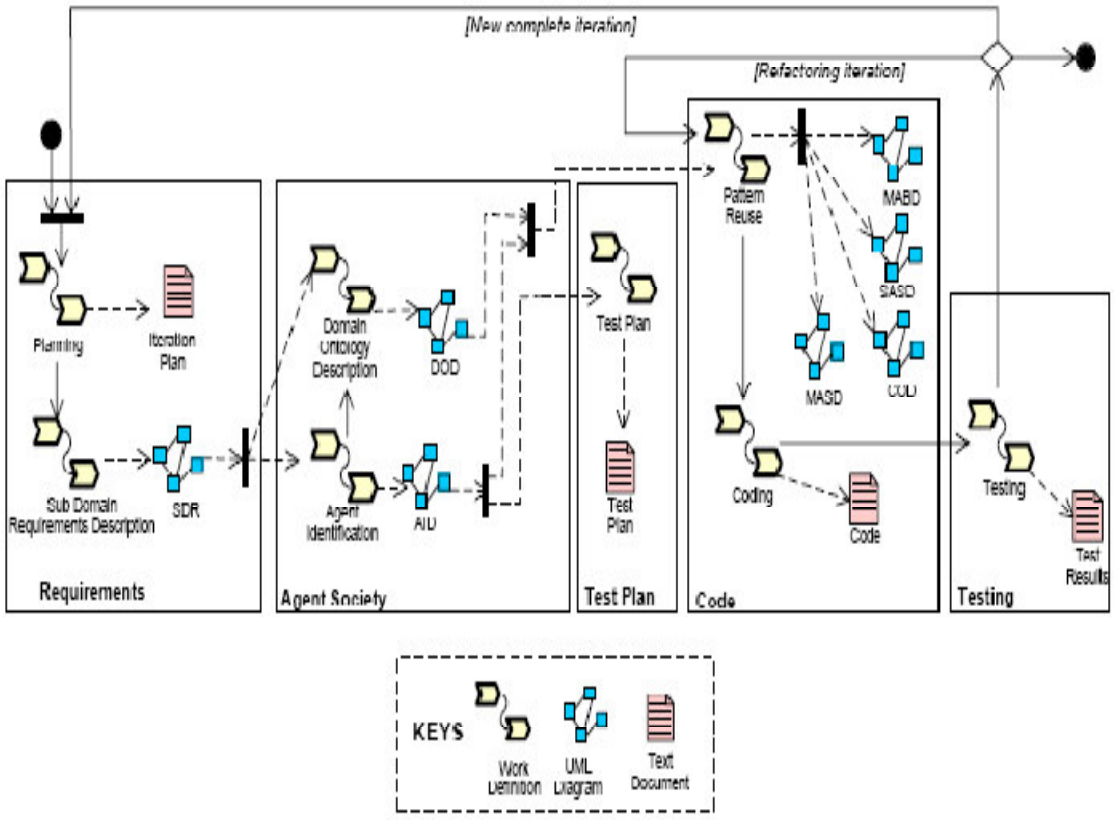


Figura 15 - O ciclo de vida do processo Agile PASSI. Fonte: (CHELLA et al., 2006)

Na Tabela 1 é descrito um comparativo entre o processo PASSI tradicional e sua versão ágil. Na tabela, pode-se observar que a quantidade de atividades (“Activities”) e produtos (“Work Products”) desenvolvidos diminuiu significativamente na versão ágil do processo. Para cada produto da tabela foi especificado se ele é realizado manualmente (M) pelo desenvolvedor ou se é parcialmente realizado por ferramentas (P) ou completamente automatizado (F).

Tabela 1 - Tabela comparativa entre a PASSI tradicional com a sua versão ágil. Fonte: (CHELLA et al., 2006)

PASSI			Agile PASSI		
Phases	**Activities**	**Work Products**	**Work Products**	**Activities**	**Phases**
System Requirement			M,1	Planning	**Requirements**
	DRD	M,1	M,1	SDRD	
	AID	M,1	M,1	AID	
	RID	M,1,scenarios	---	---	
	TSP	M,1,agents	---	---	
Agent Society	DOD	M,1	M,1	DOD	**Agent Society**
	COD	P,1	---	---	
	RD	F,1	---	---	
	PD	M,0..n	---	---	
Agent Implementation	MASD	F,1	---	---	---
	MABD	P,1	---	---	
	SASD	P,1..agents	---	---	
	SABD	M,1,agents	---	---	
Code	CR	P,1	COD (F,1) MASD (F,1) MADD (T,1) SASD (F,1..agents)	Pattern Reuse	**Code**
	CP	F,1	Code Reuse (F,1) Code (M,1)	Coding	
Deployment	DC	M,1	---	---	
Testing	Agent Test	M,1..n	Test Plan (M,1..n)	Agent Test	**Test**
	Society Test	M,1	Test Plan (M,1)	Society Test	

SAADAM (*Software Agent Development An Agile Methodology*) (CLYNCH e COLLIER, 2007) é uma metodologia para desenvolvimento de sistemas multiagente inspirada em metodologias ágeis de desenvolvimento como XP e SCRUM. Ela utiliza a linguagem AUML para realizar a modelagem do sistema.

A idéia básica da metodologia SAADAM é o enfoque na entrega de código executável ao cliente ao invés de uma extensa e detalhada documentação. Outra característica é que o código fonte é bem-testado e aperfeiçoado ao longo do processo de desenvolvimento.

O ciclo de vida de SAADAM é ilustrado na Figura 16. Além da especificação de requisitos, o processo SAADAM consiste de quatro fases: Projeto, Implementação Dirigida por Testes, Liberação e Revisão, e Melhoria.

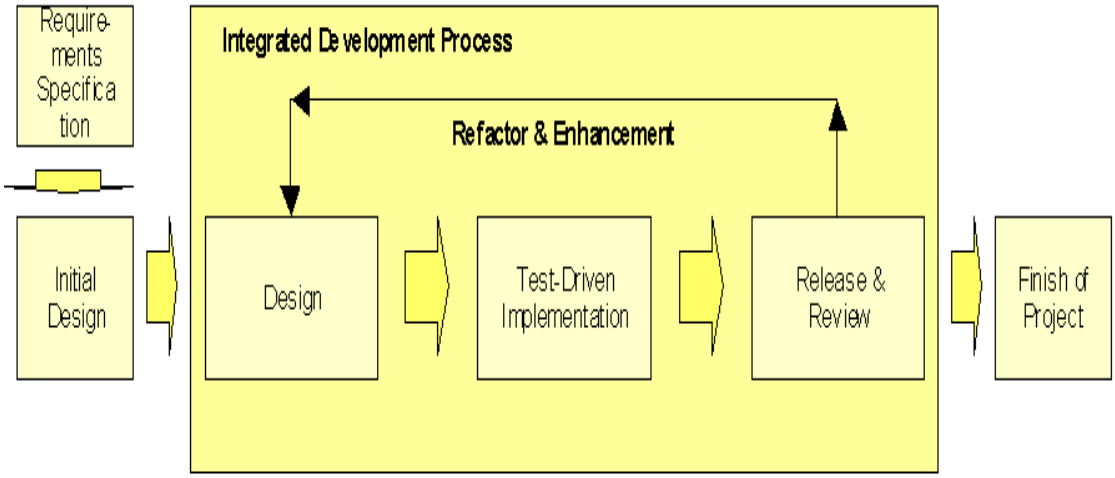


Figura 16 - Ciclo de Vida do SAADAM. Fonte: (CLYNCH e COLLIER, 2007)

A captura de requisitos é realizada rapidamente através de técnicas de "user stories" e de casos de uso. A seguir, é realizada a fase de Projeto que é realizada de forma iterativa e incremental. Nessa fase os requisitos são mapeados para decisões de projeto. São identificados os comportamentos do sistema e os papeis que serão executados por agentes. São definidas também as relações, interações e atividades dos agentes.

Na fase de Implementação Dirigida por Testes são utilizados os conceitos de "Test Agents" (TA) e a "Application Agents Under Test" (AAUT). O agente de teste (TA) é um agente que implementa um conjunto de casos de testes que devem ser satisfeitos pelo agente que estiver tendo o comportamento testado (AAUT). Cada

caso de teste representa um conjunto de testes que são realizados no agente (AAUT) para avaliar se ele satisfaz um dado requisito.

Na fase de Liberação e Revisão a parte do software desenvolvida no ciclo atual é entregue para o cliente aprovar. Durante essa fase o cliente pode testar e revisar cada parte do software para determinar se ela atende aos seus requisitos.

Na fase de Melhoria são aplicadas mudanças com intuito de mudar a estrutura interna do código fonte com intuito de torná-lo mais eficiente e mais fácil de compreender, sem alterar seu comportamento. Nesse processo o código duplicado é removido e o código com lógica muito complexa é simplificado. Essa prática de ir melhorando o código fonte ajuda a descobrir erros no código, prevenindo a inserção de defeitos no produto de software.

2.2.4 Estudo Comparativo entre Processos para Desenvolvimento de Software

A partir do estudo dos processos/metodologias RUP, OPEN SCRUM, XP do paradigma orientado a objetos e PASSI, TROPOS, SAADAM e AGILE PASSI abordados na seção anterior apresentaremos aqui um comparativo entre eles. Para realizar um comparativo é fundamental a escolha de critérios de comparação relevantes e que sirvam para evidenciar as características pelas quais os processos se diferenciam e as que possuem em comum. Esses critérios e correspondente avaliação foram definidos a partir dos trabalhos desenvolvidos em (HENDERSON-SELLERS, 2005) e (STURM e SHEHORY, 2003). O comparativo é apresentado nas Tabela 2 e Tabela 3.

A Tabela 2 apresenta as características dos processos referentes ao critério processo. Esse critério se refere ao conjunto de ações que são realizadas para produzir um produto de software. As características analisadas para esse critério são o tipo de ciclo de vida utilizado (cascata, iterativo e incremental, espiral, etc); as fases que o ciclo de vida do processo cobre (captura de requisitos, projeto, etc); esforço de utilização, essa característica se refere ao esforço intelectual para utilizar o processo; suporte de ferramentas que auxiliem a realização das fases (bom, médio e pouco) e a quantidade de documentação disponível (bom, médio, insatisfatório).

Tabela 2 – Comparativo entre processos para desenvolvimento de software relativo ao critério *processo*

Processo	Ciclo de Vida	Fases do Ciclo de Vida	Esforço de utilização	Suporte de Ferramenta	Documentação
RUP (Tradicional)	Iterativo e Incremental	Completo (Análise, Projeto, Implementação, Testes, Implantação)	Alto	Bom	Bom
OPEN (Tradicional)	Iterativo e Incremental	Média (Análise, Projeto, Implementação, Implantação)	Médio	Bom	Médio
XP (Ágil)	Iterativo e Incremental	Reduzido (Captura de requisitos rápida, projeto rápido, implementação e testes)	Baixo	Médio	Bom
SCRUM (Ágil)	Iterativo e Incremental	Reduzido (Captura de requisitos e projeto rápidos, implementação e testes)	Baixo	Médio	Bom
PASSI (Tradicional)	Iterativo	Completo (Análise, Projeto, Implementação, Implantação e Testes)	Alto	Médio	Médio
TROPOS (Tradicional)	Iterativo e Incremental	Média (Analise, Projeto e Implementação)	Alto	Médio	Médio
SAADAM (Ágil)	Iterativo e Incremental	Reduzida (Captura de Requisitos e Projeto rápidos, implementação e testes)	Médio	Pouco	Insatisfatório
AGILE PASSI (Ágil)	Iterativo e Incremental	Reduzida (Análise, Projeto, Implementação e Testes)	Médio	Médio	Insatisfatório

Na tabela, pode-se observar que todos os processos analisados, com exceção da PASSI tradicional, utilizam o ciclo de vida iterativo e incremental. Quanto às fases do ciclo de vida cobertos pelos processos todos cobrem as fases básicas de análise, projeto e implementação. As fases de testes e implantação estão presentes apenas em alguns processos como o RUP e o PASSI. Quanto ao esforço de utilização os processos tradicionais em geral são bem mais complexos do que os ágeis. E também os processos orientados a agentes são, geralmente, mais complexos do que os processos orientados a objetos.

Quanto ao suporte de ferramentas vemos que os processos orientados a objetos, até porque já são utilizados a mais tempo, possuem ferramentas de suporte com mais qualidade e em maior número. A documentação dos processos orientados a objetos é bem mais disponível e do que os processos orientados a agentes, principalmente os ágeis.

Para realizar a modelagem de um sistema os processos geralmente utilizam uma linguagem que pode ter notação gráfica ou não. Notações são símbolos usados para representar elementos em um sistema. A técnica de modelagem estabelece um conjunto de modelos que descrevem um sistema em diferentes níveis de abstração e diferentes aspectos do sistema (STURM e SHEHORY, 2003).

Na Tabela 3 são descritas, respectivamente, as propriedades nome da *linguagem de modelagem*, *acessibilidade*, *gerenciamento da complexidade*, *expressividade* e *precisão* das linguagens para modelagem adotadas pelos processos analisados na seção anterior.

A propriedade *acessibilidade* refere-se a facilidade de entendimento da linguagem para o usuário do processo. A propriedade *gerenciamento da complexidade* diz respeito ao suporte a representação de diferentes níveis de abstração pela linguagem de modelagem. A propriedade *expressividade* se refere à possibilidade de representar vários aspectos do software, como a visão estrutural e dinâmica e também de vários domínios, como sistemas distribuídos e móveis. A propriedade *precisão* se refere aos modelos criados com a linguagem possibilitarem erros de interpretação, ou seja, não possuírem ambigüidades.

Tabela 3 - Comparativo entre Processos para Desenvolvimento de Software relativo ao critério *linguagem de modelagem*

Processo	Linguagem de Modelagem	Acessibilidade	Gerenciamento da Complexidade	Precisão
RUP (Tradicional)	UML	Médio (Quantidade considerável de conceitos e diagramas a serem aprendidos)	Permite representar vários níveis de abstração	Semi-formal
OPEN (Tradicional)	UML	Médio (Quantidade considerável de conceitos e diagramas a serem aprendidos)	Permite representar vários níveis de abstração	Semi-formal
XP (Ágil)	UML (Somente alguns poucos diagramas)	Fácil (Número reduzido de diagramas é utilizado)	Permite representar vários níveis de abstração	Semi-formal
SCRUM (Ágil)	ScrUML	Fácil (Número de conceitos e diagramas bastante limitados e específicos ao processo SCRUM)	Não possui	Semi-formal
PASSI (Tradicional)	AUML	Com um conhecimento prévio de UML, os modelos podem ser facilmente compreendidos.	Permite representar vários níveis de abstração	Semi-formal
TROPOS (Tradicional)	Notação i*	Provê uma sintaxe abstrata para a construção de diagramas, cuja linguagem é baseada num pequeno conjunto de conceitos, sendo providas técnicas e ferramentas para facilitar seu uso.	Permite representar o software sob quatro perspectivas: social, intencional, orientação a processos e a objetos.	Semi-formal
AGILE PASSI (Ágil)	AUML	Com um conhecimento prévio de UML, os modelos podem ser facilmente compreendidos.	Permite representar vários níveis de abstração	Semi-formal
SAADAM (Ágil)	AUML	Com um conhecimento prévio de UML, os modelos podem ser facilmente compreendidos.	Permite representar vários níveis de abstração	Semi-formal

2.3 O desenvolvimento de software baseado no reúso

Reúso de software é o processo de criação de sistemas de software a partir de software existente, ao invés de desenvolvê-los a partir do zero (KRUEGER, 1992).

Reúso de software é o uso de componentes de software existente para construir novos sistemas. O reúso se aplica não somente a fragmentos de código, mas também para todos os produtos de trabalho intermediários gerado durante o desenvolvimento de software, incluindo documentos de requisitos, especificações de sistema, estruturas de projeto, e qualquer outra informação da qual o desenvolvedor necessita para desenvolver o software (PRIETO-DÍAZ, 1993).

O reúso informal de artefatos como código-fonte é muito comum no desenvolvimento de software. No entanto, na maioria das vezes, esses artefatos são reusados somente por quem os criou ou por poucas outras pessoas, são mal-documentados e não são testados adequadamente. Com isso, esse tipo de reúso pode introduzir erros no software construído e dificultar consideravelmente a sua manutenção. Segundo (PRIETO-DÍAZ, 1993) o problema que enfrentamos na engenharia de software não é uma falta reutilização, mas a falta de reutilização freqüente e sistemática.

O reúso sistemático de software, isto é, a incorporação de artefatos de software especialmente projetados para serem reusados ao processo de desenvolvimento, traz diversas vantagens. A mais evidente é a produtividade, uma vez que parte do esforço de desenvolvimento é eliminado através do reúso de software preexistente, diminuindo com isso o custo e prazo de entrega do software. O software também se torna mais fácil de manter e com maior qualidade, uma vez que artefatos destinados ao reúso são utilizados por diversas aplicações, são bem-documentados e testados antes de serem disponibilizados.

(PRIETO-DÍAZ, 1993) classifica o reúso de software nas seguintes perspectivas:

- **Conteúdo:** define o que será reusado. Ex.: algoritmos, procedimentos, conhecimento, componentes;
- **Escopo:** define a forma e a extensão do reúso e é dividida em horizontal e vertical. O reúso horizontal se dá através da construção de partes genéricas de software comuns a diversas aplicações, já o reúso vertical se dá no mesmo domínio de aplicação. O reúso horizontal abrange uma quantidade maior de aplicações uma vez que ele é mais genérico, mas diminui a quantidade de funcionalidades reusadas, porque aplicações de domínios diferentes, normalmente, não

compartilham muitas características. Já no reúso vertical ocorre o contrário, como as aplicações são do mesmo domínio, geralmente, elas compartilham uma quantidade grande de funcionalidades, mas o reúso fica restrito as aplicações daquele domínio;

- **Modo:** define como o reúso é conduzido, se de forma sistemática ou informal. Na forma sistemática de reúso, este é incorporado no processo de desenvolvimento de software. Sendo necessário um investimento inicial na construção dos artefatos para serem reusados e no treinamento da equipe de desenvolvimento. Já o reúso informal se dá sem procedimento, é também chamado de reúso oportunista, onde o desenvolvedor reusa partes de software a partir de suas experiências anteriores ou de indicações de outros desenvolvedores. Geralmente, esse tipo de reúso envolve um grande esforço de adaptação;
- **Técnica:** define qual a técnica será utilizada para implementar o reúso, podendo ser composicional ou gerativa. Na técnica composicional, componentes de software são selecionados e integrados para construir uma aplicação. Na técnica gerativa são utilizados software chamados de geradores de aplicação, onde o usuário específica as funcionalidades do sistema e o software gera a aplicação;
- **Intenção:** define como os elementos serão reusados. Ex.: caixa-preta (“Black-box”), caixa-branca (“White-box”). No reúso em caixa-preta o software é reusado sem nenhuma modificação como ocorre, por exemplo, na instanciação de um objeto. No reúso em caixa-branca o software é reusado com a realização de adaptações como ocorre, por exemplo, no reúso através de herança;
- **Produto:** define quais produtos serão reusados. Exemplo: código-fonte, projeto, especificação, objetos, texto, arquiteturas.

Nas próximas subseções são detalhadas as técnicas de reúso composicional e gerativa, como se dá o processo de reúso em linhas de produto através da Engenharia de Domínio e de Aplicações e, por fim, a relação entre o reúso de software e ontologias.

2.3.1 Abordagens de Reúso

As abordagens de reúso composicional e gerativa permitem reusar software preexistente na construção de um novo sistema, a diferença é que na primeira os componentes de software devem ser selecionados, adaptados e integrados de forma manual pelo desenvolvedor, enquanto que na abordagem gerativa são feitas somente especificações em alto-nível dos requisitos do sistema e o código executável é gerado automaticamente através de software chamados de geradores de aplicação.

2.3.1.1 Reúso Composicional

O reúso composicional consiste na técnica de construção de sistemas a partir da montagem de componentes de software pré-existentes. O objetivo é realizar um processo de desenvolvimento semelhante ao que acontece com o desenvolvimento de hardware. Componentes de software são como peças no equipamento de hardware que se integram através de uma interface comum. O reúso composicional pode ser realizado tanto horizontalmente quanto verticalmente, isto é, os componentes podem ser pertencentes a um mesmo domínio ou genéricos a diversos domínios de aplicação.

Existem diversas definições para o conceito de componente de software, algumas dessas são apresentadas abaixo:

- Um componente é um módulo logicamente coerente e fracamente acoplado que denota uma abstração única (BOOCH, 1987);
- Um componente de software é um empacotamento físico de software executável com uma interface pública bem definida (HOPKINS, 2000);
- Um componente é um pacote de software que oferece serviços através de interfaces (HERZUM e OLIVER, 1999);
- Um componente é um pacote coerente de artefatos de software que podem ser desenvolvidos independentemente e entregues como unidades que podem ser compostas, sem mudanças, com outros componentes para construir um sistema maior (D'SOUZA e WILSS, 1998).

Como visto nas definições anteriores, existem diversas conceitualizações para componente de software. Basicamente, as idéias se dividem em duas vertentes, a primeira são dos autores que consideram um componente como código executável e a segunda dos que consideram que um componente pode ser qualquer artefato envolvido no desenvolvimento de software.

Algumas das características dos componentes que são de consenso entre os autores são:

- Componentes são reutilizáveis;
- Componentes são acessíveis através de interfaces;
- Componentes podem ser genéricos ou pertencer a mais de um domínio de aplicação;
- Componentes se integram com outros componentes para construção de uma aplicação.

Um componente pode ser integrado com determinados tipos de outros componentes, isto é especificado através de um modelo de componentes. Segundo (KOTONYA et al., 2003), um modelo de componentes define as interações e composições especificas para componentes. Um componente pode ter uma dependência de contexto com um sistema operacional ou com alguns outros componentes. Um padrão de interação especifica o tipo de dependências de contexto que o componente pode ter. Padrões de interface são requisitos mandatórios empregados para habilitar componentes de software a diretamente interagir com outros componentes de software, ou seja, os componentes precisam ter uma interface comum para poder ser integrados.

O desenvolvimento baseado em componentes (CBD – Component-based development) é dividido em dois processos. O primeiro se refere à construção, documentação e armazenamento dos componentes destinados ao reúso em um repositório. O segundo envolve a seleção e adaptação dos componentes para compor uma aplicação.

(CRNKOVIC, 2003) define um conjunto básico de etapas para o processo de desenvolvimento de aplicações reusando componentes, ilustrado na Figura 17:

- *Seleção de componentes que podem ser usados no sistema.* Nessa etapa é realizada uma busca no repositório por componentes de software que satisfaçam os requisitos da aplicação;

- *Seleção dos componentes que se adéquam aos requisitos do sistema.* Nessa etapa, dos componentes retornados na etapa anterior são selecionados os que satisfazem os requisitos da aplicação;
- *Alternativamente, criar um componente para ser usado no sistema.* Caso a busca não retorne componentes adequados a aplicação que se quer desenvolver são criados componentes próprios;
- *Adaptar os componentes selecionados para que eles atendam as especificações de requisitos ou o modelo de componentes.* Alguns dos componentes selecionados não podem ser integrados diretamente devendo então ser adaptado. Essa adaptação pode ser realizada, por exemplo, através de parametrização;
- *Compor e distribuir os componentes usando um framework para componentes.* Nessa etapa os componentes são integrados para compor a aplicação e está é colocada em funcionamento;
- *Substituição de componentes por versões mais novas.* Essa é uma etapa de manutenção, onde componentes reusados são substituídos por outros com intuito de corrigir erros ou melhorar as funcionalidades.

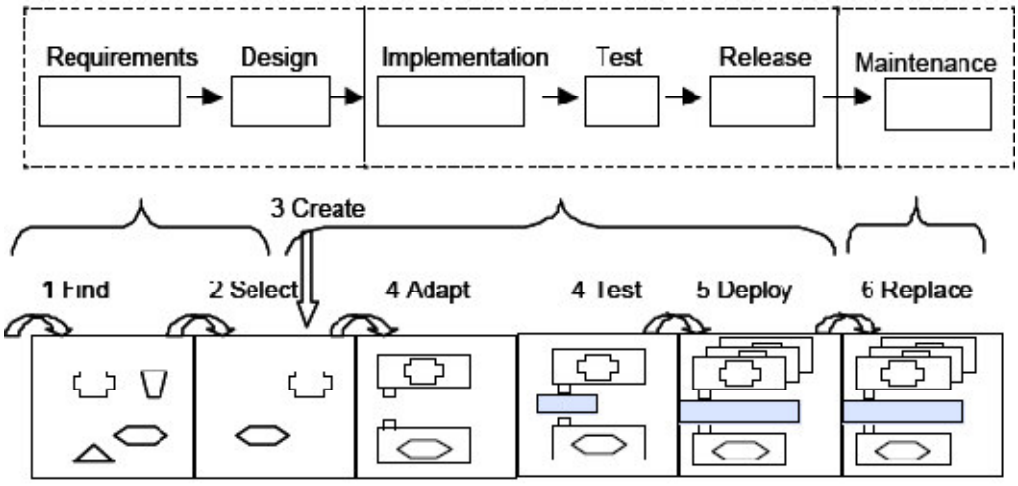


Figura 17 - O ciclo de desenvolvimento baseado em componentes (CRNKOVIC, 2003).

2.3.1.2 Reúso Gerativo

O reúso gerativo é realizado embutindo-se conhecimento do domínio e conhecimento relevante de construção de software dentro de um gerador de

aplicação específico a um domínio. Novos sistemas no domínio são criados através de especificações em uma linguagem especifica de domínio. O processo de geração pode ser completamente automatizado ou pode requerer intervenção manual. (FRAKES e KANG, 2005).

Uma linguagem específica de domínio (DSL – Domain Specific Language) é uma linguagem especifica para um domínio particular de aplicação. As características de uma DSL eficaz é a habilidade de desenvolver aplicações completas para um domínio rapidamente e efetivamente. Exemplos de linguagem de especificação são o LaTex para preparação de documentos e HTML para marcação de documentos (HUDAK, 1998).

O objetivo principal dos geradores é produzir sistemas de software a partir especificações de alto-nível (CZARNECKI, 1998). Assim, a partir de especificações em alto-nível o gerador de aplicações seleciona os componentes e os integra automaticamente e gera a aplicação conforme essas especificações.

A partir dessas definições, pode-se perceber que o reúso gerativo diminui drasticamente o esforço de desenvolvimento através da utilização de linguagens específicas de domínio e geradores de aplicações, onde são eliminadas as tarefas de seleção, adaptação e integração de componentes. No entanto, é uma abordagem de reúso com maior esforço de implantação, pois exige um custo inicial maior do que o desenvolvimento baseado em componentes, envolvendo, além da criação de componentes, a criação de linguagens específicas de domínio e de geradores de aplicação.

2.3.2 Engenharia de Linhas de Produto de Software

Uma linha de produto de software, também chamadas de famílias de aplicações, é um conjunto de sistemas de software que têm um determinado conjunto de funcionalidades em comum, e que satisfazem as necessidades de um determinado segmento de mercado ou missão, e que são desenvolvidos tendo a mesma base (CLEMENTS e NORTHRO, 2001). O paradigma da engenharia de linhas de produto de software é divido em dois processos: a Engenharia de Domínio e a Engenharia de Aplicações (POHL et al., 2005). A Figura 18 ilustra esse processo.

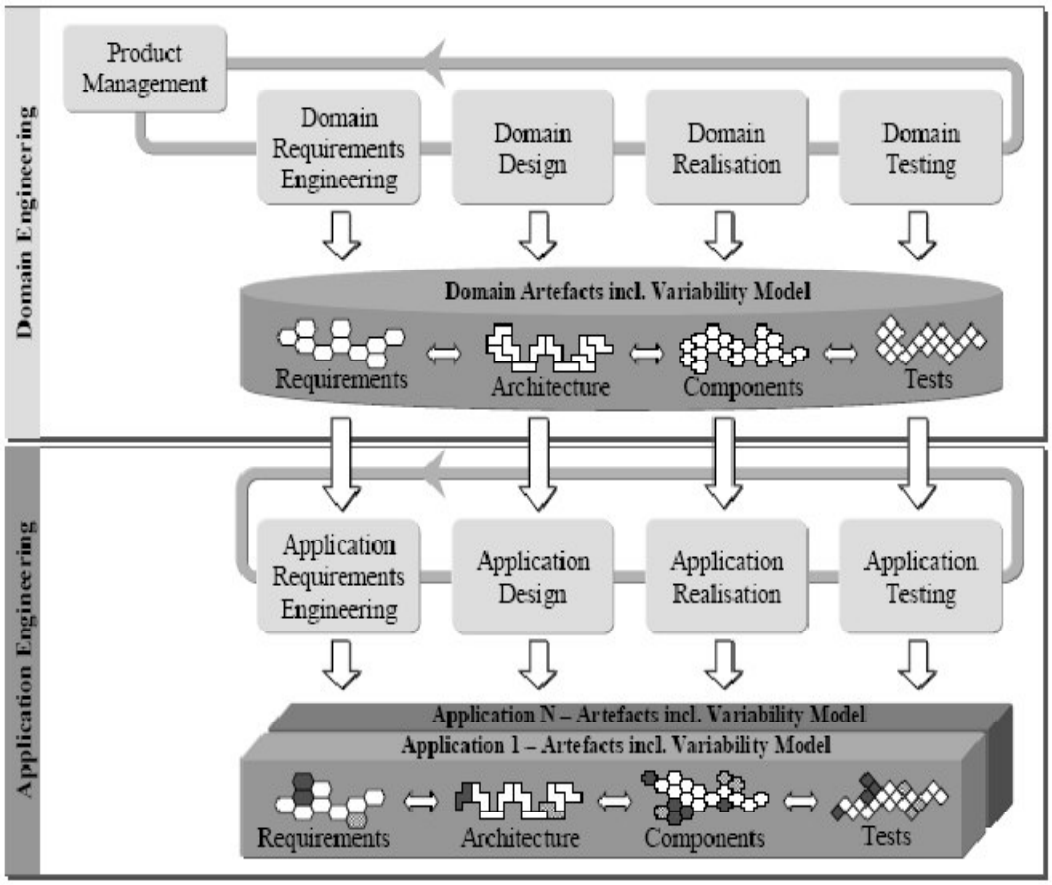


Figura 18 - Os processos da Engenharia de Linhas de Produtos. Fonte: (POHL et al., 2005).

Nas linhas de produto de software, a definição das características comuns e variáveis das aplicações da família é fundamental. Na Engenharia de Domínio são construídos artefatos pertencentes a uma família de aplicações. Nesse processo é definida a parte comum e a variável do artefato. A parte fixa é igual para todas as aplicações e a parte variável será diferente em algumas aplicações da família. Na Engenharia de Aplicações a parte fixa dos artefatos construídos no processo da Engenharia de Domínio é reusada para compor uma aplicação específica. A parte variável é selecionada conforme os requisitos da aplicação em desenvolvimento.

A variabilidade do software é a capacidade de um sistema de software ou artefato de ser mudado, customizado ou configurado para uso em um determinado contexto. Um alto grau de variabilidade nos permite o uso de um software em um número maior de contextos, isto é, o software é mais reutilizável (BOSCH, 2000).

Variabilidade é um conceito fundamental nas linhas de produto de software, pois a sua definição e gerenciamento distinguem uma linha de produção de software de um sistema individual (POHL et al., 2005).

Nas linhas de produto de software, um repositório ou biblioteca de componentes é o elo entre a Engenharia de Domínio e de Aplicações.

Durante o processo da Engenharia de Domínio, os artefatos são armazenados no repositório, devendo sofrer algum tipo de classificação e ser bem-documentado para facilitar a seleção. Na Engenharia de Aplicações, os artefatos armazenados no repositório são selecionados, adaptados e integrados para compor uma nova aplicação.

Uma abstração de software é uma especificação de alto nível, natural e sucinta que se corresponde com uma ou várias realizações num nível de representação mais detalhado. A especificação descreve o que o artefato de software faz, eliminando os detalhes irrelevantes e enfatizando a informação que, nesse nível, é importante para o usuário da abstração (GIRARDI, 1996).

2.3.2.1 A Engenharia de Domínio

A Engenharia de Domínio, também chamada de desenvolvimento *para* o reúso, é uma abordagem sistemática para a construção de abstrações de software reutilizáveis (ARANGO, 1988).

Desenvolver software *para* o reúso significa que a intenção do desenvolvedor não é a construção de uma aplicação específica, mas de um conjunto de artefatos que são comuns a uma família de aplicações.

A Engenharia de Domínio inclui três fases básicas: Análise de Domínio, Projeto de Domínio e Implementação de Domínio.

Na Análise de Domínio, são definidos os requisitos comuns e variáveis pertencentes à família de aplicações. Esses requisitos são representados em um modelo de domínio.

Um modelo de domínio é uma representação explícita das propriedades comuns e variáveis do sistema em um domínio e das dependências entre as propriedades variáveis (CZARNECKI, 1998).

No Projeto de Domínio, a arquitetura comum é estabelecida para a família de aplicações de acordo com o modelo de domínio. O projeto possui duas partes principais, o projeto global e detalhado. No projeto global é definida a arquitetura global da família de aplicações, isto é, como os componentes estão estruturados. No projeto detalhado, é definida a estrutura de cada componente da arquitetura global.

Nessa fase, também é feita a especificação das características comuns e variáveis da família de aplicações de cada componente da arquitetura. O produto dessa fase é um modelo arquitetural contendo arquitetura global, detalhada e a especificação de características comuns e variáveis da família de aplicações. Esse modelo é usado para guiar a implementação do domínio.

Na Implementação de Domínio os artefatos são implementados de acordo com o projeto global e detalhado e com a especificação de variabilidades definidos no Projeto de Domínio. Esses artefatos podem ser, por exemplo, componentes reusáveis, linguagens especificas de domínio e geradores.

2.3.2.2 A Engenharia de Aplicações

A Engenharia de Aplicações, também chamada de desenvolvimento *com* reúso, é o processo de desenvolvimento de aplicações de software individuais a partir de artefatos de software criados pelo processo da Engenharia de Domínio. Na falta de artefatos para serem reusados estes podem ser desenvolvidos especificamente para a aplicação.

A Engenharia de Aplicações possui três fases básicas: Engenharia dos Requisitos da Aplicação, Projeto da Aplicação e Implementação da Aplicação.

Na Engenharia da Aplicação, os requisitos de uma aplicação específica da família são definidos através da seleção dos requisitos comuns e variáveis a uma família de aplicação definidos no modelo de domínio da fase de Análise de Domínio.

No Projeto da Aplicação, a arquitetura da família de aplicações produzida na fase de Projeto de Domínio é reusada e são realizadas as adaptações necessárias para a aplicação que se estiver desenvolvendo.

Na Implementação da Aplicação, os componentes de software produzidos na fase de Implementação do Domínio são reusados para compor a aplicação de acordo com as características comuns e variáveis. Os componentes reusados podem precisar sofrer algum tipo de adaptação para atender algum requisito especifico da aplicação que estiver sendo desenvolvida. Depois da seleção e adaptação os componentes são integrados para compor a aplicação.

2.3.3 Reúso e Ontologias

Uma ontologia é freqüentemente definida como a especificação de uma conceitualização (GRUBER, 1995). A conceitualização refere-se à abstração de uma parte do mundo (um domínio) onde são representados os conceitos relevantes e seus relacionamentos. A especificação é formal e, portanto, processável por um sistema de computação.

As ontologias provêem uma terminologia não-ambígua que pode ser compartilhada por todos os envolvidos no processo de desenvolvimento de software. Elas podem ser tão genéricas quanto necessário permitindo o seu reúso e fácil extensão. Estas características tornam a ontologias úteis para representar o conhecimento das técnicas e metodologias para a Engenharia de Software e é um mecanismo de abstração apropriado para especificação de artefatos de software de alto nível como modelos de domínio, frameworks e padrões de software (GIRARDI et al., 2004).

De acordo com seu grau de generalidade, as ontologias podem ser classificadas em: ontologias genéricas, ontologias de domínio, ontologias de tarefa e ontologias de aplicação (GUARINO, 1998).

Uma ontologia genérica descreve conceitos gerais, tais como, espaço, tempo, matéria, objeto, sendo independentes de domínio. Uma ontologia de domínio reúne conceitos de um domínio específico e seus relacionamentos, definindo restrições na estrutura e conteúdo do conhecimento desse domínio. Uma ontologia de tarefa expressa conceitos sobre a resolução de problemas, independentemente do domínio em que ocorram, ou seja, descreve o vocabulário relacionado a uma atividade ou tarefa. Uma ontologia de aplicação descreve conceitos dependentes ao mesmo tempo de um domínio particular e de um conjunto de tarefas específicas. Estes conceitos freqüentemente correspondem a papéis desempenhados por entidades do domínio enquanto realizam certas atividades.

Os conceitos de uma ontologia de domínio ou de uma ontologia de tarefa podem ser especializados dos termos introduzidos por uma ontologia genérica. Os conceitos de uma ontologia de aplicação, por sua vez, podem ser especializações dos termos das ontologias de tarefas e de ontologias de domínio (Figura 19). As setas expressam relacionamentos de especialização.

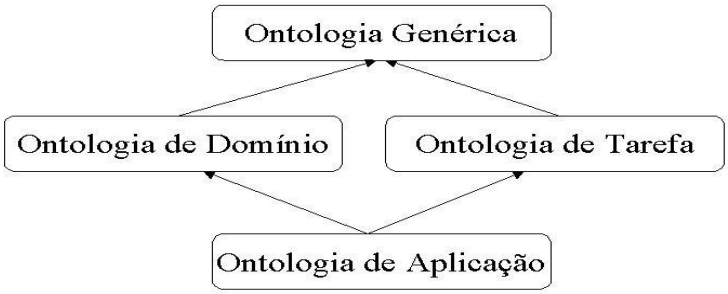


Figura 19 - Classificação das ontologias segundo o seu nível de generalidade. Fonte: (GUARINO, 1998)

As ontologias também podem ser diferenciadas segundo seu grau de formalidade. (USCHOLD, 1996) classifica uma ontologia como altamente informal, informal estruturada, semi-formal e rigorosamente formal, onde:

- **Altamente informal:** é expressa imprecisamente em linguagem natural como, por exemplo, um glossário;
- **Informal estruturada:** é expressa em linguagem natural, mas de forma estruturada e com o escopo do vocabulário bem-definido. Diminui consideravelmente o nível de ambigüidade em relação as ontologias do tipo altamente informal;
- **Semi-formal:** é expressa em uma linguagem artificial formalmente definida;
- **Rigorosamente formal:** os conceitos da ontologia são definidos em semântica formal, com teoremas e provas de suas propriedades.

As ontologias oferecem vantagens ao desenvolvimento de software, especialmente pela sua característica de compartilhamento de conhecimento independente de linguagem de representação, o que facilita o reúso de artefatos. Em (HAPPEL e SEEDOF, 2006) as vantagens do uso de ontologias são discutidas em vários aspectos do desenvolvimento de software:

- **Engenharia de Requisitos:** Na Engenharia de Software tradicional o conhecimento a respeito do domínio da aplicação fica separado do conhecimento acerca dos requisitos da aplicação, podendo trazer inconsistências entre os conceitos do domínio e o dos requisitos. Além disso, os requisitos muitas vezes são descritos em linguagem natural o

que deixa margem para ambigüidades. Através do uso de ontologias todo esse conhecimento pode ser integrado evitando tanto o problema da ambigüidade quanto da inconsistência;

- **Reúso de Componentes:** as ontologias são úteis para selecionar componentes por causa da sua expressividade semântica permitindo representar, por exemplo, relacionamentos entre partes (“part of”) e de herança (“is a”), enquanto que a maioria dos repositórios tradicionais são limitados a buscas sintáticas por palavras chaves o que traz baixa precisão por causa das palavras homônimas e baixa recuperação por causa das palavras sinônimas;
- **Implementação:** as ontologias facilitam a documentação e reúso através da descrição dos artefatos de forma semântica. Se os produtos das fases de análise e projeto forem realizados utilizando ontologias como, por exemplo, através de linguagens de modelagem semânticas, o código pode ser gerado automaticamente definindo-se regras de mapeamento de análise/projeto para implementação;
- **Linguagens de modelagem:** A utilização de uma linguagem de modelagem baseada em ontologias tem diversas vantagens em relação a linguagens baseadas no metamodelo MOF (BOOCH et al., 1999) como a UML, pois estas últimas não são baseadas no conhecimento diminuindo a precisão de consultas e não permitindo a realização de inferências. Abordagens como MDA, para transformações de modelos em código ficam prejudicados nesse sentido, pois tem que ser especificadas muitas regras de validação de modelos e de transformações de modelos em código, enquanto que, com ontologias, muitas dessas regras podem ser simplesmente inferidas através de raciocínio lógico.

