Pós-Graduação em Ciência da Computação

"U-TROPOS: uma proposta de processo unificado para apoiar o desenvolvimento de software orientado a agentes"

Por

MARIA JOCELIA SILVA

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, Outubro / 2008

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MARIA JOCELIA SILVA

"U-TROPOS: uma proposta de processo unificado para apoiar o desenvolvimento de software orientado a agentes"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR: JAELSON FREIRE BRELAZ DE CASTRO
CO-ORIENTADORA: FERNANDA MARIA RIBEIRO DE ALENCAR

RECIFE, OUTUBRO / 2008

Silva, Maria Jocelia

U-TROPOS: uma proposta de processo unificado para apoiar o desenvolvimento de software orientado a agentes / Maria Jocelia Silva. – Recife: O Autor, 2008.

xii, 177 folhas : il., fig., tab.

Dissertação (mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação, 2008.

Inclui bibliografia e apêndices.

Engenharia de software. I. Título.

005.1 CDD (22.ed.) MEI2008-090

Resumo

Atualmente existe muita expectativa em relação à tecnologia de agentes, que surge como um paradigma promissor para a engenharia de sistemas de software. Agentes são importantes e úteis porque permitem um tratamento adequado do fluxo de controle de sistemas altamente distribuídos, oferecem um mecanismo que facilita o surgimento de comportamento emergente, bem como incorporam as melhores práticas de como organizar entidades colaborativas concorrentes. Com o aumento da complexidade das aplicações baseadas em agentes, torna-se importante o uso de notações, ferramentas e metodologias específicas para seu o desenvolvimento. Nos últimos anos foram propostas várias metodologias e notações para desenvolvimento orientado a agentes, tais como Tropos. Trata-se de abordagem centrada em requisitos, que se preocupa em capturar a complexidade de aspectos sociais. Tropos é considerada uma das metodologias mais completas, pois abrange todas as fases do ciclo de desenvolvimento de sistemas multiagentes. Desde a concepção inicial do Tropos em 2000, várias versões e extensões foram propostas. Contudo, estas propostas têm adotado atividades e notações diferentes, dificultando sua adoção pelos desenvolvedores de software. Além disso, não existe um processo padronizado e unificado que oriente as suas atividades de desenvolvimento orientado a agentes. O objetivo principal desta dissertação é especificar um processo unificado de desenvolvimento orientado a agentes conforme algumas propostas existentes do projeto Tropos. A especificação será baseada na linguagem SPEM de especificação de processos. Será proposto tanto um modelo estrutural como um modelo dinâmico do processo. A base para formulação do modelo estrutural do processo é a integração de atividades extraídas das duas versões de Tropos com métodos criados para as fases de projeto arquitetural e projeto detalhado. O modelo dinâmico sugere um ciclo de vida orientado pelo modelo de desenvolvimento iterativo e incremental. O resultado é o U-Tropos: uma proposta de processo unificado para apoiar o desenvolvimento de software orientado a agentes.

Palavras chaves: Engenharia de software, Desenvolvimento de sistemas multiagentes, Metodologia Tropos

Abstract

Nowadays, there is great expectation related to agent technology, which emerges as a promising paradigm for software systems engineering. Agents are important and useful because they allow appropriate treatment of the flow of highly distributed control systems, provide a mechanism that facilitates the emergent behavior, and incorporate the best practices of collaborative entities to organize competitors. With the increasing complexity of agent-based applications, the use of specific ratings, tools and methodologies for their development is important. In recent years, methodologies and ratings for agent oriented development have been proposed, such as Tropos, which is a requirements-driven approach that captures the complexity of social aspects. Tropos is considered one of the most complete methodologies, therefore covers all stages of multiagent systems development. Since the Tropos initial design in 2000, various versions and extensions were proposed. However, these proposals have adopted different activities and ratings, hindering their adoption by software developers. Moreover, there is not a unified and standardized process to guide their agent oriented development activities. The main aim of this dissertation is to specify an agent development unified process based on some existing Tropos project proposals. The specification will use SPEM (a processes specification language). Both a process structural model and a process dynamic model will be offered. The basis for formulating the structural model is the integration of activities drawn from two Tropos versions. It will add methods designed for architectural design and detailed design phases. The dynamic model suggests a life cycle driven by the iterative and incremental development model. The result is the U-Tropos: a proposal for a unified process to support the agent oriented software development.

Key words: Software engineering, Multiagent system development, Tropos methodology.

Agradecimentos

Agradeço

a Deus, por todas as coisas;

a minha família, por ter me ensinado a chegar até aqui;

a meus amigos, pela força e apoio recebidos;

a meus amigos e colegas do CIN, pelo companheirismo e colaboração;

a meus amigos e colegas do CEFET-PE, pela colaboração e incentivo;

a minha co-orientadora Fernanda Alencar, por seu apoio e conhecimento compartilhado, que foram fundamentais em todas as fases deste trabalho;

a meu orientador Jaelson Castro pela oportunidade de participar deste conceituado grupo de trabalho;

a todos do Centro de Informática da UFPE, pela ótima estrutura para formação de profissionais e pesquisadores;

a todos que apoiaram, incentivaram e contribuíram para a realização deste trabalho.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

ADL – Architectural Description Language

AIP - Agent Interation Protocol

AO - Orientado a agentes

AOSE – Agent Oriented Software Engineering

AUML - Agent UML

BDI - Belief Desire Intention

CAME - Computer aided method engineering

CAPE - Computer aided process engineering

CASE - Computer aided software engineering

DIS - Detailed Interaction Script

FIPA - Foundation for Intelligent Physical Agents

LEL - Lexicon extended language

MSC - Message Sequence Charts

OMG - Object Management Group, Inc

OMT - Object-Modeling Technique

OO - Orientado a objetos

OPF – Open Proccess Framework

PRiM - Process Reengineering i* Methodology

RDD - Responsibility Driving Design

RiSD - Reduced i* SD models

RUP - Rational Unified Process

SD - Strategical Dependency

SDL - Specification and Description Language

SDSituation - Strategical Dependency Situation

SEP - Software Engineering Process

SIRA - Systematic Integration between Requirements and Architecture

SMA - Sistemas Multiagentes

SPEM - Software Process Engineering Metamodel

SR - Strategical Rationale

UML - Unified Modeling Language

Capítulo 1 - Introdução

Neste capítulo são apresentadas as principais motivações para realização desta dissertação e a definição do escopo/contexto em que este trabalho de pesquisa está inserido. Em seguida são definidos os objetivos do trabalho, e finalmente a sua estrutura.

1.1 Motivação

A abordagem de desenvolvimento de softwares orientado a agentes tem sido bastante explorada. Muitos trabalhos têm sido realizados para criação de processos, metodologias e técnicas que permitam a modelagem e construção destes softwares. Isto tem acontecido devido à natureza de problemas a qual o uso de agentes é indicado: Aplicações que requerem sistemas que possam tomar decisões para satisfazer seus objetivos (e.g. sistemas de recuperação de falhas, controle de tráfego, diagnóstico, etc.). Um agente é um sistema de computador situado em algum ambiente e que é capaz de ações autônomas neste ambiente para atingir seus objetivos. O agente inteligente deve operar robustamente em ambientes abertos, imprevisíveis, mutáveis e com ações que possam falhar [Wooldridge, 2002]. Agentes não são sistemas isolados. Em muitas situações eles coexistem e interagem com outros agentes. Um sistema que consiste de um grupo de agentes que podem interagir com outros é chamado Sistema Multiagentes (SMA). Tropos [Castro; Kolp & Mylopoulos, 2002] é uma das abordagens populares para desenvolvimento de software orientado a agentes e tem sido objeto de pesquisa do laboratório de engenharia de requisitos, onde esta dissertação é desenvolvida.

Tropos adota os conceitos oferecidos pelo framework i* [Yu, 1995], tais como ator, agente, posição e papel assim como as dependências sociais entre os atores, incluindo dependências de metas, tarefas e recursos. Ela usa estes conceitos não somente para modelar a fase de análise de requisitos, mas também as fases de arquitetura e projeto detalhado. Tropos foi criado em 2000 [Mylopoulos & Castro, 2000]. Esta proposta original foi evoluída e existem as versões [Castro; Kolp & Mylopoulos, 2002] e [Bresciani et al., 2004].

Tropos tem sido muito estudada por pesquisadores e desenvolvedores de software multiagentes e diversos trabalhos[1] têm sido gerados para incluir melhorias em sua proposta inicial. Porém estes trabalhos têm sido construídos em formatos e notações diferentes dificultando o seu uso na comunidade de desenvolvimento de Sistemas Multiagentes (SMA). Os desenvolvedores têm um esforço maior em unificar e integrar todos estes trabalhos, o que compromete a qualidade dos produtos gerados. Além disso, não existe um processo padronizado que oriente as suas atividades, produzindo também um esforço extra na organização do trabalho, o que pode ocasionar perda de produtividade e retrabalho. Neste

[1] www.troposproject.net

contexto, esta dissertação de mestrado propõe um processo unificado, chamado U-Tropos, que de forma iterativa e incremental, define um conjunto de atividades para guiar o desenvolvimento de sistemas multiagentes.

1.2 Escopo

O foco principal desta dissertação é a proposta Tropos e suas variantes, que propõe uma metodologia de desenvolvimento centrada nos conceitos de requisitos iniciais e suporta as seguintes fases de desenvolvimento de um software: Requisitos Iniciais, Requisitos Finais, Projeto Arquitetural, Projeto Detalhado e Implementação. Vale salientar que existem duas versões muito populares de Tropos que estão sendo usadas em diferentes grupos de pesquisas. Estas versões apresentam diferenças em suas abordagens. Além disto, várias técnicas de melhorias que foram propostas recentemente ainda não foram incorporadas ao Tropos. Uma dificuldade para adoção de Tropos é a inexistência de um modelo de processo de software explícito, que oriente o desenvolvimento de sistemas, conforme as melhores práticas de Tropos e os recentes avanços.

O escopo desta dissertação envolve uma análise das duas versões do Tropos [Castro; Kolp & Mylopoulos, 2002] e [Bresciani et al., 2004] para extração das melhores práticas com o objetivo de formar um novo processo de desenvolvimento baseado em Tropos. Além disso, algumas técnicas desenvolvidas para detalhar as atividades de requisitos, projeto arquitetural e projeto detalhados serão acrescentadas. Por último, é proposta a especificação de um modelo de processo em uma linguagem padronizada (SPEM [SPEM, 2007]) para orientar o desenvolvimento de sistemas multiagentes. O processo sugere atividades de gerenciamento, verificação e suporte, mas não faz parte do escopo desta dissertação o detalhamento destas atividades.

1.3 Objetivos

O objetivo principal desta dissertação é especificar um processo (U-Tropos) para organizar as fases de requisitos, projeto arquitetural e projeto detalhado da abordagem Tropos unificando as atividades extraídas das suas versões com técnicas recentes que possibilitam maior detalhamento destas atividades.

Os objetivos específicos para cumprir com a proposta desta dissertação são relacionados a seguir:

- Analisar as versões da abordagem Tropos para identificar uma seqüência lógica de atividades para o processo.
- Analisar as técnicas que estendem as fases de requisitos, projeto arquitetural e projeto detalhado para identificar como estas podem contribuir para detalhamento da especificação do processo.
- Identificar guias e diretrizes que contribuam para tornar mais clara as técnicas, ferramentas e modelos utilizados na metodologia.
- Descrever um modelo de processo iterativo e incremental para o desenvolvimento das atividades usando a linguagem de especificação SPEM.
- Exemplificar o uso do processo proposto em um sistema multiagente (DBSitter-AS).

1.4 Estrutura da dissertação

Além deste capítulo introdutório esta dissertação contém mais seis capítulos:

Capítulo 2 – Desenvolvimento de Software Orientado a Agentes: Neste capítulo são apresentados os conceitos da engenharia de software orientada a agentes e a motivação para o uso de agentes e sistemas multiagentes. Em seguida são revisadas algumas metodologias atualmente em uso para dar apoio ao desenvolvimento de sistemas multiagentes.