Esta seção apresentou conceitos fundamentais do desenvolvimento baseado no reúso de software. Primeiro, foi apresentada uma classificação para as diversas perspectivas de reúso e a motivação de incorporá-lo ao desenvolvimento de software. Em seguida, foram apresentadas as abordagens de reúso composicional e gerativa com intuito de fundamentar a escolha de uma dessas abordagens para se utilizada no processo MADAE-Pro. Depois, foi apresentado o

desenvolvimento em linhas de produtos de software através das Engenharia de Domínio e de Aplicações. Por fim, foram definidos os conceitos básicos a respeito de ontologias e a relação das mesmas com o reúso de software. Todos esses conceitos aqui discutidos são a base teórica para a especificação do processo MADAE-Pro, a ser apresentado no próximo capítulo.

2.4 Linguagens de Modelagem de Processos de Software

2.4.1 Conceitualização

Uma linguagem possui um léxico (um conjunto de símbolos) e uma gramática (uma especificação de como estes símbolos podem ser combinados para fazer fórmulas bem-formadas). O léxico consiste de um conjunto de símbolos (como conectivos boleanos e quantificadores) e símbolos não-lógicos (SCHLENOF et al., 2000).

Para especificar um processo de software são utilizadas linguagens específicas para modelagem de processos chamadas de PML (“Process Modeling Language”), as quais oferecem mecanismos para representar os elementos do processo como fases, tarefas e produtos e seus relacionamentos.

São inúmeras as vantagens da utilização de PML’s para representar um processo de software ao invés da simples descrição em linguagem natural. PML’s facilitam o entendimento e padronizam o processo de software. Melhoram também o gerenciamento do processo, realização de treinamentos e a criação de ferramentas que apóiem o processo de desenvolvimento de software.

2.4.2 Classificação

Algumas PML’s fornecem uma representação gráfica para os elementos do processo, outras representação simplesmente textual como em (SAEKI et al., 1989) e/ou lógica. Também é possível combinar linguagens gráficas, textuais e/ou lógicas.

A seguir são apresentadas as linguagens para modelagem de processo de software SPEM (SPEM, 2008), MVP-L (BRÖCKERS et al., 1995a) e E3

(JACCHERI et al., 1998), (JACCHERI e STÅLHANE, 2001). A linguagem SPEM é descrita em maiores detalhes, uma vez que ela é utilizada na especificação do processo MADAE-Pro. Ao final desta seção é realizado um breve estudo comparativo entre as linguagens de modelagem de processos.

2.4.3 SPEM

2.4.3.1 Conceitualização

O OMG ("Object Management Group") definiu uma arquitetura que especifica os níveis de abstração de um modelo (Figura 20). O nível M3 descreve o meta-modelo baseado no MOF ("Meta-Object Facility"), uma tecnologia adotada pela OMG para definir metadados; o nível M2, descreve o nível de meta-modelo do processo; já o nível M1 descreve um modelo de processo como RUP, XP, etc. Por fim, o nível M0 descreve um processo instanciado para um projeto específico.

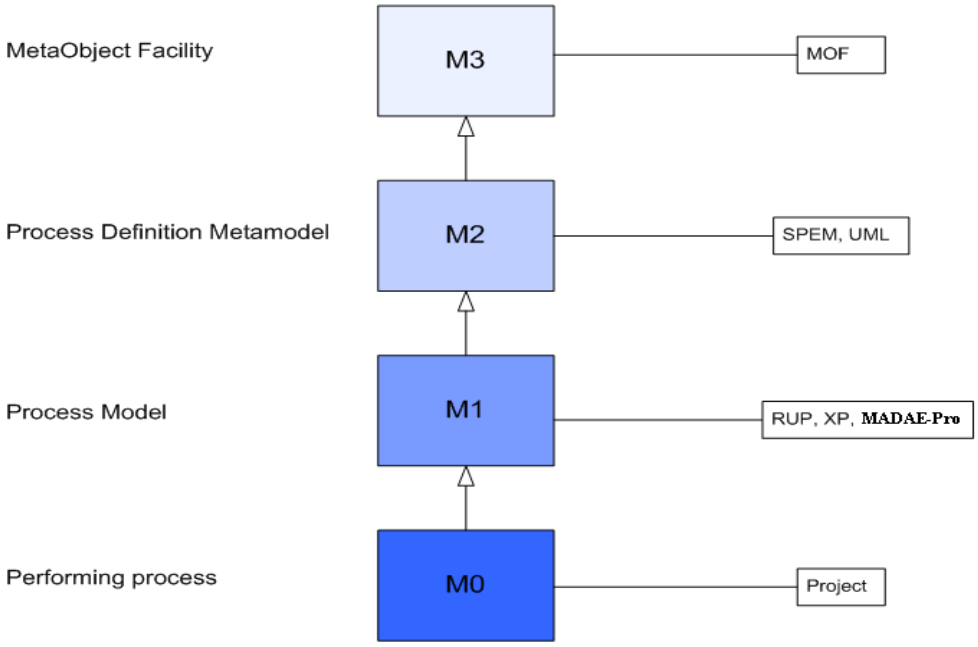


Figura 20 - Níveis de Abstração. Fonte: Adaptado de (SPEM, 2008)

SPEM (SPEM, 2008) é uma especificação do OMG para a padronização de processos de software. SPEM define como representar um processo, especificando os elementos, relacionamentos e restrições.

A idéia central do SPEM é que o processo de desenvolvimento de software é uma colaboração entre entidades ativas chamadas de papéis do processo que executam operações chamadas atividades sobre entidades concretas chamadas produtos do trabalho. Conforme esquematizado na Figura 21, em um processo de software o papel do processo (“Role”) executa atividades (“Activities”), as quais requerem ou produzem artefatos (“WorkProduct”).

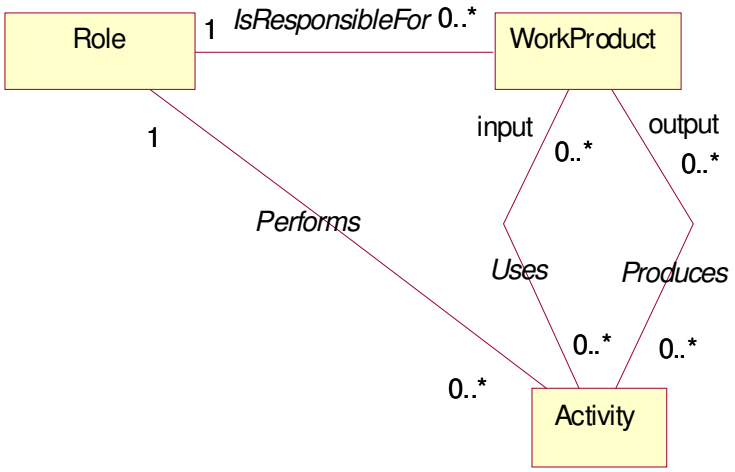


Figura 21 – Processo de Software. Fonte: (SPEM, 2008)

2.4.3.2 Estrutura

O SPEM é formado pelos pacotes SPEM_Foudation e SPEM_Extension (Figuras 22 e 23). O pacote SPEM_Foundation é um subconjunto da UML que define elementos básicos dessa linguagem de modelagem como, por exemplo, classes, relacionamentos, diagramas, etc. Existem algumas pequenas diferenças dos elementos definidos na UML e os utilizados pelo SPEM como, por exemplo, as associações na UML podem ser n-árias (relacionamento entre mais de duas classes) e no SPEM podem ser somente do tipo binária (relacionamento entre duas classes).

O segundo pacote do SPEM é o SPEM_Extension, o qual estende o pacote SPEM_Foundation para as construções e semânticas requeridas para a modelagem de processos de software. Por exemplo, a classe “Actor” definida no pacote SPEM_Foundation é mapeada para a classe “ProcessRole” no pacote SPEM_Extension, através dos mecanismos de extensão da UML como os estereótipos, vistos em maiores detalhes nas seções subseqüentes.

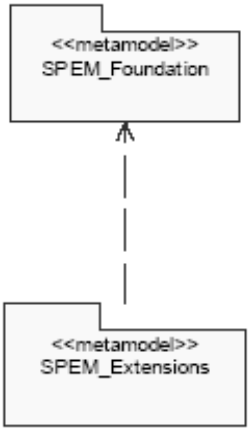


Figura 22. SPEM Foundation. Fonte: (SPEM, 2008)

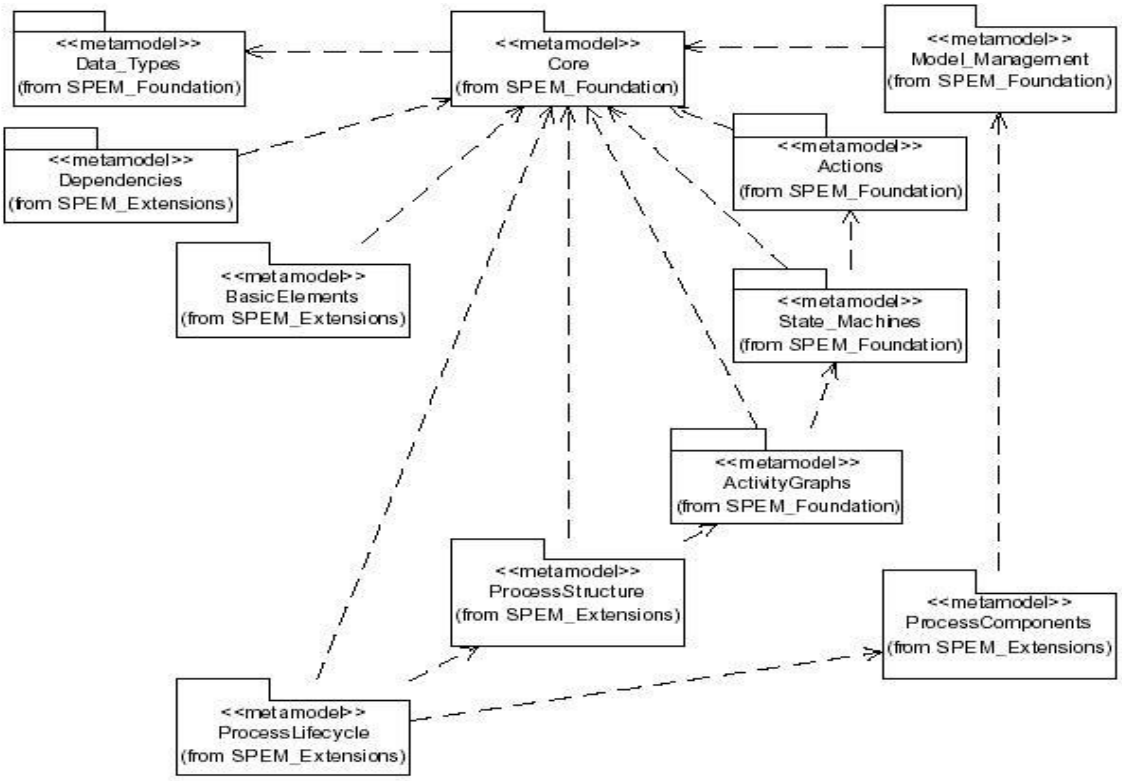


Figura 23. Estrutura dos Pacotes SPEM. Fonte (SPEM, 2008)

O pacote SPEM_FOUNDATION é subdivido nos seguintes pacotes :

- SPEM_Foundation::Data_Types: é um subconjunto do pacote Data_Types da UML e contém as definições de tipos de dados como, por exemplo, Integer, String, e Boolean;
- SPEM_Foundation::Core: este pacote é estruturado de forma similar ao pacote Core da UML e contém os elementos de modelagem que

dão suporte a base estrututural do meta-modelo. Possui elementos que definem relacionamentos como, por exemplo, “Association” e “Generalization”;

- SPEM_Foundation::Actions: é um subconjunto do pacote Common_Behaviour da UML e define elementos como “Action” e “Operation”;
- SPEM_Foundation::State_Machines: é um subconjunto do pacote “State_Machines” e define elementos que representam o conceito de máquina de estados como “Transition”, “State”, “Action”, etc;
- SPEM_Foundation::Activity_Graphs: é um subconjunto do pacote Activity_Graphs da UML e define elementos de modelagem de diagrama de atividades;
- SPEM_Foundation::Model_Management: é um subconjunto do pacote Model_Management da UML 1.4. Uma diferença no SPEM em relação à UML é que todos os elementos têm visibilidade pública e que os elementos importados para os pacotes não podem ser renomeados.

O pacote SPEM_Extension é dividido em 5 pacotes:

- SPEM_Extension::Basic Elements: define os elementos básicos para descrever um processo de software, como artefato;
- SPEM_Extension::Dependencies: define as possíveis dependências entre os elementos do processo. Por exemplo, a dependência “Impacts” que pode existir entre dois workproducts, especificando quais os efeitos que as modificações em um workproduct A podem causar em um workproduct B.

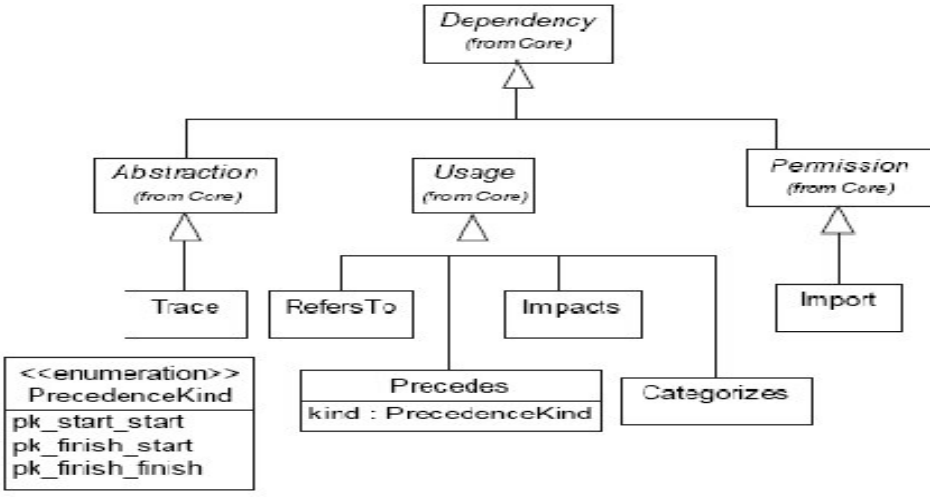


Figura 24 - Dependências SPEM. Fonte: (SPEM, 2008)

- SPEM_Extension::ProcessStrutures: define a estrutura dos elementos pelo qual o processo é descrito. Por exemplo, um elemento “Activity” é composto por um conjunto de elementos “Steps”;
- SPEM_Extension::ProcessComponents: define como o processo pode ser subdividido dentro de pacotes. Isso se dá através de pacotes, disciplines (pacotes com tarefas de mesma natureza), etc;
- SPEM_Extension::ProcessLifecycle: define um conjunto de elementos que permitem especificar o ciclo de vida do processo. Através de elementos como pré e pós-condições, fases, etc. Na Figura 25 estes elementos estão especificados:

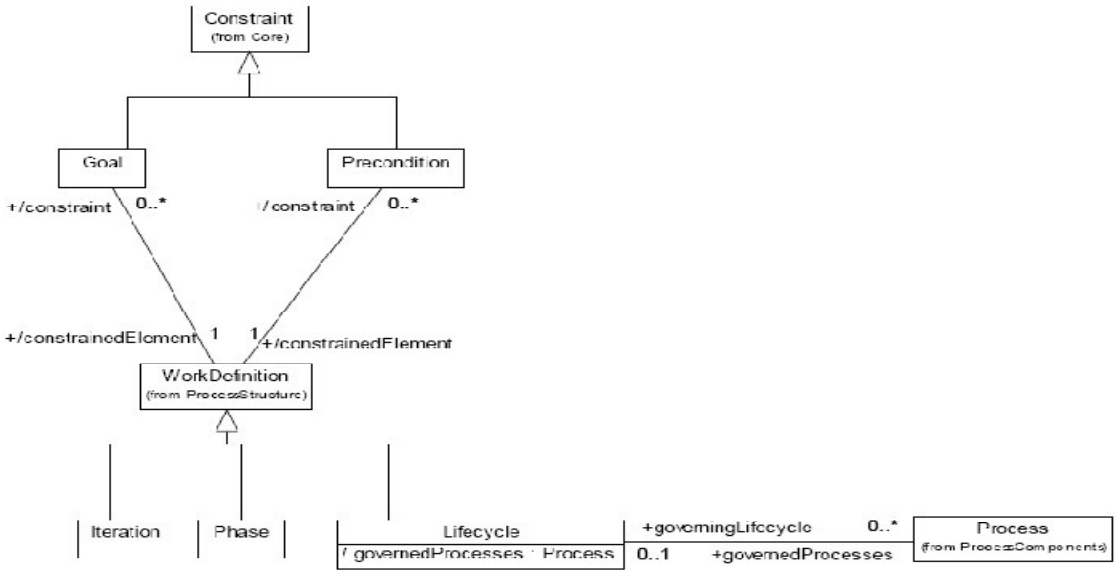


Figura 25 - Estrutura do pacote Ciclo de Vida. Fonte: (SPEM, 2008)

2.4.3.3 O Perfil SPEM

A UML, apesar de ser bastante abrangente, não consegue ser suficientemente expressiva para descrever as especificidades de alguns domínios e aplicações como algumas características dos sistemas distribuídos e da modelagem de processos. Para isso, ela permite a criação de sub-linguagens para propósitos especiais chamadas de perfis. Os perfis são criados através de mecanismos de extensão da UML. Esses mecanismos são os estereótipos, os valores atribuídos e as restrições.

Um estereótipo amplia o vocabulário da UML, permitindo a criação de novos blocos de construção que são derivados dos já existentes, mas específicos a determinados problemas (UML, 1999). Os blocos de construção da UML são divididos em itens, relacionamentos e diagramas. Os itens podem ser estruturais (classes, interfaces, casos de uso, etc), comportamentais (interações, estados e atividades), de agrupamento (pacotes) e anotacionais (notas). Os relacionamentos podem ser, por exemplo, a associação e a generalização. Existem também diversos tipos de diagramas que reúnem um conjunto de itens e de relacionamentos como, por exemplo, o diagrama de classe. Os nomes dos estereótipos são representados entre “<< >>” e podem ter uma representação gráfica chamada de ícone.

Um valor atribuído estende as propriedades dos blocos de construção da UML, permitindo a criação de novas informações na especificação de um elemento (UML, 1999). Por exemplo, pode-se criar um valor atribuído “version” com o valor 1.2 referente à versão de uma classe, sendo representado da seguinte forma “{version=1.2}”, logo após o nome da classe.

Uma restrição amplia as semânticas dos blocos de construção da UML, permitindo acrescentar novas regras ou modificar as já existentes (UML, 1999). Restrições podem ser representadas textualmente, através de notas associadas aos elementos da UML e também através da OCL. A OCL é uma linguagem formal e permite fazer a validação das regras. Por exemplo, em uma classe *Turma*, com o atributo *numeroDeAlunos* poderíamos criar textualmente a seguinte restrição “uma turma deverá ter mais do que 05 alunos”. Na OCL a mesma restrição poderia ser especificada da seguinte forma:

context *Turma* ***inv*** *:*

numeroDeAlunos > 05

Onde **context**, significa onde a restrição se aplica (determinada classe ou objeto) e **inv** é uma abreviação de invariante, que são as restrições a que os estados válidos devem obedecer, nesse caso ser maior que 5.)

A Figura 26 ilustra alguns dos mapeamentos de elementos da UML para o SPEM. Por exemplo, no diagrama de casos de usos da UML uma entidade externa que interage com o sistema é representada pelo elemento “Actor”, no SPEM esse elemento passa a se chamar “ProcessRole”, sendo alterado além do nome do elemento também a sua representação gráfica e o seu significado, que neste caso é de um elemento que realiza uma ou mais atividades do processo.

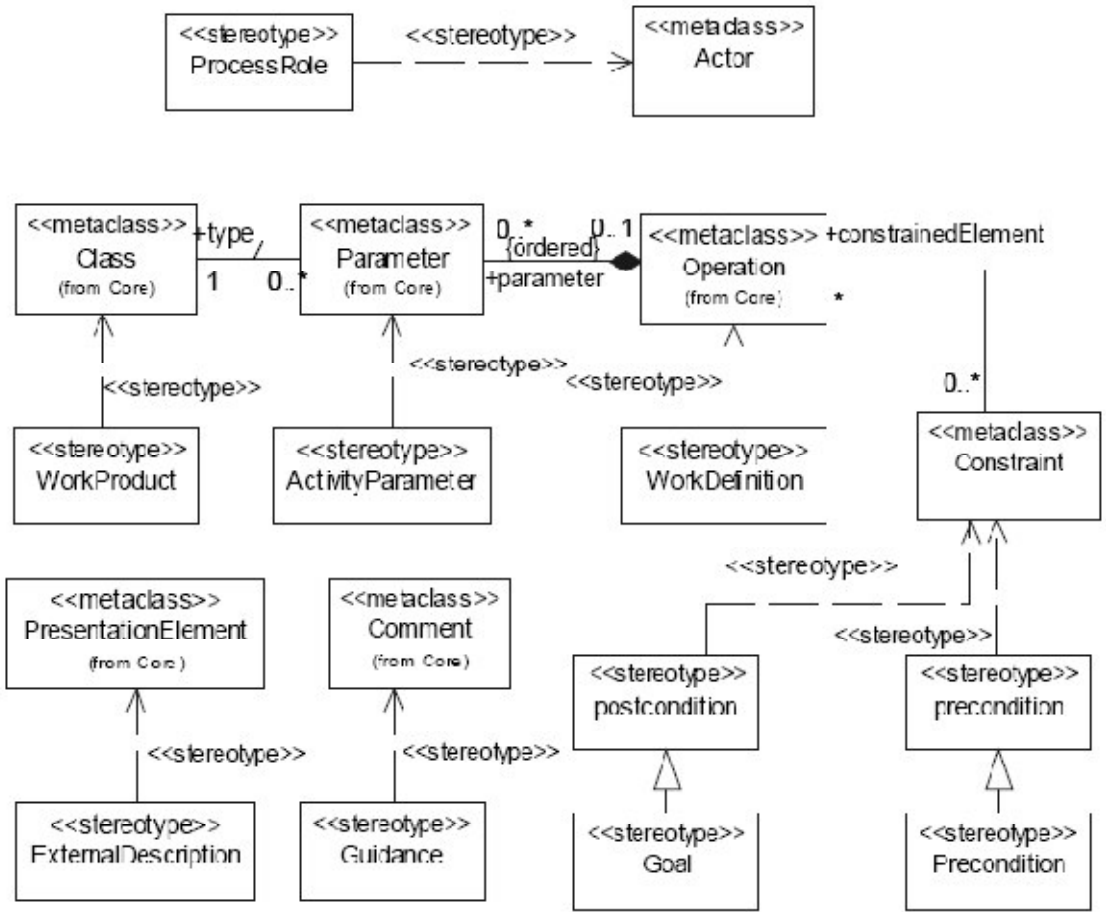
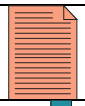
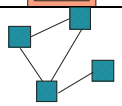
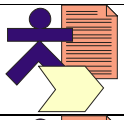
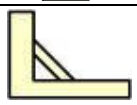


Figura 26 – Exemplo de mapeamento UML para SPEM. Fonte: (SPEM 2008)

A Tabela 4 apresenta um resumo dos principais estereótipos definidos para o SPEM, incluindo o nome, a descrição e o ícone.

Tabela 4 - Estereótipos SPEM

Esteréotipo	Descrição	Ícone SPEM
<<Process>>	Representa um processo. Ex.: RUP, OPEN.	
<<Phase>>	Representa uma fase do processo. Ex.: Implantação.	
<<WorkProduct>>	É qualquer produto ou conjunto de produtos, consumido ou modificado pelo processo, também chamado de artefato. Ex.: Interfaces com o usuário.	
<<Document>>	Notação específica para WorkProduct que representa um Documento. Ex. Documento descrevendo experiências de desenvolvimento, documento de Requisitos.	
<<UMLModel>>	Notação específica para um Workproduct que representa um Modelo UML. Ex.: Diagrama de Casos de Uso	
<<WorkDefinition>>	É uma operação que descreve o fluxo de execução do trabalho realizado numa fase do processo. Ela pode ser formada por outras subtarefas. Ex.: Modelagem de Requisitos	
<<Process Role>>	Executor das tarefas que darão origem aos WorkProducts utilizando destrezas específicas. Ex.: Analista de Sistemas.	
<<Activity>>	Uma subclasse de WorkDefinition. Descreve uma parte do trabalho realizado por um ProcessRole. Uma Atividade pode estar composta de passos. Ex.: Captura de Requisitos, um dos passos para realizar a Modelagem de Requisitos.	
<<Process Package>>	É um repositório utilizado para agrupar elementos do processo.	
<<Discipline>>	Um tipo de pacote que agrupa elementos do processo de acordo com um determinado tema. Ex. Modelagem de Negócios do processo RUP.	
<<Guidance>>	Fornecem suporte aos Papéis do Processo para a realização de algumas tarefas do processo. Pode representar técnicas, templates, etc.	

2.4.3.4 Diagramas SPEM

Para representar as diferentes perspectivas de um processo são usados os diagramas básicos da UML. No entanto, os elementos são modificados de acordo

com o perfil do SPEM (Estereótipos, valores atribuídos e restrições). Existem várias ferramentas que suportam a modelagem utilizando o SPEM como, por exemplo, o Rational Rose (ROSE, 2008) e o Microsoft Viso (VISIO, 2008). A seguir são apresentados alguns exemplos de diagramas SPEM:

Diagrama de Classe

O Diagrama de Classe permite a representação de aspectos do processo de software como herança, dependências, associações simples, comentários com links para guias, relações entre executores do processo ou entre papéis do processo e produtos, estrutura, decomposição e dependências de produtos. Os seguintes elementos não são permitidos no diagrama de classes usando SPEM: interface, template, agregação (diamante branco), associações qualificadas, associações n-árias. A Figura 27 ilustra um diagrama de classes, enfatizando os responsáveis (ex. Software Architect) por produzir cada artefato (ex. Interface Model) em um determinado processo de desenvolvimento de software.

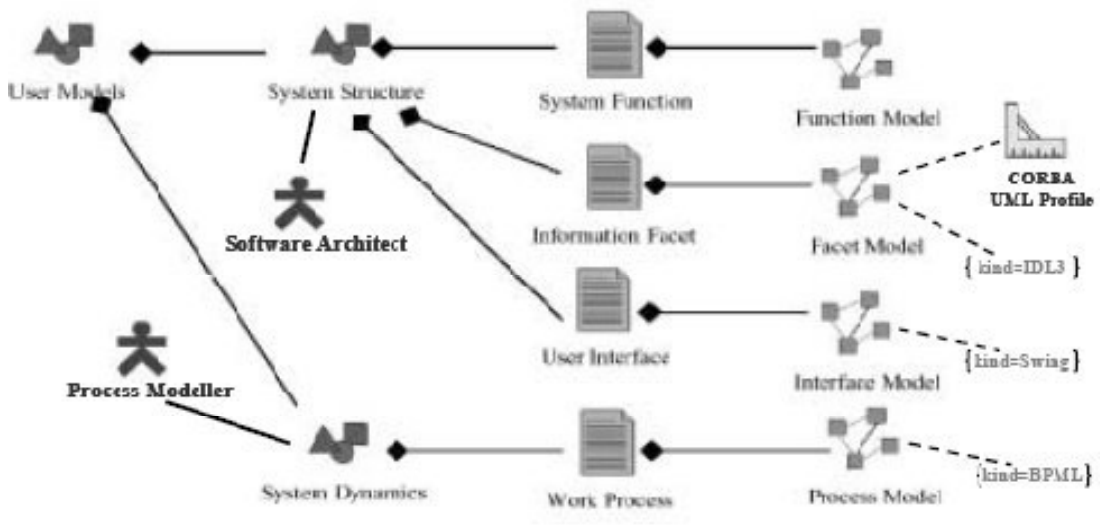


Figura 27 - Exemplo de Diagrama de Classes. Fonte: (SPEM, 2008)

Diagrama de Pacotes

Na Figura 28 é ilustrado o Diagrama de Pacote. Esse diagrama permite representar os elementos de um processo em grupos relacionados por natureza. Por exemplo, um pacote com os artefatos desenvolvidos na fase de Análise. Formas aninhadas e não-aninhadas desse diagrama podem ser usadas.

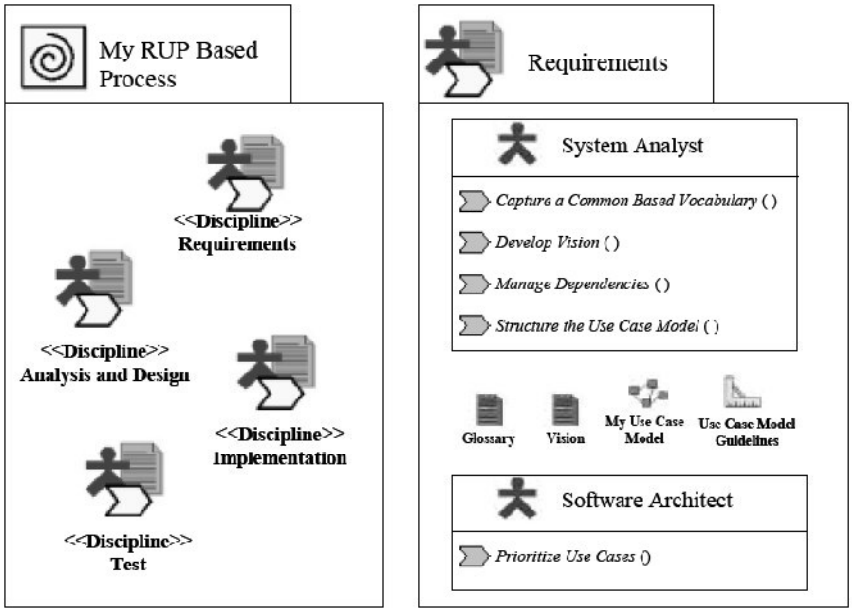


Figura 28 – Exemplo de Diagrama de Pacotes. Fonte: (SPEM, 2008)

Diagrama de Casos de Uso

Na Figura 29 um exemplo de diagrama de casos de usos é ilustrado. Esses diagramas são úteis para mostrar as relações entre papéis do processo e as atividades desenvolvidas por eles e também para mostrar a relação entre as atividades desenvolvidas e as fases as quais elas pertencem.

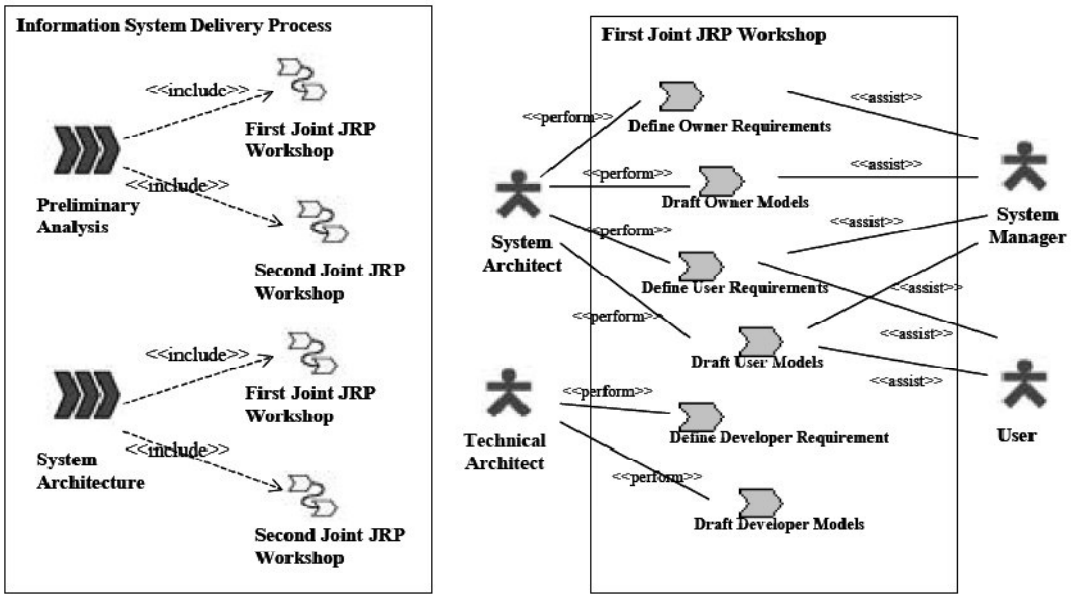


Figura 29 – Exemplo de Diagrama de Casos de Uso. Fonte: (SPEM, 2008)

Diagrama de Atividades

Na Figura 30 é ilustrado o diagrama de atividades. Esse diagrama permite representar as seqüências de atividades com as entradas e saídas de produtos de trabalho, como também o fluxo de estados dos objetos. As raias de natação podem ser usadas para separar as responsabilidades de diferentes papéis do processo.

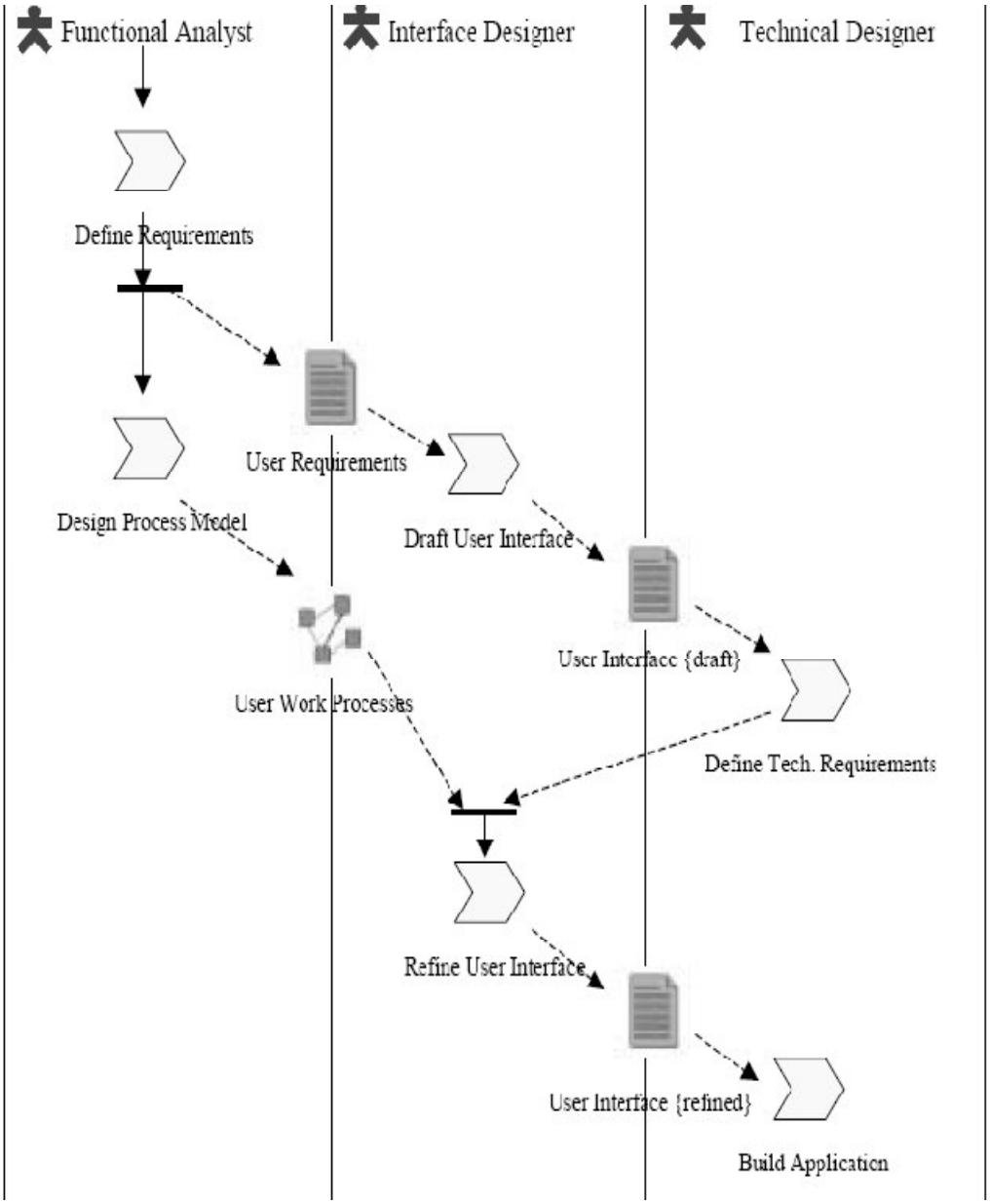


Figura 30 - Exemplo de Diagrama de Atividades. Fonte: (SPEM, 2008)

2.4.4 MVP-L

A linguagem para representação de processos de software MVP-L (“Multi View Process Modeling Language”) permite modelar processos, produtos, recursos e atributos de qualidade.

A principal idéia dessa linguagem é dividir a descrição do processo em visões, onde cada visão representa um aspecto do processo. A representação do processo é feita de forma natural, isto é, ela é fácil de ser entendida não somente por máquinas como também por pessoas. Ela também permite a representação de diferentes tipos de modelos e é formal.

A linguagem MVP-L possui uma representação gráfica para definir o processo de software simples, formada por um conjunto de figuras geométricas. A linguagem gráfica é complementada com uma descrição textual baseada em regras com a sintaxe definida em (BRÖCKERS et al., 1995b) e que pode ser compilada em um editor desenvolvido em C++. A Figura 31 e a Figura 32 ilustram, respectivamente, a representação gráfica e textual provida pela linguagem MVP-L. Na Figura 31 as elipses representam os produtos, o retângulo representa um processo elementar (em alguns processos de software esse elemento corresponde a uma atividade ou tarefa realizada no processo) e o paralelogramo representa um recurso do processo. Na descrição textual do modelo o processo é detalhado e são, por exemplo, especificadas as importações e exportações de elementos, juntamente com os critérios de importação e exportação, são definidos também quais os produtos estão sendo consumidos, produzidos ou modificados pelo processo.

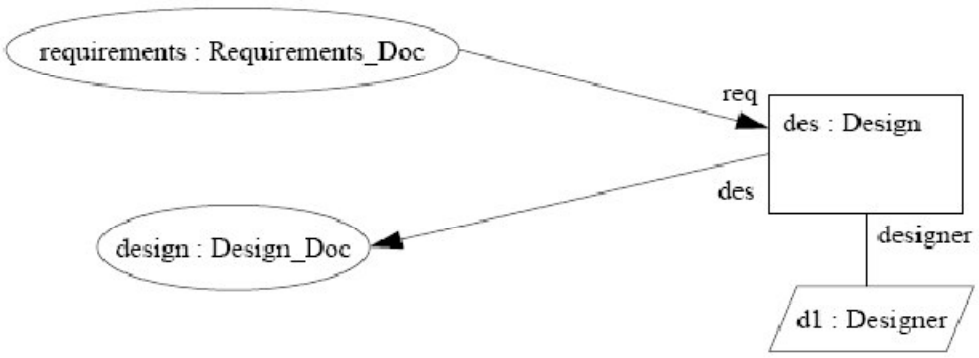


Figura 31 - Exemplo de um diagrama de projeto. Fonte: (BRÖCKERS et al., 1995b)

```
project_plan sample
imports
    product_model Requirements_Doc, Design_Doc;
    resource_model Designer;
    process_model Design;
objects
    requirements : Requirements_Doc;
    design : Design_Doc;
    d1 : Designer;
    des : Design;
...
end project_plan sample

process_model Design () is
...
product_flow
    consume
        req : Requirements_Doc;
    produce
        des : Design_Doc;
```

Figura 32 - Fragmento da descrição textual do processo da Figura 17. Fonte: (BRÖCKERS et al., 1995).

2.4.5 E3

E3 é uma linguagem para modelagem de processos de software orientada a objetos com uma notação gráfica, o que facilita a representação e comunicação dos modelos de processo (JACCHERI et al., 1998).

E3 possui um conjunto de classes e associações para descrever as entidades básicas e relacionamentos envolvidos em um processo de software, que podem ser personalizadas pelo desenvolvedor através de herança. Essa linguagem possui três níveis de abstração: criação, definição e instanciação. No nível de criação, as classes e associações do processo são criadas através de especialização. No nível de definição, as classes e associações criados no nível de criação são conectadas por associações formando um template. No nível de instanciação, classes e associações pertencentes a um template são instanciadas em objetos e ligações entre objetivos que representa as entidades pertencentes a um modelo do projeto que estiver sendo modelado.

As classes utilizadas por essa linguagem para modelagem de processos são: “Task” que representa atividades realizadas em um processo; “Role” que define

um conjunto de responsabilidades e um conjunto de destrezas; “Data” que representa um artefato como, por exemplo, código fonte; “Tool” que representa uma ferramenta ou um procedimento. Essas classes possuem diversos relacionamentos entre si, como por exemplo: “**binary**(objeto1, objeto2)”, que representa a associação entre dois elementos do processo; “**agregation**(objeto1, objeto2)”, que representa um composição entre dois elementos do processo, “**subtask**(tarefa1, tarefa2), que define a decomposição das tarefas em subtarefas ”; “**responsible**(task,role)”, que define os responsáveis de cada tarefa; “**preorder**(task1,task2)”, que significa que a tarefa2 só pode ser iniciada depois da realização da tarefa1.

Essa linguagem é suportada pela ferramenta CASE P-Draw (Process Draw) que permite modelar o processo. Essa ferramenta possibilita analisar, desenvolver, verificar e reusar modelos E3. Os modelos são representados seguindo os níveis de abstração de criação, definição e instanciação. Na Figura 33 a criação da classe “Requirement Analysis” é ilustrada, representando a fase de Análise de Requisitos e um conjunto de outras classes associadas a ela (ex. Preliminary Domain Description) representando as atividades realizadas.

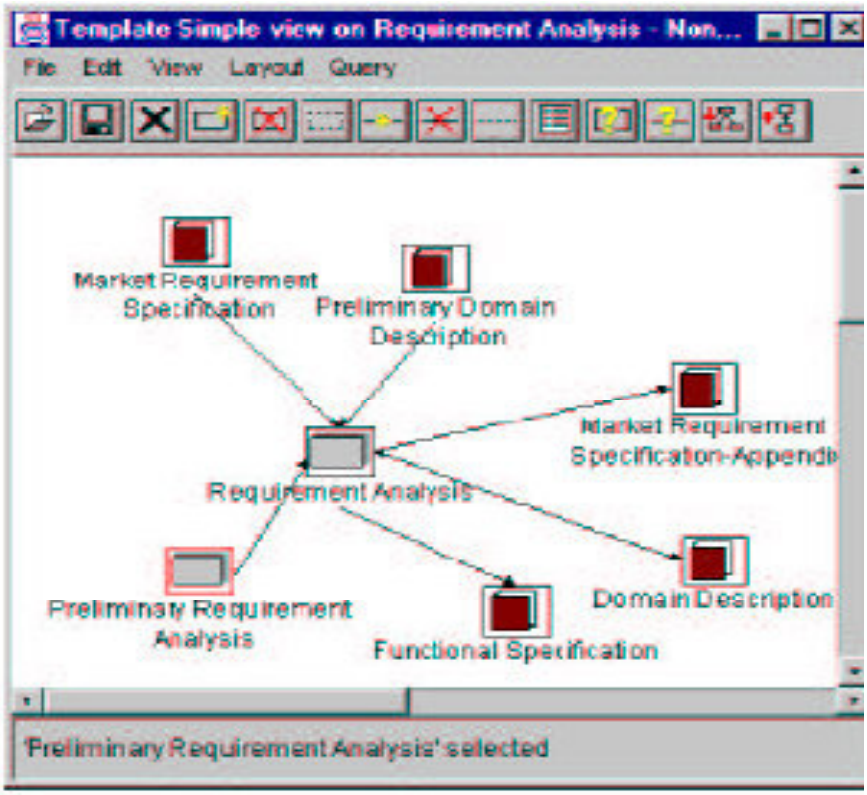


Figura 33 - Visão da classe Análise de Requisitos. Fonte: (JACCHERI e STÅLHANE, 2001)

2.4.6 Estudo Comparativo das Linguagens para Especificação de Processo de Software

Na Tabela 5, algumas características requeridas para uma linguagem de modelagem de processos de software são resumidas. Pode-se visualizar na tabela que quanto ao nível de formalidade, a linguagem MVP-L possui o maior nível de formalidade, podendo ser, inclusive, compilada, enquanto que o SPEM e E3 possuem um nível formalidade menor, porque o SPEM permite a validação dos modelos através da definição de restrições em OCL e E3 permite a verificação das propriedades dos modelos, mas ambas não permitem a execução do processo.

As linguagens MVP-L e E3 possuem uma notação gráfica para representar os elementos do processo básico, principalmente a MVP-L. Já a linguagem SPEM possui uma notação mais expressiva, com maior número de elementos, relacionamentos entre eles e diagramas disponíveis para a modelagem de processos.

A documentação disponível para as linguagens MVP-L e E3 é bastante escassa e com poucos exemplos, já a linguagem SPEM é bem mais detalhada, no entanto, também possui poucos exemplos da modelagem de processos de software reais na documentação oficial, definida pelo OMG. Contudo, vários exemplos de sua aplicação foram encontrados em artigos e relatórios técnicos publicados por terceiros.

Quanto a aprendizagem da linguagem, ou seja, o seu entendimento e aplicação, a linguagem E3 é a de fácil aprendizado, já que ela usa conceitos já bastante conhecidos da orientação a objetos como herança, agregação e os elementos dos modelos são em pequeno número e relativamente simples. A linguagem SPEM, por ser predominantemente visual, também é fácil de aplicar e entender, principalmente para quem já conhece a UML. Já linguagem MVP-L por ser melhor expressa na linguagem baseada em regras do que em notação visual, devido a notação grafica ser definida conceitualmente, mas não ser suportada ainda pela ferramenta de modelagem é a que possui a maior curva de aprendizado. Assim o usuário deverá aprender a sintaxe da linguagem baseada em regras, o que aumenta um pouco o tempo de aprendizado.

Por fim, quanto ao suporte de ferramentas, SPEM é a que possui o melhor suporte e em maior número e, a linguagem MVP-L é a que possui o menor suporte, pois não permite a modelagem gráfica do processo.

Tabela 5 - Comparativo entre Linguagens para Modelagem de Processos

Linguagem	Formalidade	Notação Gráfica	Documentação	Curva de aprendizagem	Suporte de Ferramentas
SPEM	Semi-formal	Média	Razoável	Baixa	Bom
MVP-L	Formal	Simples	Pouca	Médio	Pouco
E3	Semi-formal	Média	Pouca	Baixa	Médio

Das linguagens aqui apresentadas, SPEM além de ser uma das mais utilizadas atualmente para modelagem de processos e metodologias multiagente como, em (COSSENTINO et al., 2003)(GLEIZES et al., 2003) é a que mais se adéqua ao nosso propósito de realizar a especificação do processo de forma a garantir que esta seja ao mesmo tempo de fácil entendimento e tenha um nível de formalidade razoável para possibilitar a automatização do processo.

2.5 Considerações Finais

Este capítulo apresentou aspectos importantes de processos de desenvolvimento de software dos paradigmas orientado a objetos e a agentes, também fez uma breve descrição do reúso de software e dos benefícios da sua incorporação no processo de desenvolvimento. No próximo capítulo, um processo para o desenvolvimento de sistemas multiagente baseado no reúso sistemático de software será apresentado.

3 MADAE-Pro – UM PROCESSO BASEADO NO CONHECIMENTO PARA A ENGENHARIA DE DOMÍNIO E APLICAÇÕES MULTIAGENTE

3.1 Introdução

O MADAE-Pro é um processo para o desenvolvimento e reúso de famílias de sistemas multiagente. Ele integra dois subprocessos complementares. Um é baseado nos conceitos da Engenharia de Domínio, isto é, visa construir artefatos reutilizáveis que representem uma família de aplicações e o outro, baseado na Engenharia de Aplicações, guia o desenvolvimento de uma aplicação específica reutilizando os produtos do primeiro subprocesso.

Outra característica do MADAE-Pro é ser um processo baseado no conhecimento. Isto é, a definição das suas fases, tarefas e produtos são classes da ONTORMAS e os relacionamentos entre esses conceitos são especificados através dos “slots” dessa ontologia. Durante o desenvolvimento das famílias de sistemas multiagente as classes da ONTORMAS são instanciadas formando uma base de conhecimento, o que faz da ONTORMAS também um repositório para os modelos desenvolvidos com o MADAE-Pro.

O MADAE-Pro envolve o uso de metodologias que especificam “como” realizar as tarefas de desenvolvimento, de um ciclo de vida que define a “ordem” de realização dessas atividades de desenvolvimento, de papéis do processo que representa “quem” é responsável por realizar cada tarefa e de tecnologias (ferramentas e linguagens) que apóiam os papéis do processo na realização de suas atividades.

Na rede semântica ilustrada na Figura 34 estão os principais elementos envolvidos no MADAE-Pro. O processo adota as metodologias MADEM e MAAEM. A MADEM guia o desenvolvimento de artefatos de software reusáveis integrantes de uma família de aplicações multiagente enquanto a MAAEM, o desenvolvimento de uma aplicação de software multiagente particular que pode ocorrer sem reuso ou reutilizando artefatos de software desenvolvidos a partir da metodologia MADEM. Uma técnica está associada com cada fase de desenvolvimento para guiar as tarefas de desenvolvimento das metodologias. As técnicas GRAMO, DDEMAS e DIMAS integram a metodologia MADEM, e estão associadas, respectivamente, as suas fases de Análise de Domínio, Projeto de Domínio e Implementação de

Domínio. As técnicas SRAMO, ADEMAS e AIMAS integram a metodologia MAAEM e estão associadas, respectivamente, as suas fases de Engenharia dos Requisitos da Aplicação, Projeto da Aplicação e Implementação da Aplicação.

O ciclo de vida adotado no MADAE-Pro é o iterativo e incremental, devido a complexidade inerente ao desenvolvimento de sistemas multiagente. O processo é suportado pela ferramenta ONTORMAS, que é utilizada para guiar as tarefas de desenvolvimento, realizar a modelagem visual, documentar e armazenar os artefatos produzidos durante a execução do processo. A linguagem para modelagem de sistemas multiagente adotada é a MADAE-ML. Esta linguagem estabelece uma representação gráfica para os modelos e conceitos de modelagem das metodologias MADAEM e MAAEM. Por fim, o processo define os papéis do processo (ex. Programador, Analista de Sistemas), encarregados da realização de uma ou mais tarefas durante a execução do processo.

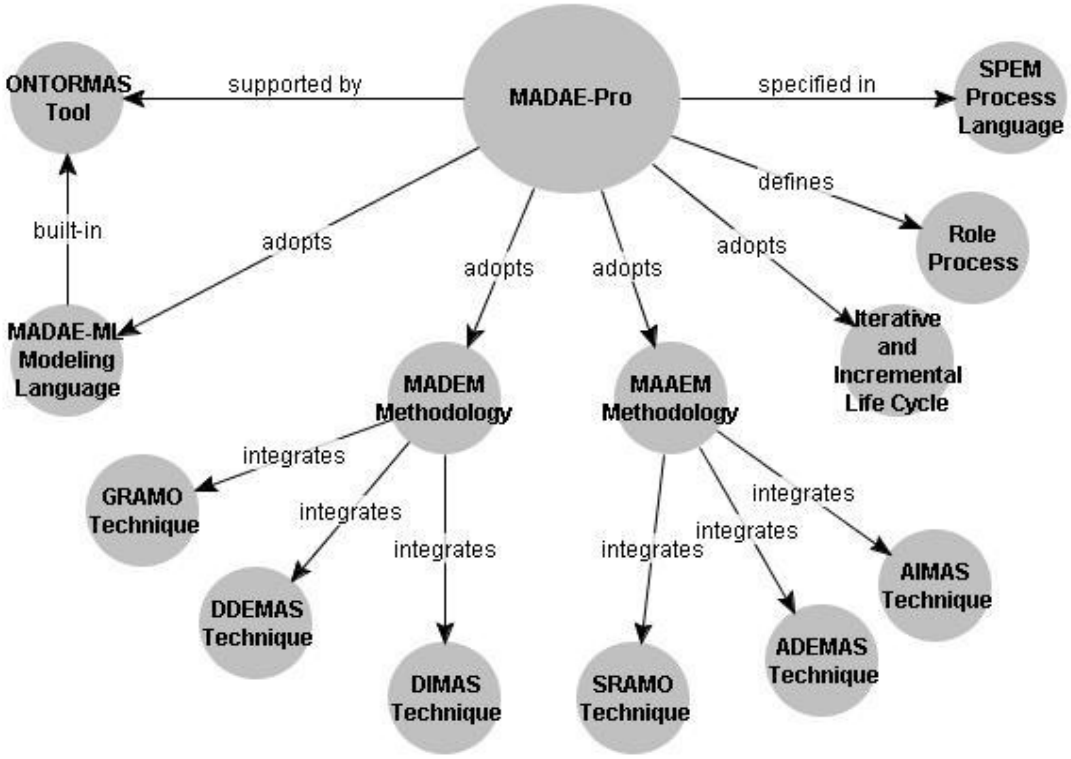
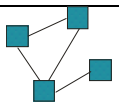
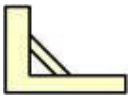


Figura 34 - Elementos envolvidos no MADAE-Pro

O MADAE-Pro está especificado na linguagem de modelagem de processos SPEM. Na Tabela 6, um resumo com a correspondência entre os estereótipos definidos na especificação SPEM, previamente descritos no capítulo 2, e os elementos do processo MADAE-Pro são apresentados.

Tabela 6 - Estereótipos SPEM e sua correspondência no processo MADAE-Pro

SPEM	MADAE-Pro	Descrição	Ícone
Process	**Processo**	Representa um processo de desenvolvimento de software. Ex.: MADAE-Pro, RUP, etc.	
Phase	**Fase**	Representa uma fase do processo. Ex.: Análise da Aplicação.	
WorkProduct	**Produto**	É qualquer produto ou conjunto de produtos, consumido ou modificado pelo processo. Ex.: Especificação da Aplicação.	
Document	**Documento**	Notação específica para *Workproduct* que representa um Documento. Ex. Documento descrevendo experiências de desenvolvimento, um documento de requisitos.	
UMLModel	**Modelo**	Notação específica para um Workproduct que representa um Modelo. (No MADAE-Pro, são utilizados modelos na linguagem MADAE-ML e não os da UML).	
WorkDefinition	**Tarefa**	É uma operação que descreve o fluxo de execução do trabalho realizado numa fase do processo. Ela pode ser formada por outras subtarefas. Ex.: Modelagem de Objetivos	
Process Role	**Papel do Processo**	Executor das tarefas que darão origem aos WorkProducts utilizando destrezas específicas. Ex.: Analista de Sistemas.	
Activity	**Subtarefa**	Uma subclasse de WorkDefinition. Descreve uma parte do trabalho realizado por um ProcessRole. Uma Atividade pode estar composta de passos. Ex.: Modelagem de Objetivos Específicos é uma subtarefa da tarefa de Modelagem de Objetivos.	
Process Package	**Pacote do Processo**	É um repositório utilizado para agrupar elementos do processo. Ex.: Pacote que agrupa as Disciplinas do MADAE-Pro.	
Discipline	**Disciplina**	Um tipo de pacote que agrupa elementos do processo de acordo com um determinado tema. Ex. Elementos da fase de Análise da Aplicação	
Guidance	**Destreza**	Fornecem suporte aos Papéis do Processo para a realização de algumas tarefas do processo. Ex.: A Técnica SRAMO que suporta as tarefas da fase de Análise da Aplicação.	

Quanto ao reúso o processo MADAE-Pro se enquadra na classificação de (PRIETO-DÍAZ, 1993), descrita no capítulo 2 da página 53, da seguinte forma:

- **Quanto ao conteúdo:** *componentes*, que são, no nível de especificação, os modelos desenvolvidos com instâncias da ONTORMAS, descrita na seção 3.3.2. No nível de implementação, os componentes são agentes de software.
- **Quanto ao escopo:** *vertical*, pois o reúso é realizado em um mesmo domínio de aplicação.
- **Quanto ao modo:** *sistemático*, utilizando os processos da Engenharia de Domínio e de Aplicações.
- **Quanto à técnica:** composicional.
- **Quanto à intenção**: caixa-branca (“*white-box”)*, o reúso tanto no nível de especificação quanto no nível de implementação permite a realização de adaptações.
- **Quanto ao Produto**: modelos de Domínio, Frameworks, Agentes.

Até aqui foi apresentada uma visão geral do MADAE-Pro no contexto dos conceitos discutidos no capítulo 2. As seções seguintes apresentam o processo MADAE-Pro sob duas perspectivas. A seção 3.2 apresenta o processo MADAE-Pro sob a perspectiva do seu ciclo de vida, onde é enfatizada a “ordem” de realização das tarefas de desenvolvimento em cada fase do processo. A seção 3.3 apresenta o processo no contexto das metodologias MADEM e MAAEM, onde é enfatizado “como” as tarefas de desenvolvimento são realizadas.

Para ilustrar os modelos produzidos em cada tarefa são utilizados exemplos extraídos da ONTOSERS (MARIANO, 2008), uma família de aplicações para recomendação de informações na Web Semântica, e da INFOTRIB (MARIANO, 2008), uma aplicação para recomendação de informações no domínio tributário. Nessa seção, a linguagem de modelagem de sistemas multiagente MADAE-ML adotada pelo processo e a ferramenta ONTORMAS, que dá suporte as tarefas de desenvolvimento são descritas em mais detalhes. Por fim, na seção 3.4, são feitas as considerações finais deste capítulo.

3.2 O ciclo de vida

A Figura 35 ilustra o ciclo de vida do processo MADAE-Pro, composto por seis fases de desenvolvimento. As três primeiras, Análise de Domínio, Projeto de Domínio e Implementação de Domínio, referem-se à Engenharia de Domínio Multiagente, onde se constrói artefatos genéricos destinados ao reúso. E as três últimas, Análise da Aplicação, Projeto da Aplicação e Implementação da Aplicação, referem-se à Engenharia de Aplicação Multiagente, onde se constrói uma aplicação específica a partir do reúso dos artefatos desenvolvidos na Engenharia de Domínio Multiagente.

O ciclo de vida é iterativo, incremental e dirigido por objetivos. O desenvolvimento é realizado a partir de sucessivos incrementos, com o intuito de reduzir a sua complexidade.

O ciclo inicia-se a partir da decisão de desenvolvimento de uma nova família de aplicações ou de uma aplicação específica (novo objetivo geral), conforme ilustrado no losango da figura.

O ciclo reinicia-se com a decisão de incrementar aquelas através da introdução de um novo objetivo específico (manutenção evolutiva) ou de alterar um objetivo específico preexistente (manutenção evolutiva e/ou corretiva). Iterações também podem ocorrer entre as fases, com o objetivo de aprimorar os produtos da modelagem.

Uma técnica está associada a cada fase de desenvolvimento para guiar a realização das tarefas de modelagem: as técnicas GRAMO (GIRARDI e FARIA, 2004), DDEMAS (GIRARDI e LINDOSO, 2005), DIMAS que integram a metodologia MADEM e as técnicas SRAMO (LINDOSO e GIRARDI, 2006), ADEMAS (LINDOSO e GIRARDI, 2005) e AIMAS que integram a metodologia MAAEM. Nas próximas seções, o subciclo de vida de cada fase definida no ciclo de vida do processo MADAE-Pro é apresentado.

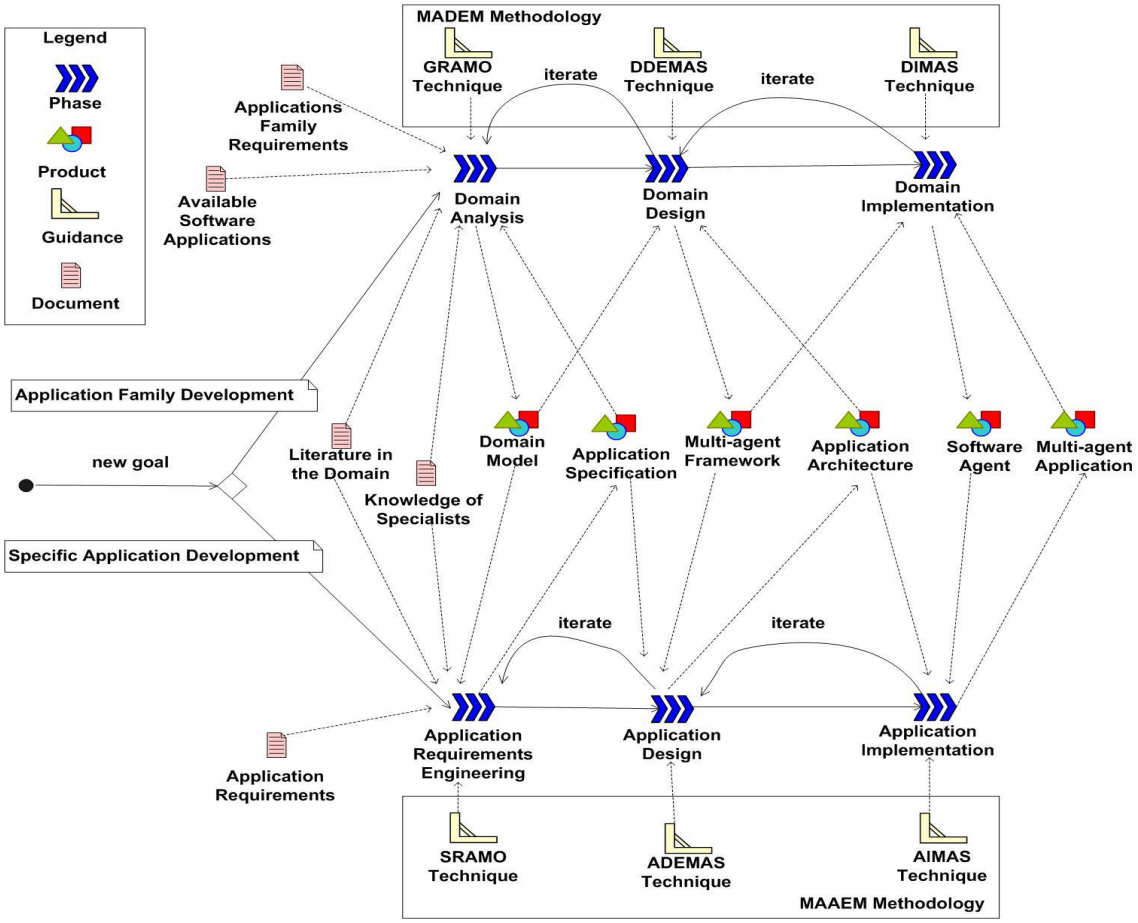


Figura 35 – O Ciclo de Vida do Processo MADAE-Pro

3.2.1 A Fase de Análise de Domínio

A fase de Análise de Domínio produz um Modelo de Domínio, que representa os requisitos comuns e específicos de uma família de aplicações multiagente. Essa fase é guiada pela técnica GRAMO.

A Figura 36 apresenta o subciclo de vida da fase de Análise de Domínio. O fluxo de realização das tarefas inicia-se com a modelagem de conceitos que utiliza como insumos conhecimento de especialistas no domínio, aplicações já desenvolvidas e a literatura disponível no domínio para definir o escopo inicial do domínio. Ao final desta tarefa, tem-se como produto o Modelo de Conceitos, uma rede semântica com os principais conceitos do domínio. A seguir é feita a modelagem de objetivos que utiliza como insumos os requisitos da família de aplicações para determinar os objetivos geral e específicos da família, as entidades

externas com as quais é necessário interagir e as responsabilidades necessárias para alcançar esses objetivos. Ao final desta tarefa é produzido o Modelo de Objetivos. O próximo passo é a realização da modelagem de papéis, onde cada responsabilidade definida no Modelo de Objetivos é associada a um papel que será encarregado delas. Nesse modelo, também é definido qual é o conhecimento que o papel necessitará para a realização de suas responsabilidades e as pré e pós condições que a realização dessas responsabilidades deverá atender. O produto dessa tarefa é um Modelo de Papéis. A seguir é realizada a modelagem de interações entre papéis. Essa tarefa representa a forma em os papéis se comunicam para realizarem suas responsabilidades a fim de atingirem os objetivos específicos. O produto dessa tarefa é um conjunto de Modelos de Interações entre Papéis, um para cada objetivo específico no nível inferior da hierarquia de objetivos no Modelo de Objetivos. Por último, é realizada a prototipagem da interface com o usuário, cujo objetivo é identificar as interações dos usuários com o sistema e construir um protótipo da interface com o usuário.

Para cada tarefa de modelagem descrita anteriormente, é realizada a subtarefa de modelagem de variabilidades com objetivo de identificar e representar as características comuns e variáveis da família de aplicações. Ao final desta fase, conforme o losango da Figura 36, passa-se para a fase de Projeto de Domínio ou inicia-se um novo ciclo para realizar um refinamento dos modelos a fim de adicionar novos requisitos ou alguma eventual correção.

Os papéis do processo “System Analyst” e “Domain Expert”, ilustrados na Figura 37, são os responsáveis pela realização das tarefas da fase de Análise de Domínio.

Essas tarefas são realizadas em conjunto pelos dois papéis do processo. O papel do processo “System Analyst” é o principal responsável pela realização e coordenação das tarefas, enquanto que “Domain Expert” o auxilia através do conhecimento que possui do domínio.

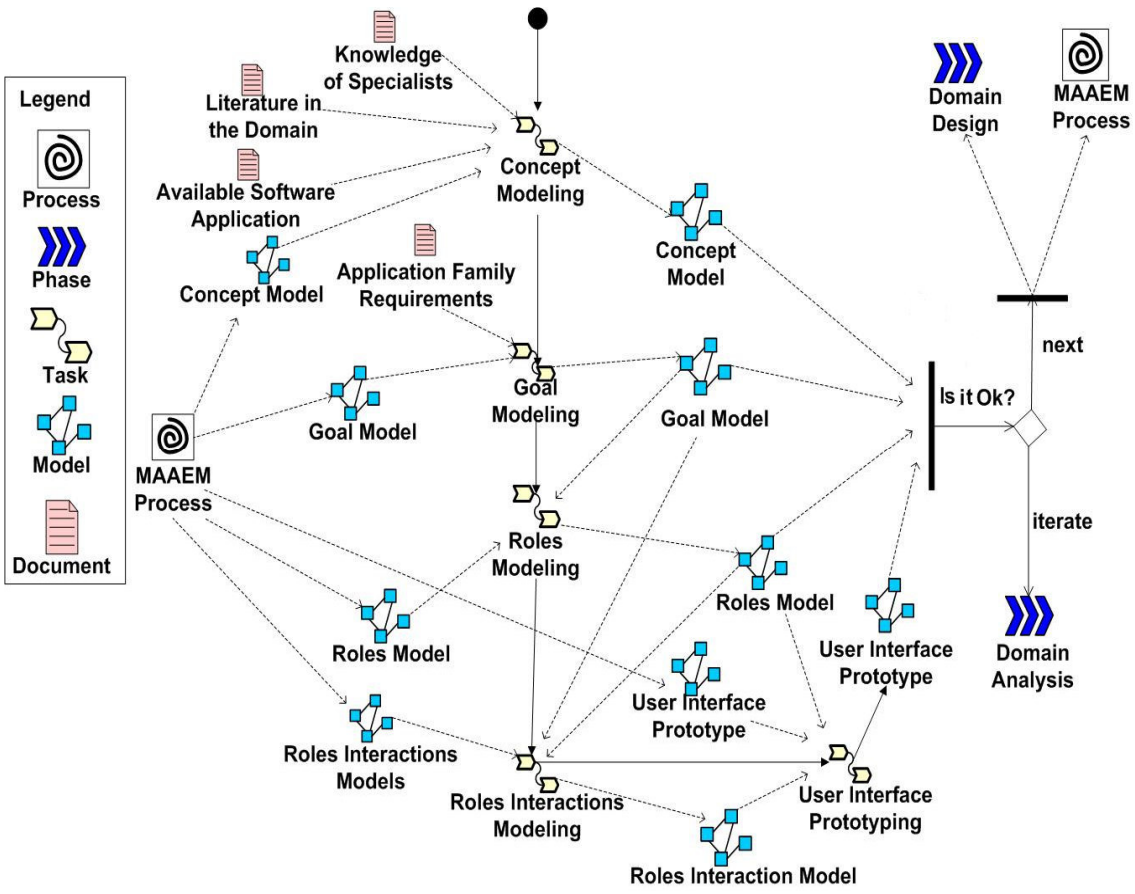


Figura 36. Subciclo de vida referente à fase de Análise de Domínio

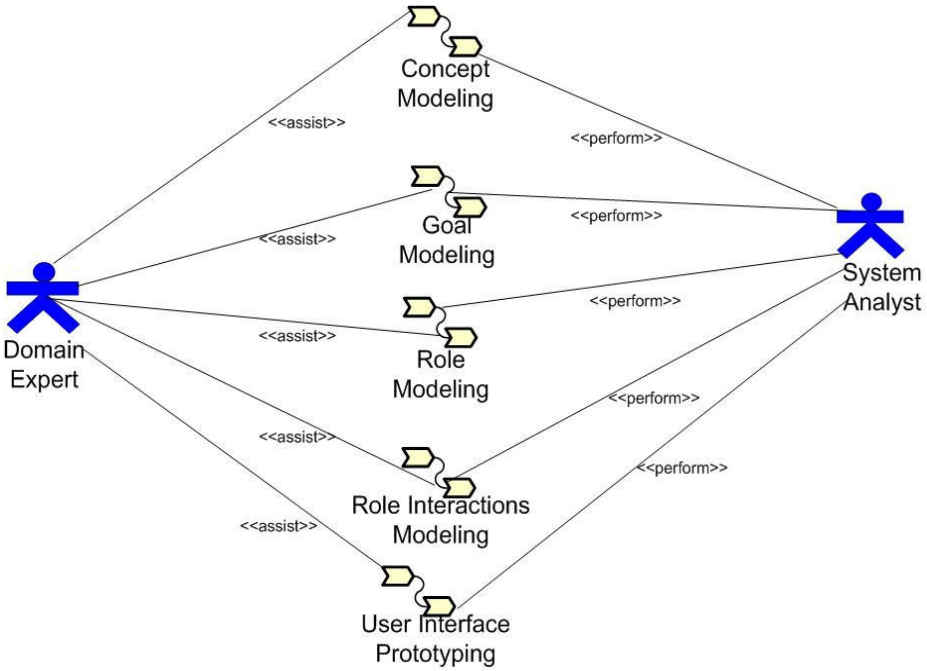


Figura 37 - Papéis do Processo da Fase de Análise de Domínio

3.2.2 A Fase de Projeto de Domínio

A fase de Projeto de Domínio visa definir uma solução de projeto aos requisitos de uma família de aplicações multiagente definidos na fase de Análise de Domínio. Nessa fase é realizado o Projeto Arquitetural, onde se constrói um modelo arquitetural da sociedade multiagente e o Projeto do Agente, que define a estrutura interna de cada agente da sociedade.

O subciclo de vida da fase de Projeto de Domínio, ilustrado na Figura 38, consiste na realização das tarefas de Projeto Arquitetural e Projeto do Agente, guiadas pela técnica DDEMAS. O subciclo inicia-se com a realização da subtarefa de modelagem do conhecimento da sociedade multiagente, onde é definido o conhecimento compartilhado pelos agentes da sociedade. Os insumos utilizados nessa tarefa são os *Modelos de Interações entre Papéis* da fase de Análise de Domínio, uma vez que nesse modelo é especificado o conhecimento trocado pelos papéis nas suas interações e possivelmente algum *Modelo do Conhecimento da Sociedade Multiagente* desenvolvidos pela metodologia MAAEM que seja do mesmo domínio de problema da família de aplicações que estiver sendo desenvolvida. Ao final dessa tarefa, obtém-se um *Modelo do Conhecimento da Sociedade Multiagente*, contendo o conhecimento compartilhado pelos agentes da sociedade. O próximo passo, é a modelagem da sociedade multiagente, onde são definidos os agentes da sociedade. Esse modelo é baseado no Modelo de Papéis da fase de Análise de Domínio, uma vez que os papéis são mapeados para agentes em função da semelhança de responsabilidades e de seus requisitos não-funcionais, e possivelmente de algum *Modelo da Sociedade Multiagente* desenvolvidos pela metodologia MAAEM. A seguir é realizada a tarefa de modelagem de interações entre agentes, onde são definidas as iterações que o agente deverá realizar para atingir o objetivo geral da família de aplicações. Esse modelo é baseado no Modelo da Sociedade Multiagente, a partir do qual são obtidos os agentes da sociedade. Ao final dessa tarefa obtem-se o Modelo de Interações entre Agentes, onde estarão definidas as interações dos agentes para atingir o objetivo da família de aplicações. A próxima tarefa é a modelagem dos mecanismos de cooperação e coordenação, onde é definido como os agentes irão cooperar e estar organizados a fim de atingir os objetivos da família de aplicações. Ao final dessa tarefa, obtém-se um Modelo de Cooperação e Coordenação da Sociedade Multiagente e finaliza-se a tarefa de

Projeto Arquitetural. A seguir é dado início a tarefa de Projeto do Agente, que é divida nas subtarefas de modelagem de conhecimento do agente e modelagem das ações do agente que produzem, respectivamente, um Modelo de Conhecimento do Agente e um Modelo de Ações do Agente para cada agente da sociedade. Na modelagem do conhecimento do agente, é definido o conhecimento particular de cada agente da sociedade e na modelagem das ações do agente são definidas as ações que permitiram aos agentes realizar as suas responsabilidades.

Os papéis do processo definidos para as tarefas da fase de Projeto de Domínio são “Domain Designer” e “System Analyst”, ilustrados na Figura 39. As tarefas desta fase são realizadas em conjunto pelos dois papéis do processo. O papel do processo “Domain Designer” é o principal responsável pela realização e coordenação das tarefas, enquanto que o “System Analyst” o auxilia através do conhecimento que possui sobre os requisitos da família.

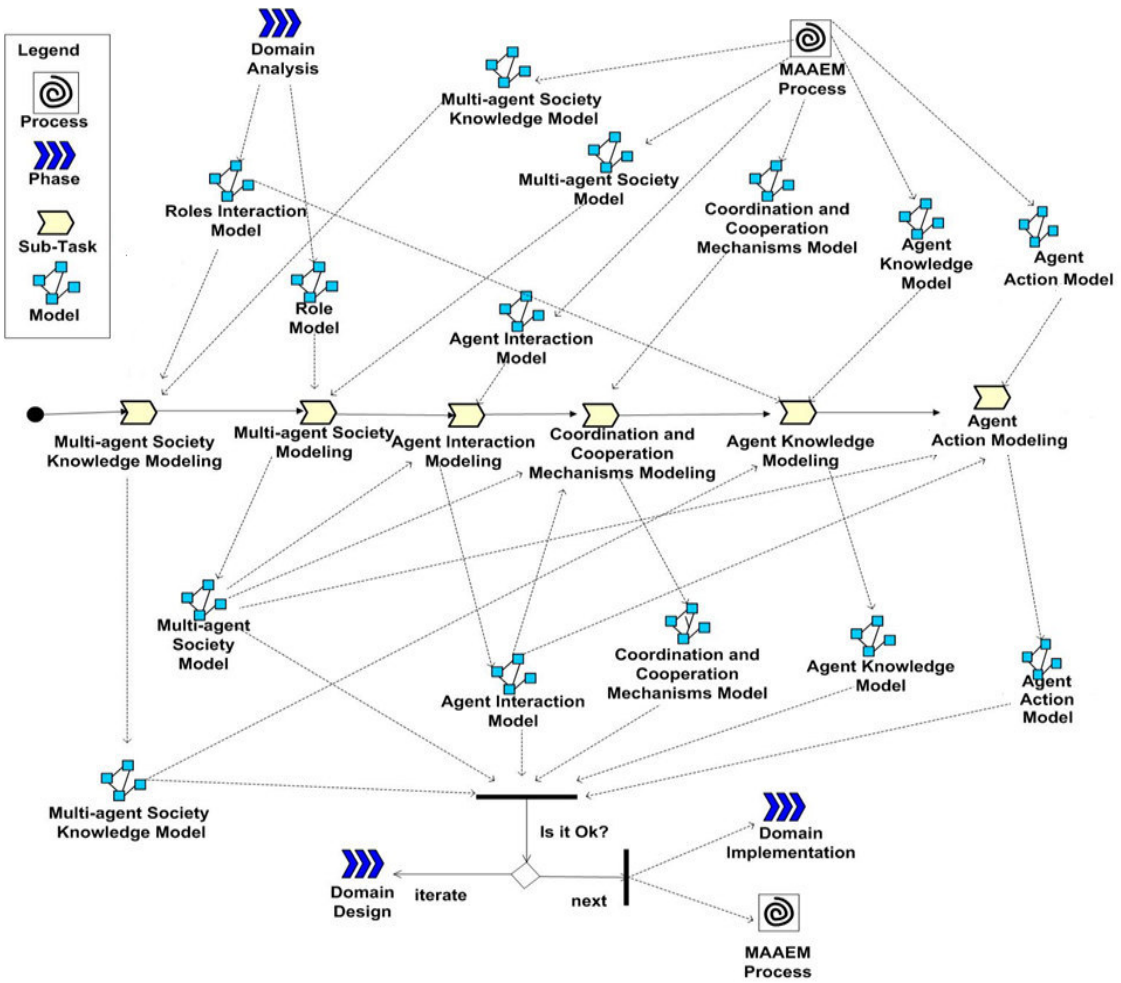


Figura 38. Subciclo de vida referente à fase de Projeto de Domínio

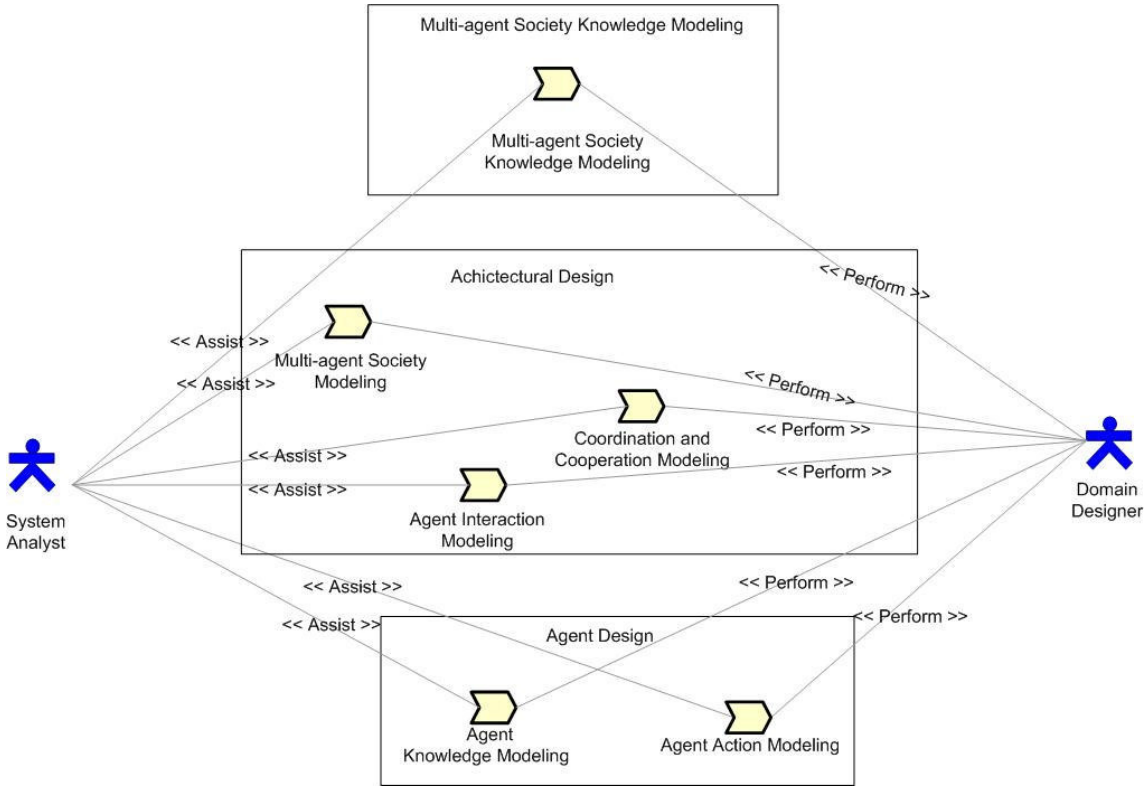


Figura 39 - Papéis do Processo da Fase de Projeto Domínio

3.2.3 A Fase de Implementação de Domínio

Na fase de Implementação de Domínio, os conceitos de projeto são mapeados para modelos específicos a uma plataforma de implementação de agentes. Para isso, tem-se adotado os frameworks JADE (BELLIFEMINE et al., 2003) e JESS (FRIEDMAN-HILL et al., 2003). Os modelos produzidos no subciclo de vida da fase de implementação de agentes são modelados de acordo com as diretrizes da técnica DIMAS.