Capítulo 3 – O projeto Tropos: o projeto Tropos propõe uma das abordagens mais completas para o desenvolvimento de sistemas multiagentes. Este capítulo apresenta a visão geral do Tropos, suas fases, alguns conceitos básicos e as técnicas que compõe a proposta. Também são apresentadas novas técnicas e diretrizes que surgiram com o objetivo de melhorar o trabalho original.

Capítulo 4 - Processo de desenvolvimento de Software: Neste capítulo são apresentados os conceitos e modelos de processos usados para o desenvolvimento de software. Atenção especial é dada aos modelos de processo para desenvolvimento de sistemas multiagentes. O SPEM, que é uma abordagem bastante popular para descrição de processos de desenvolvimento de software, também é apresentado. Por fim, é indicado como Tropos poderá ser descrito através de SPEM.

Capítulo 5 – U-Tropos: Nesse capítulo é apresentada a principal contribuição desta dissertação, o processo U-Tropos. O U-Tropos é um processo interativo e incremental

especificado em SPEM que integra diversas técnicas, métodos e diretrizes às atividades da abordagem Tropos.

Capítulo 6 - Exemplo de aplicação do processo: Esse capítulo tem o objetivo de apresentar um exemplo para uso prático da proposta desta dissertação. Será utilizado o desenvolvimento do sistema multiagentes DBSitter-AS, que tem como objetivo a detecção e correção autônoma de falhas em banco de dados. Nesse contexto, é apresentado um guia para a aplicação do processo dividido em iterações que geram os produtos de trabalho das atividades de requisitos, projeto arquitetural e projeto detalhado.

Capítulo 7 - Conclusões e trabalhos futuros: Neste capítulo são apresentadas as considerações finais sobre o desenvolvimento deste trabalho, assim como as principais contribuições, abrangência e limitações encontradas. Serão mostrados alguns trabalhos futuros que complementem esta proposta e que possam direcionar futuras pesquisas nesta área.

Capítulo 2 - Desenvolvimento de Software Orientado a Agentes

A engenharia de software orientada a agentes tem sido objeto de várias pesquisas e aplicações para o desenvolvimento de sistemas complexos que exigem características como autonomia e aprendizagem. Este capítulo apresenta os conceitos e a motivação para o uso de agentes e sistemas multiagente e em seguida, as metodologias atualmente em uso para apoiar o desenvolvimento de sistemas multiagentes.

2.1 Introdução

Os sistemas de computadores da atualidade, geralmente são grandes sistemas distribuídos e interconectados por redes de computadores. Está se tornando raro encontrar sistemas comerciais ou acadêmicos que não sejam conectados a internet. As tarefas que os homens são capazes de automatizar e delegar aos computadores se tornam cada vez mais complexas. Quanto mais complexa uma tarefa, mais existe a tendência desta ser delegada a um sistema de computadores. Mas para que esta atribuição ocorra com segurança, é necessário atribuir aos sistemas conceitos e metáforas que refletem como os humanos entendem o mundo. Isto implica que os sistemas de computadores devem ter a habilidade de operar independente da intervenção humana e a habilidade de representar interesses humanos específicos quando interagindo com pessoas ou outros sistemas.

A partir deste contexto, surgiu um novo campo na ciência da computação: Os sistemas orientados a agentes [Wooldridge, 2002]. Um agente é um sistema de computador situado em algum ambiente, e que é capaz de ações autônomas neste ambiente para alcançar os seus objetivos. Um agente não tem controle completo sobre o ambiente, ele tem apenas um controle parcial naquilo que ele pode influenciar. Uma agente monitora seu ambiente e tem um conjunto de ações que ele pode executar. Estas ações são as capacidades do agente de modificar o seu ambiente. Cada ação tem precondições que determinam em quais situações as ações podem ser executadas. Um agente é considerado inteligente quando possui características como reatividade, pro atividade e habilidade social.

Um sistema multiagentes é um sistema computacional que consiste de um conjunto de agentes que interagem entre si ou trabalham juntamente para atingir suas metas. Em um sistema multiagentes, os agentes se relacionam através de delegação de tarefas, transferências de dados, compromissos, sincronização de ações, etc. Estes relacionamentos são possíveis dentro de uma organização que prover suporte para distribuição de tarefas, dados, recursos e a coordenação de ações [Ferber, 1999].

A tecnologia de agentes saiu da esfera de pesquisas em universidades e laboratórios e está sendo aplicada em diversas áreas comerciais, industriais e de entretenimento[2]. Existem diversos aplicativos em uso e cada vez mais está se usando esta tecnologia para

[2] Veja tendências em: AAMAS'08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track. Publisher International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC. Estoril, Portugal. 2008.

desenvolvimento de sistemas complexos. As principais áreas de aplicação na indústria são: Fabricação, Controle de processo, Telecomunicações, Controle de tráfego aéreo e Sistemas de transportes. No comércio, pode ser citada a Gerência de informação, o Comércio eletrônico e a Gerência de processo de negócio como as principais aplicações que fazem uso de agentes. Na área de entretenimento, os agentes são usados em jogos, cinema interativo e realidade virtual. Também são úteis na área médica para monitoramento de pacientes e planos de saúde.

Um paradigma de desenvolvimento de sistemas para se tornar mais estabelecido na comunidade da Ciência da Computação deve prover tecnologias para suportar o seu desenvolvimento. Muitas contribuições têm sido feitas para a engenharia de software orientada a agentes. Existe uma grande quantidade de metodologias (e.g. GAIA [Zambonelli; Jennings & Wooldridge, 2003], Tropos [Mylopoulos & Castro, 2000], MASE [Wood & DeLoach, 2001]), framework de desenvolvimento (e.g. JACK [JACK, 2007], JADE [Bellifemine et al., 2003], JADEX [Pokahr; Braubach & Lamersdorf, 2005], Whitestein's Living Systems [Whitestein, 2008]) e aplicações do mundo real, como por exemplo, soluções providas pelo Whitestein Information Technology Group [Whitestein, 2008].

O objetivo deste capítulo é estudar as metodologias usadas na engenharia de sistemas orientados a agentes. A próxima seção apresenta as principais metodologias e são citadas algumas ferramentas de desenvolvimento usadas por estas metodologias.

2.2 Metodologias para desenvolvimento de SMA

Várias metodologias para o desenvolvimento de sistemas orientados a agentes têm sido propostas nos últimos anos. Estas podem ser divididas em duas correntes: As metodologias inspiradas ou adaptadas do desenvolvimento orientado a objetos e as metodologias que adaptam a engenharia de conhecimento ou outras técnicas.

Henderson-Sellers & Giorgini (2005) fizeram um estudo da genealogia das metodologias orientadas a agentes mais conhecidas (Figura 2-1). Neste estudo identificou-se que várias metodologias reconhecem uma descendência direta de métodos orientados a objetos. Em particular, MASE [Wood & DeLoach, 2000] recebe influência de Kendall, Malkoun & Jiang (1995) e herda muitas características de AAII [Kinny; Georgeff & Rao, 1996]. O-MaSE [DeLoach, 2006] é uma extensão de MaSE acrescentando conceitos de interação e organização. AAII, por sua vez foi influenciada pela OMT [Rumbaugh et al., 1991] que é uma metodologia orientada a objetos. Da mesma forma a Metodologia OO Fusion [Coleman et al., 1994] foi uma forte influência para o projeto de GAIA [Zambonelli;

Jennings & Wooldridge, 2003], que por sua vez influenciou SODA [Omicini, 2000]. Duas outras abordagens têm sido usadas como base para extensões orientadas a agentes. O RUP [Kruchten, 1999] influenciou ADELFE [Bernon et al., 2002] e MESSAGE [Caire et al., 2001]. Prometheus [Padgham & Winikoff, 2002] também usa diagramas e conceitos OO quando estes são compatíveis com o paradigma orientado a agentes. E PASSI [Cossentino & Potts, 2002] unifica idéias de OO e SMA, usando UML como sua principal notação. MAS-CommonKADS [Iglesias et al., 1998] é baseada em conceitos da inteligência artificial, embora ainda assuma que tem influência da OMT.

Algumas metodologias não reconhecem qualquer influência da orientação a objetos. Por exemplo, a metodologia Tropos [Mylopoulos & Castro, 2000] é fortemente influenciada pelos conceitos do i* [Yu, 1995]. Outras abordagens não assumem descendência direta da metodologia OO, como por exemplo, Nemo [Huget, 2002], MASSIVE [Lind, 2000], Cassiopeia [Collinot; Drogoul & Benhamou, 1996]. A abordagem OPEN [Graham; Henderson-Sellers & Younessi, 1997] para o desenvolvimento OO tem também sido estendida para suportar agentes. Esta abordagem é chamada Open / Agents [Debenham & Henderson-Sellers, 2002].

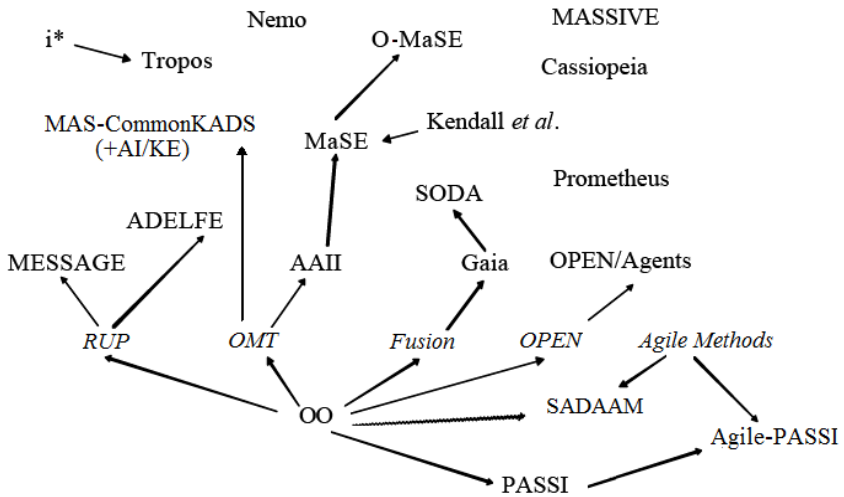


Figura 2-1 – Genealogia de algumas metodologias AO. Adaptado de Henderson-Sellers & Giorgini (2005)

Recentemente, o desenvolvimento orientado a agentes tem sido influenciado pelas práticas do desenvolvimento ágil [Cockburn, 2000]. A abordagem Agile-PASSI [Chella et al., 2004] é uma versão ágil de PASSI. A metodologia SADAAM [Clynch & Collier, 2007] também utiliza técnicas derivadas de uma variedade de métodos ágeis incluindo o manifesto

ágil [AGILE, 2008], a modelagem ágil [Ambler, 2008], Extreme Programming [Beck, 1999] e desenvolvimento orientado a testes (TDD - Test-Driven Development) [Beck, 2003].

Em seguida são apresentadas as características principais de algumas técnicas e metodologias. Inicia-se pela linguagem de modelagem Agent UML (AUML) que é uma extensão para a UML e é muito usada como notação em muitas metodologias orientadas a agentes. Depois são apresentadas brevemente algumas metodologias: MAS-CommonKADS por ter origem na engenharia do conhecimento e inteligência artificial. Gaia e MaSE são metodologias que cobrem as fases de análise e projeto e se tornaram muito populares, sendo citadas como referência em vários trabalhos de pesquisas em engenharia de software orientada a agentes. PASSI e Tropos têm a característica de abordar todo o ciclo de desenvolvimento desde os requisitos até a implementação. Para a descrição gráfica das fases das diversas metodologias a seguir, será utilizada a notação SPEM [SPEM, 2007].