O subciclo de vida da fase de Implementação de Domínio, ilustrado na Figura 40, inicia-se com a realização das tarefas de modelagem de comportamentos, onde as ações dos agentes e as responsabilidades são mapeadas do modelo de ações do agente para o modelo de comportamentos do agente seguindo os conceitos específicos de uma plataforma de implementação de agentes como JADE. A seguir é realizada modelagem de atos de comunicação, onde as interações identificadas no modelo de interações entre agentes são mapeadas para uma linguagem de comunicação específica como a FIPA-ACL (FIPA, 2008). Por fim,

a tarefa de implementação dos agentes é realizada, onde a partir dos modelos de Comportamentos do Agente e do Modelo de Atos de Comunicação os agentes são codificados. Parte dessa codificação pode ser gerado automaticamente a partir desses dois modelos.

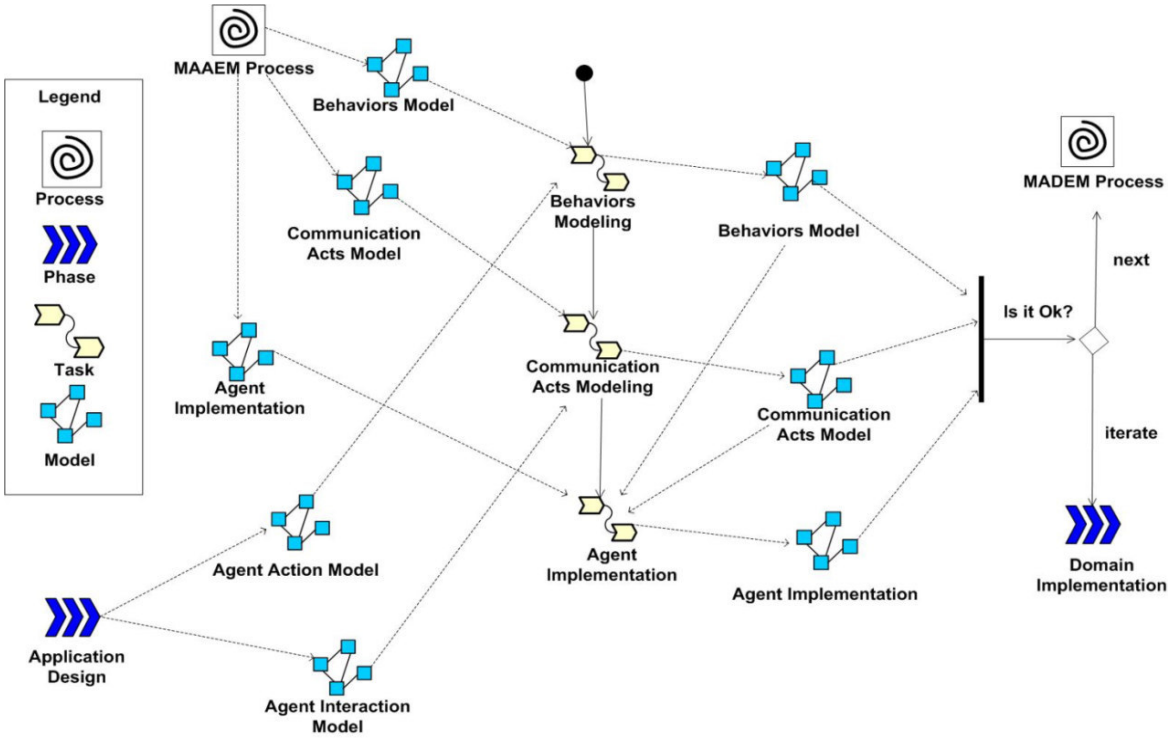


Figura 40. Subciclo de vida referente à fase de Implementação de Domínio

Os papel do processo “Programmer”, ilustrado na Figura 41, é o responsável pela realização das tarefas da fase de Implementação de Domínio e é supervisionado pelo papel “Domain Designer”.

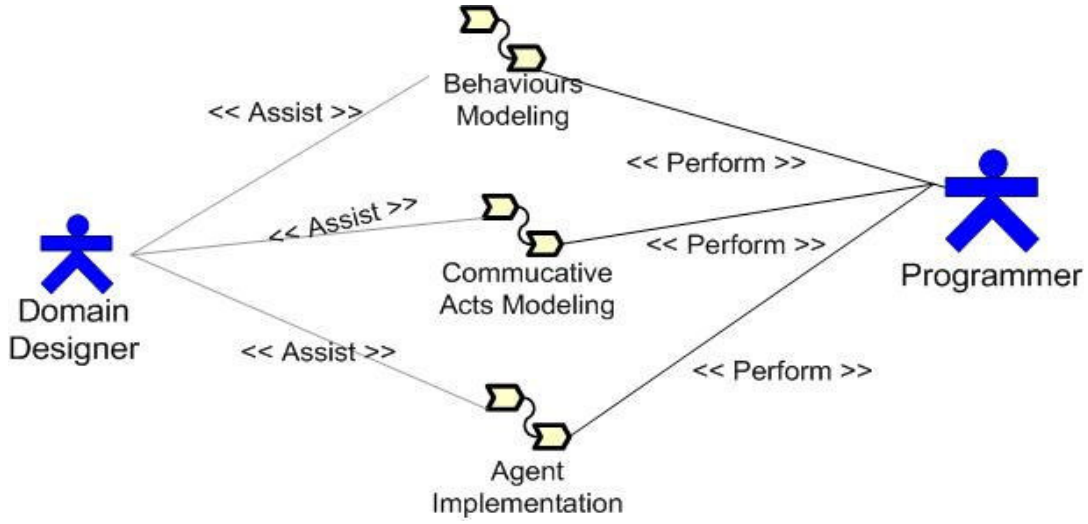


Figura 41 - Papéis do Processo da Fase de Implementação do Domínio

3.2.4 A Fase de Engenharia dos Requisitos da Aplicação

No subciclo de vida da fase de Engenharia dos Requisitos da Aplicação os requisitos de uma aplicação particular são identificados. Para isso, são reusados modelos desenvolvidos na Engenharia de Domínio, através da modelagem de conceitos, objetivos, papéis, interação entre papéis e prototipação da interface com o usuário seguindo as diretrizes da técnica SRAMO.

O subciclo da fase de Engenharia dos Requisitos da Aplicação, ilustrado na Figura 42, inicia-se com a Modelagem de Conceitos, na qual é realizada uma "tormenta de idéias" sobre os conceitos do domínio e estes são representados em uma rede semântica. A seguir é feita a Modelagem de Objetivos. Nessa tarefa, primeiro busca-se Modelos de Objetivos preexistentes na ONTORMAS, a fim de tornar essa tarefa mais produtiva. Se não houver modelos disponíveis para reúso, então se definem os objetivos geral e específicos da aplicação multiagente e as responsabilidades necessárias para alcançá-los através de um documento de requisitos da aplicação. Ao final desta tarefa o Modelo de Objetivos será produzido. O próximo passo é reusar Modelos de Papéis preexistentes que estejam associados aos objetivos específicos presentes no Modelo de Objetivos, através das responsabilidades que compartilham. Caso não haja Modelos de Papéis disponíveis para reúso, então se realiza a Modelagem de Papéis, a partir de responsabilidades identificadas no Modelo de Objetivos que são associadas a papéis (cada responsabilidade é associada a um papel), é definido ainda o conhecimento requerido para realização dessa responsabilidade, as entidades externas com as quais é necessário interagir e as suas pré e pós condições. Na tarefa seguinte, é realizada a modelagem de interações entre papéis. Nessa tarefa são reusados Modelos de Interações entre Papéis desenvolvidos no processo da Engenharia de Domínio que estejam associados as mesmas responsabilidades definidas no Modelo de Objetivos ou, no caso de não haver modelos disponíveis para reúso, esse modelo é construído identificando-se as necessidades de comunicação dos papéis definidos no Modelo de Papéis para alcançar os objetivos específicos da aplicação. Por último, é realizada a prototipação da interface com o usuário, onde é possível reusar protótipos de interface genéricos e especializar para a aplicação específica ou defini-la a partir do Modelo de Interações entre Papéis.

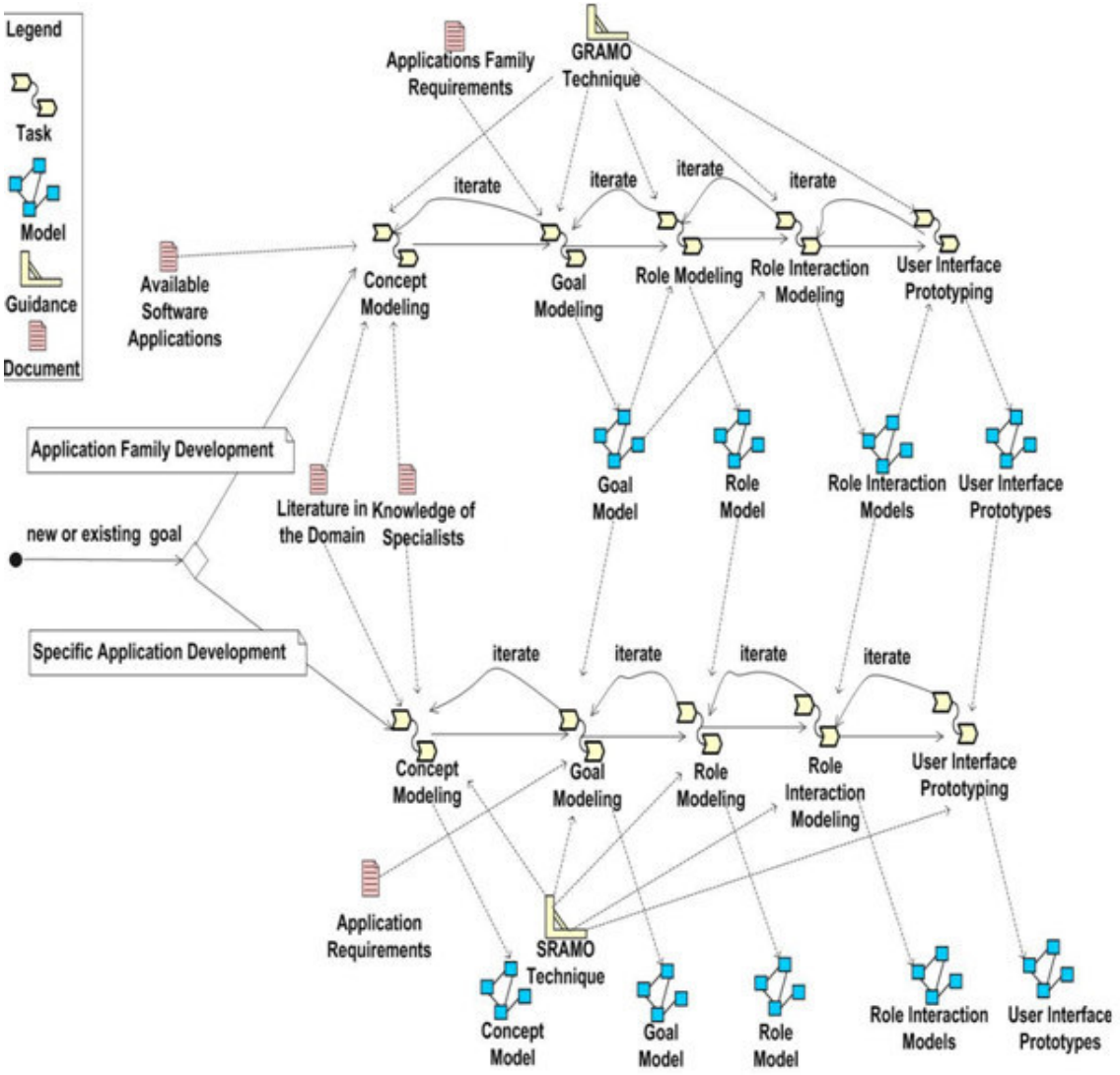


Figura 42. Subciclo de vida referente à fase de Engenharia dos Requisitos da Aplicação

3.2.5 A Fase de Projeto da Aplicação

Na fase de Projeto da Aplicação, o desenvolvedor poderá reutilizar soluções de projeto relacionadas a uma família de aplicações e adaptá-la aos requisitos específicos da aplicação em desenvolvimento. Esta fase consiste em realizar o Projeto Arquitetural da Sociedade Multiagente, incluindo seus mecanismos de coordenação e cooperação e o Projeto do Agente, onde é feito o projeto interno de cada agente da sociedade, modelando sua estrutura de conhecimento e seus comportamentos, seguindo as diretrizes da técnica ADEMAS.

O subciclo da fase de Projeto da Aplicação é ilustrado na Figura 46. Ele inicia-se com a realização da subtarefa de modelagem do conhecimento da sociedade multiagente, onde o conhecimento compartilhado pelos agentes da sociedade é representado em uma rede semântica, gerando o Modelo de Conhecimento da Sociedade Multiagente. A outra possibilidade é o reúso de Modelos do Conhecimento da Sociedade Multiagente desenvolvido no subprocesso da Engenharia de Domínio. O próximo passo é a modelagem da sociedade multiagente. Aqui são definidos os agentes da sociedade a partir dos papéis definidos no Modelo de Papéis da fase de Análise da Aplicação ou se já houver Modelos da Sociedade Multiagente desenvolvidos no subprocesso da Engenharia de Domínio da mesma família esse é reusado. A seguir, a modelagem de interações entre agentes é realizada, onde as interações dos agentes da sociedade são definidas. Aqui também podem ser reusados modelos preexistentes. A próxima tarefa é a modelagem dos mecanismos de cooperação e coordenação, onde é definido como os agentes irão cooperar e estar organizados na sociedade. Para isso, usam-se padrões já popularizados na comunidade, como o padrão de mercado, mestre-escravo, camadas, etc. Modelos pré-existentes também podem ser reusados nesse passo. Após a construção do Modelo dos Mecanismos de Cooperação e Coordenação finaliza-se a tarefa de projeto arquitetural e dá-se inicio a tarefa de projeto do agente.

No projeto do agente a modelagem de conhecimento é realizada, onde é definido o conhecimento particular de cada agente da sociedade. A seguir, a modelagem das ações do agente é realizada, onde são definidas as ações que o agente terá para realizar suas responsabilidades. Aqui também podem ser reusados Modelos de Conhecimento e Modelos de Ações desenvolvidos no subprocesso da Engenharia de Domínio.

Ao final da fase de Projeto da Aplicação, conforme o losango da Figura 43, passa-se para a fase de Projeto de Domínio ou inicia-se um novo ciclo para realizar um refinamento dos modelos a fim de adicionar novos requisitos ou alguma eventual correção.

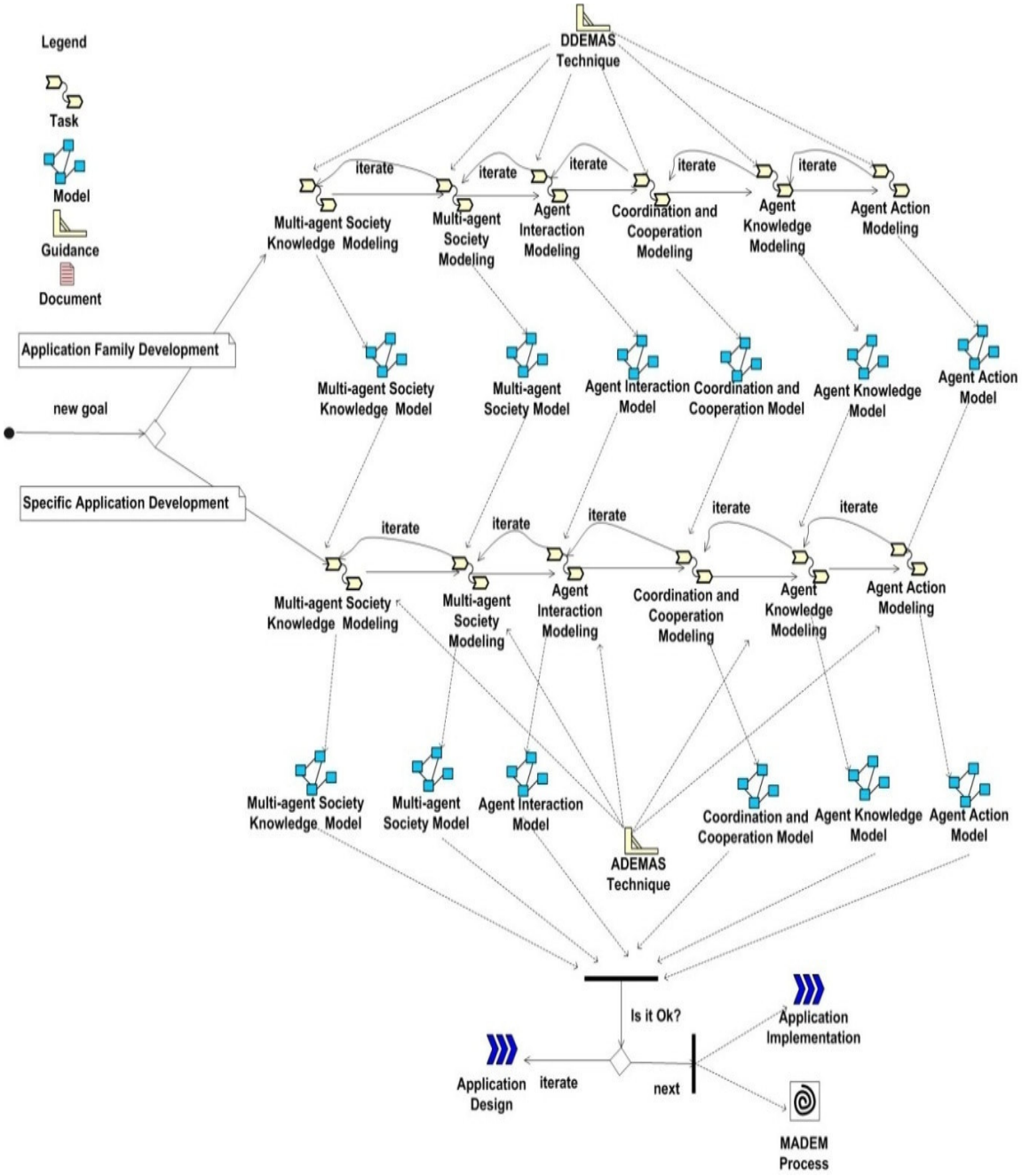


Figura 43. Subciclo de vida referente à fase de Projeto da Aplicação

3.2.6 A Fase de Implementação da Aplicação

No subciclo da fase de Implementação, ilustrado na Figura 44, são construídos o Modelo de Comportamentos e o Modelo de Atos de Comunicação. Com base nesses dois modelos é realizada a codificação da aplicação multiagente. No Modelo de Comportamentos são especificados os comportamentos que os

agentes terão e no Modelo de Comunicação as interações que eles precisam realizar para cumprir suas responsabilidades. Esses modelos são construídos de forma semelhante aos da fase de Implementação de Domínio e o papel do processo é o mesmo, a diferença é que aqui é uma aplicação específica é implementada a partir do reúso de agentes de software genéricos pertencentes a uma família de aplicações multiagente.

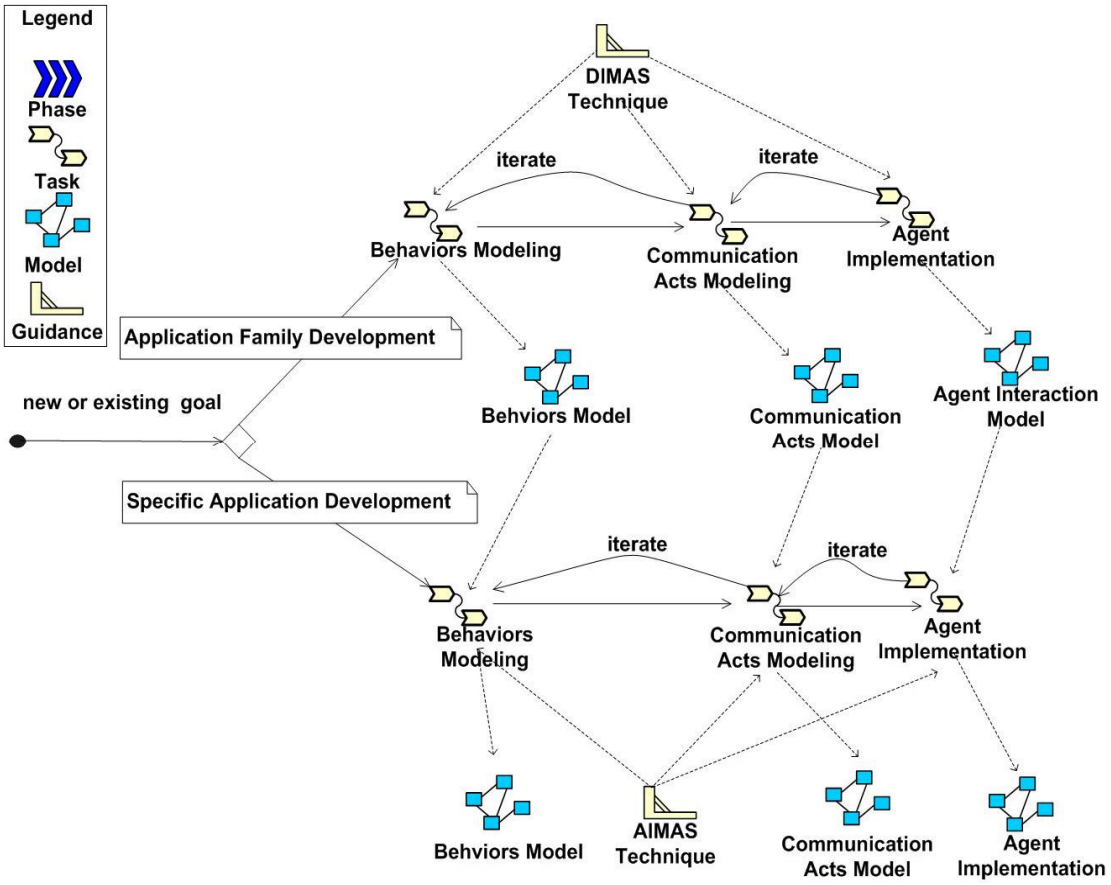


Figura 44. Subciclo de vida referente à fase de Implementação da Aplicação

3.3 As metodologias MADEM e MAAEM

A partir desta seção, apresenta-se detalhes de como os modelos são construídos seguindo as diretrizes das metodologias MADEM e MAAEM e de como é realizado o reúso. No entanto, antes de detalhar a construção dos modelos, apresenta-se a linguagem de modelagem MADAE-ML e a ferramenta ONTORMAS.

3.3.1 A Linguagem de Modelagem MADAE-ML

MADAE-ML ("Multi-agent Domain and Application Engineering Modeling Language") é uma linguagem visual baseada em ontologias para modelagem de sistemas multiagente. Essa linguagem é suportada pela ferramenta ONTORMAS e para sua utilização no processo MADAE-Pro. Na Figura 45, uma rede semântica é ilustrada com os principais conceitos da linguagem MADAE-ML utilizados para realizar a modelagem de sistemas multiagente de acordo com as metodologias MADAEM e MAAEM.

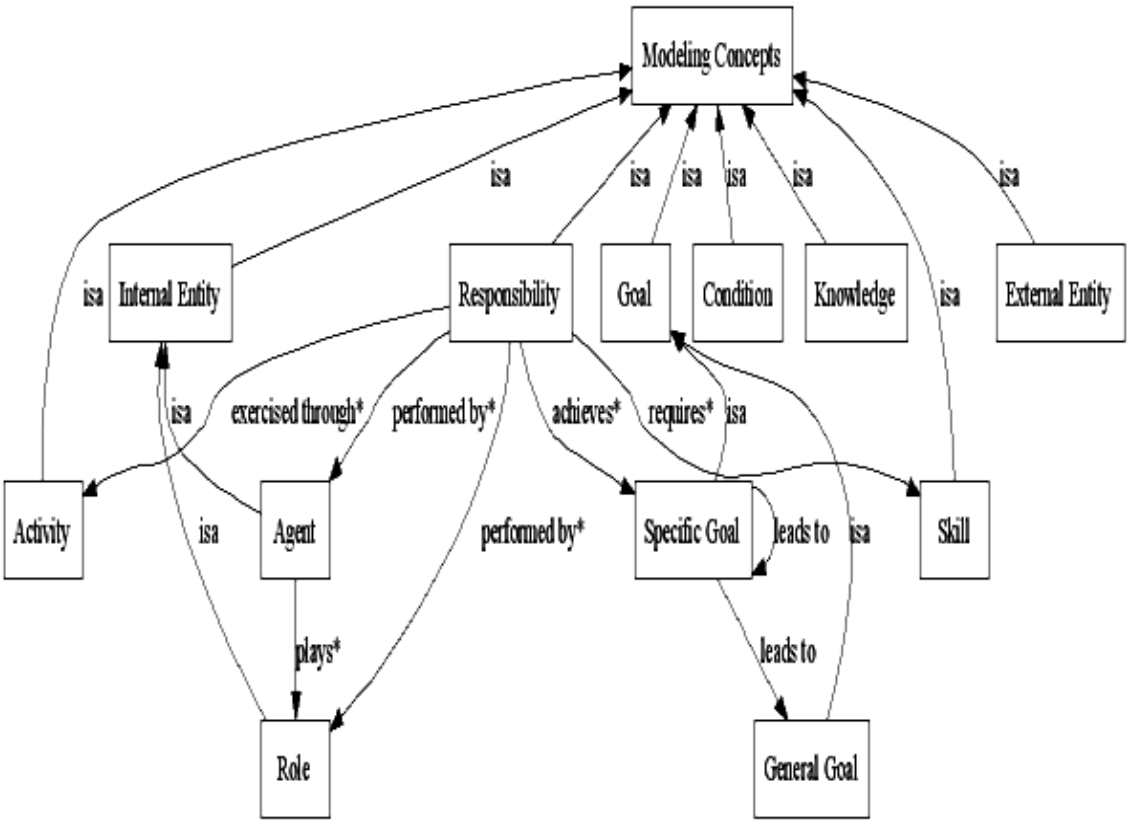


Figura 45 - Conceitos básicos de Modelagem das metodologias MADAEM e MAAEM

Na Tabela 7 os conceitos de modelagem da linguagem MADAE-ML são listados. Um determinado agrupamento desses conceitos constitui um modelo produzido no MADAE-Pro. Na coluna "*nome"* é definido o nome do conceito, na coluna "*descrição"* o conceito é definido, na coluna "*usado por"* são listados os modelos dos quais um determinado conceito pode fazer parte.

Tabela 7 - Conceitos da Linguagem MADAE-ML

Nome	Descrição	Usado por	Símbolo
Domain Concept	Representa qualquer conceito do domínio.	Concepts Model	
Responsibility	Representa uma responsabilidade a ser desempenhada por um Papel e/ou Agente	Goal Model Role Model Multi-agent Society Model	
General Goal	Representa o Objetivo Geral do Sistema Multiagente.	Goal Model	
Specific Goal	Representa um Objetivo Específico do Sistema Multiagente.	Goal Model	
External Entity	Representa uma entidade externa ao Sistema com a qual o Papel ou Agente interage.	Goal Model Role Model Role Interaction Model Multi-agent Society Model Agent Interaction Model Coordination and Cooperation Model	
Role	Representa um Papel, uma entidade que terá uma ou mais responsabilidades necessárias para alcançar um objetivo específico.	Role Model Role Interaction Model	
Knowledge	Representa conhecimento de domínio.	Role Model Multi-agent Society Model Multi-agent Society Knowledge Model Agent Knowledge Model Agent Action Model	
Condition	Representa uma condição a ser satisfeita, previamente ou após a execução de uma responsabilidade.	Role Model Multi-agent Society Model Agent Action Model	
Action	Representa uma ação a ser executada por um agente para o cumprimento de sua Responsabilidade.	Agent Action Model	
Skill	São habilidades ou conhecimentos necessários para a execução de alguma responsabilidade ou atividade específica.	Role Model Multi-agent Society Model	

Agent	É uma entidade de software com autonomia, responsável pela obtenção dos objetivos do Sistema através de uma ou mais tarefas.	Multi-agent Society Model Agent Interaction Model Agent Action Model	
Organizational and Process Pattern	Representa padrões de processos e organizações.	Pattern System	
Analysis Pattern	Representa um Padrão de Análise	Pattern System	
Architectural Pattern	Representa um Padrão Arquitetural	Pattern System	
JADE Behaviour	Representa um comportamento de um agente JADE.	Model of Behaviours, Communicative Acts Model	
Method JADE Behaviour	Representa um método de um comportamento de um agente JADE.	Model of Behaviours, Communicative Acts Model	
FIPA-ACL Message	Representa uma mensagem com uma determinada performativa da linguagem de comunicação entre agentes FIPA-ACL.	Communicative Acts Model	

3.3.2 A ONTORMAS

A ONTORMAS é uma ontologia que representa uma conceitualização das metodologias MADEM e MAAEM. Ela é uma extensão da ONTOMADEM (*Ontology for Multi-Agent Application Domain Engineering Methodology*) (GIRARDI e LEITE, 2008) que conceitualizava apenas a metodologia MADEM.

Na ONTORMAS foram expressas todas as diretrizes para o desenvolvimento de sistemas multiagente especificadas por essas metodologias. A ontologia foi desenvolvida no ambiente de desenvolvimento de sistemas baseados no conhecimento Protégé (PROTÉGÉ, 2008), como um sistema de frames, no qual cada frame representa um conceito e os slots representam as características e relacionamentos entre esses conceitos.

A ONTORMAS é formada por um conjunto de classes organizadas hierarquicamente, tendo como superclasses principais: “Variable Concepts”, “Modeling Concepts”, “Modeling Tasks” e “Modeling Products”. A superclasse “Variable Concepts” e suas correspondentes subclasses são utilizadas para especificar a variabilidade de uma família de aplicações multiagente (Figura 46). Isso é realizado através da definição de “pontos de variação” e de “variantes”. Um ponto de variação é a representação de um conceito sujeito a variação. Uma variante representa uma variação do conceito propriamente dito. Uma variante poderá representar um grupo de variações alternativas ou opcionais. A superclasse “Modeling Concepts” especifica os conceitos fundamentais de modelagem presentes nas metodologias MADEM e MAAEM. Esses conceitos são básicos para o desenvolvimento de sistemas multiagente e estão especificados em níveis de abstração indo do nível mais abstrato (Fase de Análise dos Requisitos da Aplicação) ao mais concreto (Fase de Implementação). Os conceitos de modelagem podem ter entre si a relação de sinonímia, generalização ou especialização. Na ONTORMAS, conceitos específicos de uma plataforma de implementação de agentes, como o JADE, também podem ser representados para possibilitar a geração automática de código.

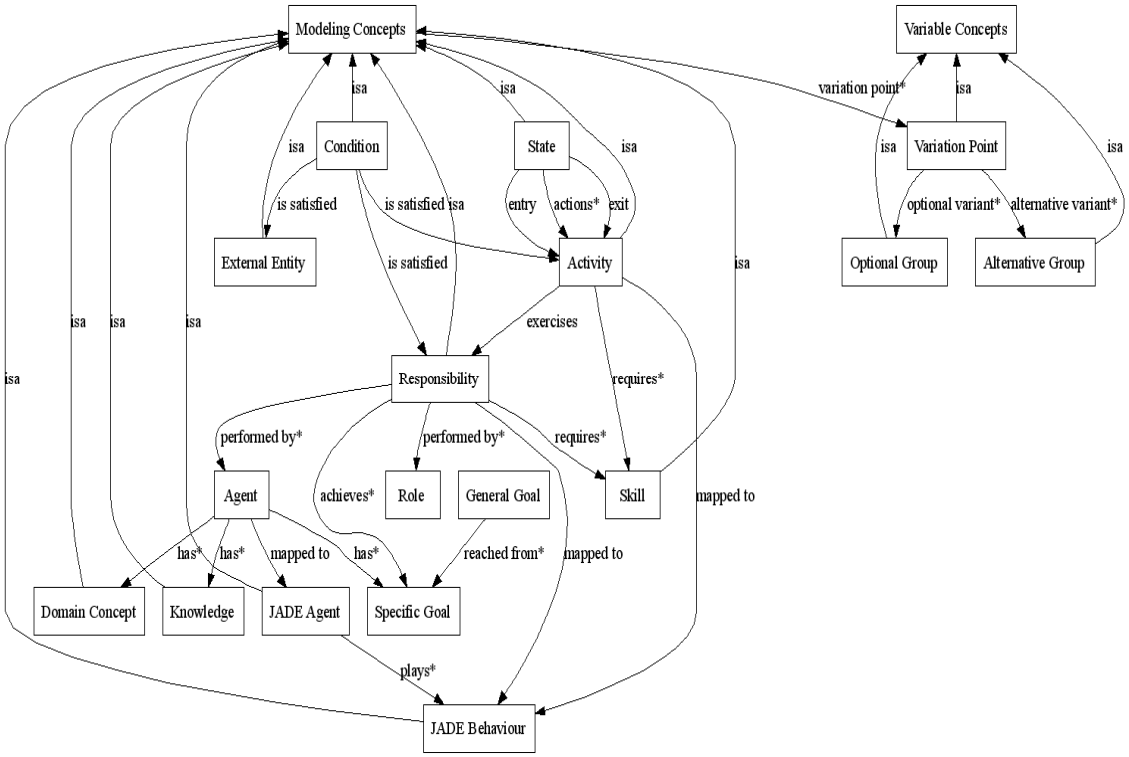


Figura 46 - Rede semântica com os principais conceitos de modelagem especificados pelas metodologias MADEM e MAAEM

Na superclasse “Modeling Tasks” e em suas correspondentes subclasses são definidas as tarefas de modelagem especificadas pelas metodologias MADEM e MAAEM.

A Figura 47 ilustra a representação das tarefas realizadas na fase de Análise de Domínio e na de Análise dos Requisitos da Aplicação. Essas tarefas são divididas em “Domain Engineering Tasks”, cujas subtarefas são relacionadas à metodologia MADEM e “Application Engineering Tasks”, relacionadas à metodologia MAAEM.

Na superclasse “Modeling Products” estão definidas as classes que representam os possíveis produtos de modelagem produzidos através das metodologias MADEM e MAAEM. Alguns destes produtos como “Domain Model” são compostos por subprodutos, isso é especificado através do relacionamento “subproducts”.

Na Figura 48, classes e instâncias do Modelo de Objetivo de ambas as metodologias são ilustradas.

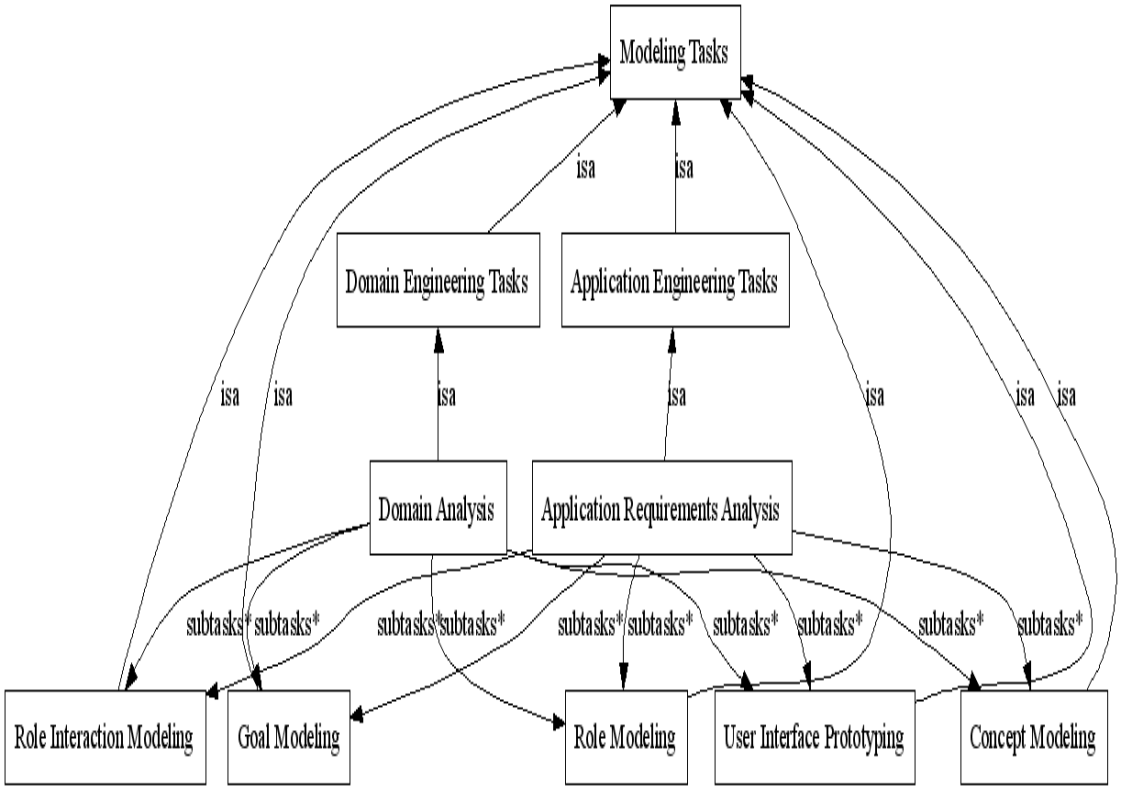


Figura 47 - Rede semântica das tarefas e subtarefas referentes à fase de Análise das metodologias MADEM e MAAEM

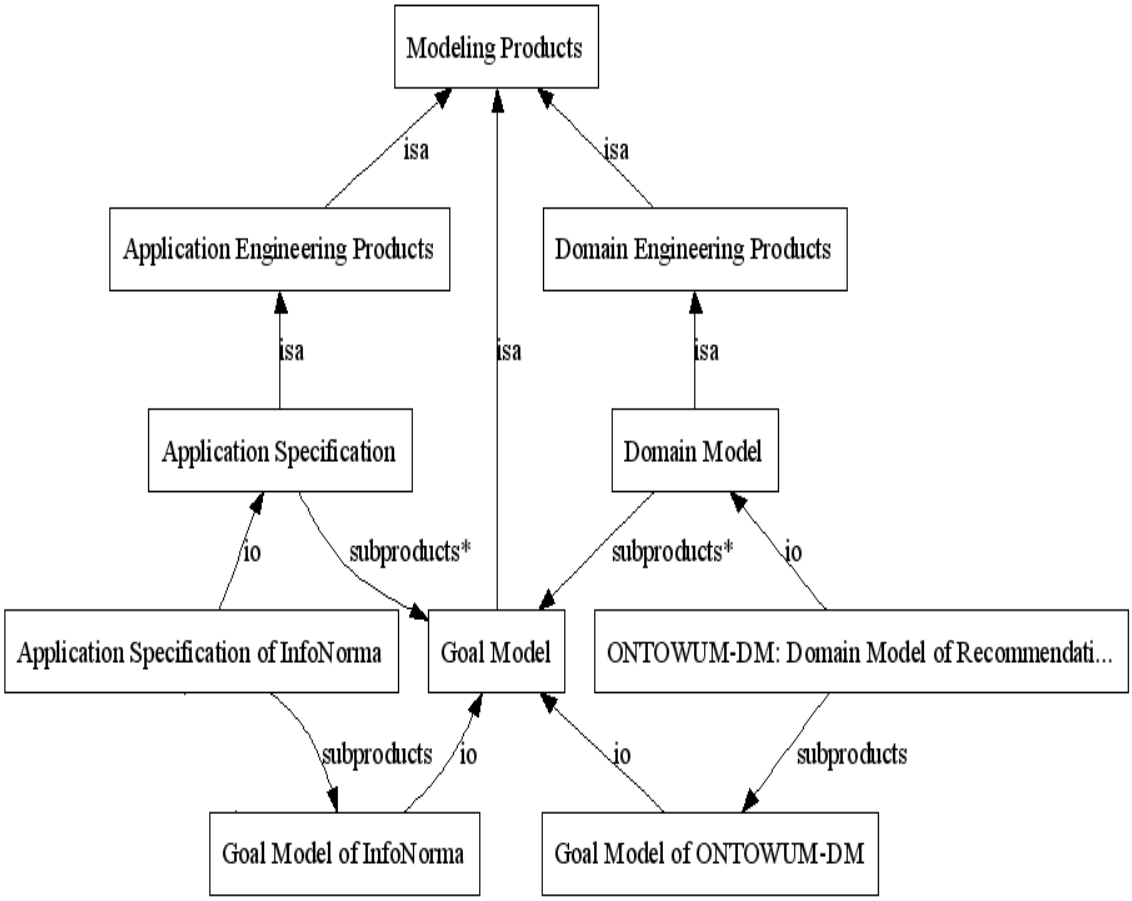


Figura 48 - Rede semântica exemplificando o relacionamento entre classes e instâncias de produtos presentes nas metodologias MADEM e MAAEM

A ONTORMAS é utilizada no MADAE-Pro para realizar três tarefas básicas. A primeira é para realizar a criação dos modelos definidos no processo. A segunda é como repositório desses artefatos e, por último, como ferramenta de seleção, adaptação e integração dos componentes.

A aplicação das metodologias MADEM e MAAEM e os produtos das mesmas são obtidos através da instanciação dos correspondentes conceitos na hierarquia de classes da ONTORMAS. A linguagem de modelagem utilizada em todos os produtos presentes na ONTORMAS é a MADAE-ML que foi apresentada na seção anterior.

A Figura 49 ilustra a criação do modelo de domínio ONTOWUM-DM e seus respectivos subprodutos. ONTOWUM-DM descreve os requisitos comuns e variáveis de uma família de aplicações multiagente para fornecer recomendações personalizadas através da Mineração de Uso. Na ONTORMAS, para criar um Modelo de Domínio é necessário a instanciação das classes que especificam suas

tarefas de modelagem (“Concept Modeling”, “Goal Modeling”, “Role Modeling”, “Role Interaction Modeling” e “User Interface Prototyping”) e das que especificam os seus subprodutos (“Concept Model”, “Goal Model”, “Role Model”, “Role Interaction Models” e “Prototype of the User Interface”). Uma vez que o Modelo de Domínio e seus respectivos subprodutos estejam instanciados, tornam-se parte da base de conhecimento da ONTORMAS, sendo então utilizados em consultas e inferências através de raciocínio lógico, facilitando o seu reúso.

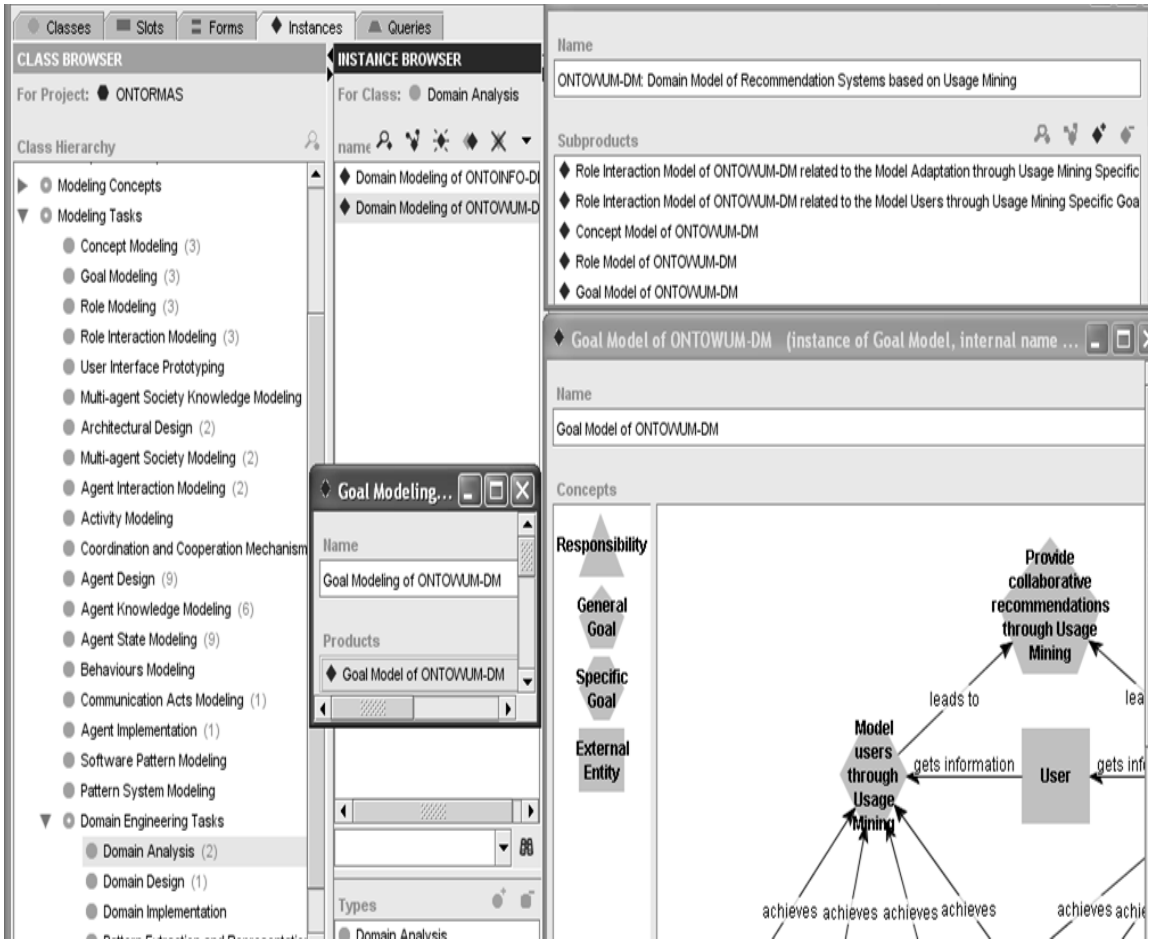


Figura 49 - Processo de Instanciação da ONTORMAS para criação de um Modelo de Domínio

3.3.3 A metodologia MADEM

A MADEM é uma metodologia para o desenvolvimento de artefatos de software reusáveis integrantes de uma família de aplicações multiagente. Uma família de aplicações é definida como um conjunto de aplicações de software similares que compartilham algumas características em comum e outras específicas.

Para a especificação do domínio de problema a ser resolvido, a MADEM orienta a realização da modelagem de objetivos, papéis, interações de entidades de uma organização e prototipação da interface com usuário. As entidades possuem conhecimento e o usam para exibir comportamento autônomo. Uma organização é composta de entidades com objetivos geral e específicos que estabelecem o que a organização pretende alcançar. A execução dos objetivos específicos permite alcançar o objetivo geral da organização. Por exemplo, um sistema de informação pode ter o objetivo “satisfazer as necessidades de informação de uma organização” e os objetivos específicos de “satisfazer as necessidades de informação dinâmica ou em longo prazo”. Os objetivos específicos são alcançados através da execução de responsabilidades que as entidades têm para a execução de papéis com certo grau de autonomia.

Os papéis têm destrezas sobre uma ou um conjunto de técnicas que suportam a execução de responsabilidades. Pré-condições e pós-condições podem necessitar serem satisfeitas para ou após a execução de uma responsabilidade. Conhecimento pode ser requerido e produzido na realização de uma responsabilidade. Por exemplo, uma entidade pode atuar como o papel “recuperador” com a responsabilidade de executar atividades para satisfazer as necessidades de informações dinâmicas de uma organização. Outra entidade pode atuar como o papel de “filtrador” com a responsabilidade de executar atividades para satisfazer as necessidades de informações em longo prazo das organizações.

Destrezas podem ser, por exemplo, as regras da organização que as entidades conhecem para acessar e estruturar suas fontes de informação.

Algumas vezes, as entidades têm que se comunicar com outras entidades internas ou externas para cooperar na execução de uma responsabilidade. Por exemplo, uma entidade atuando como o papel de “filtrador” pode necessitar interagir com um usuário (entidade externa) para observar seu comportamento e requisitar a inferência do seu perfil de interesses de informação.

Um resumo das tarefas realizadas, dos insumos requeridos e dos produtos obtidos em cada fase de desenvolvimento de uma aplicação multiagente seguindo as diretrizes da metodologia MADEM é descrito na Tabela 8.

Tabela 8 - Fases, Insumos, Tarefas e Produtos da metodologia MADEM

Fases	Insumos	Tarefas		Produtos		
Análise de Domínio	Conhecimento de Especialistas, Literatura no Domínio, Aplicações de Software Existentes.	Modelagem de Conceitos		Modelo de Conceitos		Modelo de Domínio
	Requisitos da família de aplicações	Modelagem de Objetivos		Modelo de Objetivos		
	Modelo de Objetivos	Modelagem de Papéis		Modelo de Papéis		
	Modelo de Objetivos	Modelagem de Interações entre Papéis		Modelo de Interações entre Papéis		
	Modelo de Papéis					
	Modelo de Interações entre Papéis	Prototipagem da Interface com o usuário		Protótipo da Interface com o usuário		
	Modelo de Papéis					
Projeto de Domínio	Modelo de Conceitos	Modelagem Arquitetural	Modelagem do Conhecimento da Sociedade Multiagente	Modelo do Conhecimento da Sociedade Multiagente	Modelo Arquitetural	Modelo do Framework Multiagente
	Modelo de Interações entre Papéis					
	Modelo de Papéis		Modelagem da Sociedade de Agentes	Modelo da Sociedade Multiagente		
	Modelo da Sociedade Multiagente		Modelagem das Interações entre Agentes	Modelo de Interações entre Agentes		
	Modelo de Objetivos (Requisitos Não-Funcionais)		Modelagem dos Mecanismos de Cooperação e Coordenação	Modelo dos Mecanismos de Cooperação e Coordenação		
	Modelo da Sociedade Multiagente					
	Modelo das Interações entre Agentes					
	Modelo de Interações entre Papéis	Projeto do Agente	Modelagem do Conhecimento do Agente	Modelos do Conhecimento do Agente	Modelos de Agente	
	Modelo do Conhecimento da Sociedade Multiagente					
	Modelo da Sociedade Multiagente		Modelagem das Ações dos Agentes	Modelos de Ações do Agente		
	Modelo de Interações entre Agentes					
Implementação de Domínio	Modelos de Ações	Modelagem de Comportamentos		Modelo de Comportamentos		Modelo de Implementação da Sociedade Multiagente
	Modelo de Interações entre Agentes	Modelagem de Atos de Comunicação		Modelo de Atos de Comunicação		
	Modelo de Comportamentos	Implementação dos Agentes		Agentes de Software Executáveis		
	Modelo de Atos de Comunicação					
Extração e Representação de Padrões	Experiências de Desenvolvimento e outras fontes disponíveis	Extração e Representação de Padrões		Padrões de Sistema e de Software		

Na seção 3.2 foi apresentado o ciclo de vida do processo MADAE-Pro utilizando a linguagem SPEM, com os modelos produzidos em cada fase. A seguir discute-se como estes modelos estão estruturados internamente (metamodelo), isto é, quais conceitos e relacionamentos são permitidos em cada modelo. Para ficar mais claro, o entendimento dos metamodelos serão dados exemplos de instâncias dos mesmos. Essas instâncias foram extraídas da ONTOSERS, uma família de aplicações para recomendação de informações na Web Semântica, e de INFOTRIB, uma aplicação para recomendação de informações para o domínio tributário.

Na Figura 50, um esquema da representação de metamodelos MADAE-Pro está representado, onde cada modelo definido pelo processo (ex. modelo de conceitos, modelo de objetivos) possui um metamodelo representado na linguagem SPEM com seu correspondente na ferramenta ONTORMAS. Nesse metamodelo, são definidos quais os conceitos permitidos naquele modelo (conjunto de classes na ONTORMAS), quais os relacionamentos (slots na ONTORMAS) e qual a multiplicidade desses relacionamentos (facetas na ONTORMAS). A partir da instanciação dos metamodelos são criados os modelos.

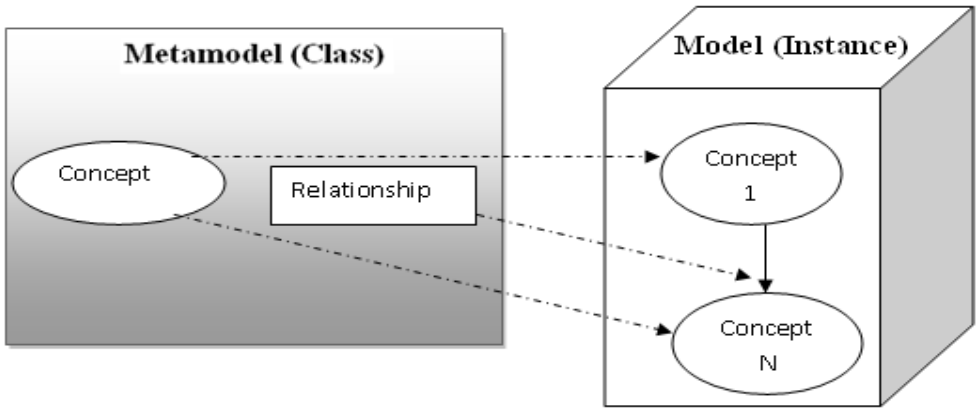


Figura 50 . Esquema de Metamodelos do MADAE-Pro

A fase de Análise de Domínio aborda a construção de um Modelo de Domínio, no qual são especificados os requisitos atuais e futuros de uma família de aplicações em um determinado domínio, considerando o conhecimento do domínio e as experiências de desenvolvimento extraídas a partir de especialistas e de aplicações já desenvolvidas nesse domínio. Na fase de Análise de Domínio, a técnica GRAMO recomenda a realização das seguintes tarefas: Modelagem de Conceitos, onde é realizada uma “tormenta de idéias” sobre os conceitos do domínio e seus relacionamentos; Modelagem de Objetivos, na qual são identificados e representados os objetivos de uma família de aplicações, as entidades externas com as quais coopera e as responsabilidades necessárias para alcançá-los; Modelagem de Papéis, onde são associadas as responsabilidades identificadas na tarefa de Modelagem de Objetivos aos papéis que serão encarregados delas; Modelagem de Interações entre Papéis, onde é identificado como as entidades externas e internas cooperam para alcançar um objetivo específico e a Prototipação da Interface com o

Usuário, cujo objetivo é identificar as interações dos usuários com o sistema e construir um protótipo da interface. O produto desta fase é um Modelo de Domínio.

Na Engenharia de Domínio, a modelagem de variabilidades especifica as características comuns e variáveis da família de aplicações. As características comuns são aquelas compartilhadas por todas as aplicações da família, enquanto que as características variáveis são as que se diferenciam em algumas aplicações da família. A Modelagem de Variabilidades no processo MADAE-Pro, é baseada na especificação de pontos de variação e variantes. Um ponto de variação representa um conceito sujeito à variação e uma variante representa grupos de variações alternativas ou opcionais desse conceito. Na Figura 51, é ilustrado o metamodelo de variabilidades, onde temos os conceitos já definidos anteriormente de “Variation Point”, “Alternative Group”, “Optional Group”, “Variant” e “Modeling Concepts”.

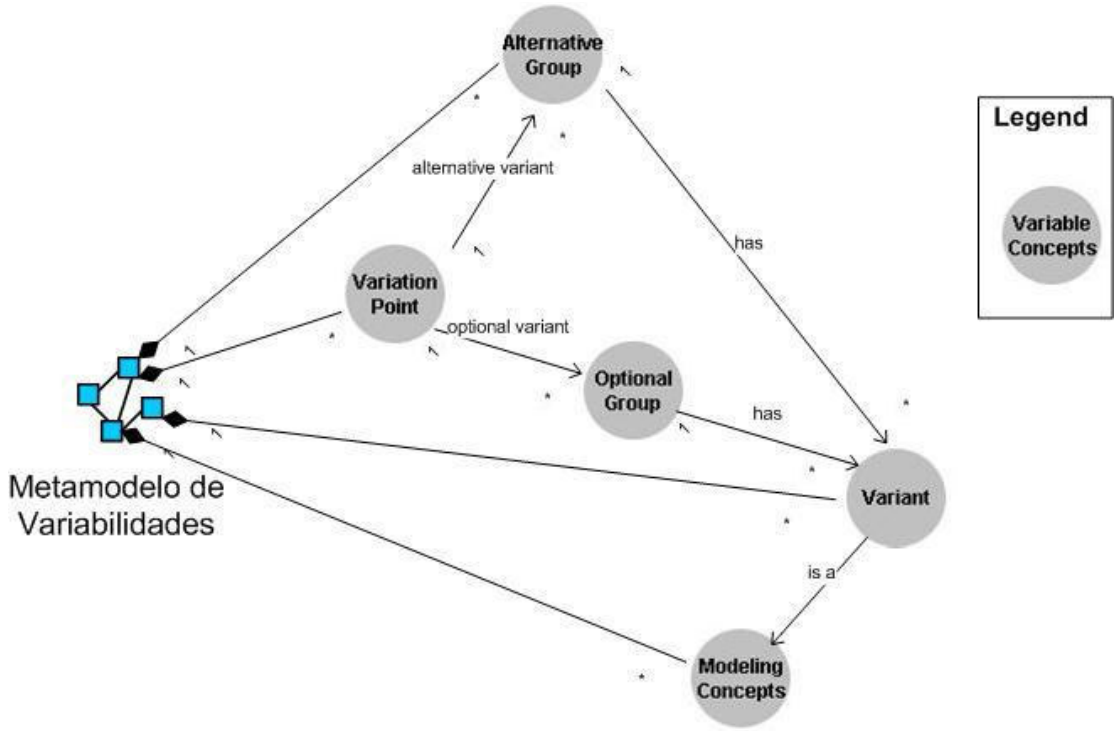


Figura 51 - Metamodelo de Variabilidades

O Modelo de Conceitos

O Modelo de Conceitos tem por objetivo reunir o máximo de informações relevantes ao domínio. Nele são representados os conceitos do domínio e suas relações em uma rede semântica, onde os conceitos representam os nodos e suas relações os arcos. Este modelo é construído com base em possíveis Modelos de

Conceitos preexistentes, no conhecimento de especialistas no domínio, em aplicações já desenvolvidas e na literatura disponível no domínio. Na Figura 52, é ilustrado graficamente o metamodelo de conceitos, onde estão definidos os conceitos e relacionamentos disponíveis. Uma instância do Modelo de Conceitos da família de sistemas ONTOSERS é ilustrado na Figura 53. Nesse modelo, conceitos-chave do domínio de sistemas de recomendação de informação estão relacionados.

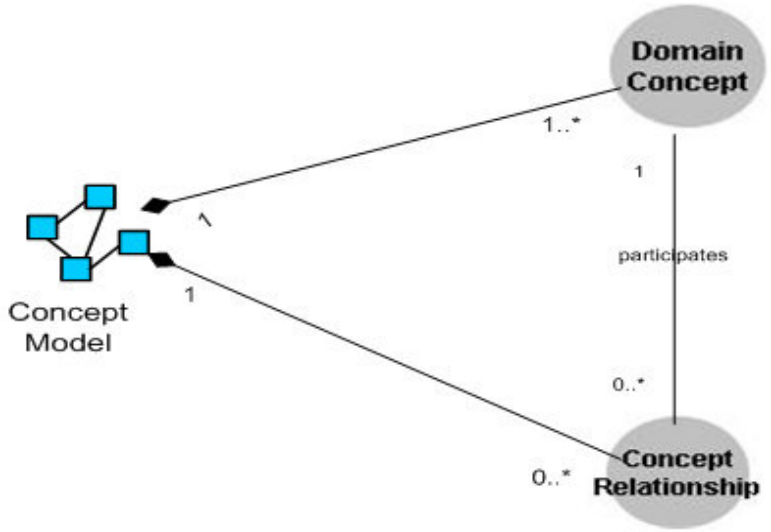


Figura 52 - Metamodelo de Conceitos

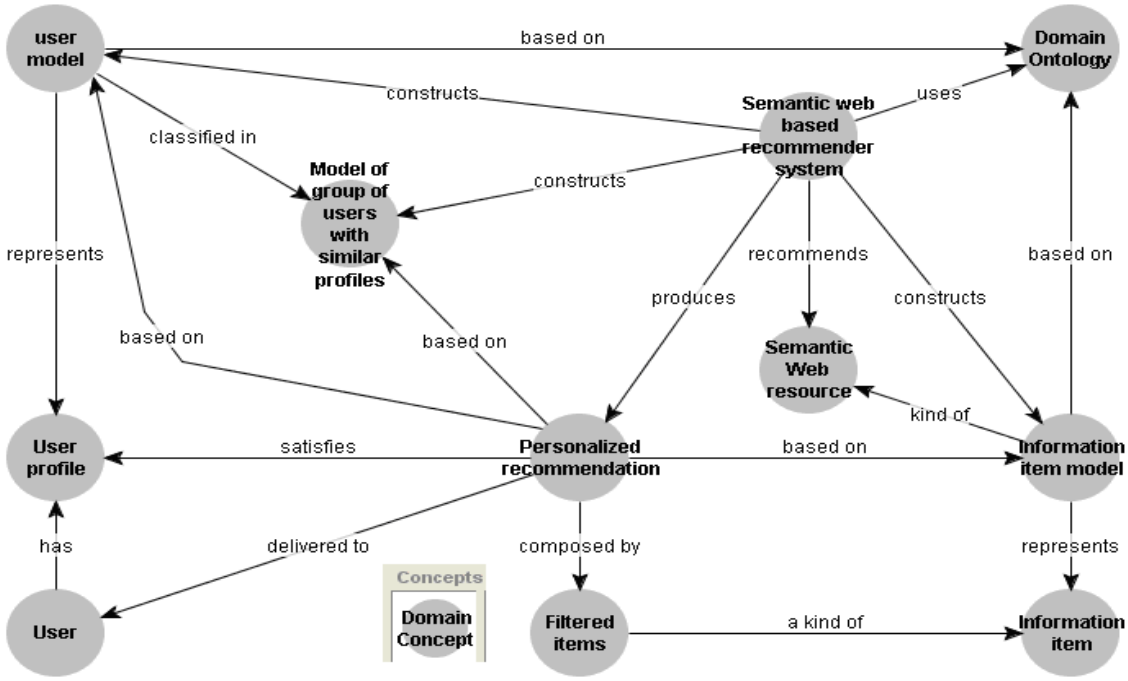


Figura 53 - Modelo de Conceitos ONTOSERS

O Modelo de Objetivos

Na tarefa de modelagem de objetivos são identificados os objetivos comuns e variáveis da família de aplicações multiagente, as entidades externas, e as responsabilidades necessárias para alcançá-los.

O Modelo de Objetivos é construído com base em possíveis modelos de objetivos preexistentes e a partir dos requisitos de uma família de aplicações multiagente. Nesta tarefa, a modelagem de variabilidade é realizada identificando-se pontos de variação em objetivos específicos relacionados com grupos de responsabilidades variantes.

A Figura 54 ilustra o metamodelo de objetivos, com os conceitos e os relacionamentos semânticos existentes entre esses conceitos. O conceito “General Goal”, representa o objetivo geral da família de aplicações multiagente; “Specific Goal”, representa um objetivo específico associado ao objetivo geral; “Responsibility”, representa uma responsabilidade necessária para alcançar um ou mais objetivos específicos; “External Entity”, representa uma entidade externa com a qual é necessário interagir para alcançar objetivos específicos.

O conceito “General Goal” está relacionado a dois ou mais conceitos “Specific Goal” através do relacionamento semântico “leads to”, isso significa que dois ou mais objetivos específicos conduzem ao objetivo geral.

O conceito “Responsibility” está associado a “Specific Goal” pelo relacionamento semântico “achieves”, isso denota que um objetivo específico será alcançado através da execução de uma ou mais responsabilidades.

O conceito “External Entity” está associado a “Specific Goal”, indicando que para alcance de um objetivo específico pode haver necessidade de interação com entidades externas.

O conceito “Specific Goal” está associado a ele mesmo através do relacionamento “leads to” para indicar os casos onde existe uma hierarquia de objetivos específicos.

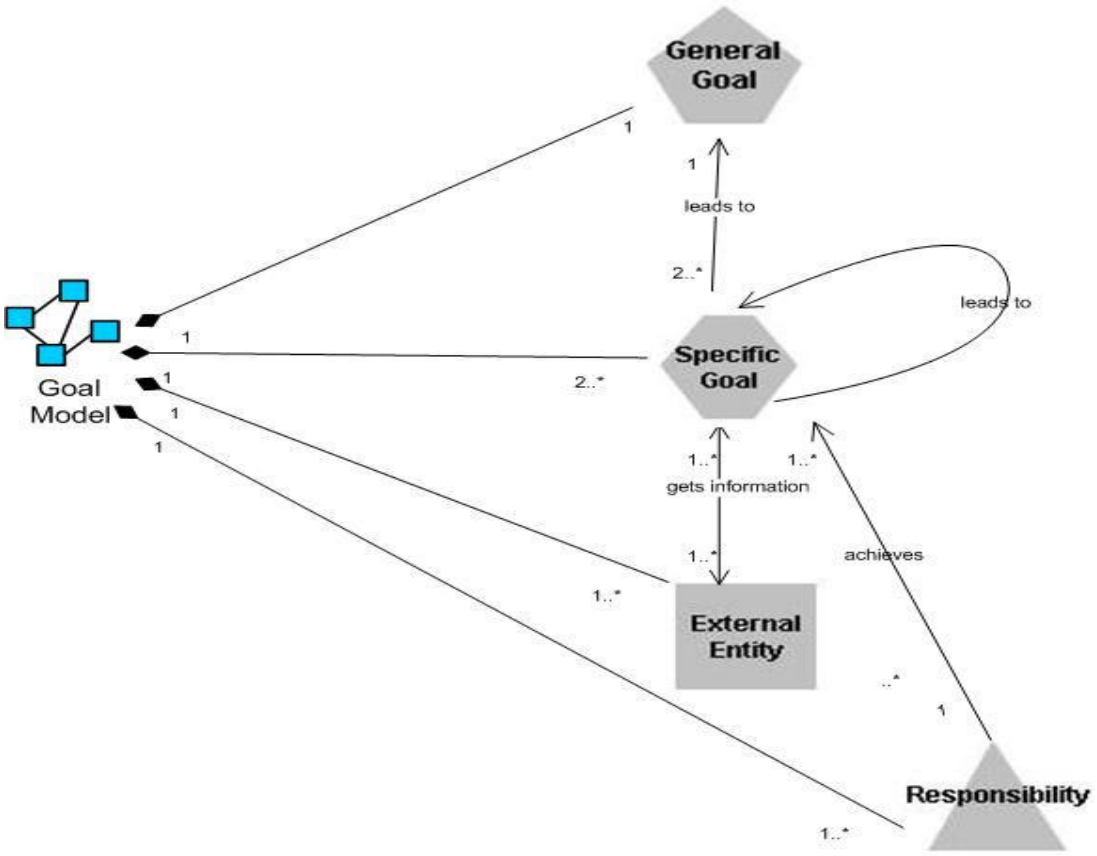


Figura 54 – Metamodelo de Objetivos

A Figura 55 ilustra o modelo de objetivo de ONTOSERS. O objetivo geral "Provide Recommendations using Semantic Web Technology" é alcançado através dos objetivos específicos "Model Users", "Filter Information" e "Deliver Recommendations". Para alcançar o objetivo específico "Filter Information", é necessário realizar a responsabilidade "Ontology Instance User Model Creation and Update", que também contribui para alcançar o objetivo específico "Model Users". Além dessas são necessárias as responsabilidades "Grouping of user models", "Information Items based on Ontology Instance Representation" e "Similarity Analysis". A responsabilidade "Grouping of Users Models" permite identificar grupos de usuários com interesses similares. O objetivo específico "Model Users" tem um ponto de variação com grupos de responsabilidades para aquisição do perfil de usuário, sendo possível escolher entre três variantes alternativas: "Implicit Profile Acquisition", "Explicit Profile Acquisition" ou ambas. A última responsabilidade, "Ontology Instance User Model Creation and Update" é necessária para todas as

aplicações da família. O objetivo específico “Filter Information” tem um ponto de variação que tem como variantes alternativas: a responsabilidade “Grouping of users models”, necessária quando se usa a filtragem colaborativa; e a responabilidade “Information Items based on Ontology Instance Representation” quando se está usando a Filtragem baseada no conteúdo. O objetivo específico “Deliver Recommendations” não possui pontos de variação, pois as responsabilidades “Similarity Analysis”, “Personalized Recommendations Production” and “Delivery of Personalized Recommendations” são necessárias para todas as aplicações da família.

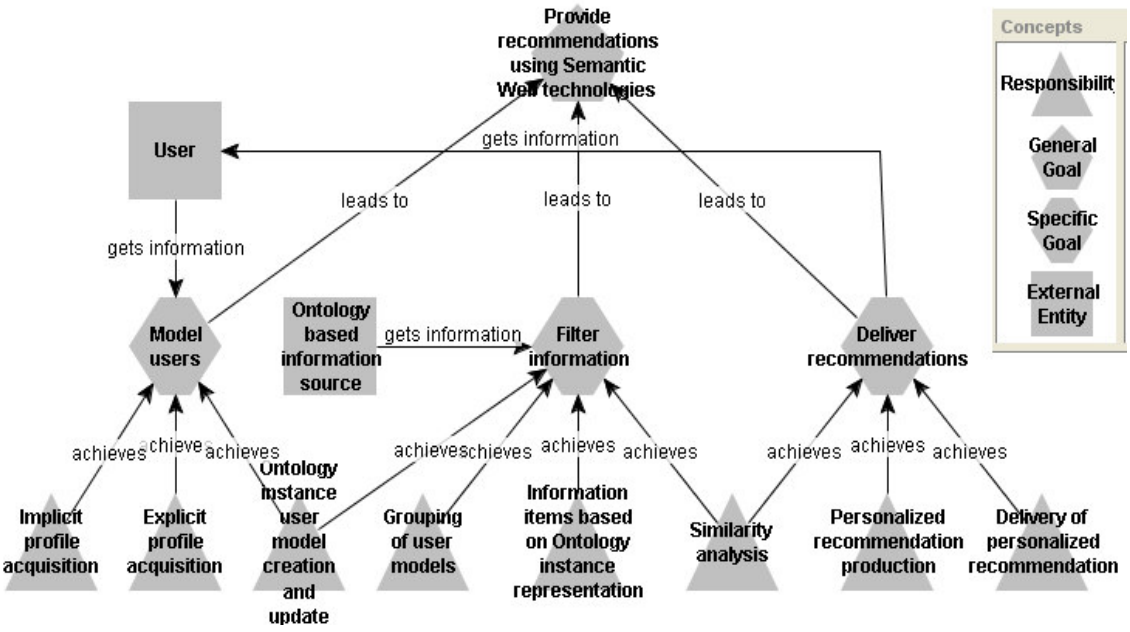


Figura 55 - Modelo de Objetivos ONTOSERS

A Figura 56 ilustra a modelo de variabilidade do objetivo específico “Model User” do modelo de objetivos do ONTOSERS. O objetivo específico “Model Users” tem um grupo de pontos de variação com grupos de responsabilidade para aquisição do perfil do usuário, sendo possível escolher entre três alternativas variantes: “Implicit Profile Acquisition”, “Explicit Profile Acquisition” ou ambas.

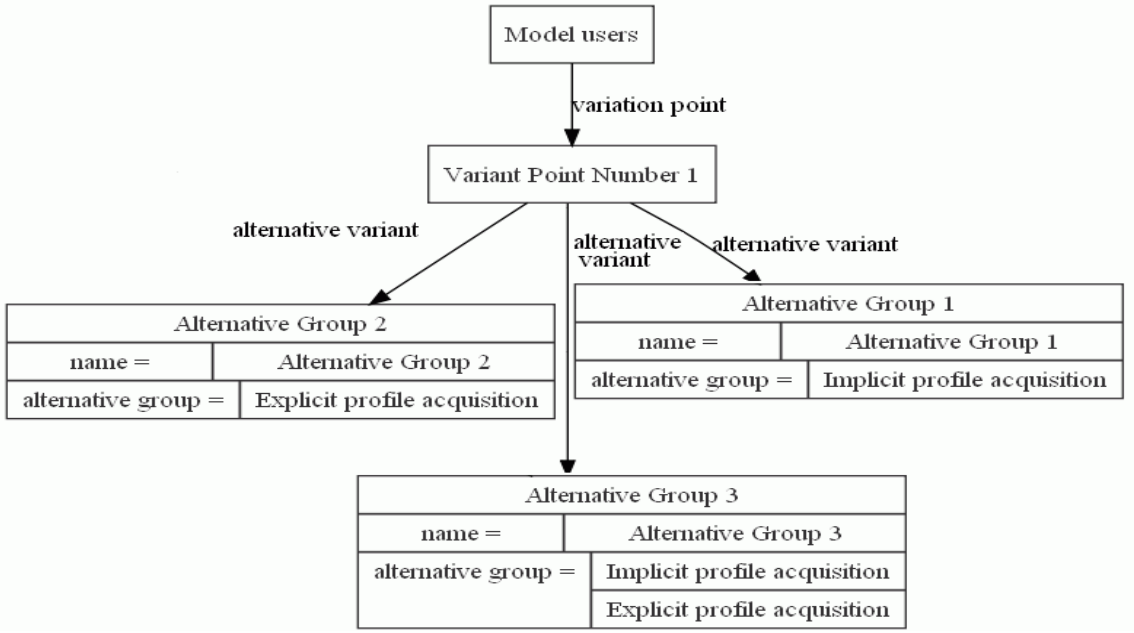


Figura 56 - Modelo de Variabilidade para o objetivo específico "Model users"

O Modelo de Papéis

A tarefa de modelagem de papéis associa as responsabilidades, comuns ou variantes, identificadas na tarefa de modelagem de papéis aos papéis que serão encarregados delas. As pré e pós condições que devem ser satisfeitas antes e depois da execução de uma responsabilidade são identificadas. O conhecimento requerido a partir de outras entidades (papéis ou entidades externas) para a realização das responsabilidades e o conhecimento produzido a partir da sua execução é identificado. Esta tarefa produz um conjunto de modelos de papéis, um para cada objetivo específico ou, tendo um ou mais pontos de variação, um modelo de papéis para cada variante, especificando papéis, responsabilidades, pré e pós condições, conhecimento e os relacionamentos entre estes conceitos.

A Figura 57 ilustra o metamodelo de papéis. Os conceitos definidos nesse metamodelo são: “Role”, “Responsibility”, “Knowledge”, “External Entity” e “Condition”. O conceito “Role” possui o relacionamento “in charge of” com o conceito “Responsibility”, significando que um papel é encarregado pela realização de uma responsabilidade. O conceito “Responsibility” possui os relacionamentos semânticos “used by” e “produces” com o conceito “Knowledge”, isso significa que para a/depois da realização de uma responsabilidade conhecimento pode ser requerido/produzido. O conceito “External Entity” tem o relacionamento semântico “satisfied by” e “has

knowledge" com os conceitos "Condition" e "Responsibility", respectivamente, isso significa que uma entidade externa é satisfeita através de uma condição e tem conhecimento que pode ser utilizado por uma responsabilidade. A Figura 58 ilustra o Modelo de Papéis do ONTOSERS.

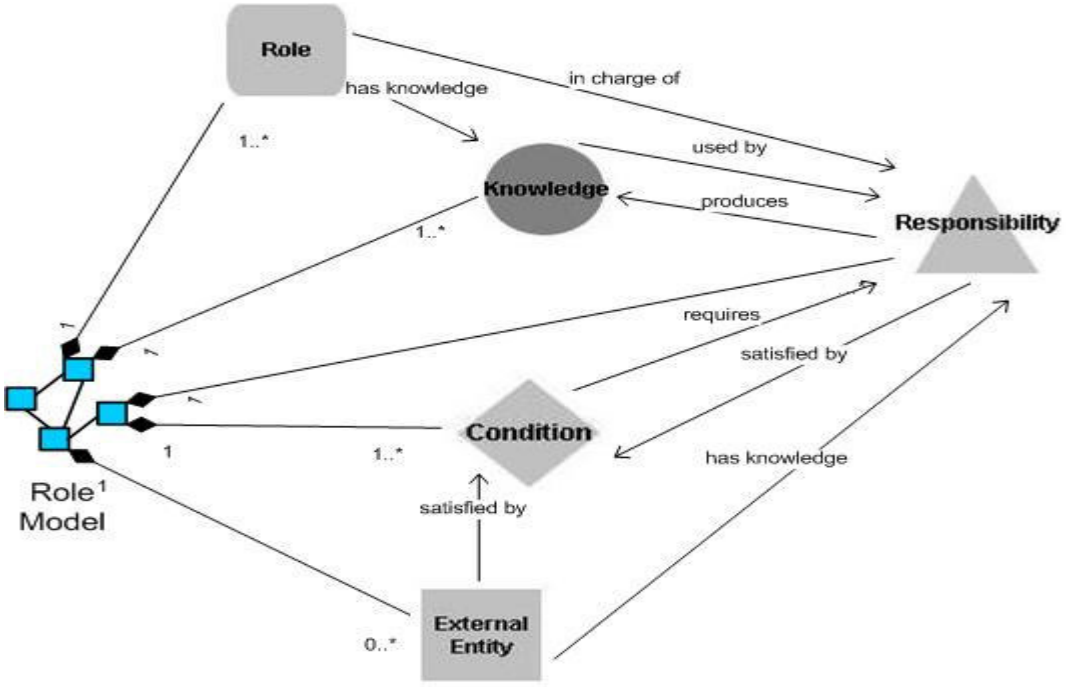


Figura 57 – Metamodelo de Papéis

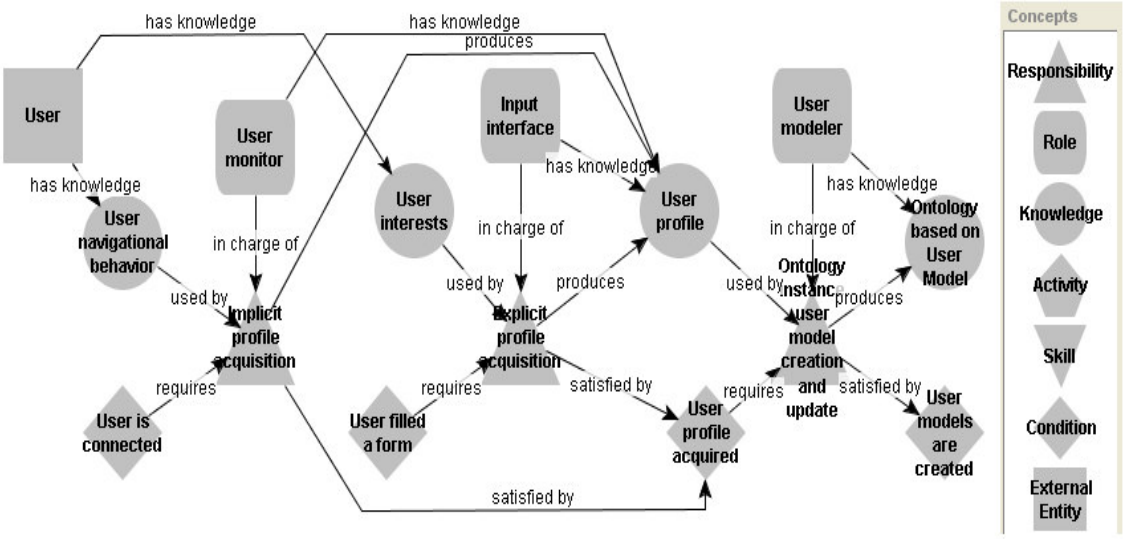


Figura 58 - Modelo de Papéis ONTOSERS

A Figura 59 mostra algumas variantes do modelo de papéis de ONTOSERS produzido através da tarefa de modelagem de variabilidades. Cada grupo de responsabilidade é uma variante alternativa que dá origem a um Modelo de Papéis. A figura mostra o relacionamento semântico entre o papel “User Monitor”, derivado da responsabilidade “Implicit profile acquisition” e dos grupos de variantes alternativas “Implicit User Modeling Group” e “Implicit-Explicit User Modeling Group”. Para esses grupos de responsabilidades alternativos foram construídos os Modelos de Papéis “Implicit User Modeling ONTOSERS Role Model” e “Implicit-Explicit User Modeling ONTOSERS Role Model”.