2.2.1 AUML (Agent UML)

Os diagramas de UML (Unified Modelling Language) [UML, 2007] foram projetados para suportar a descrição dos aspectos inerentes a orientação a objetos. Em Odell et al. (2000), um agente é uma extensão do conceito de objeto, o que permite explorar a UML para representar características do paradigma Orientado a Agentes. Odell et al. (2000) afirma que a UML é insuficiente para modelagem de agentes e sistemas baseados em agentes e propõe extensões à UML, criando assim a AUML (Agent UML).

A AUML propõe um subconjunto de extensões para a UML com o objetivo de especificar protocolos de interação de agentes e outras noções baseadas em agentes. Um protocolo de interação de agentes (AIP – Agent Interation Protocol) descreve um padrão de comunicação como uma seqüencia de mensagens entre agentes e as restrições do conteúdo destas mensagens. A especificação de AIP resolve problemas na fase de análise e projeto de sistemas e prover uma semântica formal e notação gráfica amigável.

Usam-se pacotes da UML para representar *templates* para o AIP com o objetivo de focar na finalidade da interação e não na ordem precisa das mensagens trocadas. Diagramas de seqüência (Figura 2-2) são usados para modelar elementos básicos de comunicação entre agentes. Por exemplo, a Figura 2-2 mostra um protocolo de interação genérico, modelado na forma de um template. São indicados os papéis dos agentes (Iniciador e Participante), possíveis restrições durante as interações (e.g. a existência de prazos) e as ações de comunicação presentes no protocolo de comunicação (e.g. chamada-de-proposta, recusa, etc).

Algumas extensões são adicionadas para expressar os agentes e seus papéis, atos de comunicação de agentes e *threads* de interação são representados para suportar concorrência. Diagramas de colaboração também podem ser usados para descrever um padrão de interação entre agentes.

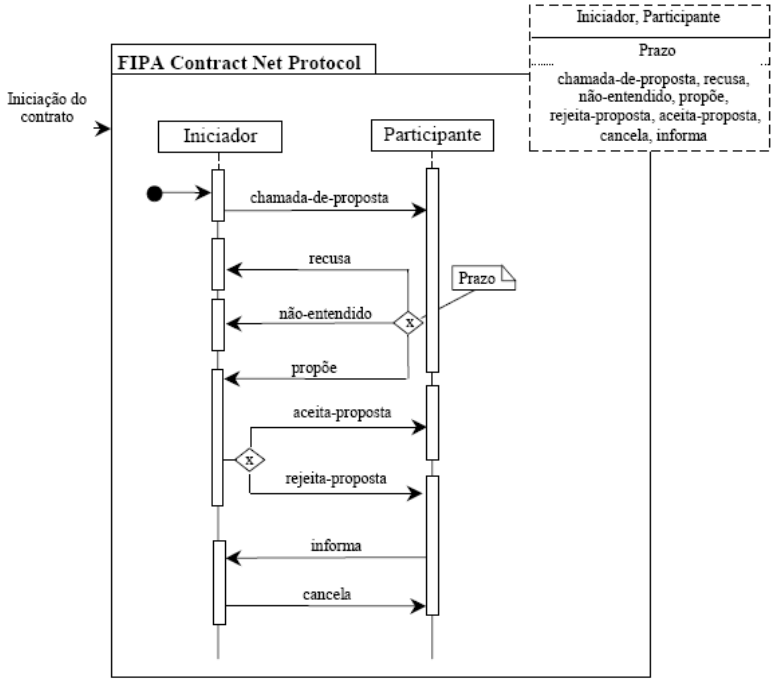


Figura 2-2 – Diagrama de seqüência representando um protocolo de interação entre agentes. Adaptado de Odell et al.(2000)

O diagrama de atividade é usado para especificar semânticas de *thread* de processamento mais claras expressando as operações e os eventos que as iniciam. Por exemplo, na Figura 2-3, o Agente Vendedor aceita, monta, envia e fecha o pedido. A execução começa no estado de partida, representado por um círculo vazado e termina no estado final, representado por um círculo preenchido. O diagrama de atividade é especialmente útil em protocolos complexos que envolvam processamento concorrente. Também pode ser usado para expressar o processamento interno de um agente simples. O diagrama de estado geralmente não é usado para expressar AIP por que ele é centrado em estados e não em agentes (ou processos). Contudo um agente poderia embutir as restrições de transição de estados garantindo que as restrições do protocolo de interação sejam encontradas. Portanto, ele é mais apropriado para expressar o comportamento interno do agente.

A AUML também adiciona uma forma de representar papéis de agentes aos modelos UML existentes. Os diagramas de seqüência, colaboração e atividades recebem estas mudanças. Os modelos da UML original também poderiam ser utilizados, mas ficariam

complexos e ilegíveis. Diagramas de distribuição (implantação) também podem ser usados e estendidos para expressar mobilidade de agentes.

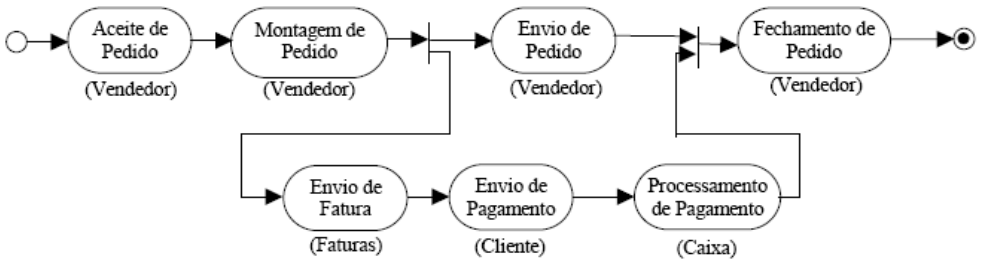


Figura 2-3 – Diagrama de atividade representando atividades e papel de agentes. Adaptado de Odell et al.(2000)

2.2.2 MAS-CommonKADS

MAS-CommonKADS [Iglesias, 1998] é uma extensão da metodologia CommonKADS [Schreiber et al., 1999] com o acréscimo de características relevantes aos sistemas multiagentes. O CommonKADS é uma metodologia para desenvolvimento de sistemas baseados em conhecimento. O MAS-CommonKADS adicionou ao CommonKADS técnicas de metodologias OO tais como a OMT [Rumbaugh et al., 1991] e RDD (Responsibility Driving Design) [Wirfs-Brock; Wilkerson & Wiener, 1990] e técnicas da engenharia de protocolos para descrever protocolos de agentes tais como SDL (Specification and Description Language) [Belina & Hogrefe, 1999] e MSC (Message Sequence Charts) [Rudolph; Grabowski & Graubmann,1996]. O modelo de processo da metodologia combina a abordagem orientada a riscos com a abordagem orientada a componentes.

MAS-CommonKADS segue o modelo de ciclo de vida proposto por CommonKADS e engloba as seguintes fases (Figura 2-4): (i) fase de Conceituação, que envolve as atividades de elicitação dos requisitos e a geração de casos de uso para descrever os requisitos informais e testar o sistema; (ii) fase de Análise, que determina os requisitos do sistema partindo do enunciado do problema. São gerados cinco modelos na fase de análise: o modelo de organização é usado para analisar a organização humana em que o sistema será inserido e descreve a organização dos agentes de sistema e sua relação com o meio; o modelo de tarefas descreve as tarefas que os agentes realizam, os objetivos de cada tarefa, sua decomposição e os métodos de resolução de problemas para resolver cada objetivo; o modelo de agentes especifica a capacidade de raciocínio dos agentes, suas habilidades, serviços providos, sensores, grupos de agentes a que pertence e a classe de agente. Um agente pode ser um agente humano, software, ou qualquer entidade capaz de empregar uma linguagem de comunicação de agentes; o modelo de comunicação descreve as interações entre um agente

humano e um agente software; o modelo de coordenação descreve as interações entre agentes de software; e o modelo de experiência descreve o conhecimento necessário aos agentes para atingir seus objetivos; (iii) fase de Projeto, que determina a arquitetura da rede multiagentes e de cada agente. Utiliza o modelo de projeto que descreve a arquitetura e o projeto do sistema multiagentes como passo prévio a sua implementação; na (iv) fase de Codificação cada agente é implementado e testado e na fase de (v) Integração, o sistema completo é testado.

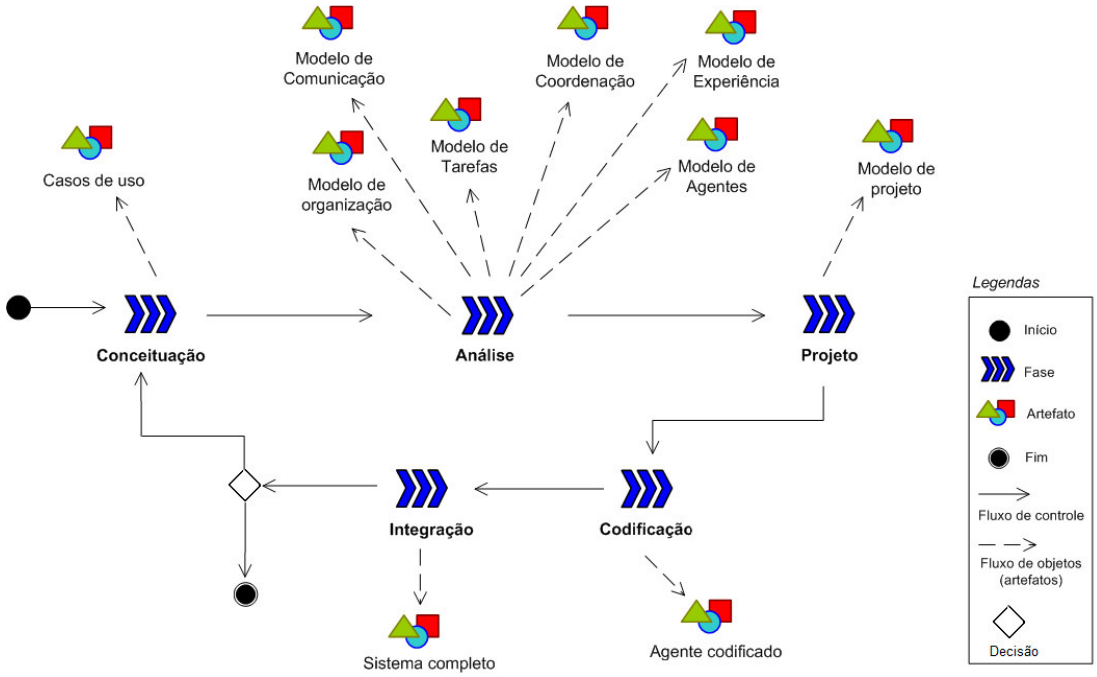


Figura 2-4 – Fases da Metodologia MAS-CommonKADS

2.2.3 GAIA

A metodologia Gaia foi criada por Wooldrige, Jennings & Kinny (1999) como uma metodologia específica para o desenvolvimento de sistemas baseados em agentes. Uma versão mais recente foi proposta por Zambonelli; Jennings & Wooldridge (2003). Em Gaia o sistema é modelado como uma sociedade ou organização de agentes. Abaixo da organização estão os papéis que os agentes podem assumir dentro desta. Um papel é definido por três atributos: responsabilidades, permissões e protocolos. As responsabilidades determinam as funções que cada papel executa. Existem dois tipos de responsabilidades: propriedades de progresso (do inglês *liveness*), que descrevem as ações e os estados que os agentes devem buscar, de acordo com as condições do ambiente; e propriedades de segurança (do inglês *safety*), que indicam os estados que os agentes devem manter continuamente. As permissões são direitos dos papéis sobre determinados recursos.