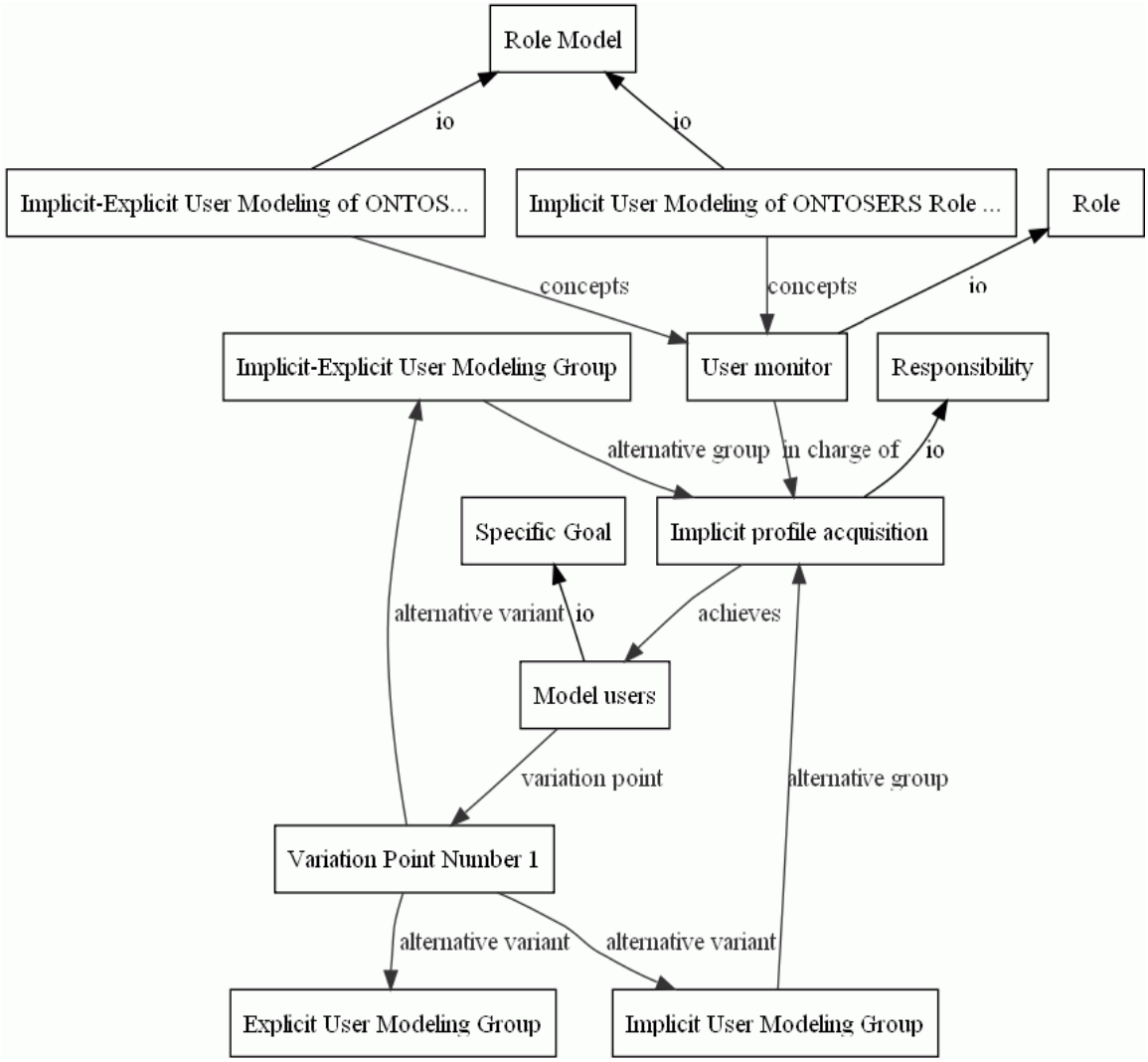


Figura 59 – Variabilidades do Modelo de Papéis

O Modelo de Interações entre Papéis

O Modelo de Interações entre Papéis representa a cooperação das entidades externas (usuários, outros sistemas, etc) e internas (papéis) para alcançar um objetivo específico. Esse modelo é construído reusando Modelos de Interações entre Papéis preexistentes e com base no Modelo de Objetivos e no Modelo de Papéis. O Modelo de Objetivos na associação entre um objetivo específico para cada Modelo de Interações entre Papéis e o Modelo de Papéis para associar papéis responsáveis do alcance desses objetivos específicos.

Na Figura 60 o metamodelo de papéis é ilustrado. Nesse metamodelo, estão disponíveis os conceitos de “Role”, “External Entity” e “Role Interaction”. O conceito de “Role Interaction” representa as interações entre dois conceitos “Role” ou entre um conceito “Role” e um conceito de “External Entity”, através do relacionamento semântico “participates”.

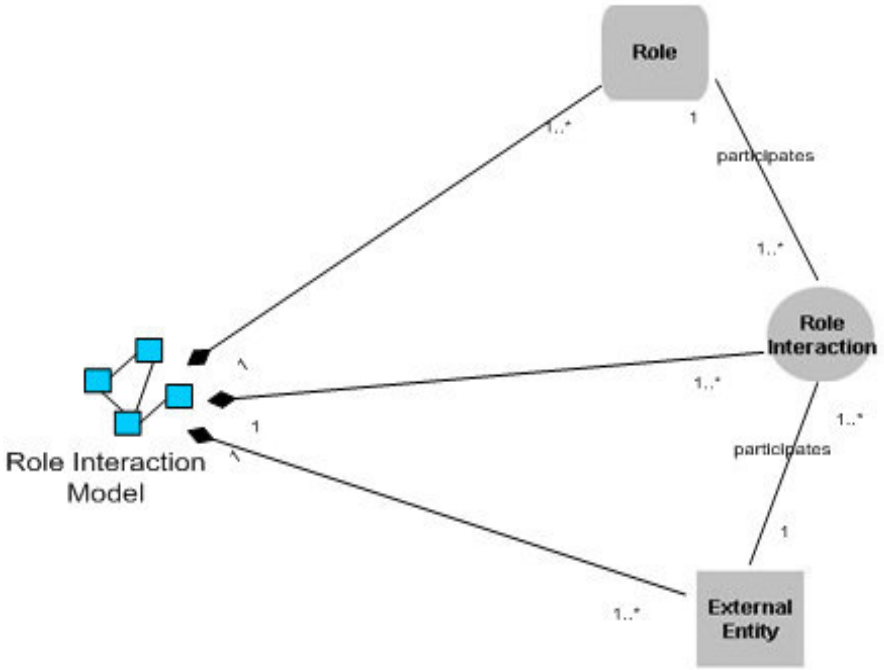


Figura 60 - Metamodelo de Interações entre Papéis

A Figura 61, apresenta parte do modelo de variabilidades do ONTOSERS, onde podemos observar que, para o objetivo específico “Model Users”, temos três possíveis grupos de interações alternativas, um para cada grupo de responsabilidades alternativas anteriormente definidas no modelo de objetivos. Para cada grupo de interações alternativas é construído um Modelo de Interações entre Papéis. A Figura 62 ilustra o Modelo de Interações entre Papéis da ONTOSERS. Nesse modelo, a modelagem de variabilidades é realizada criando-se pontos de variação nos objetivos específicos, cujas variantes são grupos de interações entre papéis.

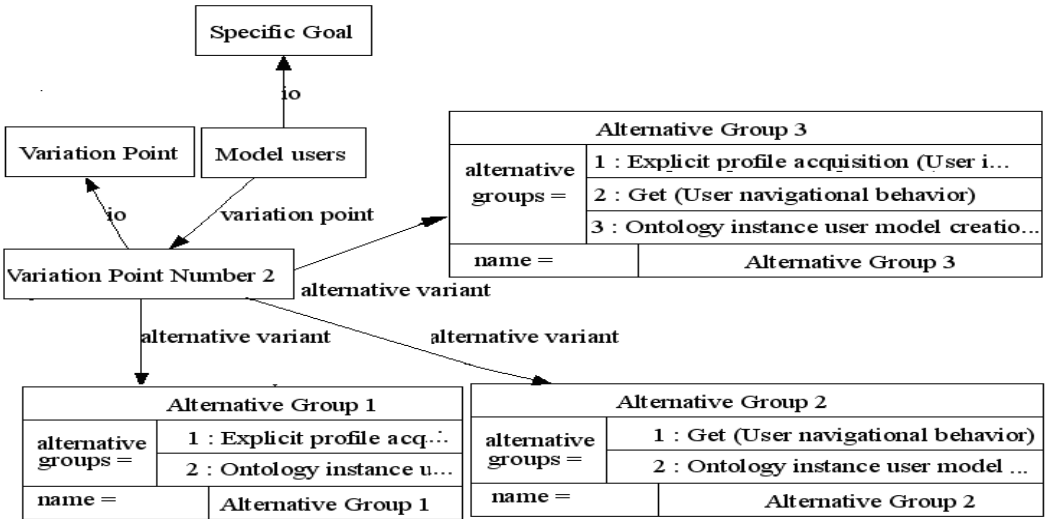


Figura 61 - Modelo de Variabilidades referente ao Objetivo Específico “Model Users”

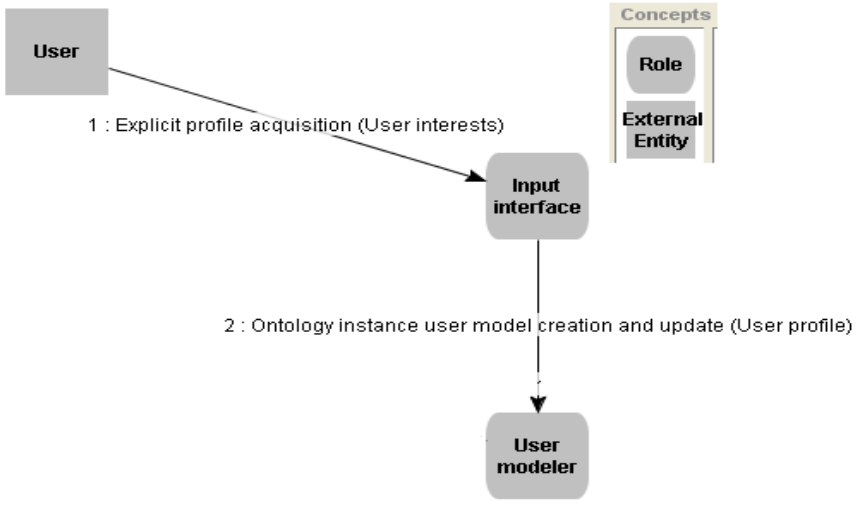


Figura 62 - Modelo de Interações entre Papéis ONTOSERS relacionado ao objetivo específico “Modelar Usuário” e a responsabilidade variante “Aquisição explícita de perfil”

O Protótipo de Interface com o Usuário

O Protótipo da Interface com o Usuário representa as interações dos usuários com o sistema. Esse protótipo é construído a partir do reúso de protótipos de interface com os usuários preexistentes e com base no Modelo de Papéis e no Modelo de Interações entre Papéis.

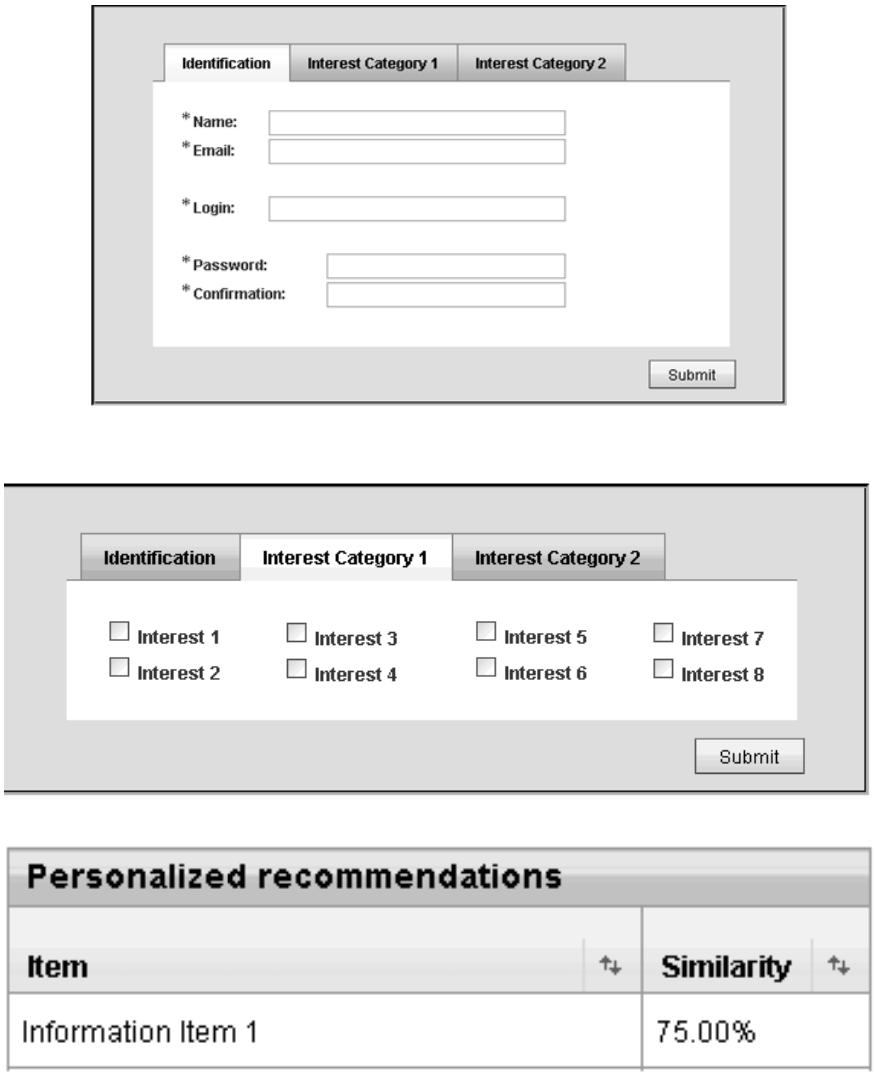


Figura 63 – Principais Telas do Protótipo de Interface Usuário da família ONTOSERS

A fase de Projeto de Domínio aborda o projeto arquitetural e detalhado de *frameworks* multiagente provendo uma solução aos requisitos de uma família de aplicações multiagente especificadas em um Modelo de Domínio. Nesta fase, a

técnica DDEMAS recomenda a realização das três seguintes subfases: Modelagem do Conhecimento da Sociedade Multiagente, a qual identifica e representa semanticamente os conceitos compartilhados por todos os agentes em sua comunicação; Projeto Arquitetural, que estabelece um modelo arquitetural da sociedade multiagente incluindo seus mecanismos de coordenação e cooperação; e o Projeto do Agente, que define o projeto interno de cada agente, modelando sua estrutura e comportamento.

O Modelo de Conhecimento da Sociedade Multiagente

O Modelo do Conhecimento da Sociedade Multiagente representa os conceitos que os agentes da sociedade precisam entender para comunicar entre si. Este modelo é construído a partir do reúso de Modelos do Conhecimento da Sociedade Multiagente desenvolvidos na Engenharia de Aplicações e com base no Modelo de Interações entre Papéis. O conhecimento trocados pelos papéis durante as suas interações são integrados e refinados neste modelo.

Na Figura 64 o metamodelo do conhecimento da sociedade multiagente é ilustrado. Nesse metamodelo, estão disponíveis os conceitos de “Knowledge”, que representa o conhecimento do domínio e de “Knowledge Relationship”, que representa os possíveis relacionamentos entre conceitos de conhecimento. “Part of”, “kind of” e “is a” são exemplos de “Knowledge Relationship”. Na Figura 65, está parte do Modelo do Conhecimento da Sociedade Multiagente de ONTOSERS.

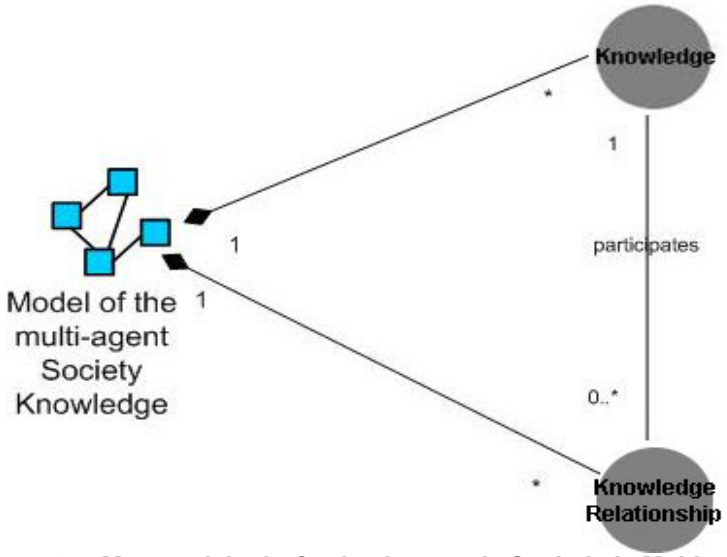


Figura 64 - Metamodelo do Conhecimento da Sociedade Multiagente

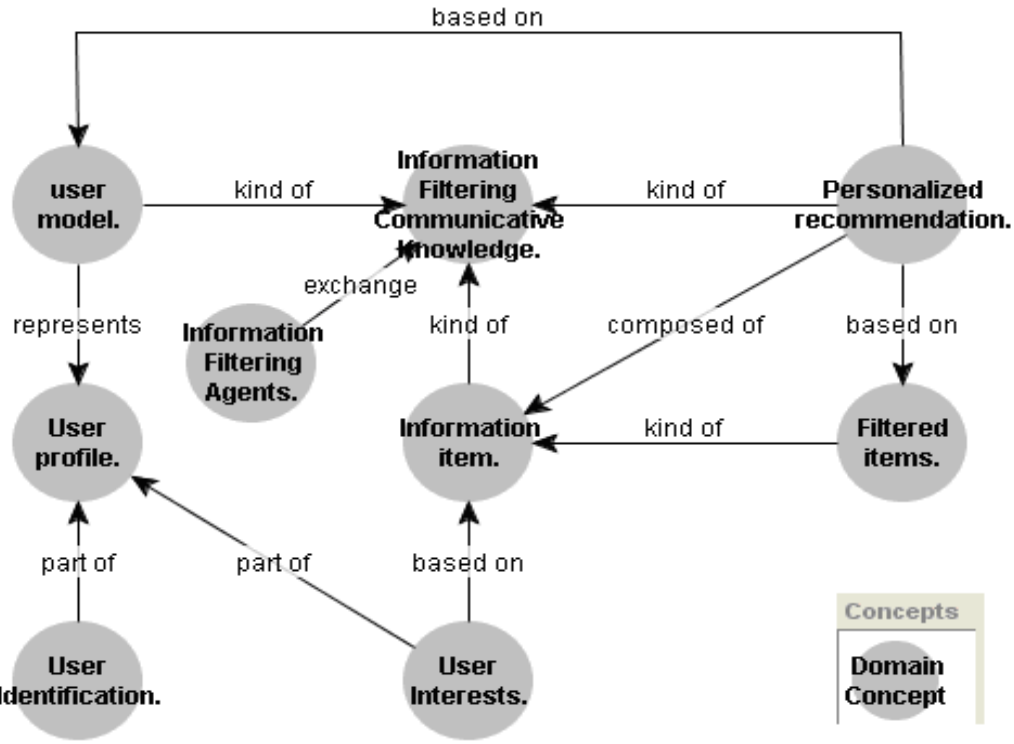


Figura 65 – Parte do Modelo do Conhecimento da Sociedade Multiagente do ONTOSERS

O Modelo da Sociedade Multiagente

O Modelo da Sociedade Multiagente representa os agentes da sociedade, juntamente com o conhecimento, as destrezas e as pré e pós-condições necessárias para realização das suas responsabilidades. Esse modelo é construído a partir do reúso de Modelos da Sociedade Multiagente preexistentes e com base no Modelo de Papéis da fase de Análise de Domínio. Cada papel presente no Modelo de Papéis é mapeado para agentes. Pré e pós-condições, responsabilidades, conhecimento e entidades externas também são obtidas a partir do Modelo de Papéis. Um agente pode representar um ou mais papéis de acordo com a afinidade entre suas responsabilidades, número de interações entre eles ou critério de coesão funcional.

Na Figura 66 o metamodelo da sociedade multiagente é ilustrado. Nesse metamodelo, os conceitos representados são: “Agent”, representando os agentes da sociedade; “Responsibility”, representando as responsabilidades dos agentes; “Knowledge”, representando o conhecimento que o agente usa e produz durante a realização de suas responsabilidades; “Condition”, representando as pré e pós-condições que devem ser satisfeitas para/depois da realização de uma

responsabilidade; “Skill”, que representa uma habilidade que o agente deve possuir para realizar uma determinada responsabilidade; “Activity”, representando as atividades que compõem uma responsabilidade; “External Entity”, representando as entidades externas com as quais os agentes interagem para realizar suas responsabilidades e os seus relacionamentos: “in charge of”, que indica que um agente será encarregado uma responsabilidade; “requires”; utilizado quando para realização de uma responsabilidade é necessário atender uma pré ou pós-condição; “required by”, representando que uma destreza é necessária para realização de uma determinada responsabilidade; “has knowledge”, que indica o conhecimento fornecido por entidades externas; “exercised through”, representando a realização de uma responsabilidade através de um conjunto de atividades; “used by” e “produces”, representando o conhecimento requerido e produzido através da realização de uma atividade. A Figura 66 ilustra o modelo da sociedade multiagente do ONTOSERS para o agente Filtrador. Aqui a modelagem de variabilidades é realizada adicionando-se um ponto de variação na responsabilidade cujas variantes são as possíveis destrezas.

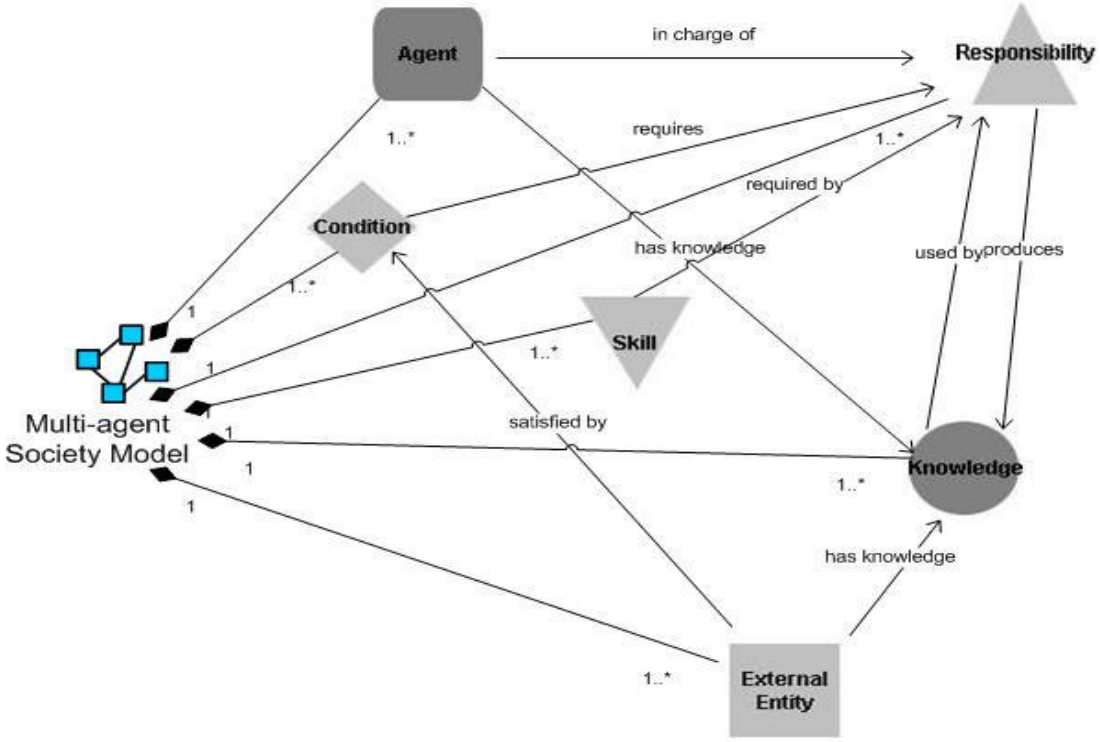


Figura 66 – Metamodelo da Sociedade Multiagente

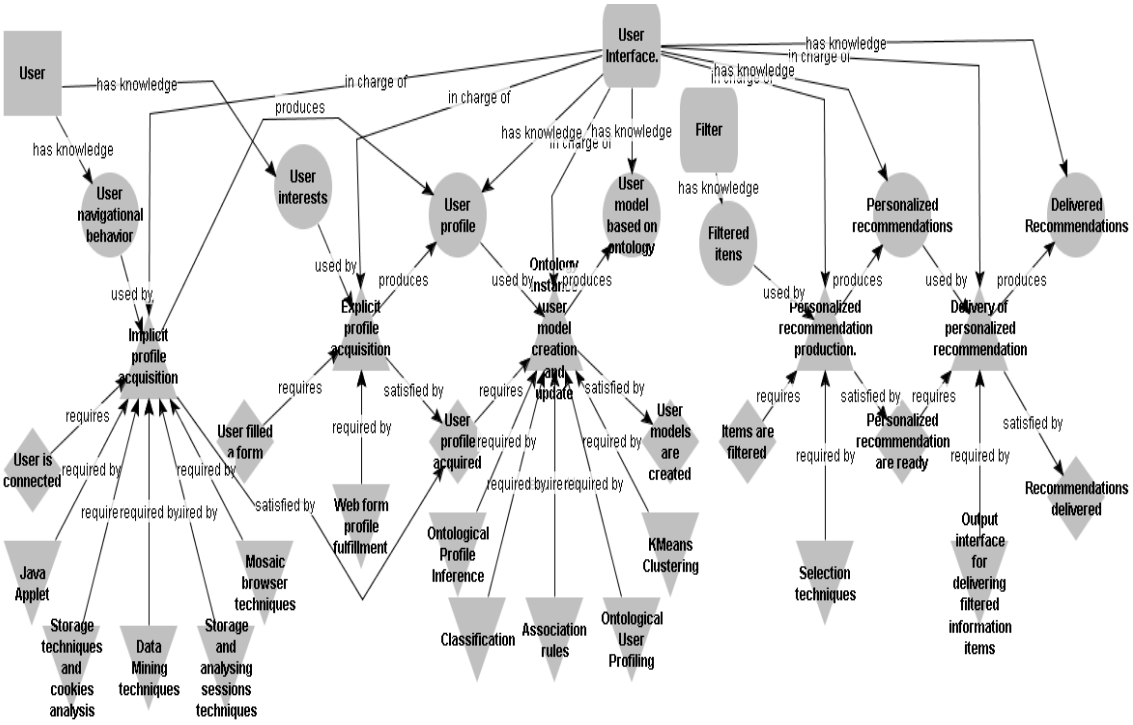


Figura 67 - Modelo da Sociedade Multiagente ONTOSERS para o Agente Filtrador

O Modelo de Interações entre Agentes

O Modelo de Interações entre Agentes representa as interações entre os agentes da sociedade. Este modelo é semelhante a um diagrama de seqüencia da UML. Fazendo uma analogia entre estes dois modelos, um agente ou entidade externa no Modelo de Interações entre Agentes seria um objeto no diagrama de seqüencia da UML, possuindo linha de vida e trocando mensagens entre si. As mensagens são especificadas seguindo as diretrizes da FIPA-ACL. Este modelo é construído a partir do reúso de Modelo de Interações entre Agentes preexistentes e com base no Modelo da Sociedade Multiagente. A partir do Modelo da Sociedade Multiagente são obtidos os agentes, as entidades externas com que eles interagem e o conhecimento trocado em suas comunicações.

Na Figura 68, o metamodelo de interações entre agentes é ilustrado. Nesse metamodelo são representados os conceitos de “Agent”, representando cada agente da sociedade, “Knowledge”, representando o conhecimento trocado nas comunicações entre agentes e “External Entity”, representando as entidades externas com as quais os agentes têm necessidade de se comunicar. O conceito “Agent Interaction” representa as interações entre agentes e “Agent Notification from

External Entity" representa as interações entre um agente e uma entidade externa. Na Figura 69 é ilustrado o Modelo de Interações entre Agentes da ONTOSERS.

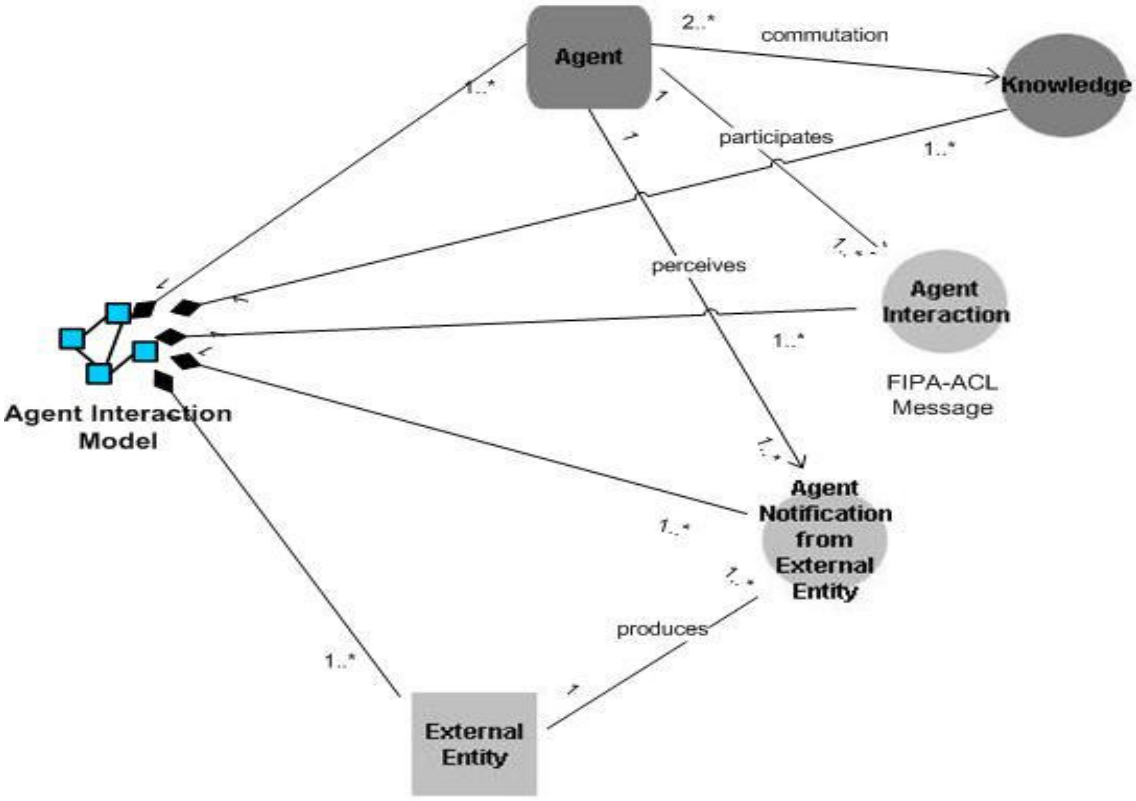


Figura 68 – Metamodelo de Interações entre Agentes

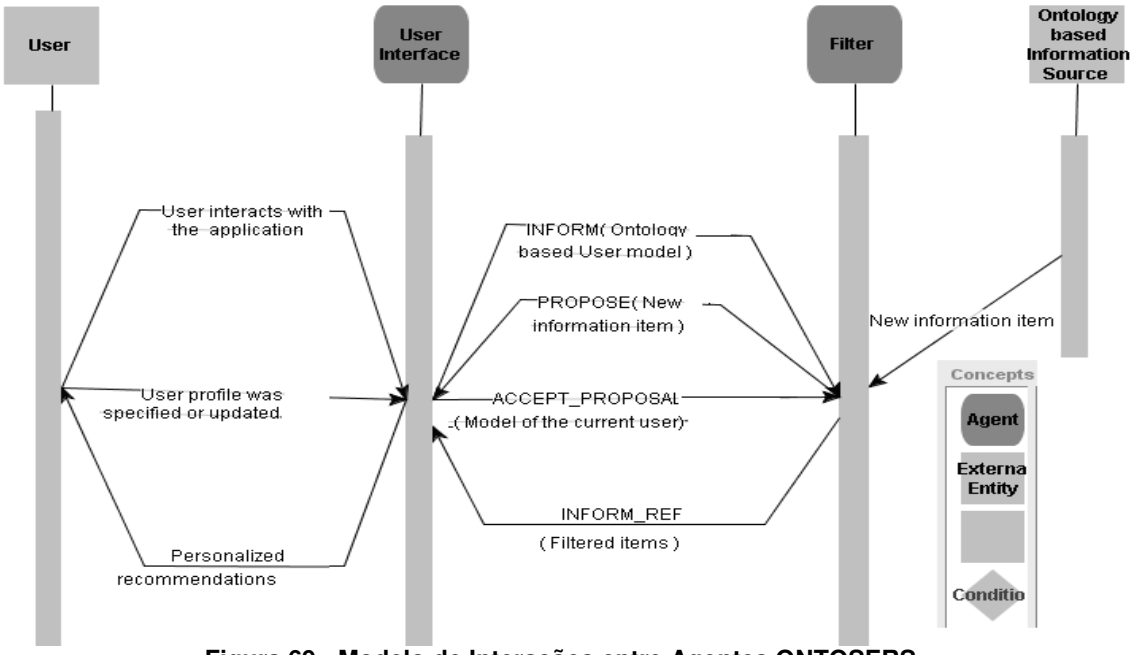


Figura 69 - Modelo de Interações entre Agentes ONTOSERS

O Modelo de Cooperação e Coordenação

Para o Modelo de Cooperação e Coordenação ser construído é reusado ou criado um mecanismo de cooperação e coordenação entre agentes que satisfaçam os requisitos da família de aplicações presentes no Modelo de Objetivos. Esse modelo especifica como os agentes da sociedade vão cooperar, organizar-se e interagir para realizar uma determinada tarefa. O padrão de cooperação determina como os agentes vão interagir para realizar suas responsabilidades. Exemplos de padrões de cooperação são o quadro-negro e o federativo (Girardi, 2004). O padrão de coordenação especifica como os agentes estarão organizados. Eles podem estar em um mesmo nível hierárquico como é o caso do padrão de mercado, onde os agentes são classificados em consumidores e vendedores ou em níveis de hierarquia diferentes como, por exemplo, o mecanismo de mestre-escravo, onde os agentes são classificados em gerentes (mestres) e em trabalhadores (escravos). Os agentes trabalhadores são coordenados por um gerente que distribui as tarefas entre estes e espera o resultado. Esses padrões estão incluídos na ONTORMAS, como instâncias da classe “Architectural Pattern”. O Modelo de Cooperação e Coordenação é construído a partir do reúso de Modelos de Cooperação e Coordenação desenvolvidos preexistentes e com base no Modelo de Sociedade Multiagente, Modelo de Interações entre Agentes e no Modelo de Objetivos. A partir do Modelo da Sociedade Multiagente são identificados os agentes da Sociedade.

No Modelo de Interações entre Agentes, o conjunto de interações entre agentes é identificado, auxiliando na escolha da melhor arquitetura de cooperação/coordenação. A partir do Modelo de Objetivos serão obtidos os requisitos não-funcionais que devem ser atendidos na escolha da arquitetura de cooperação/coordenação.

Na Figura 70, o metamodelo de cooperação e coordenação é ilustrado. Nesse metamodelo o conceito de “Architectural Pattern” representando o padrão arquitetural que os agentes estarão organizados e o conceito de “External Entity”, que representa as entidades externas com as quais os agentes interagem.

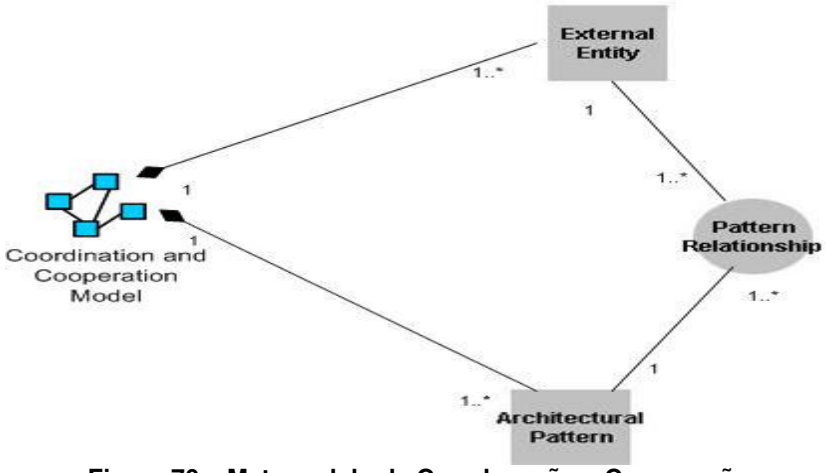


Figura 70 – Metamodelo de Coordenação e Cooperação

O Modelo de Conhecimento do Agente

O Modelo de Conhecimento do Agente representa o conhecimento particular de cada agente da sociedade. Ele é construído a partir do reúso de Modelos de Conhecimento do Agente desenvolvidos na Engenharia de Aplicações e com base no Modelo de Interações entre Papéis, o qual provê o conhecimento compartilhado entre os papéis durante as suas interações, e no Modelo do Conhecimento da Sociedade Multiagente, do qual se obtém os conhecimentos específicos de cada agente. Na Figura 71, o metamodelo do conhecimento do agente é ilustrado. Esse metamodelo possui os conceitos de “Knowledge” e “Knowledge Relationship” que representam, respectivamente, o conhecimento particular de cada agente e o relacionamento entre esses conhecimentos.

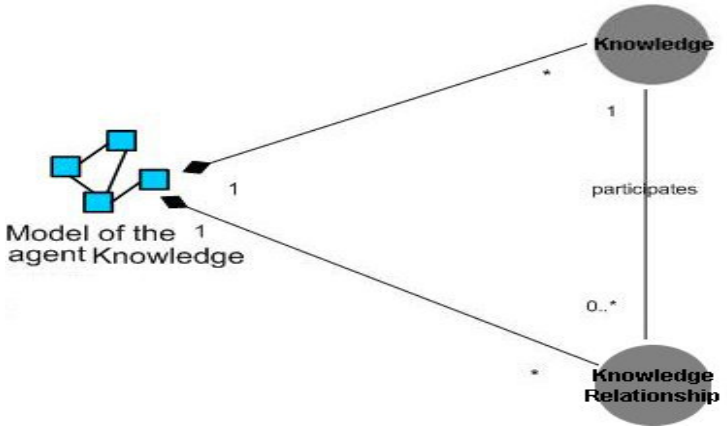


Figura 71 - Metamodelo do Conhecimento do Agente

Na Figura 72, o Modelo de Conhecimento do Agente Interface Usuário do ONTOSERS é ilustrado.

A variabilidade no Modelo de Conhecimento do agente “Interface Usuário” do ONTOSERS é representada na Figura 73. Um ponto de variação foi colocado sobre o conhecimento “Perfil do Usuário”, que tem como variantes os conhecimentos “Interesses do Usuário“ e “Comportamento Navegacional do Usuário”.

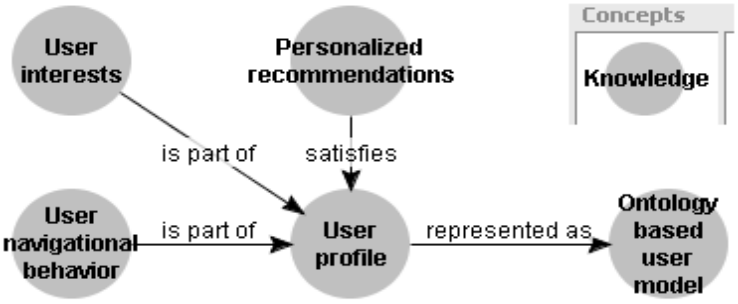


Figura 72 – Modelo do Conhecimento do Agente Interface Usuário

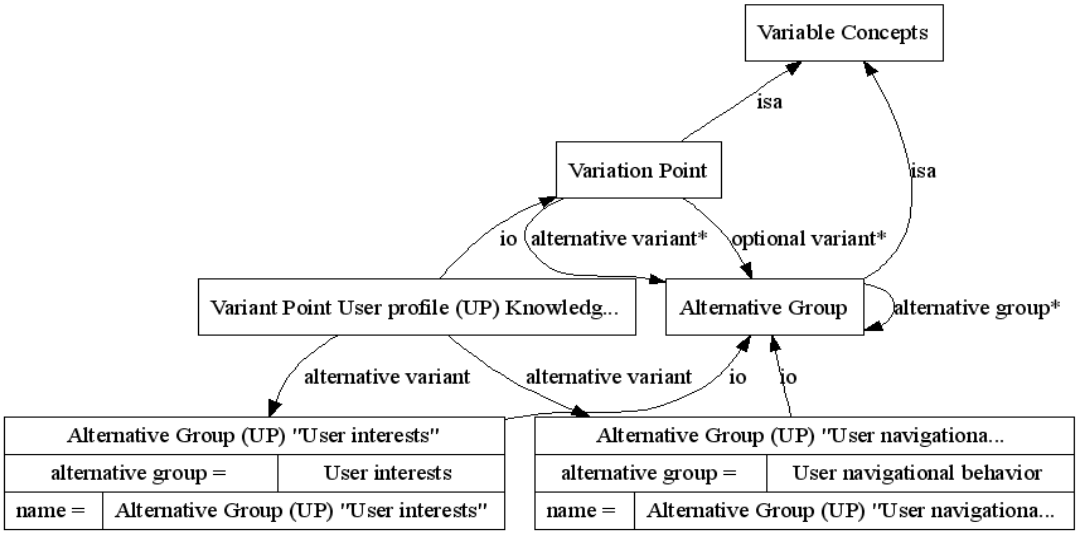


Figura 73 – Modelo de Variabilidades Agente Interface com Usuário

O Modelo de Ações do Agente

O Modelo de Ações do Agente representa as ações que o agente terá no seu ambiente de acordo com as responsabilidades de que tiver sido encarregado. Nesse modelo as percepções que o agente tem do ambiente: um evento ou as

possíveis mensagens que ele poderá receber de outros agentes ou de entidades externas que irão disparar as suas ações são representadas. Existem duas formas de mapeamento dessas percepções para ações. A primeira, é quando uma percepção é mapeada diretamente para uma ação. Nesse caso, o agente é considerado tipo reativo. A outra forma é quando ocorre um planejamento e na busca da ação adequada de acordo com a percepção do agente para atingir um objetivo. Nesse caso, o agente é considerado do tipo deliberativo e usa um motor de inferência no seu processo de raciocínio. Para realizar uma ação o agente pode necessitar de uma destreza (uma destreza é uma habilidade que o agente deverá possuir para a execução de suas responsabilidades) e também atender a uma determinada pré-condição. Ao termino de uma ação, uma pós-condição também pode ser necessária.

É construído um Modelo de Ações do Agente para cada agente da sociedade. Na Figura 74 é ilustrado o metamodelo das ações do agente.

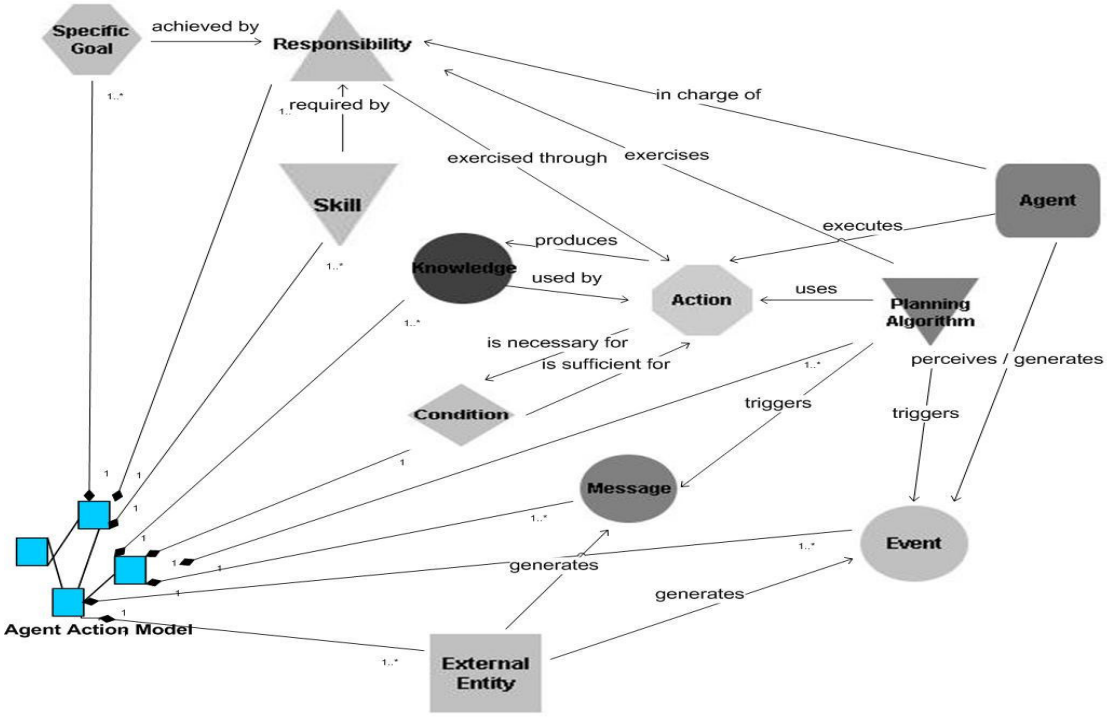


Figura 74 - Metamodelo de Ações do Agente

Na fase de Implementação do Domínio, a técnica DIMAS recomenda o mapeamento de modelos de projeto para agentes, comportamentos e atos de comunicação envolvidos no framework JADE, o qual tem sido adotado como plataforma de implementação. Um Modelo de Implementação da Sociedade

Multiagente é construído como produto desta fase da MADEM, composto de um Modelo de Comportamentos e de um Modelo de Atos de Comunicação.

O Modelo de Comportamentos

O Modelo de Comportamentos representa os comportamentos dos agentes em uma determinada linguagem/plataforma de implementação. Para cada responsabilidade identificada no Modelo de Atividades é associado um comportamento. Este modelo é construído a partir de Modelos de Comportamentos preexistentes e a partir do mapeamento de agentes, responsabilidades e atividades presentes no Modelo de Atividades. Um agente é mapeado para uma classe de agente JADE, uma responsabilidade é mapeada para um comportamento JADE e uma atividade é mapeada para um método JADE. Estão incluídos na base de conhecimento da ONTORMAS os possíveis comportamentos e métodos JADE, como instâncias das classes “JADE Behaviour” e “Method JADE Behaviour”, respectivamente.

Na Figura 75, o metamodelo de comportamentos é ilustrado. Nesse metamodelo, o conceito “JADE Agent”, representa um agente JADE; “JADE Behaviour”, representa um comportamento JADE e “Method JADE Behaviour”, representa um método do comportamento JADE.

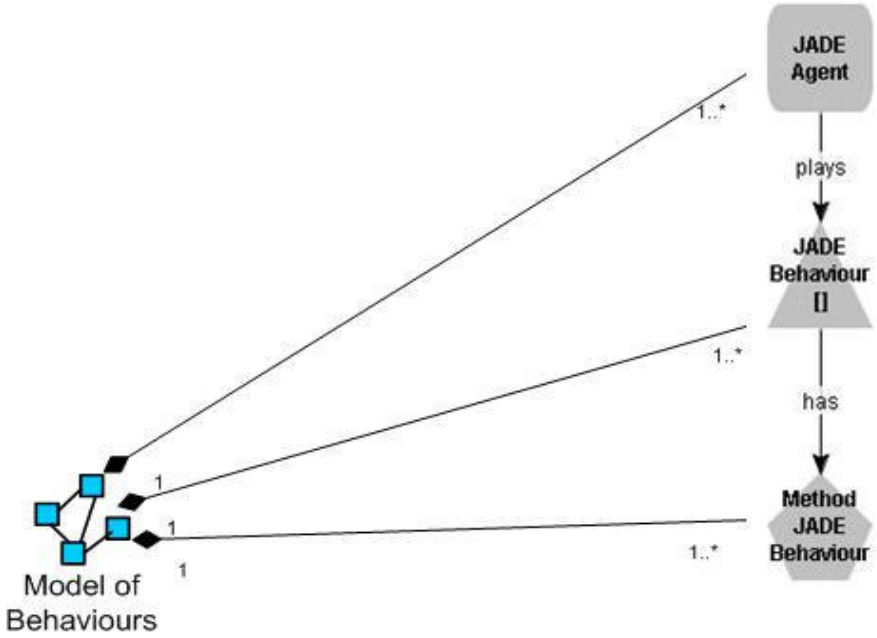


Figura 75 - Metamodelo de Comportamentos

O Modelo de Atos de Comunicação

No Modelo de Atos de Comunicação as mensagens trocadas entre agentes presentes no Modelo de Interações entre Agentes são mapeadas para a linguagem FIPA-ACL (FIPA, 2008). As interações identificadas no Modelo de Interações entre Agentes são mapeadas para uma linguagem de comunicação entre agentes. A partir do Modelo de Comportamentos e do Modelo de Atos de Comunicação a codificação de agentes genéricos a uma família de aplicações é realizada. Agentes desenvolvidos na Engenharia de Aplicações também podem ser reusados nesta tarefa.

Na Figura 76, o metamodelo de atos de comunicação é ilustrado. Nesse metamodelo o conceito “JADE Agent”, representa um agente JADE; “JADE Behaviour”, representa um comportamento JADE; “Method JADE Behaviour”, representa um método do comportamento JADE e “Performative”, representa uma performativa presente na especificação da linguagem de comunicação entre agentes FIPA-ACL.

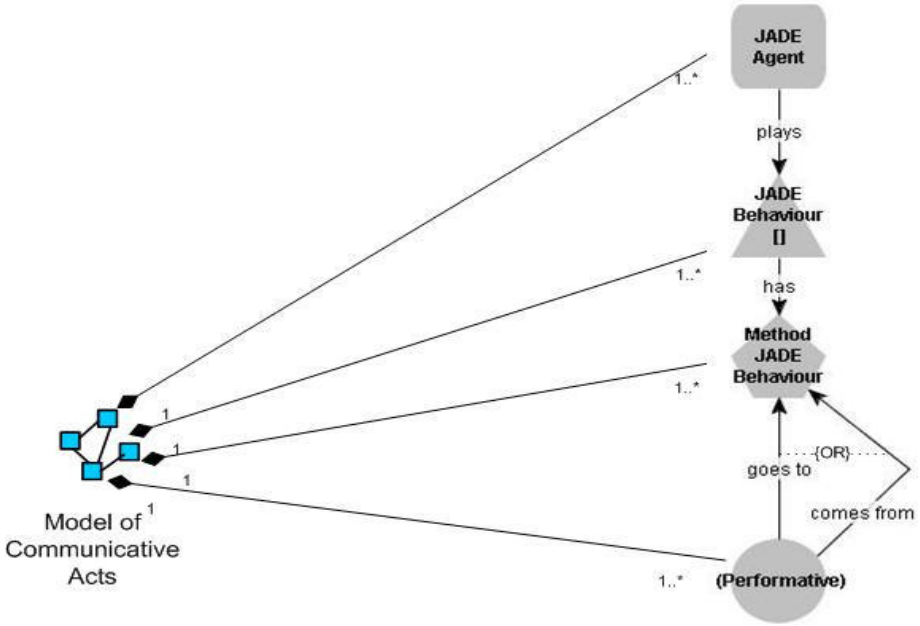


Figura 76 - Metamodelo de Atos de Comunicação

A Tabela 9 descreve a correspondência entre os conceitos do modelo de implementação para a plataforma de agentes JADE/JESS. O agente, quando for do

tipo reativo irá utilizar somente conceitos do framework JADE, enquanto que os do tipo deliberativo irão utilizar uma combinação dos frameworks JADE e JESS.

Tabela 9 - Correspondência entre conceitos de projeto e de implementação

Reative Agent	**Design Concept**	**JADE Concept**
	1 Agent	1 Agent Class
	1 Responsibility of each Agent	1 JADE Behaviour Class
	1 Action	Method of JADE Behaviour Class
	1 Multi-agent Knowledge Society Model	1 JADE Ontology (shared of all society agents)
	1 Knowledge instance of each Agent	1 Attribute of JADE Class
Deliberative Agent	**Design Concept**	**JADE/JESS Concept**
	1 Agent	1 Agent Class
	1 Responsibility of each Agent	1 JESS Rule
	1 Actions	JESS Rules
	1 Multi-agent Knowledge Society Model	1 JADE Ontology (shared of all society agents)
	1 Knowledge instance of each Agent	1 JESS Fact in JESS File

3.3.4 A metodologia MAAEM

A MAAEM é uma metodologia para a análise, o projeto e a implementação de aplicações multiagente através da reutilização de artefatos de software anteriormente produzidos na Engenharia de Domínio Multiagente. Sendo inspirada no desenvolvimento baseado em componentes, ela envolve a seleção, adaptação e composição desses artefatos para a construção de uma aplicação. Na fase de análise dos requisitos, pretende-se identificar e especificar os requisitos de uma determinada aplicação, partindo de modelos de domínio, elaborados na fase correspondente da Engenharia de Domínio Multiagente.

Desse modo, a tarefa central do desenvolvedor é reusar um conjunto de requisitos da família em um domínio. Tais requisitos específicos da aplicação, quando provenientes da família, são levantados a partir da seleção dentre os requisitos comuns ou variáveis no domínio. Um resumo das tarefas realizadas, dos insumos requeridos e dos produtos obtidos em cada fase de desenvolvimento de uma aplicação multiagente seguindo as diretrizes da metodologia MAAEM está

descrito na Tabela 10. Nessa tabela, têm-se além dos insumos presentes na Tabela 10, os produtos desenvolvidos pela metodologia MADEM.

Tabela 10 - Fases, Insumos, Tarefas e Produtos da metodologia MAAEM

Fases	Insumos	Tarefas		Produtos		
Engenharia dos Requisitos da Aplicação	Modelo de Conceitos (MADEM)	Modelagem de Conceitos		Modelo de Conceitos		Especificação dos Requisitos da Aplicação
	Conhecimento de Especialistas, Literatura no Domínio, Aplicações de Software Existentes					
	Modelo de Objetivos (MADEM)	Modelagem de Objetivos		Modelo de Objetivos		
	Requisitos da família de aplicações					
	Modelo de Papéis (MADEM)	Modelagem de Papéis		Modelo de Papéis		
	Modelo de Objetivos					
	Modelo de Interações entre Papéis (MADEM)	Modelagem de Interações entre Papéis		Modelo de Interações entre Papéis		
	Modelo de Objetivos					
	Modelo de Papéis					
	Protótipo da Interface com o usuário (MADEM)	Prototipagem da Interface com o usuário		Protótipo da Interface com o usuário		
	Modelo de Papéis					
	Modelo de Interações entre Papéis					
Projeto da Aplicação	Modelo do Conhecimento da Sociedade Multiagente (MADEM)	Modelagem Arquitetural	Modelagem do Conhecimento da Sociedade Multiagente	Modelo do Conhecimento da Sociedade Multiagente	Modelo Arquitetural	Arquitetura da Aplicação
	Modelo de Conceitos					
	Modelo de Interações entre Papéis					
	Modelo da Sociedade Multiagente (MADEM)		Modelagem da Sociedade de Agentes	Modelo da Sociedade Multiagente		
	Modelo de Papéis					
	Modelo de Interações entre Agentes (MADEM)		Modelagem das Interações entre Agentes	Modelo de Interações entre Agentes		
	Modelo da Sociedade Multiagente					
	Modelo dos Mecanismos de Cooperação e Coordenação (MADEM)		Modelagem dos Mecanismos de Cooperação e Coordenação	Modelo dos Mecanismos de Cooperação e Coordenação		
	Modelo de Objetivos (Requisitos Não-Funcionais)					
	Modelo da Sociedade Multiagente					
	Modelo das Interações entre Agentes					
	Modelos do Conhecimento do Agente (MADEM)	Projeto do Agente	Modelagem do Conhecimento do Agente	Modelos do Conhecimento do Agente	Modelos do Agente	
	Modelo de Interações entre Papéis					
	Modelo do Conhecimento da Sociedade Multiagente					
	Modelos das Ações do Agente (MADEM)		Modelagem das Ações dos Agentes	Modelos das Ações do Agente		
	Modelo da Sociedade Multiagente					
	Modelo de Interações entre Agentes					
Implementação da Aplicação	Modelo de Comportamentos (MADEM)	Modelagem de Comportamentos		Modelo de Comportamentos		Modelo de Implementação da Aplicação
	Modelos de Ações					
	Modelo de Atos de Comunicação (MADEM)	Modelagem de Atos de Comunicação		Modelo de Atos de Comunicação		
	Modelo de Interações entre Agentes					
	Agentes de Software Executáveis (MADEM)	Implementação dos Agentes		Agentes de Software Executáveis		
	Modelo de Comportamentos					
	Modelo de Atos de Comunicação					

O Modelo de Conceitos

Para criação do Modelo de Conceitos, identificam-se alguns conceitos-chave da aplicação, tomando-se tais conceitos como parâmetros para o reúso e levando em conta a variabilidade de cada um. Novos conceitos podem ser adicionados para melhor entendimento do domínio de problema da aplicação. A composição do modelo decorre do relacionamento final de todos os conceitos presentes no modelo, sejam os reutilizados sejam os novos, de maneira que resulte uma representação uniforme e coerente do conhecimento básico sobre o domínio de problema da aplicação em desenvolvimento. Na Figura 77 o Modelo de Conceitos de Infotrib é ilustrado.

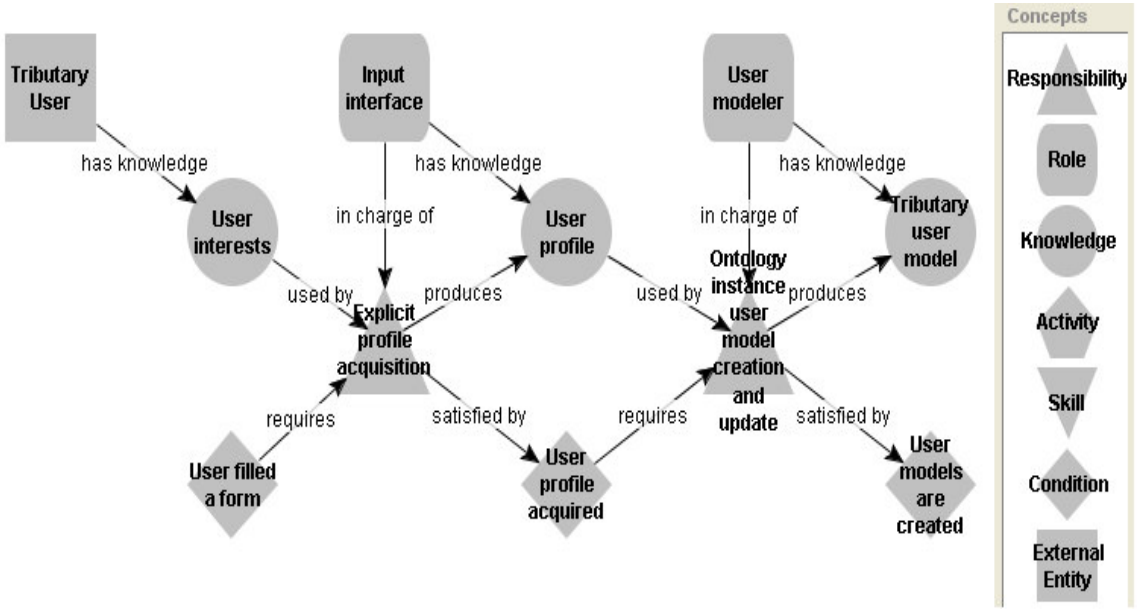


Figura 77 - Modelo de Conceitos INFOTRIB

O Modelo de Objetivos

O Modelo de Objetivos é construído a partir da seleção no repositório de objetivos, responsabilidades e entidades externas pertencentes à mesma família da aplicação em desenvolvimento. O objetivo geral é comum a todas as aplicações da família, os objetivos específicos podem ser comuns ou variar conforme a aplicação da família. Assim, quando o objetivo da aplicação corresponder ao objetivo geral de um dado modelo de objetivos no repositório da ONTORMAS, então este e todos os

objetivos específicos comuns serão selecionados. Os objetivos específicos variáveis que contribuam para o alcance do objetivo geral da aplicação também deveram ser selecionados, respeitando o tipo da variabilidade (alternativa ou opcional). Após isso, é feita também a seleção de responsabilidades e entidades externas dentre aquelas contidas nos modelos de objetivos reutilizados. Uma vez que tenham sido selecionados todos os objetivos, responsabilidades e entidades externas, eles são então eventualmente adaptados para compor o Modelo de Objetivos, que conterá: no seu nível superior, o objetivo geral da aplicação; nos níveis intermediários, os objetivos específicos e as entidades externas e no nível inferior, as responsabilidades. Na Figura 78, o modelo de objetivos de Infotrib é ilustrado.

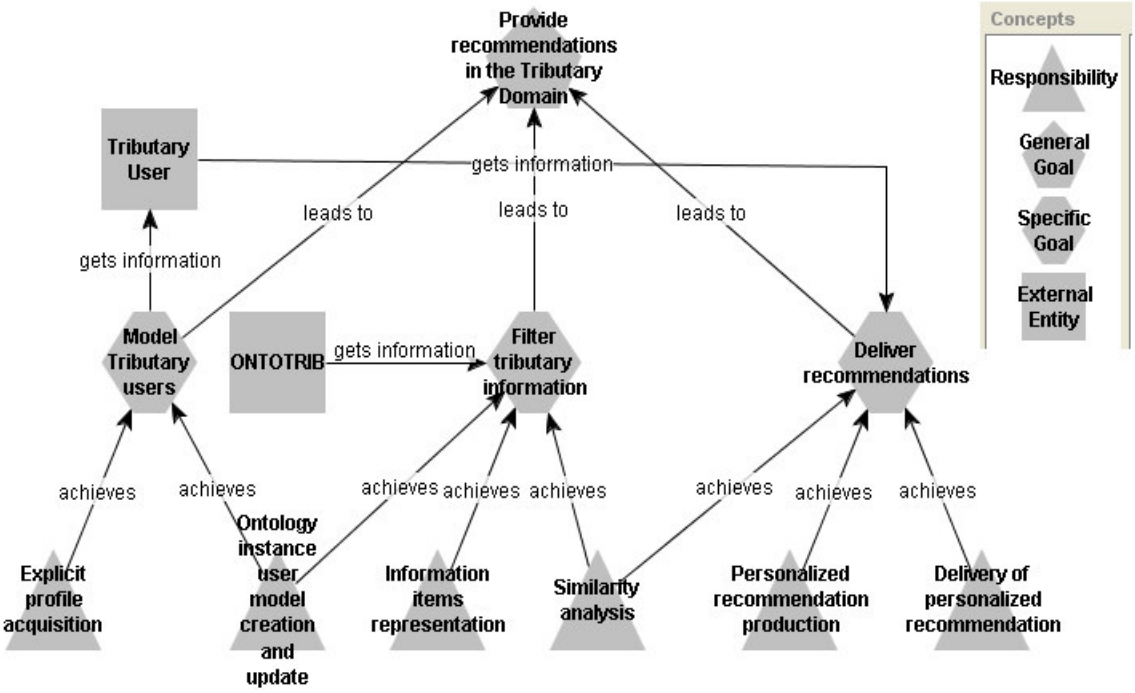


Figura 78 - Modelo de Objetivos INFOTRIB

O Modelo de Papéis

As entidades externas podem também ser reusadas a partir do Modelo de Objetivos. Como uma responsabilidade do Modelo de Objetivos é atribuída a um papel no Modelo de Papéis, pode-se facilmente inferir a entidade externa através do relacionamento existente entre uma entidade externa e um objetivo específico e deste com responsabilidades no Modelo de Objetivos. O Modelo de Papéis resultará

de uma composição dos conceitos selecionados e adaptados e de novos conceitos adicionados. Na Figura 79, o Modelo de Papéis do Infotrib é ilustrado.

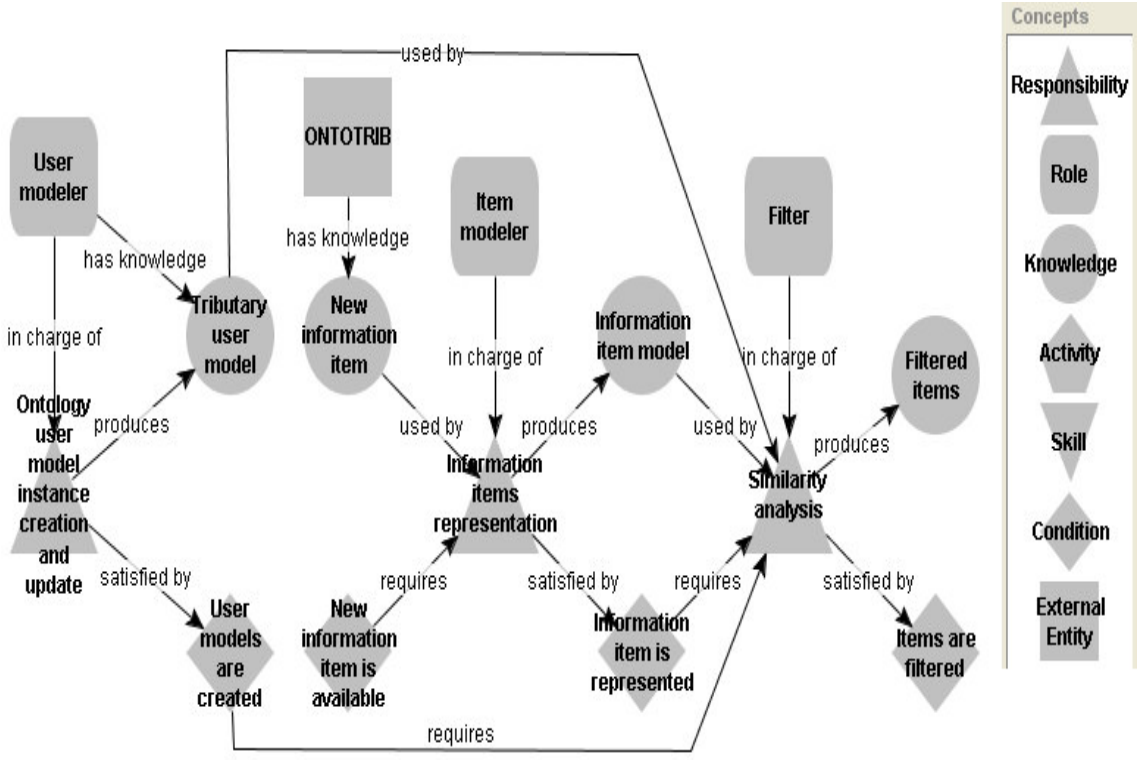


Figura 79 - Modelo de Papéis INFOTRIB, para o objetivo específico "Filtrar Informações"

O Modelo de Interações entre Papéis

Para cada objetivo específico de nível mais baixo na hierarquia do Modelo de Objetivos é construído um Modelo de Interações entre Papéis. Nesse modelo, também são definidas as mensagens trocadas entre os papéis e entre papéis e entidades externas e os conhecimentos trocados. Como as interações dependem fundamentalmente dos papéis e entidades externas envolvidas, então, o reúso será em função da diversidade de papéis reusados. Assim, quanto mais papéis forem reusados a partir de um único modelo maior será a quantidade de interações reusadas. Quando os papéis forem reusados de modelos diferentes ou novos papéis forem adicionados não será possível reusar interações. Na Figura 80 é ilustrado o Modelo de Interações entre Papéis do Infotrib.

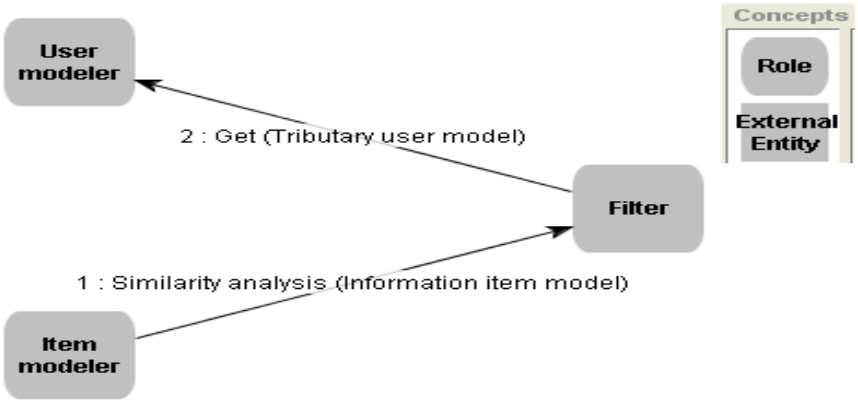


Figura 80 - Modelo de Interações entre Papéis para o objetivo específico "Filtrar Usuário"

O Protótipo de Interface com Usuário

Na Prototipação da Interface com Usuário as possíveis interações que os usuários terão com o sistema são identificadas e um protótipo é construído. Protótipos de Interface Usuários genéricos disponíveis na ONTORMAS podem ser reusados nessa tarefa. Na Figura 81 um exemplo de parte do protótipo de interface usuário do Infotrib é ilustrado.

Figura 81 – Parte do Protótipo de Interface Usuário da INFOTRIB

Na fase de Projeto da Aplicação, o desenvolvedor irá reutilizar algumas soluções de projeto relacionadas a uma família de aplicações e adaptá-la aos

requisitos específicos da aplicação em desenvolvimento. Essa fase consiste de três tarefas: Modelagem do Conhecimento da Sociedade Multiagente, a qual representa os conceitos compartilhados por todos os agentes em sua comunicação; o Projeto Arquitetural, que estabelece um modelo arquitetural da sociedade multiagente incluindo seus mecanismos de coordenação e cooperação; e o Projeto do Agente, que define o projeto interno de cada agente da sociedade, modelando sua estrutura de conhecimento e seus comportamentos.

O Projeto Arquitetural é composto pelos seguintes modelos: Modelo do Conhecimento da Sociedade Multiagente, Modelo da Sociedade Multiagente, Modelo de Interações entre Agentes, Modelo de Atividades e Modelo de Cooperação e Coordenação. No Projeto do Agente é construído um Modelo de Comportamentos e um Modelo de Estados para cada agente da sociedade.

O Modelo do Conhecimento da Sociedade Multiagente

No Modelo do Conhecimento da Sociedade Multiagente, os conceitos compartilhados pelos agentes da sociedade e representados em uma rede semântica são identificados. Esse modelo utiliza como insumo o conhecimento trocado entre os papéis em sua comunicação e os conceitos presentes no Modelo de Conceitos. Modelos de Conhecimento da Sociedade Multiagente pertencentes à mesma família de aplicações também podem ser reusados. Na Figura 81 o Modelo de Conhecimento da Sociedade Multiagente do Infotrib é ilustrado.

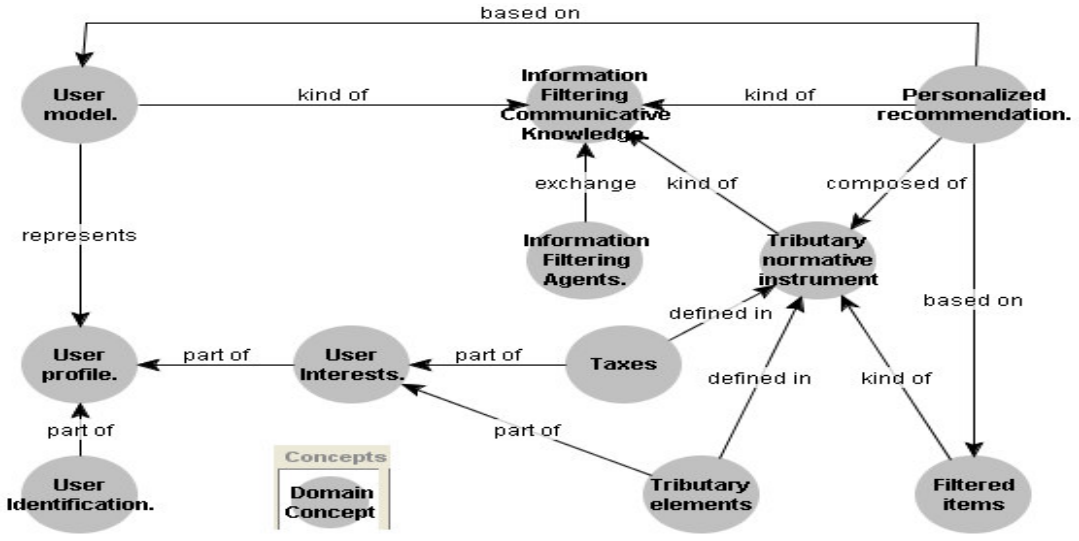


Figura 82 - Modelo do Conhecimento da Sociedade Multiagente INFOTRIB

O Modelo da Sociedade Multiagente

Na construção do Modelo da Sociedade Multiagente o ponto de partida são os papéis identificados na fase de Engenharia dos Requisitos da Aplicação. Os papéis servem como critério para seleção de agentes. Por exemplo, se um Modelo de Papéis reusado possui o papel “Filter” que está associado às responsabilidades “Content based similarity analysis” e “Legislative repository monitoring” através do relacionamento “in charge of”, os agentes da mesma família de aplicações encarregados dessas responsabilidades serão reusados. Os conceitos de conhecimento, pré e pós-condições e entidades externas são automaticamente reusados a partir do Modelo de Papéis através do relacionamento que eles possuem com os papéis e com as responsabilidades. Para finalizar o Modelo da Sociedade Multiagente é necessário adicionar o conceito de Destreza. Uma destreza é uma habilidade que o agente deverá possuir para a execução de suas responsabilidades. Na Figura 82 o Modelo da Sociedade Multiagente de Infotrib é ilustrado.

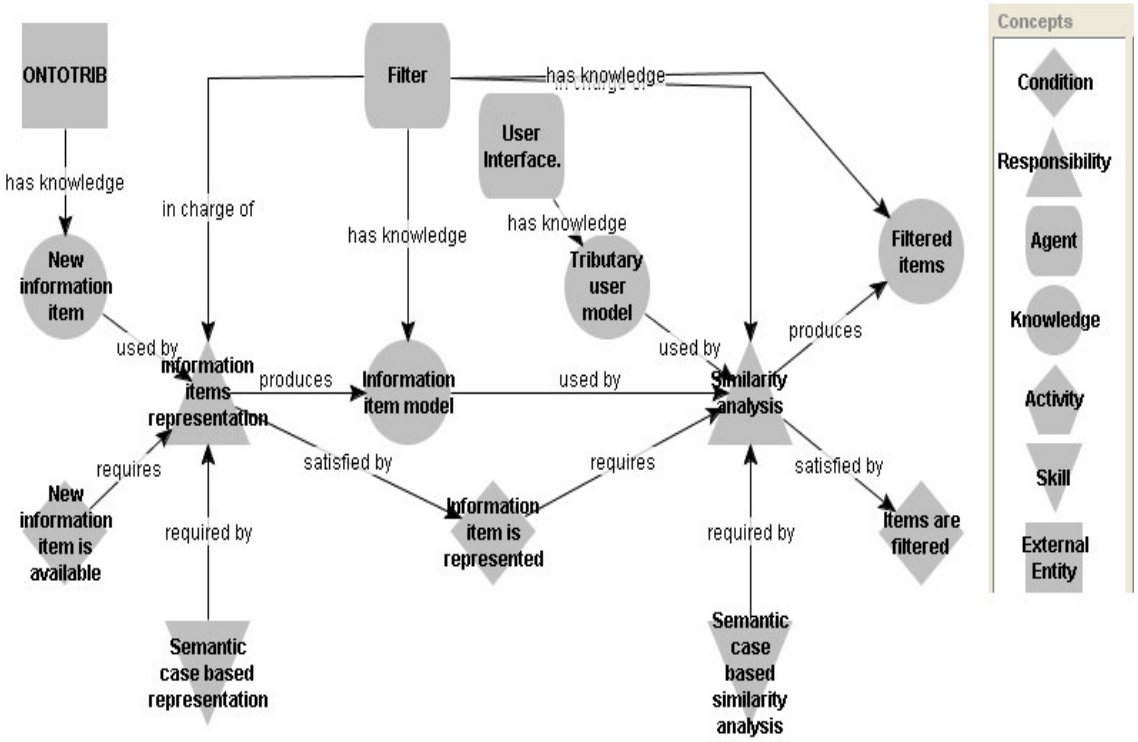


Figura 83 - Modelo da Sociedade Multiagente INFOTRIB (Agente Filtrador)

O Modelo de Interações entre Agentes

Para a construção do Modelo de Interações entre Agentes são utilizados como insumo as interações presentes no Modelo de Interações entre Papéis, selecionadas de acordo com o mapeamento de papéis para agentes. Agentes do Modelo da Sociedade Multiagente podem ser utilizados como parâmetro para selecionar um Modelo de Interações entre Agentes na base de conhecimento da ONTORMAS. As interações serão reusadas quando o agente emissor e receptor da mensagem fizer parte do Modelo da Sociedade Multiagente. Na Figura 84 o Modelo de Interações entre Agentes da Infotrib é ilustrado.

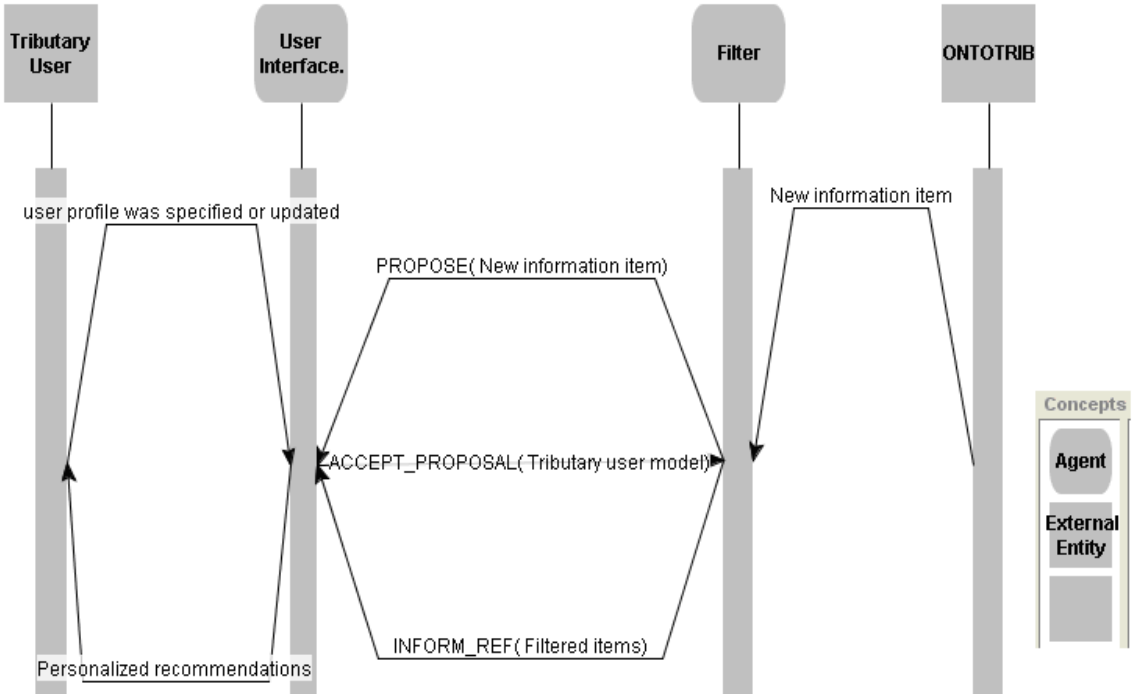


Figura 84 - Modelo de Interações entre Agentes

O Modelo dos Mecanismos de Cooperação e Coordenação

A Modelagem dos Mecanismos de Cooperação e Coordenação é realizada através da seleção de padrões arquiteturais no repositório ONTORMAS. Para tanto, a descrição do objetivo da aplicação é comparada com o problema descrito em cada padrão e com seu contexto de aplicação, no caso, o dos sistemas multiagente. Por exemplo, na escolha de um padrão arquitetural em camadas, uma ou mais camadas podem ser reusadas se fazendo as adaptações necessárias à

aplicação em desenvolvimento. Na Figura 82 o Modelo dos Mecanismos de Cooperação e Coordenação do Infotrib é ilustrado.