A metodologia compreende duas fases (Figura 2-5): a primeira é a Análise, que utiliza os conceitos acima e modela os papéis e as interações entre estes papéis. No processo de análise, primeiramente identificam-se os papéis do sistema gerando um protótipo do modelo de papéis. Em seguida, para cada papel, os protocolos (padrões de interações) associados são identificados e documentados gerando o modelo de interações. Usando-se o modelo de interações, elabora o modelo definitivo de papéis. Estes três primeiros passos são repetidos até chegar a um nível satisfatório de análise; a segunda fase de Gaia é o Projeto, que tem como objetivo transformar os modelos abstratos da análise em modelos suficientemente detalhados para aplicação em outras metodologias de desenvolvimento de software, como as orientadas a objetos. Gaia não sugere que a fase de implementação use ferramentas específicas para agentes. O projeto é constituído pela construção de três modelos: modelo de agentes, modelo de serviços e modelo de conhecimento. O modelo de agentes documenta os agentes e papéis que serão usados pelo sistema em construção e as instâncias de agentes que podem surgir em tempo de execução. Utiliza-se um diagrama específico para visualizar os papéis que o agente executa. O modelo de serviços descreve as funções desempenhadas para cada papel de agente, especificando suas principais propriedades. O modelo de conhecimento define a comunicação entre os diversos tipos de agentes.

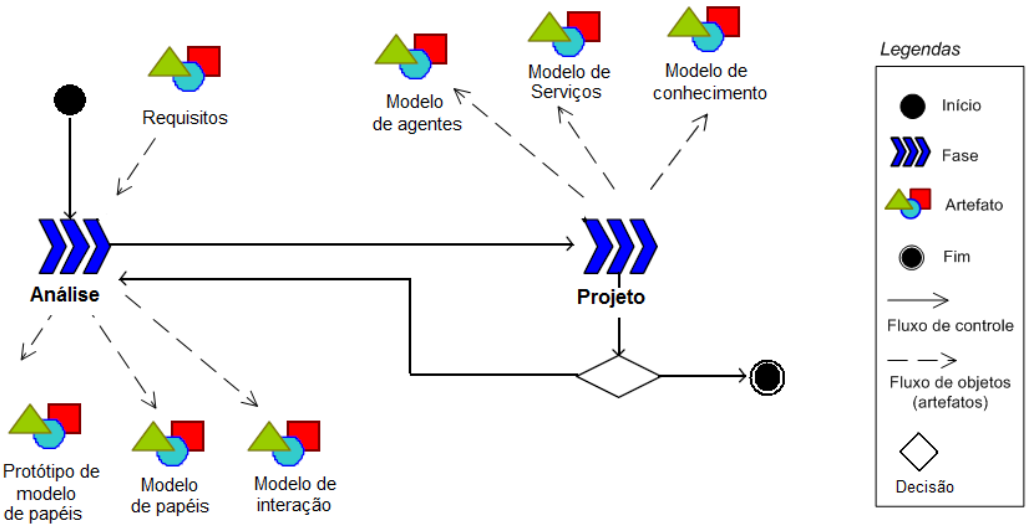


Figura 2-5 – Fases da metodologia GAIA. Adaptado de FIPA (2008)

2.2.4 MaSE e O-MaSE

A Metodologia MaSE (Multiagent Systems Engineering Methodology) [Wood & DeLoach, 2001] é similar a Gaia em sua generalidade e domínio da aplicação. Porém MaSE fornece um suporte à criação automática de código para agentes através de suas ferramentas. A primeira versão de MaSE apresentava algumas deficiências, tais como, a falta de um mecanismo para

interação dos agentes com o ambiente e o uso de organizações de tamanho fixo. Para solucionar estas deficiências, MaSE foi estendida com conceitos organizacionais originando uma nova versão chamada O-MaSE [DeLoach, 2006].

O-MaSE é composta de três fases principais (Figura 2-6): inicia com a fase de Requisitos, onde os requisitos são documentados em um Diagrama hierárquico de metas representado por um modelo de classes da UML; a segunda fase é a Análise, onde a organização é definida no Modelo organizacional usando diagramas de caso de uso, seqüência e atividades da UML. A ontologia usada na organização também é capturada e documentada em um Modelo de domínio. A identificação dos papéis define o conjunto inicial de papéis do sistema que são representados no Modelo de papéis; a última fase é o Projeto, onde os modelos da análise são transformados em construções utilizáveis para a implementação de um sistema multiagente. A fase do Projeto é composta por quatro passos: criação das classes de agentes, projeto dos protocolos de interação, definição dos planos e da política da organização. Os Modelos de classes de agentes são definidos por meio de papéis executados, capacidades ou serviços fornecidos. Os Modelos de protocolos são usados para definir os protocolos de passagem de mensagem entre classes de agentes. O comportamento dos agentes é definido usando Modelo de planos. O Modelo de capacidades inclui as capacidades, ações e operações realizadas pelos agentes. Por fim, o Modelo de política define um conjunto de regras especificadas formalmente para descrever como uma organização se comporta em situações particulares. O projeto do sistema implica na definição do sistema a ser implementado, uma vez que já foram definidos os diagramas para identificar as características dos agentes no sistema. O ambiente de desenvolvimento AgentTool III (ou aT3)[3] é utilizado para apoiar o processo de análise, projeto e implementação do sistema.

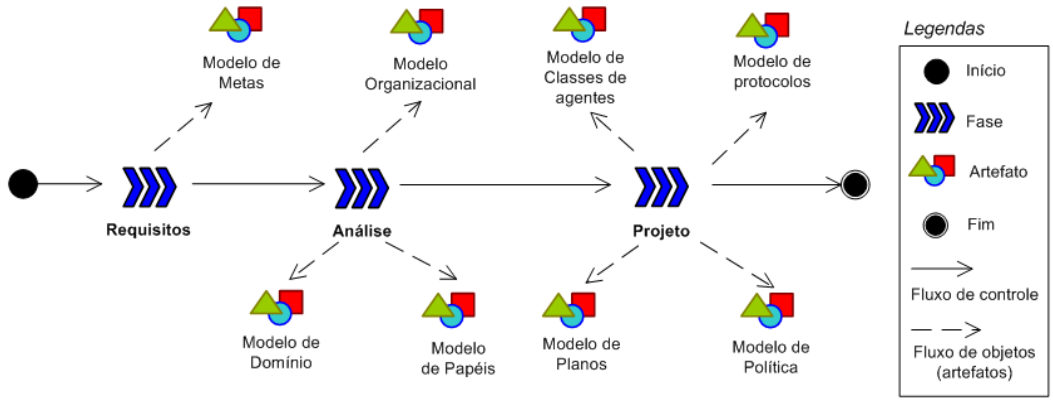


Figura 2-6 - Fases da metodologia MASE

[3] http://agenttool.projects.cis.ksu.edu/

2.2.5 PASSI

PASSI (Process for Agent Societies Specification and Implementation) [Cossentino & Potts, 2002] é uma metodologia para desenvolver sociedades de sistemas multiagentes, que propõe atividades para todas as fases desde os requisitos até a codificação. Ela adota conceitos tanto da orientação a objetos, quanto da inteligência artificial e usa a UML como notação.

Está dividida em cinco fases e doze passos para o processo de construir sistemas multiagentes (Figura 2-7): (i) a fase de requisitos é responsável pela definição do modelo de requisitos do sistema, que produz descrições de funcionalidades baseadas em caso de uso e adaptações de acordo com o paradigma de agentes. Esta fase é subdividida em descrição do domínio, identificação de agentes, identificação de papéis e especificação de tarefas; (ii) a fase sociedade de agentes cria o modelo de sociedade de agentes. Este modelo representa as interações sociais e dependências entre os agentes que executam uma parte da solução. A fase é subdividida em descrição da ontologia, descrição dos papéis e descrição de protocolos; (iii) a fase de implementação gera o modelo de implementação dos agentes, que modela uma solução em termos de classes e métodos. Está dividida em dois passos que são a definição da estrutura dos agentes e a descrição do comportamento dos agentes; (iv) a fase codificação modela a solução no nível de codificação e tem os passos reuso de código e conclusão do código; e (v) a fase de implantação (distribuição) gera o modelo de implantação que modela a distribuição de partes do sistema em unidades de processamento de hardware e é composto de um único passo que é a configuração da implantação.

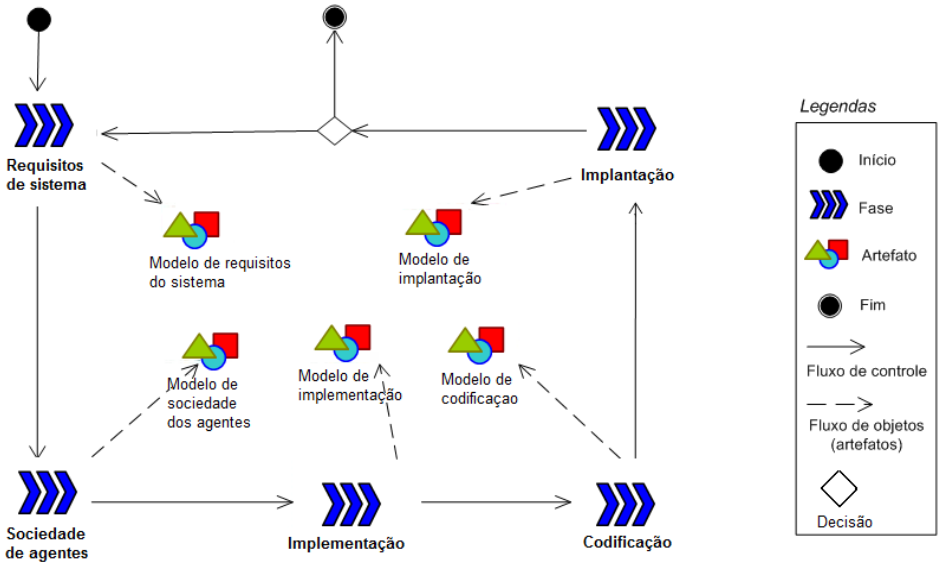


Figura 2-7 – Fases da metodologia PASSI

A abordagem Agile-PASSI [Chella et al., 2004] é uma versão ágil de PASSI. O Agile PASSI acrescenta uma fase de testes conforme as práticas do desenvolvimento ágil, que sugere que os testes devem começar o mais cedo possível dentro do processo de desenvolvimento, já sendo definido no planejamento dos requisitos do sistema.

2.2.6 TROPOS

A metodologia Tropos [Mylopoulos & Castro, 2000] [Castro; Kolp & Mylopoulos, 2002] [Bresciani et al., 2004] é orientada a requisitos no sentido que é baseada em conceitos usados durante a fase de análise de requisitos iniciais. O objetivo de usar conceitos da engenharia de requisitos é reduzir a incompatibilidade entre o sistema e seu ambiente. Para este fim, foram utilizados os conceitos oferecidos pelo framework i* [Yu 1995]. Tropos está descrita em duas versões que serão mais detalhadas no Capítulo 4.

Tropos é dividido em cinco fases de desenvolvimento de software (Figura 2-8): (i) requisitos iniciais, cujo objetivo é o entendimento de um problema a partir da análise e estudo da configuração organizacional onde o problema está inserido. Produz o modelo de dependências estratégica para descrever a rede de relacionamentos entre os atores e o modelo de raciocínio estratégico para justificar o raciocínio que cada ator tem sobre seu relacionamento com outros atores; (ii) requisitos finais, onde é incluído explicitamente o sistema com o seu ambiente operacional, a partir de requisitos funcionais relevantes e requisitos de qualidade (os chamados requisitos não funcionais). São produzidos os modelos revisados de dependência estratégica e raciocínio estratégico; (iii) projeto arquitetural, cujo objetivo é a definição global da arquitetura do software em termos de subsistemas e suas dependências. É produzido o modelo de projeto arquitetural, que modela a estrutura do sistema em partes relativamente pequenas e intelectualmente gerenciáveis, que descreve como os papéis de agentes trabalham; (iv) projeto detalhado que detalha o comportamento de cada componente arquitetural através de linguagens de comunicação entre agentes, mecanismos de transporte de mensagens, ontologias, protocolos de interação de agentes da comunidade de programação de agentes. Produz o diagrama de classes de agentes, diagrama de interações, diagrama de capacidades e diagrama de planos; e (v) implementação onde é proposto o uso do ambiente de desenvolvimento orientado a agentes, chamado JACK Intelligent Agents [JACK, 2007] para implementar os agentes.