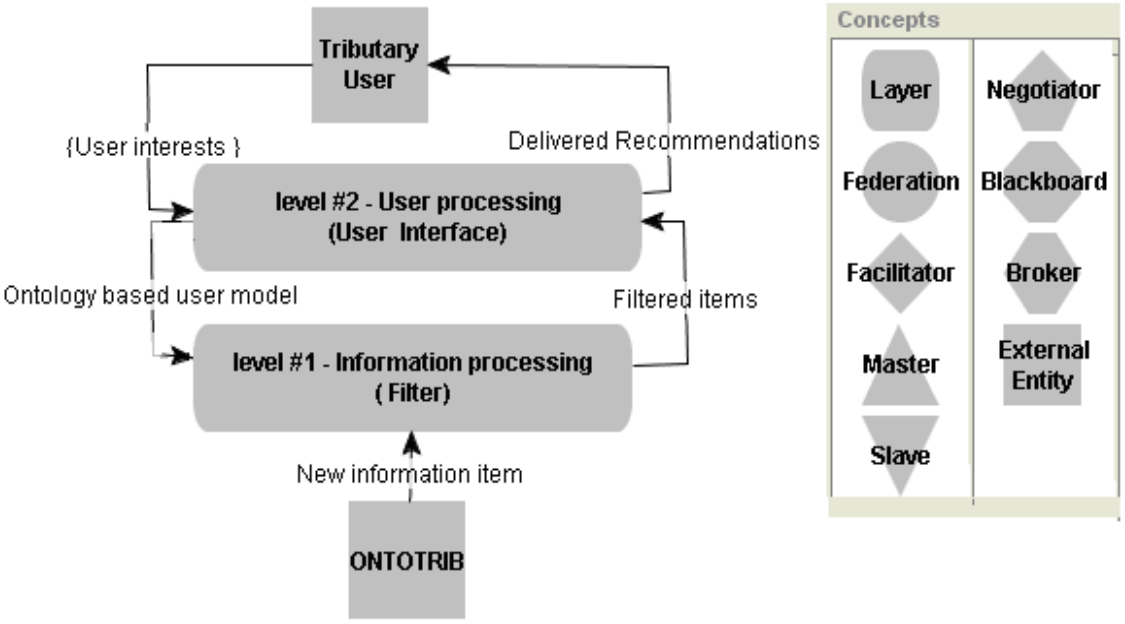


Figura 85 - Modelo de Cooperação e Coordenação INFOTRIB

O Modelo do Conhecimento do Agente

No Modelo de Conhecimento do Agente, o conhecimento de cada agente da sociedade é definido. Os agentes reusados no Modelo da Sociedade Multiagente são utilizados como parâmetros da consulta na ONTORMAS, onde são selecionados Modelos de Conhecimento e Modelo de Estados na base de conhecimento da ONTORMAS. Na Figura 86 é ilustrado o Modelo do Conhecimento do Agente Filtrador da aplicação Infotrib.

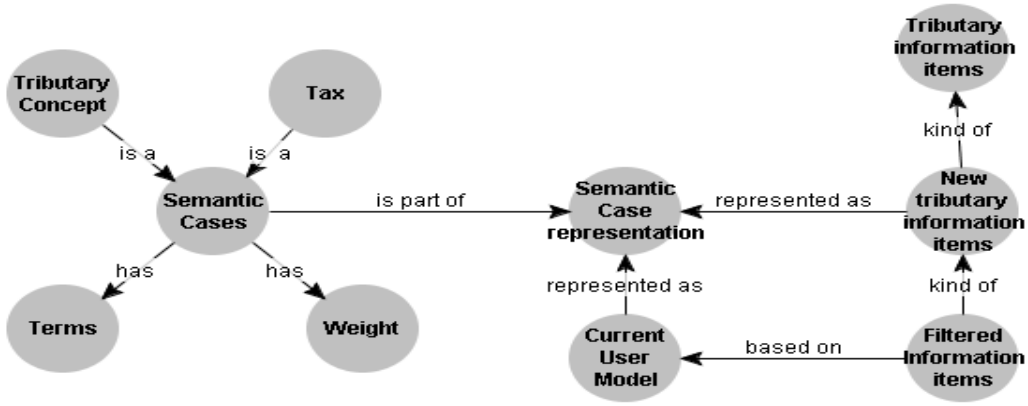


Figura 86 - Modelo do Conhecimento do Agente Filtrador

O Modelo das Ações do Agente

O Modelo das Ações do Agente representa as ações que o agente realiza de acordo com as suas responsabilidades e as percepções do ambiente. O agente pode ser do tipo reativo ou deliberativo. No agente reativo as ações são mapeadas diretamente a partir das percepções, já no caso do agente deliberativo a partir das percepções, ele realiza planejamento para decidir que ações tomar. No Modelo das ações do Agente, o planejamento é realizado a partir das percepções que são o conhecimento das entidades externas, do conjunto de ações, do estado atual do mundo (pré-condição), do estado desejado (pós-condição) para determinar uma seqüência de ações que leve do estado atual do mundo para o estado desejado. Na Figura 87 é ilustrado o Modelo de Ações do Agente Filtrador da Aplicação Infotrib.

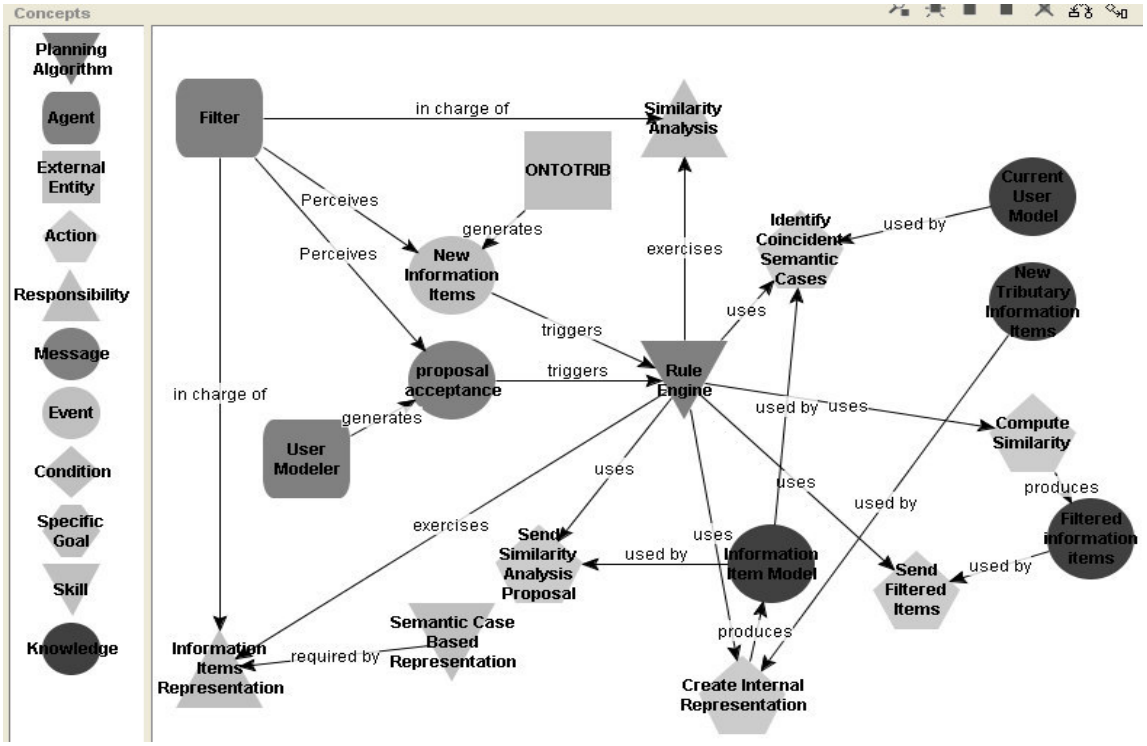


Figura 87 - Modelo de Ações do Agente Filtrador

Na fase de Implementação da Aplicação, os agentes são identificados a partir do modelo de ações do agente da fase anterior, sendo associado a cada uma de suas responsabilidades comportamentos em uma determinada linguagem/plataforma de desenvolvimento de agentes, como o JADE. São ainda identificadas as interações entre agentes presentes no modelo de interações entre

agentes, sendo estas mapeadas para uma determinada linguagem de comunicação entre agentes, como a FIPA-ACL. O reúso, se dá através da complementação do código gerado, para a qual o programador utiliza agentes de software anteriormente implementados na fase de Implementação do Domínio. A seleção dos agentes ocorre conforme a semelhança de comportamentos e comunicações. A adaptação torna-se necessária em função dos detalhes que diferem de uma aplicação para outra. Na Figura 88 é ilustrado o Modelo de Comportamentos do Agente Filtrado do Infotrib e na Figura 89 o Modelo de Atos de Comunicação.

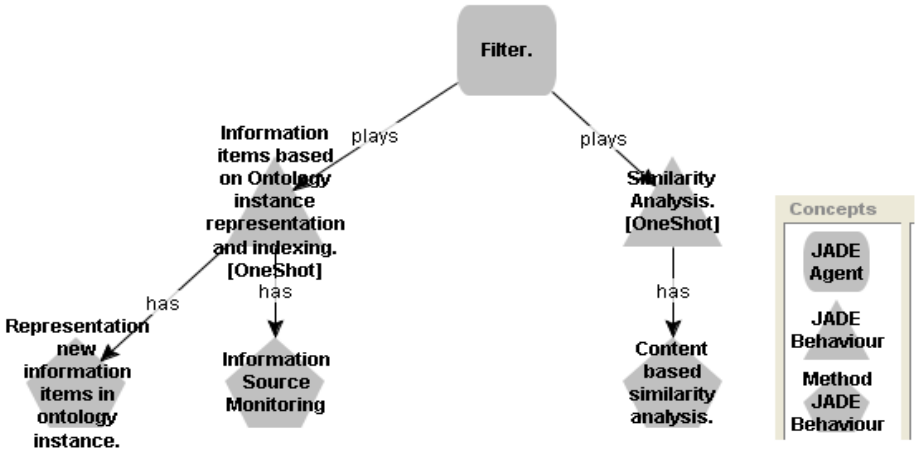


Figura 88 - Modelo de Comportamento do Agente Filtrador

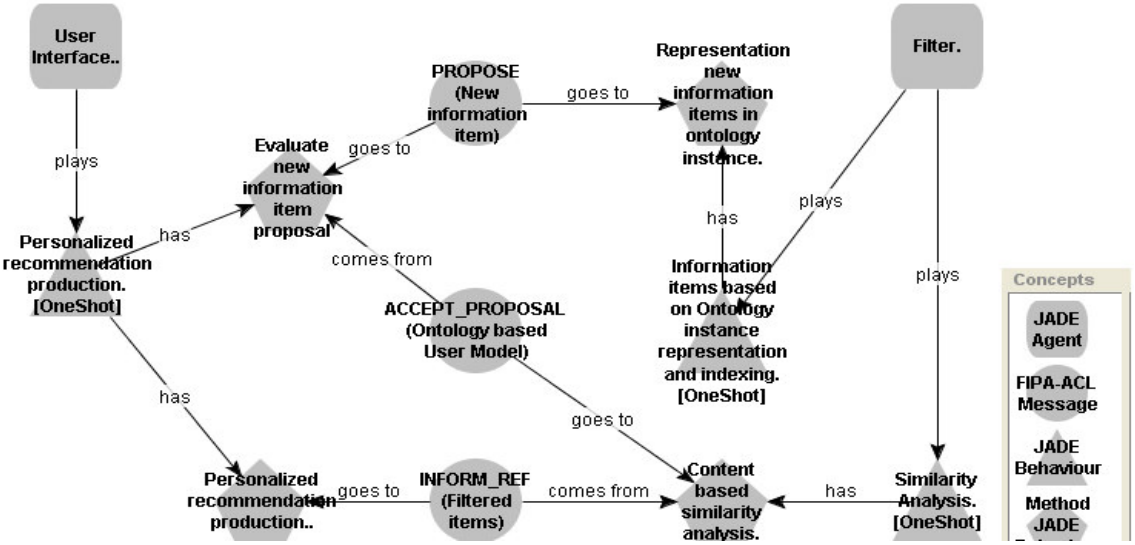


Figura 89 - Modelo de Atos de Comunicação dos Agentes

3.4 Considerações Finais

Este capítulo apresentou o processo MADAE-Pro sob duas perspectivas. Primeiro, foi apresentado o seu ciclo de vida, enfatizando quais as tarefas realizadas, quais os artefatos que cada uma utiliza e produz, quais as técnicas utilizadas para desenvolver esses artefatos, em que ordem e quais os responsáveis por realizar cada tarefa. Em seguida, as metodologias que enfatizam como as tarefas são realizadas durante a realização do processo MADAE-Pro foram apresentadas. Também foi apresentada a linguagem MADAE-ML para modelagem de sistemas multiagente e a ferramenta ONTORMAS que funciona como um repositório dos artefatos desenvolvidos no processo e como ferramenta de modelagem.

A família de aplicações ONTOSERS (MARIANO, 2008) e a aplicação específica InfoTrib (MARIANO, 2008) foram desenvolvidas utilizando MADAE-Pro e contribuíram para o seu aperfeiçoamento. As principais modificações realizadas no MADAE-Pro em função do desenvolvimento dessas aplicações foram:

- Anteriormente ao desenvolvimento de ONTOSERS e InfoTrib, a modelagem de variabilidades no MADAE-Pro era realizada somente na fase de Análise de Domínio. Essa tarefa era realizada classificando-se alguns dos conceitos presentes nos Modelo de Objetivos e no Modelo de Papéis em: *mandatório,* para artefatos comuns a toda família de aplicações; *alternativo,* onde exatamente um conceito de um conjunto de conceitos pode ser incluído no artefato, e *opcional*, quando o conceito pode ou não ser incluído no artefato. Com o desenvolvimento de ONTOSERS e InfoTrib, descobriu-se que essa abordagem não era eficiente, pois os conceitos são variáveis em relação a outros e não de forma independente como era feito e também que a modelagem deve ser realizada em todas as fases da Engenharia de Domínio. Então, toda a modelagem de variabilidades do MADAE-Pro foi redefinida, incorporando-se os conceitos de Pontos de Variação e Variantes em todas as fases da Engenharia de Domínio, conceitos estes que

tem se mostrado eficazes para especificar os conceitos comuns e variáveis da família;

- Anteriormente, o conceito de requisitos não-funcionais, como confiabilidade e segurança, não era definido no processo. Durante o desenvolvimento de ONTOSERS e InfoTrib, houve a necessidade da utilização desse conceito. Então, foi definido que um ou mais requisitos não-funcionais são associados aos conceitos de objetivo geral e de objetivo específico do Modelo de Objetivos;

- Durante a modelagem de ONTOSERS e InfoTrib, os conceitos de "Destreza" e "Atividade" foram eliminados da fase de Análise de Domínio e de Engenharia dos Requisitos da Aplicação. Esses conceitos foram eliminados, porque representam detalhes da solução de um problema, sendo então mais apropriado utilizá-los na fase de projeto.

- Outra mudança, foi a definição do Modelo de Ações do Agente a partir da unificação do Modelo de Estados e do Modelo de Atividades. O Modelo de Ações do Agente permite representar características próprias dos agentes deliberativos, como mecanismo de inferência, que o Modelo de Atividades não permitia. Os estados do agente também foram incorporados nesse modelo, eliminando assim o Modelo de Estados.

4 CONCLUSÕES

Neste trabalho, MADAE-Pro foi proposto, um processo baseado no conhecimento para a Engenharia de Domínio e de Aplicações Multiagente. O processo definiu um ciclo de vida iterativo e incremental, as metodologias MADEM e MAAEM, a linguagem MADAE-ML e a ferramenta ONTORMAS para o desenvolvimento de famílias de aplicações e para o seu reúso na construção de aplicações específicas.

4.1 Principais Contribuições

No contexto do grupo GESEC, o MADAE-Pro trouxe como contribuição a incorporação dos seguintes trabalhos:

- A revisão e refinamento da metodologia MADEM, e suas técnicas GRAMO, DDEMAS e DIMAS;
- A revisão e refinamento da metodologia MAAEM, com suas técnicas SRAMO, ADEMAS, AIMAS;
- A revisão e refinamento da ferramenta ONTORMAS;
- A formalização da linguagem MADAE-ML;

Na elaboração do processo, todos esses produtos foram revisados e atualizados. Vários modelos definidos nas técnicas/metodologias foram modificados, alguns foram integrados em um único modelo. Em função das alterações realizadas nos modelos, a linguagem MADAE-ML e a ONTORMAS também foram atualizadas.

A definição de um ciclo de vida iterativo e incremental também foi uma importante contribuição, uma vez que o desenvolvimento utilizando as metodologias MADEM e MAAEM era puramente seqüencial e, portanto, não adequado para o desenvolvimento de sistemas complexos, como os típicos sistemas multiagente.

O MADAE-Pro foi utilizado no desenvolvimento da família de aplicações ONTOSERS e da aplicação específica INFOTRIB. Esses estudos de casos foram desenvolvidos paralelamente a especificação do processo, com isso também contribuíram de forma significativa para a sua elaboração.

A utilização da linguagem SPEM na especificação do MADAE-Pro, permitiu a obtenção de uma representação formal do processo facilitando a comunicação e entendimento pelos usuários bem como seu processamento computacional. Anteriormente, as metodologias MADEM e MAAEM e suas técnicas estavam especificadas apenas em linguagem textual, o que dificultava a sua utilização pelas possíveis ambigüidades encontradas.

No contexto da Engenharia de Software Multiagente, o MADAE-Pro oferece duas principais contribuições. A primeira é o fato de todos os conceitos de desenvolvimento utilizadas pelo MADAE-Pro, bem como as tarefas de desenvolvimento e os modelos serem representados como classes em uma ontologia. Assim, durante o desenvolvimento de uma determinada aplicação essas classes são instanciadas, formando uma base de conhecimento. Os relacionamentos semânticos existentes entre as classes e instancias da base de conhecimento tem se mostrado extremamente útil no desenvolvimento de software. Isso possibilita, por exemplo, realizar tarefas como a geração automática de parte dos modelos e a checagem de consistência de forma automática através de inferências sobre a base de conhecimento. A segunda contribuição é a integração dos processos da Engenharia de Domínio e da Engenharia de Aplicações no desenvolvimento de software. Isso nos permite obter as vantagens do reúso desde a concepção do software, uma vez que reusamos desde os modelos da fase de análise até o código fonte. Além disso, como os artefatos são projetados para serem reusados a seleção dos mesmos é realizada mais facilmente.

Entretanto, a atual versão do MADAE-Pro possui ainda algumas limitações. As versões de um mesmo produto de software, não possuem mecanismos de controle de versões na ONTORMAS. Atualmente, é realizado somente “backups” dos arquivos quando é necessário fazer alguma modificação nos modelos.

A ONTORMAS oferece suporte a seleção de artefatos através de busca exploratória, busca simples por palavras-chaves e busca avançada através do plugin Algernon, mas a integração e adaptação desses artefatos é realizada de forma manual.

Ainda como contribuição, os seguintes trabalhos foram publicados em periódico, conferências e no site do grupo GESEC:

- “A Process for Multi-Agent Domain And Application Engineering: The Domain Analysis And Application Requirements Engineering Phases” (LEITE e GIRARDI, 2009);
- “A Case Study on Domain Analysis of Semantic Web Multi-Agent Recommender Systems” (MARIANO et al., 2008);
- “An Ontology for Multi-Agent Domain and Application Engineering” (LEITE et al., 2008).
- “MAAEM: A Multi-agent Application Engineering Methodology” (LEITE et al., 2008);
- “ONTORMAS: Uma ferramenta dirigida por ontologias para a Engenharia de Domínio e de Aplicações Multiagente” (LEITE e GIRARDI, 2008);
- “A Case Study on the Application of the MAAEM Methodology for the Specification Modeling of Recommender Systems in the Legal Domain” (DRUMOND et al., 2007);
- “A Knowledge-based Tool for Multi-Agent Domain Engineering” (GIRARDI e LEITE, 2008).
- “ProEDM: Um processo para a Engenharia de Domínio Multiagente” (LEITE e GIRARDI, 2007a);
- “ProEAM: Um processo para a Engenharia de Aplicações Multiagente” (LEITE e GIRARDI, 2007b);
- “MADAE-Pro: Processo para a Engenharia de Domínio e de Aplicações Multiagente” (LEITE e GIRARDI, 2008).

4.2 Trabalhos Futuros

Para melhorar a efetividade do MADAE-Pro alguns trabalhos já estão sendo desenvolvidos e são sugeridos os seguintes trabalhos futuros:

- O ambiente MADAE-IDE (“Multi-agent Domain and Application Engineering - Integrated Development Environment”) (CAVALCANTE, 2008), atualmente nas fases de concepção e projeto, oferecerá suporte completo ao processo, ou seja, permitirá o desenvolvimento de aplicações com e sem reúso de artefatos. Ela consiste em uma aplicação com interface gráfica com o usuário desenvolvida utilizando

um conjunto de regras na linguagem JESS e que acessa a base de conhecimento da ONTORMAS. Graças à riqueza semântica fornecida pelas ontologias, parte considerável dos modelos será gerada de forma automática ou semi-automática a partir de inferências sobre a base de conhecimento;

- Formalização e integração no MADAE-Pro das fases de Teste de Domínio e Teste da Aplicação. Essas fases são importantes para garantir a qualidade no desenvolvimento de software. A fase de Teste de Domínio é responsável pela validação e verificação dos artefatos de software reusáveis pertencentes a uma família de aplicações, já a fase de Teste da Aplicação pela validação e verificação de uma aplicação específica;

- Incluir um sistema de controle de versões na ONTORMAS e, posteriormente, no ambiente MADAE-IDE. Isso irá fornecerá um histórico das modificações realizadas nos projetos de software e possibilitará também armazenar versões estáveis do projeto, evitando assim, o retrabalho em caso de eventuais erros na versão atual do projeto;

- Desenvolvimento ou adoção de uma ferramenta que apóie o gerenciamento do projeto de software no MADAE-Pro;

- Aperfeiçoamento da linguagem de modelagem de sistemas multiagente baseada em ontologias MADAE-ML;

- Avaliação e refinamento do processo através do desenvolvimento de novos estudos de caso na construção de famílias de sistemas e aplicações multiagente específicas;

- Disponibilização do processo e das ferramentas para o público em geral, através da criação de um site na Web.

REFERÊNCIAS

ARANGO, G. **Domain Engineering for Software Reuse.** Ph.D. Thesis, Department of Information and Computer Science, University of California, Irvine, 1988.

AMBLER., S. **Agile Modeling: Effective Practices for Extreme Programming and the Unified Process**. Wiley, 224 p. ISBN: 0471202827, 2002.

BAUER, B., MULLER, J., ODELL, J. **Agent UML: A formalism for specifying multiagent interaction.** In Proc. of the 1st Int. Workshop on Agent-Oriented Software Engineering, AOSE'00, pages 91--104, Limerick, Ireland, 2001.

BELLIFEMINE, F., CAIRE, G., POGGI, A., RIMASSA, G. **JADE A White Paper.** Exp v. 3 n. 3, Sept., 2003, http://jade.tilab.com/

BRESCIANI, P. GIORGINI, P. Giunchiglia, F., Mylopoulos, J., Perini, A. **TROPOS: An Agent-Oriented Software Development Methodology.** In Journal of Autonomous Agents and Multi-Agent Systems. May 2004. Kluwer Academic Publishers.

BOOCH, G., RUMBAUGH, J., e JACOBSON, I. **UML Guia do Usuário.** Editora Campus. 1999.

BOOCH, G. **Software Components with Ada: Structures, Tools, and Subsystems**. Benjamin-Cummings, Redwood City, CA, 1987.

BOSCH, J. **Design & Use of Software Architectures: Adopting and evolving a product-line approach**. Addison.Wesley, 2000.

BRÖCKERS, A., CHRISTOPHER, M., LOTT, H., DIETER, M. **MVP-L Language Report Version 2**. Technical Report 265/95, Department of Computer Science, University of Kaiserslautern, Germany, 1995a.

BRÖCKERS, A. DIFFERDING, C., HOISL, B., KOLLNISCHKO, F., Lott, C., et al. **A graphical representation schema for the software process modeling language MVP-L**, 1995b.

BUSETTA, P., Ronnquist, R., Hodgson, A., Lucas, A. **Jack intelligent agents - components for intelligent agents in java**, Technical report, Agent Oriented Software Pty. Ltd., Melbourne, Australia, 1998.

CAVALCANTE, U. **Um ambiente de desenvolvimento de software para o reuso composicional no desenvolvimento de sistemas multiagente.** Proposta de Dissertação (Mestrado em Engenharia de Eletricidade – Área de Ciência da Computação) - Universidade Federal do Maranhão, São Luís, 2008.

CERNUZZI, L., COSSENTINO, M., and ZAMBONELLI, F. **Process models for agent-based development**. Engineering Applications of Artificial Intelligence, 2005.

CERNUZZI, L., ZAMBONELLI, F. **Multi-Agent Abstractions and Organizations and the i* Framework**, Proceedings of the 3rd International i* Workshop, pp. 17-20, 2008.

CLEMENTS, P., NORTHROP. L. **Software Product Lines: Practices and Patterns.** Addison-Wesley Professional, 2001.

CLYNCH, N., COLLIER, R. **SADAAM: Software Agent Development - An Agile Methodology**, International Workshop on Languages, methodologies, and Development tools for multi-agent Systems (LADS'007), 2007.

CHELLA, A. COSSENTINO, M. and SABATUCCI L. **Tools and patterns in designing multi-agent systems with passi.** In 6th WSEAS Int.Conf. on Telecommunications and Informatics (TELE-INFO 04), Cancun, Mexico, May 2004.

CHELLA, A., COSSENTINO, M., SABATUCCI L., SEIDITA, V. **Agile PASSI: An agile process for designing agents,** Journal of Comput. Syst. Scientific Enginering, v. 21, 2006.

COSSENTINO, M., POTTS, C. **PASSI: a process for specifying and implementing multi-agent systems using UML**, 2002.

COSSENTINO, M., SABATUCCI, L., SEIDITA, V. **SPEM description of the PASSI process rel. 0.3.4**, Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni,2003.

CRNKOVIC, I. **Component-Based Software Engineering - New Challenges in Software Development**, Journal of Computing and Information Technology, vol 11, pages 151-162, 2003.

CZARNECKI, K. **Generative Programming Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models**, Ph.D. Thesis, Department of Computer Science and Automation, Technical University of Ilmenau, 1998.

DAM, K., WINIKOFF, M. **Comparing Agent-Oriented Methodologies**, In: Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems, 2003.

D'SOUZA, D., WILLS A., **Objects, Components, And Frameworks with UML – The Catalysis Approach**, Addison-Wesley, 1998.

DRUMOND, L., GIRARDI, R., LEITE, A. Architectural **Design of a Multi-Agent Recommender System for the Legal Domain**, Eleventh International Conference on ARTIFICIAL INTELLIGENCE and LAW, Ed. ACM. a ser publicado. 04 a 08 de junho de 2007.

FAHMI., S. CHOI., H. **"Life Cycles for Component-Based Software Development"** citworkshops, pp. 637-642, 2008.

FIPA. Foundation for Intelligent Physical Agents. Disponível em: <http://www.fipa.org/>. Acessado em: 08 jan. 2007.

FUGGETTA, A. **Software Process: a Roadmap.** Proceedings of the Conference on the Future of Software Engineering, June 4-11, Limerick (Ireland), ACM Press, New York (USA), pp. 25-34, 2000.

FRIEDMAN-HILL E. **JESS in Action**, Manning Publications, 2003.

FRAKES, W., KANG, K., **Software Reuse Status and Future**. IEEE Trans. On Software Engineering, No 7, p. 529-536, 2005.

KOTONYA, G. SOMMERVILLE, I. HALL, S. **Towards A Classification Model for Component-Based Software Engineering Research**, euromicro, p. 43, 2003.

GHEZZI, C., JAZAYERI, M., MANDRIOLI, D., **Fundamentals of Software Engineering.** Prentice Hall International, Upper Saddle River, NJ (USA), 1991.

GIORGINI, P., KOLP, M., MYLOPOULOS, J., & PISTORE, M. **The Tropos methodology: An overview.** In F. Bergenti, M. P. Gleizes, & F. Zambonelli

(Eds.), Methodologies and software engineering for agent systems. Boston: Kluwer Academic Publishing, 2004.

GIRARDI, R. **Engenharia de Software baseada em Agentes**, Anais do IV Congresso Brasileiro de Ciência da Computação (CBCOMP 2004), Ed. UNIVALI, Itajaí, Santa Catarina, Brasil, pp. 913-937, 2004.

GIRARDI, R., FARIA, C. **An Ontology-Based Technique for the Specification of Domain and User Models in Multi-Agent Domain Engineering.** CLEI Electronic Journal, V. 7, N. 1, pp. 7, 2004.

GIRARDI, R., FARIA, C., MARINHO, L. **Ontology-based Domain Modeling of Multi-Agent Systems**. Proceedings of the Third International Workshop on Agent-Oriented Methodologies at International Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA 2004), Ed. Cesar Gonzalez-Perez, pp. 51--62. Vancouver, Canada. October 24th to 28th, 2004.

GIRARDI, R., LEITE. **A Knowledge-based Tool for Multi-Agent Domain Engineering**, Knowledge-Based Systems Journal, v. 21, Ed. Elsevier, pp. 604-611, 2008.

GIRARDI, R., MARINHO, L. **A Domain Model of Web Recommender Systems based on Usage Mining and Collaborative Filtering**, Requirements Engineering Journal, vol.12 n. 1, Ed. Springer-Verlag, pp. 23-40. London, 2007.

GIRARDI, R., LINDOSO, A. **DDEMAS: A Domain Design Technique for Multi-agent Domain Engineering**, Lecture Notes in Computer Science, Perspectives in Conceptual Modeling: ER 2005 Workshops CAOIS, BP-UML, CoMoGIS, eCOMO, and QoIS, Volume 3770/2005, ISSN 0302-9743., Ed. Springer-Verlag GmbH, pp. 141-150. Klagenfurt, Austria, 2005.

GLEIZES, P., MILLAN, T., PICARD, G. **ADELFE: Using SPEM Notation to Unify Agent Engineering Processes and Methodology**, Intitut de Recherche en Informatique de Toulouse IRIT/2003-10-R, 2003.

GRAHAM, I., HENDERSON-SALLERS B., YOUNESSI H., **The Open Process Specification**, Addison-Wesley, 1997.

GRUBER, T. **Towards Principles for the Design of Ontologies Used for Knowledge Sharing**. In: *International Journal of Human-Computer Studies*,1995.

GUARINO, N. **Formal Ontology and Information Systems**, Proceedings of FOIS'98, Trento, Italy. Amsterdan, IOS Press, pp. 3-15, 6-8 June, 1998.

HAPPEL, H. SEEDOF, S. **Applications of Ontologies in Software Engineering**, 2nd Int. Workshop on Semantic Web Enabled Software Engineering (SWESE 2006).

HENDERSON-SELLERS, B., GIORGINI, P. (eds.), **Agent-Oriented Methodologies**. Idea Group Publishing, 2005.

JACCHERI, L., PICCO, G., LAGO, P. **Eliciting Software Process Models with the E3 Language**, ACM Transactions on Software Engineering and Methodology, vol. 7, no. 4, 1998.

HERZUM, P., e OLIVER S., **Business Components Factory: A Comprehensive Overview of Component-Based Development for the Enterprise**, John Wiley & Sons, Incorporated, 1999.

HOPKINS, J. **Component Primer**, Communications of the ACM, 43(10), pp. 27-30, October 2000.

P. HUDAK. **Modular Domain Specific Languages and Tools**. In Fifth International Conference on Software Reuse, pages 134--142, Victoria, Canada, June 1998.

JACCHERI, M. L., STÅLHANE, T. **Evaluation of the E3 Process modelling language and tool for the purpose of model creation**, Lecture Notes in Computer Science, V. 2188, 2001.

KROLL, P., KRUCHTEN P. **The Rational Unified Process Made Easy: A Practitioner ' s Guide to the RUP**, Addison Wesley, 2003.

KRUEGER, C. W. 1992. Software reuse. ACM Comput. Surv. 24, 2, 131-183. DOI= http://doi.acm.org/10.1145/130844.130856, 1992.

LEMOINE, M. GAUDI, G. **From UML to Z**, In book Formal methods for embedded distributed systems: how to master the complexity, isbn 1-4020-7996-6, pp.65-88, Kluwer Academic Publishers, Norwell, MA, USA, 2004.

LEITE, A., GIRARDI, A. **ONTORMAS: Uma ferramenta dirigida por ontologias para a Engenharia de Domínio e de Aplicações Multiagente**, The XI

Iberoamerican Workshop on Requirements Engineering and Software Environment , Pernambuco, 2008.

LEITE, A., GIRARDI, R. **ProEDM: Um processo para a Engenharia de Domínio Multiagente**, Versão 1.0. Relatório Técnico. Fevereiro, 2007a. Disponível em: < http://gesec.deinf.ufma.br/publicacoes/detalhes/index.php?id=109>. Acessado em: nov., 2008.

LEITE, A., GIRARDI, R. **ProEAM: Um processo para a Engenharia de Aplicações Multiagente**, Relatório Técnico. Fevereiro, 2007b. Disponível em: <http://gesec.deinf.ufma.br/publicacoes/detalhes/index.php?id=113>. Acessado em: nov., 2008.

LEITE, A., GIRARDI, R. **MADAE-Pro: Processo para a Engenharia de Domínio e de Aplicações Multiagente** , Versão 1.0. Relatório Técnico. Fevereiro, 2008. Disponível em:<http://gesec.deinf.ufma.br/publicacoes/detalhes/index.php?id=119 >. Acessado em: nov., 2008.

LEITE, A. GIRARDI, R. CAVALCANTE, U.: **An Ontology for Multi-Agent Domain and Application Engineering**. In: The 2008 IEEE International Conference on Information Reuse and Integration (IEEE IRI-08), p. 98-103, Las Vegas. Proceedings of the 2008 IEEE International Conference on Information Reuse and Integration, 2008a.

LEITE, A., GIRARDI, R. CAVALCANTE, U. **MAAEM: A Multi-agent Application Engineering Methodology,** Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE), Knowledge Systems Institute, Redwood City, Boston, 2008b.

LEITE, A., GIRARDI, R. Adriana Leite e Rosario Girardi. **A Process For Multi-Agent Domain And Application Engineering: The Domain Analysis And Application Requirements Engineering Phases**, Proceedings of the 11th International Conference on Enterprise Information Systems, Ed. INSTIIC. Milan, Italy, 2009.

LINDOSO, A. **Uma metodologia baseada em ontologias para a engenharia de aplicações multiagente.** Dissertação (Mestrado em Engenharia de Eletricidade – Área de Ciência da Computação) - Universidade Federal do Maranhão, São Luís, 2006.

LINDOSO, A. GIRARDI, R. **Uma Técnica baseada em Ontologias para o Reuso de Padrões de Software e de Frameworks no Projeto de Aplicações Multiagente**, First Workshop on Software Engineering for Agent-oriented Systems (SEAS 2005), 19º Simpósio Brasileiro de Engenharia de Software (XIX SBES). Uberlândia, Minas Gerais, Brasil, 2005.

LINDOSO, A. GIRARDI, R. **The SRAMO Technique for Analysis and Reuse of Requirements in Multi-agent Application Engineering**, IX Workshop on Requirement Engineering, Cadernos do IME, v. 20, Ed. UERJ, pp. 41-50. Rio de Janeiro, 2006.

MARIANO, R., GIRARDI, R., LEITE, A., DRUMOND, L. MARANHÃO, D. **A Case Study on Domain Analysis of Semantic Web Multi-agent Recommender Systems.** In: Proceedings 3th International Conference on Software and Data Technologies, Porto. Portugal, 2008.

MELLOR, S., SCOTT, K., UHL, A. WEISE, D. **MDA distilled: principles of**

Model-Driven Architecture. Addison-Wesley, 2004.

MICROSOFT VISIO 2003. Disponível em: <http://www.microsoft.com/brasil/office/visio/default.asp>. Acessado em: 01 jul. 2008.

MOHAMED, AHMED-NACER, M. 2001. **Towards a New Approach on Software Process Evolution.** In Proceedings of the ACS/IEEE international Conference on Computer Systems and Applications (June 25 - 29, 2001). AICCSA. IEEE Computer Society, Washington, DC, 345.

PATTON, J. **Don't Know What I Want, But I Know How to Get It**. Disponível em: < http://www.agileproductdesign.com/blog/dont_know_what_i_want.html>. Acessado em: 01 nov. 2008.

PERKINS, D. **SPSL Language Reference Manual", Department of Computational Science, University of Saskatchewan**, September, 1978.

POHL, K., BÖCKLE, G., VAN DER LINDEN, F., **Software Product Line Engineering – Foundations, Principles, and Techniques**. Springer, Heidelberg 2005.

PRESSMAN, R. S. **Engenharia de Software**. 5a edição, Editora McGraw-Hill, 2002

PRIETO-DÍAZ, R. 1993. **Status Report: Software Reusability**. *IEEE Softw.* 10, 3 (May. 1993), 61-66. DOI= http://dx.doi.org/10.1109/52.210605.

PROTÉGÉ. Disponível em: <http://protege.stanford.edu>, Acessado em: 19 jul. 2008.

RISING, L., JANOFF, N. **The Scrum Software Development Process for Small Teams**. IEEE Software, July/August 2001.

ROLLAND, C. **Modeling the Requirements Engineering Process, 3rd European-Japanese Seminar on Information Modelling and Knowledge Bases**, 1993.

ROSE - Rational Rose Enterprise. Disponível em: < www.rational.com/products/rose/>. Acessado em: 10 mai. 2008.

SAEKI, M., HORAI, H., and ENOMOTO, H. 1989. **Software development process from natural language specification.** In Proceedings of the 11th international Conference on Software Engineering (Pittsburgh, Pennsylvania, United States). ICSE '89. ACM, New York, NY, 64-73. DOI= http://doi.acm.org/10.1145/74587.74594

SCHLENOFF, C., GRUNINGER M., TISSOT, F., VALOIS, J., LUBELL, J., LEE, J., **The Process Specification Language (PSL): Overview and Version 1.0 Specification**, NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD., 2000.

SOFTHOUSE. Scrum in five minutes. Disponível em: <http://www.softhouse.se/Uploades/Scrum_eng_webb.pdf>., Acessado em: 10 mai. 2008.

SOMMERVILLE, I. **Engenharia de Software**, 6 ed. Tradução: André Maurício de Andrade Ribeiro. Addison-Wesley, 2004.

SPEM - Software Process Engineering Metamodel Specification. Disponível em: <http://www.omg.org/technology/documents/formal/spem.htm>. Acessado em: 30 jun. 2008.

STURM A., SHEHORY, O. **A Framework for Evaluating Agent-Oriented Methodologies**. In M. Winikoff P. Giorgini, B. Henderson-Sellers, editor, Proc. of the Int. Bi-Conference Workshop on Agent-Oriented Information Systems, AOIS 2003.

USCHOLD, M. **Building Ontologies: Towards A Unified Methodology**. Proceedings Expert Systems 96. Cambridge, 1996.

OPEN - Object-oriented Process, Environment, and Notation. Disponível em: < http://www.open.org.au/>. Acessado em: 20 jul. 2008.

XP – Extreme Programming. Disponível em: <http://www.extremeprogramming.org/rules/commit.html>. Acessado em: 30 jun. 2008.

WAUTELET, Y., KOLP, M., NGUYEN T., **Agile Tropos. Adopting Industry Best Software Development Practices in Agent Oriented Development**, Forum des Recherches Doctorales, Louvain School of Management (LSM), December, 2006.