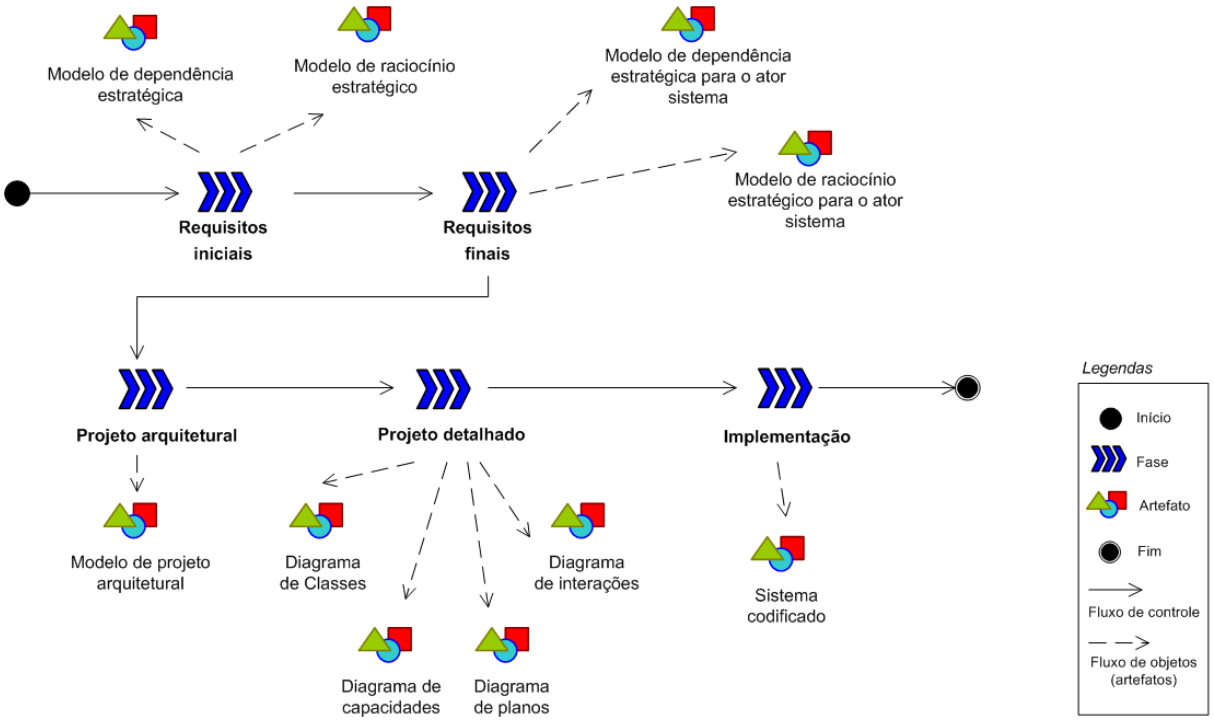


Figura 2-8 – Fases da metodologia Tropos

2.3 Considerações Finais

Existe muito esforço em criar metodologias para orientar o desenvolvimento de sistemas multiagentes. Isto tem ocorrido devido ao interesse crescente no uso de sistemas orientados a agente na indústria de software. As metodologias têm como objetivo fornecer métodos, técnicas e ferramentas para facilitar o desenvolvimento desses softwares, que são complexos por sua natureza.

Neste capítulo foi feita uma breve revisão das metodologias orientadas a agentes, incluindo MAS-CommonKADS, Gaia, MaSE, PASSI e Tropos (Tabela 2-1). Estas metodologias foram escolhidas por serem as mais populares no desenvolvimento orientado a agentes. O objetivo foi analisar a abordagem de cada uma dessas metodologias, suas atividades e produtos de trabalho gerados para caracterizar o que a comunidade de sistemas orientados a agentes considera como importante em uma metodologia. Observa-se que a maioria das metodologias tem raízes na engenharia de software orientada a objetos, inclusive fazendo uso da UML e extensões, como a AUML. MAS-CommonKADs, Gaia, MaSE / O-MaSE, PASSI têm influência da orientação a objetos em seus conceitos ou produtos de trabalhos, fazendo uso de artefatos da UML para modelar os produtos de trabalhos. Tropos usa conceitos da modelagem organizacional do i* [Yu, 1995] em sua versão original [Mylopoulos & Castro, 2000], mas as versões posteriores adotaram a modelagem UML na

fase de projeto detalhado. A UML (com extensões para conceitos de agentes) tem sido adotada, de forma geral, pela comunidade de desenvolvimento de agentes como a linguagem de modelagem para sistemas orientados a agentes.

Tabela 2-1 – Metodologias para desenvolvimento orientado a agentes

Metodologia	**Influência**	**Fases**	**Produtos gerados**
MAS-CommonKADS [Iglesias, 1998]	Engenharia do conhecimento e Orientação a objetos	Conceituação	Casos de Uso
		Análise	Modelo de organização Modelo de tarefas Modelo de comunicação Modelo de coordenação Modelo de experiência Modelo de agentes
		Projeto	Modelo de projeto
		Codificação	Agente codificado
		Integração	Sistema completo
GAIA [Wooldrige, Jennings & Kinny 1999] e [Zambonelli; Jennings & Wooldridge, 2003]	Orientação a objetos	Análise	Protótipo de modelo de papéis Modelo de papéis Modelo de interação
		Projeto	Modelo de agentes Modelo de serviço Modelo de conhecimento
O-MaSE [DeLoach, 2006]	Orientação a objetos	Requisitos	Modelo de Metas
MaSE [Wood & DeLoach, 2001] e O-MaSE [DeLoach, 2006]		Análise	Modelo organizacional Modelo de domínio Modelo de papeis
		Projeto	Modelo de classes de agentes Modelo de protocolo Modelo de planos Modelo de Política
PASSI [Cossentino & Potts, 2002] e Agile PASSI [Chella et al., 2004]	Orientação a objetos	Requisitos	Modelo de requisitos
		Sociedade de agentes	Modelo de sociedade de agentes
		Implementação	Modelo de implementação
		Codificação	Modelo de codificação
		Implantação	Modelo de implantação
Agile PASSI [Chella et al., 2004]		Testes	Modelo de testes
Tropos [Mylopoulos & Castro, 2000], [Castro; Kolp & Mylopoulos, 2002] e [Bresciani et al., 2004]	i* (modelagem organizacional)	Requisitos iniciais	Modelo de dependência estratégica Modelo de raciocínio estratégico
		Requisitos finais	Modelo de dependência estratégica Modelo de raciocínio estratégico
		Projeto arquitetural	Modelo de projeto arquitetural
		Projeto detalhado	Diagrama de classes Diagrama de interações Diagrama de Planos Diagrama de capacidades
		Implementação	Sistema codificado

Todas as metodologias têm uma forte preocupação com a análise e projeto do sistema. MAS-CommonKADS, PASSI e Tropos geram modelos de definição de requisitos e modelos para orientar a implementação. O-MaSE acrescentou a fase de requisitos à versão inicial da

metodologia MaSE. A versão mais recente de PASSI (Agile-PASSI) acrescentou uma fase explícita de testes para orientar o desenvolvimento ágil de agentes. Existe hoje uma preocupação em atualizar as metodologias, investindo nas fases de requisitos e acrescentando modelos de processo para guiarem o desenvolvimento. A fase de requisitos é crítica em um processo de desenvolvimento, uma vez que todo o sistema é baseado nos conceitos definidos nesta fase. Modelos de processos para orientar o desenvolvimento também tem sido preocupação da comunidade de desenvolvimento de agentes, como apresentado no capítulo 4. Ágile-PASSI é um bom exemplo de extensões baseada em modelos de processo.

Tropos é a única que usa conceitos extraídos das fases de requisitos iniciais, que propõe o entendimento do problema antes da construção do sistema. Além disso, é uma abordagem que está em constante evolução[4], tendo sido gerados diversas extensões a versão original. No próximo capítulo, esta metodologia será estudada mais detalhadamente.

[4] www.troposproject.net

Capítulo 3 – O projeto Tropos

O projeto Tropos propõe uma das abordagens mais completas para o desenvolvimento de sistemas multiagentes. Este capítulo apresenta a visão geral do Tropos, suas fases, alguns conceitos básicos e as técnicas que compõe a proposta. Também são apresentadas novas técnicas e diretrizes que surgiram com o objetivo de melhorar o trabalho original.

3.1 Introdução

Os processos básicos do ciclo de vida de um software propostos pela ISO/IEC-12207:1994 consistem em cinco partes principais: aquisição, suprimento, desenvolvimento, operação e manutenção de produtos de software. O projeto Tropos [Mylopoulos & Castro, 2000] propõe o processo de desenvolvimento de produtos de softwares orientados a agentes, englobando as atividades de definição de requisitos (iniciais e finais), projeto arquitetural, projeto detalhado e implementação.

O nome Tropos é derivado da palavra grega “tropé”, que significa "facilmente mutável" ou "facilmente adaptável". A metodologia é orientada a requisitos no sentido que é baseada em conceitos usados durante a fase de análise de requisitos iniciais. O objetivo de usar conceitos da engenharia de requisitos é reduzir a incompatibilidade entre o sistema e seu ambiente. Para este fim, foram utilizados os conceitos oferecidos pelo framework i* [Yu 1995] que, são geralmente aplicados na fase de requisitos. Contudo, em Tropos, eles também são usados nas fases de arquitetura e projeto detalhado.

Tropos foi desenvolvida por diversos grupos de pesquisa e possui duas principais versões publicadas. Embora estas abordagens incluam as mesmas atividades, elas apresentam diferenças na seqüência de realização das etapas das atividades e nos artefatos gerados [Silva, M. J. et al., 2007]. Nesta dissertação, estas duas abordagens são chamadas de Tropos’02 [Castro; Mylopoulos & Kolp, 2002] e Tropos’04 [Bresciani et al., 2004]. Diversos trabalhos têm sido desenvolvidos estendendo as duas abordagens principais. Alguns trabalhos como o framework SIRA [Bastos, 2005] que, descreve como gerar uma configuração arquitetural a partir dos requisitos do sistema e o modelo de projeto detalhado de [Silva, C.T.L.L. et al.; 2007a] utilizam a abordagem Tropos’02. Alguns trabalhos para gerenciamento de segurança [Giorgini & Mouratidis, 2005] e análise de risco [Asnar; Gorgini & Mylopoulos, 2006] utilizam a abordagem Tropos’04. O comitê técnico de metodologias da FIPA [FIPA, 2007] propôs a especificação de Tropos [FIPA, 2008] usando a notação de especificação de processos SPEM [SPEM, 2007]. O objetivo foi a criação de um meta-modelo que pudesse ser usado para descrever as metodologias e estruturas dos sistemas multiagentes. A versão utilizada nesta descrição foi a Tropos’04. O Spiral Tropos [Wautelet; Kolp & Achbany, 2005] é a descrição da metodologia Tropos’02 como um processo de desenvolvimento em espiral. O objetivo deste processo é estender a metodologia Tropos para incluir as vantagens do modelo de desenvolvimento em espiral para softwares orientados a agentes.

Neste capítulo são apresentados os conceitos usados por Tropos e as principais variações existentes. Inicia descrevendo o framework i*, depois o Tropos e suas variações e por fim as técnicas de melhoria e extensão do Tropos. São apresentadas as técnicas mais conhecidas para detalhamento das fases de requisitos iniciais e finais e técnicas que estendem as fases de projeto arquitetural e projeto detalhado. As técnicas citadas anteriormente para gerenciamento de requisitos, análise de risco e segurança, não são detalhadas, pois não fazem parte do escopo desta dissertação.

Para um melhor entendimento, neste trabalho, são usados exemplos ilustrativos dos conceitos apresentados. Para isto é utilizado um sistema multiagentes chamado DBSitter-AS [Maciel, 2007] que foi modelado usando os conceitos do i* e fazendo uso de um subconjunto de atividades extraídas do Tropos'02 e Tropos'04 [Silva, M.J et al., 2007]. O DBSitter-AS é um sistema multiagentes que tem como objetivo monitorar um banco de dados e resolver falhas que podem ocorrer durante seu uso.

3.2 Framework i*

O framework i* [Yu, 1995] foi criado sob a premissa que, o entendimento mais profundo de um processo pode ser obtido a partir de uma visão estratégica e intencional. O nome i* está relacionado à noção de intencionalmente distribuído. Desta forma ele se baseia em uma unidade central: o ator estratégico e intencional. O ator intencional não somente executa atividades e produz entidades, mas tem motivações, intenções e raciocínio por trás de suas ações. Os aspectos intencionais de um ator podem ser caracterizados por metas, crenças, habilidades e compromissos. Um ator é estratégico quando ele não está meramente focalizado em encontrar suas metas imediatas, mas está interessado nas implicações de seus relacionamentos com outros atores. O i* consiste de dois modelos: o modelo de dependência estratégica (Modelo SD) e o modelo de raciocínio estratégico (Modelo SR).

O Modelo SD provê uma descrição intencional de um processo em termos de uma rede de relacionamentos de dependências entre atores. O modelo pode ajudar a identificar as partes interessadas (*do inglês stakeholders*) no sistema, analisar vulnerabilidades e oportunidades, reconhecer padrões de relacionamentos e identificar mecanismos para minimizar vulnerabilidades. O modelo SD consiste de um conjunto de nós e ligações. Cada nó representa um ator e cada ligação entre dois atores representa uma dependência entre estes.

Os conceitos utilizados no modelo SD são atores, metas, *metas-soft*[5], tarefas, recursos e dependências. Os atores representam papéis exercidos pelas pessoas, funções ou cargos ocupados, organizações ou subdivisões da organização, sistemas ou envolvidos no contexto onde sistema está inserido. As metas representam os interesses estratégicos dos atores, ou seja, suas intenções, necessidades ou objetivos almejados para cumprir o seu papel dentro do ambiente em que está inserido. As *metas-soft* também representam os interesses estratégicos dos atores. Neste caso, estes interesses têm natureza subjetiva, isto é, não são medidas de forma concreta, mas são geralmente usadas para descrever desejos dos atores relacionados aos atributos de qualidade com que eles querem que suas metas sejam satisfeitas. As tarefas representam a forma de executar alguma atividade, isto é, indicam como realizar alguma ação para obter a satisfação de uma meta ou *meta-soft*. Os recursos representam dados ou informações que podem ser fornecidas ou recebidas por um ator.

As dependências representam um acordo entre dois atores. O ator que depende de outro ator é chamado *depender* e o ator que satisfaz a dependência de outros atores é chamado *dependee*. O objeto da dependência é chamado *dependum*. O tipo da dependência descreve a natureza do acordo entre os atores. A dependência de metas ocorre quando um ator tem uma meta a cumprir, mas depende de outros atores para a satisfação desta meta. Então esta dependência representa uma delegação da responsabilidade de atingir uma meta de um *depender* para um *dependee*. Neste caso, o *depender* não especifica como ele quer que a meta seja executada, ele apenas repassa a responsabilidade para o ator *dependee*, que assume o compromisso de executá-la de acordo com suas capacidades. A dependência de *metas-soft* é semelhante à dependência de metas, com a diferença que o *dependum* é uma *meta-soft*. O ator *depender* delega a responsabilidade de obtenção da *meta-soft* ao ator *dependee* e este assume a compromisso de satisfazer esta *meta-soft*. A dependência de tarefas ocorre quando um ator tem uma atividade a cumprir, mas depende de outros atores para a sua execução. Esta dependência representa uma delegação de responsabilidade de como realizar uma atividade. O *depender* especifica como ele quer que a atividade seja executada já repassando o plano de como esta deve ser executada. O ator *dependee* assume o compromisso de executar a atividade da forma como é especificado no plano recebido. A dependência de recursos ocorre quando um ator para cumprir suas responsabilidades, depende de informações fornecidas por outros atores. Esta dependência representa a troca de informações entre os atores. O ator

[5] Os termos em inglês são traduzidos nesta dissertação: Goal é traduzido para Meta. Softgoal não tem tradução na língua portuguesa, mas para manter a padronização é assumido o termo *Meta-soft*.

depender depende de um recurso fornecido por um ator *dependee* que se responsabiliza em prover este recurso.

A Figura 3-1 mostra um exemplo de diagrama SD. Os círculos representam os atores do modelo SD que, interagem ente si, através das suas dependências, representadas pelas ligações. O ator1 depende do ator2 para satisfazer suas metas e metas-soft. Neste caso o ator1 é o depender e o ator2 o dependee. O ator2 tem dependências do ator1 para realizar tarefas e obter recursos. Portanto, o ator2 é o *depender* e o ator1 o *dependee*. Os *dependums* caracterizam os tipos de dependências, isto é, metas, metas-soft, recursos e tarefas.

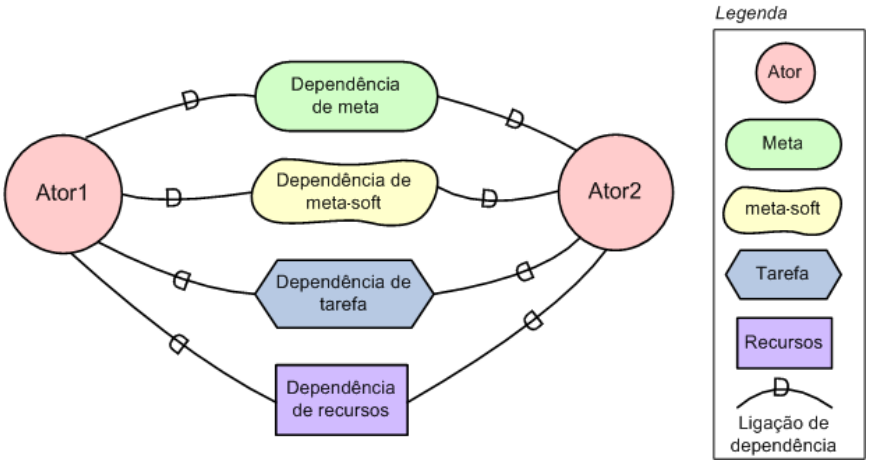


Figura 3-1 – Exemplo do modelo SD

Embora o Modelo de dependência estratégica mostre dicas sobre o processo estruturado com os atores e seus relacionamentos, ele não é suficiente para apoiar o processo de sugerir, explorar e avaliar soluções alternativas. Este é o papel do Modelo de raciocínio estratégico (Modelo SR). O modelo SR inclui os mesmos quatro tipos de nós principais: as metas, tarefas, recursos e *metas-soft* e adiciona três tipos de ligações: análise meio-fim[6], decomposição e contribuições. A relação entre estes novos elementos se dá no contexto de um único ator, que no diagrama SR é representado por uma fronteira, ou seja, um círculo que delimita a análise dos elementos de um único ator.

Os conceitos de ator, metas, *metas-soft*, tarefas e recursos são os mesmos descritos anteriormente no modelo SD. Na ligação de decomposição, uma tarefa é modelada em termos de decomposição em subcomponentes. Estes subcomponentes podem ser metas, tarefas, recursos e/ou *metas-soft*. Uma decomposição da tarefa em metas indica o objetivo que aquela tarefa quer atingir. Quando uma tarefa é decomposta em uma subtarefa, a subtarefa define uma ação a ser realizada. Uma decomposição em um recurso implica em uma entidade (física

[6] Os termos em inglês são traduzidos nesta dissertação: Means-end é traduzido para Meio-Fim

ou informacional) que será usada para a tarefa. Um *meta-soft* serve como uma meta de qualidade para a tarefa e guia a seleção entre alternativas em outras decomposições da tarefa. Ao realizar a análise meio-fim podemos identificar várias tarefas que seriam alternativas para execução de uma meta. Estas tarefas são exclusivas, então ao final da análise, deve-se optar por uma destas, ou seja, as outras não serão executadas. O fim pode ser um uma meta, tarefa ou recurso e o meio geralmente é uma tarefa. Quando o fim é uma Meta e o meio é uma Tarefa, esta tarefa especifica o "como atingir a meta" através de suas decomposições em componentes. Quando o fim é um recurso e o meio é a tarefa, esta tarefa especifica o "como produzir o recurso" através de suas decomposições em componentes. A ligação entre *Meta-soft* e Tarefa é um caso especial chamado Contribuição. A tarefa é o meio para satisfazer as restrições de qualidade da meta-soft. Esta satisfação pode ser negativa ou positiva. Desta forma esta ligação é representada de forma diferente das demais ligações Meio-Fim, sendo representado com um atributo extra que identifica o tipo de contribuição da tarefa para a *meta-soft* (++ significa satisfeito, + significa parcialmente satisfeito, -- significa proibido, - significa conflito).

Um exemplo do modelo SR é mostrado na Figura 3-2. O círculo pontilhado representa a fronteira do ator2. O elemento *Meta* representa a meta principal do ator2 que é refinada por uma análise meio-fim nas *Tarefa1* e *Tarefa2*. Estas tarefas são as alternativas para realização da Meta. A *Tarefa1* é refinada por uma *decomposição* em duas subtarefas. A *subtarefa1* tem contribuição negativa sobre a *meta-soft* enquanto que a *tarefa2* tem contribuição positiva. A análise das contribuições ajuda ao ator decidir qual tarefa deve ser executada.

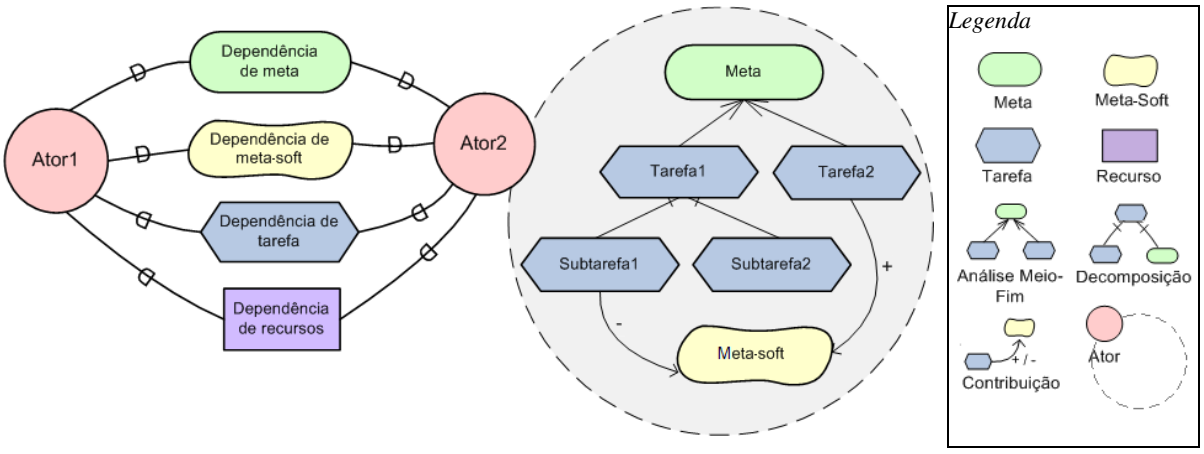


Figura 3-2 – Exemplo do Diagrama SR

3.3 A abordagem Tropos – visão geral

Tropos foi proposta em 2000 [Mylopoulos & Castro, 2000]. Esta versão original foi evoluída em duas versões [Castro; Kolp & Mylopoulos, 2002] [Bresciani et al., 2004], que são ambas utilizadas por pesquisadores em estudos e para modelagem de sistemas multiagentes. A proposta de Castro, Kolp & Mylopoulos (2002) é mais voltada à estruturação dos elementos sociais (pessoas envolvidas), intencionais (metas dos envolvidos), processos (negócios e responsabilidades) e objetos (objetos, classe e seus relacionamentos). A proposta de Bresciani et al. (2004) aborda mais profundamente conceitos do paradigma de desenvolvimento orientado a agentes e propõe mudanças nos elementos do framework i* original. Em ambas as versões, Tropos é dividida em cinco fases de desenvolvimento de software, que são requisitos iniciais, requisitos finais, projeto arquitetural, projeto detalhado e implementação. A seguir, são apresentadas as características destas fases.

3.3.1 Requisitos Iniciais

Os requisitos iniciais são o entendimento de um problema a partir da análise e estudo da configuração organizacional onde o problema está inserido. Durante esta fase os engenheiros de requisitos capturam e analisam as intenções das partes interessadas[7]. Estas são modeladas como metas, que a partir de uma análise orientada a metas geram os requisitos funcionais e não funcionais do sistema. Os modelos do i* são usados para representar as informações definidas nesta fase, que são os atores sociais que dependem de outros para atingir suas metas, realizar suas tarefas e obter recursos. O modelo de dependências estratégica é usado para descrever a rede de relacionamentos entre os atores enquanto, o modelo de raciocínio estratégico descreve e justifica o raciocínio que cada ator tem sobre seu relacionamento com outros atores. A Figura 3-3 exemplifica um modelo de dependência estratégica (SD) para o sistema DBSitter-AS na fase requisitos Iniciais. Esta figura mostra cinco atores: “Administrador de BD”, “Gerência de informação”, ”Sistema de Informação”, ”RH” e ”Diretoria executiva” que dependem uns dos outros para obter seus objetivos, executar suas tarefas e prover os recursos. Cada ator tem dependências entre si, representados pelas conexões entre eles.

O modelo de raciocínio estratégico (SR) representa o raciocínio que cada ator social usa para obter suas metas, executar suas tarefas e prover os recursos de sua responsabilidade. A

[7] *Partes interessadas* é a tradução assumida nesta dissertação para o termo *stakeholder* escrito na língua inglesa

Figura 3-4 mostra um fragmento de um modelo de raciocínio estratégico para o ator Administrador de BD da organização que irá utilizar o sistema DBSitter-AS. O ator "Administrador de BD" tem como meta o "Gerenciamento de dados executado" e também deve satisfazer uma meta-soft de "Gerenciamento eficiente". Para fazer isto ele pode realizar seu trabalho usando duas alternativas diferentes que são "Automatizar tarefas" ou "Usar administração de dados tradicional". Ambas contribuem para o gerenciamento eficiente, mas a primeira alternativa gera uma solução mais satisfatória.

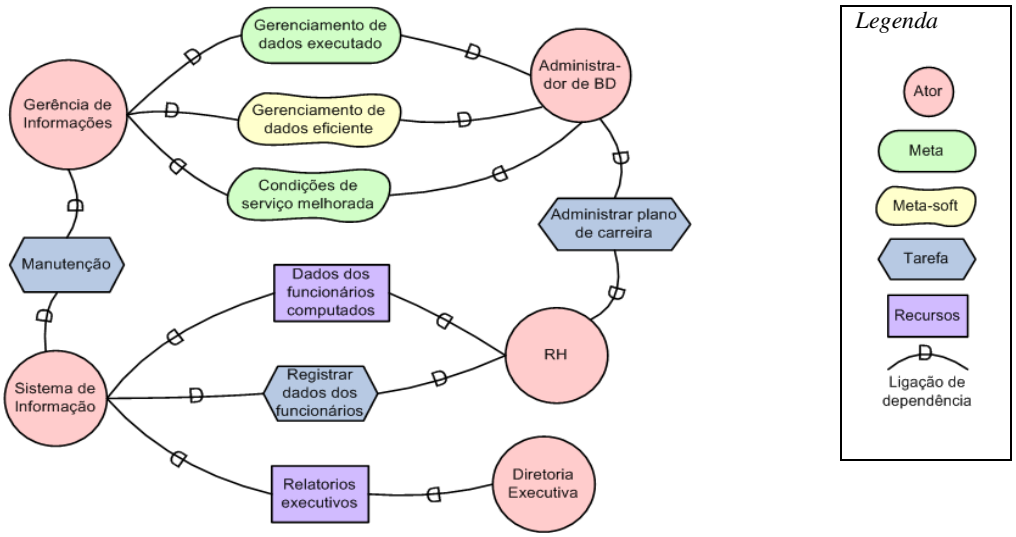


Figura 3-3 – Modelo SD na fase requisitos iniciais

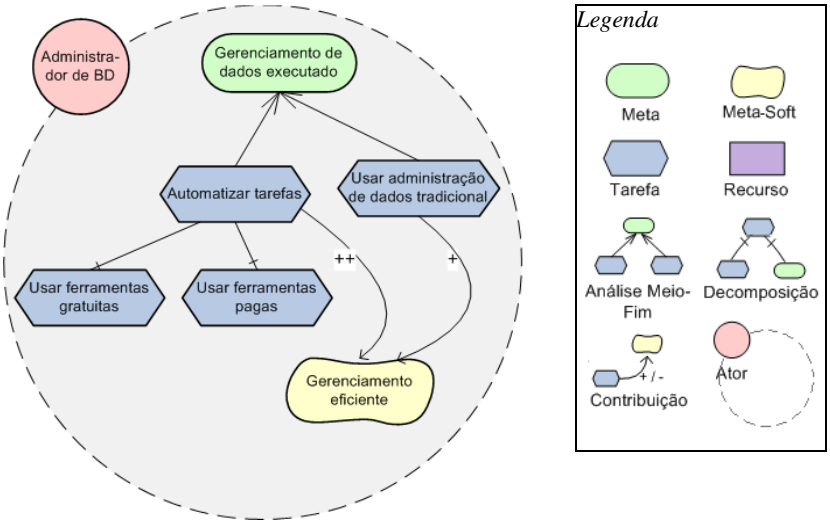


Figura 3-4 - Modelo SR na fase de requisitos iniciais

3.3.2 Requisitos Finais

Nesta fase, o sistema é descrito dentro de seu ambiente operacional, a partir de requisitos funcionais relevantes e requisitos de qualidade ou não-funcionais. Os Modelos SD e SR

iniciado nos requisitos iniciais são complementados acrescentando-se o ator *sistema-que-será desenvolvido* e a análise das metas na perspectiva deste ator. O novo sistema é representado como um ou mais atores que participam de um modelo de dependência estratégica juntamente com outros atores do ambiente operacional do sistema. À medida que a análise prossegue, o sistema recebe responsabilidades adicionais e atua como o *depender* de várias dependências. Posteriormente, o sistema pode ser decomposto em vários sub-atores que assumem algumas destas responsabilidades. Os modelos SD e modelos SR são atualizados nesta fase para representação dos novos atores, suas dependências e as responsabilidades dos atores.

A Figura 3-5 mostra um fragmento de um Modelo SD para o ator DBSitter-AS que é introduzido no modelo SD existente. O ator "Administrador de BD" depende do DBSitter-AS para que este "resolva falhas de forma autônoma" e deseja que ele garanta "segurança" e "confiabilidade", além de ter o "desempenho aperfeiçoado".

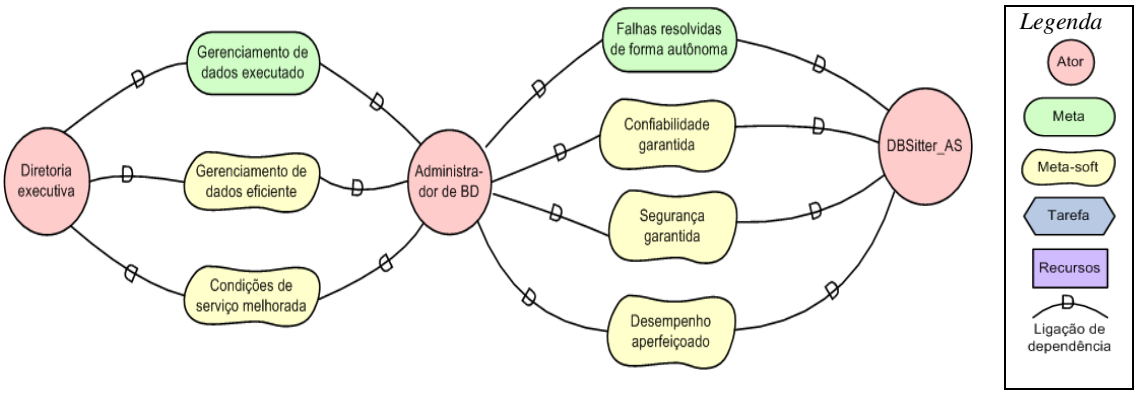


Figura 3-5 – Modelo SD na fase requisitos finais

A Figura 3-6 apresenta um exemplo do modelo SR que relaciona alguns requisitos funcionais e não-funcionais do ator sistema. As metas podem ser analisadas como os requisitos do usuário que devem ser satisfeito pelos requisitos de sistema. As tarefas representam os requisitos funcionais do sistema. As *metas-soft* representam os requisitos não funcionais. Já os recursos são dados ou informações recebidas ou providas pelo sistema.

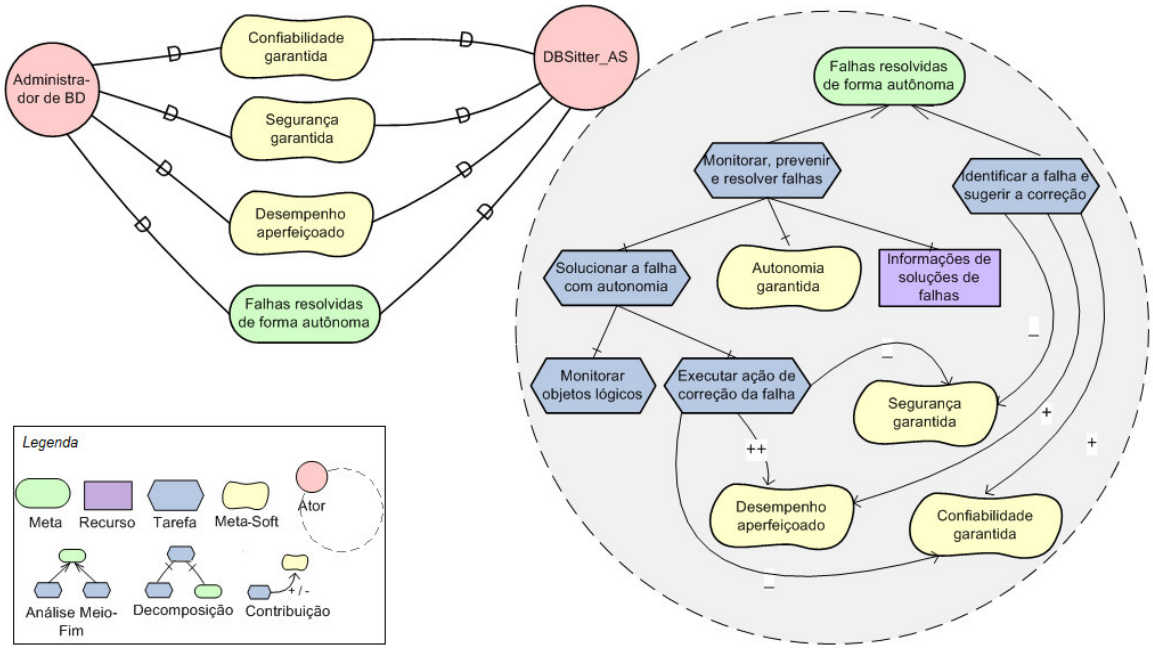


Figura 3-6 – Modelo SR na fase de requisitos finais

3.3.3 Projeto arquitetural

Durante a fase de projeto arquitetural é feita a definição global da arquitetura do software em termos de subsistemas e suas dependências. A arquitetura do sistema constitui um modelo de estrutura do sistema relativamente pequeno e intelectualmente gerenciável, que descreve como os componentes trabalham. A arquitetura é influenciada por projetistas de sistemas e por fatores técnicos e sociais. Durante o projeto arquitetural as análises são concentradas nos atores do sistema e suas responsabilidades, que foram definidos durante a fase requisitos finais. Levando em consideração tanto as funcionalidades desejadas como os requisitos de qualidade, os arquitetos de software podem desenvolver o sistema usando qualquer estilo arquitetural existente [Sommerville, 2005], que incluem estilos conhecidos como: Componentes Independentes, Call-and-return, Data flow, Data centered entre outras.

Em Tropos, a análise de requisitos é feita a partir de uma análise meio-fim dos estilos arquiteturais comparados às *metas-soft* do sistema (atributos de qualidade). São atribuídas métricas que definem quais estilos arquiteturais contribuem mais fortemente para atingir as metas de qualidade do sistema e a partir disto são selecionados os estilos arquiteturais.

A versão original da metodologia Tropos [Mylopoulos & Castro, 2000] analisava os atributos de qualidade em relação aos estilos tradicionais tais como camada, arquitetura orientada a objetos ou invocação explícita. Mas estes estilos não se adaptavam a proposta de sistemas multiagentes. A partir da versão Tropos'02 [Castro; Kolp & Mylopoulos, 2002], foram definidos os estilos arquiteturais organizacionais [Kolp; Castro & Mylopoulos, 2001]

[Kolp; Giorgini; Mylopoulos, 2006] para guiar a arquitetura de aplicações distribuídas e cooperativas como os sistemas multiagentes. Estes estilos arquiteturais se baseiam em conceitos da pesquisa em administração organizacional. São exemplos destes estilos arquiteturais [Fuxman et al., 2001]: *flat structure, pyramid, joint venture, structure-in-5, takeover, arm's length, vertical integration, co-optation, bidding*, etc.

Estes estilos organizacionais foram avaliados e comparados com os atributos de qualidade de sistemas de componentes autônomos coordenados. Podem ser considerados como atributos de qualidade deste tipo de sistema: *Previsibilidade, Segurança, Adaptabilidade, Coordenação, Cooperação, Disponibilidade, Integridade, Modularidade ou Agregabilidade.* O resultado desta análise compõe um catálogo de correlações entre os estilos organizacionais (linhas) e os atributos de qualidade (colunas) mostrado na Tabela 3-1. As correlações são contribuições segundo a notação do framework NFR [Chung et al., 2000] com o significado: "++" é uma contribuição positiva e suficiente; "+" é uma contribuição positiva e parcial; "-" é uma contribuição negativa e parcial; e "--" é uma contribuição negativa e suficiente.

Tabela 3-1 – Catálogo de estilos arquiteturais

	Prev.	***Seg.***	***Adapt.***	***Coord.***	***Coop.***	***Disp.***	***Integr.***	***Mod.***	***Agreg.***
Flat Structure	--	--	-			+	+	++	-
Structure-in-5	+	+		+	-	+	++	++	++
Pyramid	++	++	+	++	-	+	--	-	
joint venture	+	+	++	+	-	++		+	++
Bidding	--	--	+	-	++	-	--	++	
Take over	++	++	-	++	--	+		+	+
Arm's Length	-	--	+	-	++	--	++	+	
Hierarchical Contracting			+	+	+	+		+	+
Vertical Integration	+	+	-	+	-	+	--	--	--
Cooptation	-	-	++	++	+	--	-	--	

No exemplo utilizado (Figura 3-5), foram identificadas as *metas-soft confiabilidade, segurança* e *desempenho*. As correlações com os estilos arquiteturais organizacionais são analisadas e identifica-se o melhor estilo arquitetural. *Confiabilidade* está relacionada à *segurança* e *desempenho* a *tempo de resposta*, que segundo o Framework NFR, está relacionado a *disponibilidade*. Desta forma o melhor estilo arquitetural é aquele que tiver contribuições mais positivas para os atributos de qualidade *segurança* e *disponibilidade*. Neste caso, os estilos mais apropriados seriam *Piramide, Joint Venture ou Takeover*. Se disponibilidade for mais crítico ao sistema que segurança, o estilo selecionado é o *Joint-*

Venture, se não, pode-se fazer uma análise nos estilos *Piramide* ou *Takeover* para selecionar o mais apropriado.

O Modelo de dependência estratégica (SD) é usado para expressar a arquitetura do sistema, apresentando cada componente como um ator. Suponha que o estilo selecionado seja o Joint-Venture. O estilo Joint-Venture envolve um acordo entre dois ou mais parceiros para operar em larga escala e reusar a experiência e conhecimento dos outros parceiros. Estes parceiros se dividem em: um coordenador, cuja finalidade é gerenciar o compartilhamento do conhecimento e dos recursos; parceiros principais que interagem para trocar, prover e receber serviços, dados e conhecimento; e parceiros secundários que suprem serviços ou suportam tarefas para o núcleo da organização.

A Figura 3-7 apresenta um exemplo de um sistema que usa o estilo arquitetural Joint-Venture. Nesta figura, visualizamos cinco atores. O ator Coordenador, que tem a responsabilidade de coordenar os recursos compartilhados. Ele se relaciona com os atores *Resolvedor de Falhas e Relator de Informações*. Estes são os parceiros principais que interagem para cumprir as metas da organização. O ator *Wrapper* é um ator secundário que prover o serviço de interação com outro ator externo, o Servidor de e-mail.

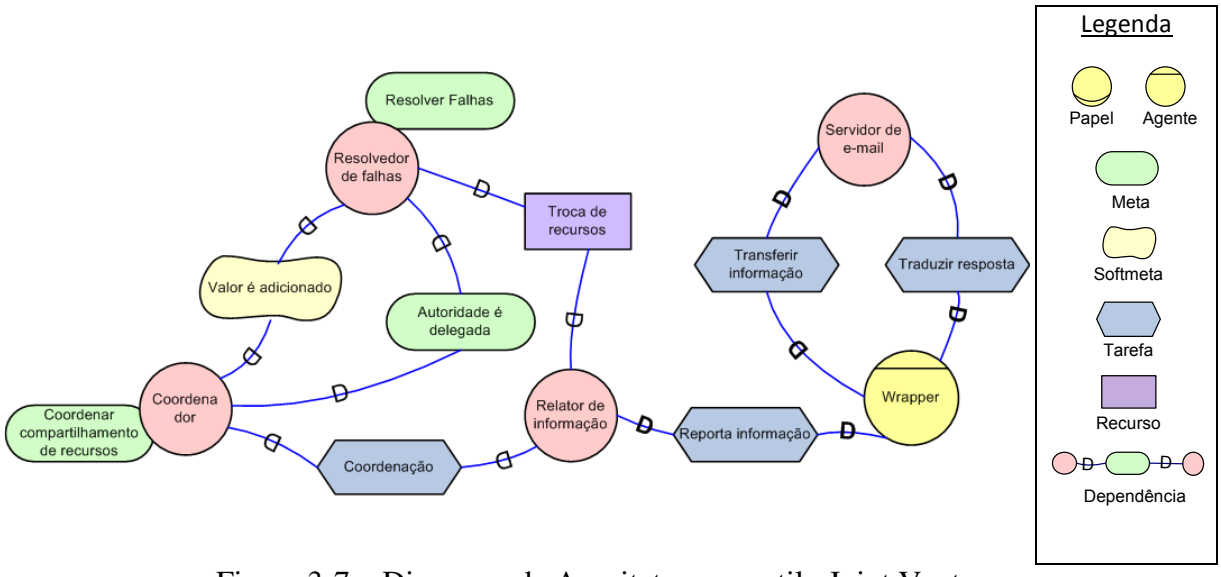


Figura 3-7 – Diagrama de Arquitetura no estilo Joint Venture

Outro passo do projeto arquitetural é definir como as metas atribuídas para os atores podem ser atendidas por agentes através do uso de padrões sociais. Tropos usa alguns padrões sociais, tais como os padrões *federados* [Hayden; Carrick & Yang, 1999]: *broker, matchmaker, mediator, monitor, embassy, wrapper, contractnet*. O padrão *Wrapper*, por exemplo, pode ser usado para se comunicar com atores externos ao sistema. A Figura 3-7 mostra um exemplo do uso de um padrão social *Wrapper*. Cada padrão social possui um conjunto de capacidades que são um conjunto de atividades que estes podem realizar para

obtenção de suas metas. Para cada capacidade um ator possui um plano que pode ser aplicado em situações diferentes. Estas capacidades são catalogadas e permitem definir qual papel cada agente pode realizar em um domínio particular.

3.3.4 Projeto detalhado

O objetivo desta fase é detalhar o comportamento de cada componente arquitetural. Para isto são usadas linguagens de comunicação entre agentes, mecanismos de transporte de mensagens, ontologias, protocolos de interação de agentes da comunidade de programação de agentes. Na versão Tropos'02 adotou-se a AUML [Bauer; Muller & Odell, 2001], uma extensão da UML proposta pela FIPA e o grupo de agentes da OMG [OMG, 2007]. Os atores e elementos intencionais são representados em diagramas de classes da UML estendidos com estereótipos que identificam estes elementos. As interações entre os atores são representadas por diagramas de seqüência da AUML e os planos por diagramas de estado ou diagramas de atividades.

No exemplo do sistema DBSitter-AS, os agentes podem ser detalhados em diagramas de classes da AUML (Figura 3-8) e o diagrama de seqüência (Figura 3-9) é usado para representar a interação de três agentes Controlador de bufferCache, Coordenador e Wrapper firebird. Este diagrama representa as mensagens trocadas entre os agentes para formação do plano global de resolução de falhas.

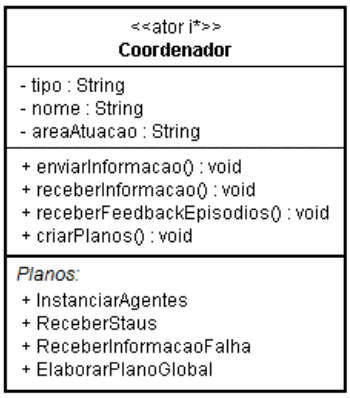


Figura 3-8 – Classe Agente Coordenador

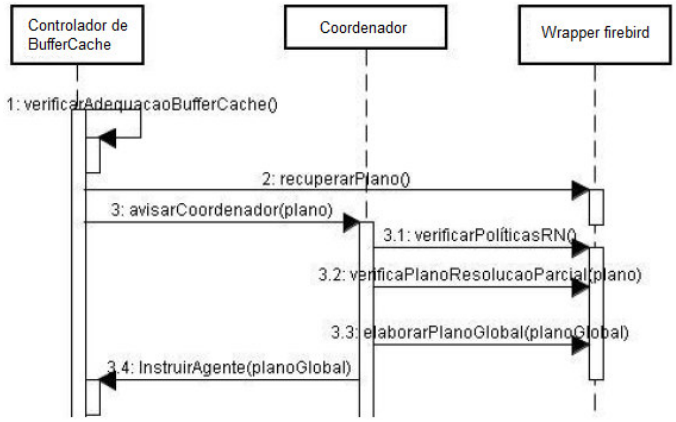


Figura 3-9 – Diagrama de interações entre os agentes

As capacidades dos atores são representadas por diagrama de atividades (Figura 3-10). Cada nó do diagrama representa uma ação que será realizada pelo agente para cumprir com suas tarefas e atingir as metas propostas para si. A capacidade modelada no exemplo é "Elaborar Plano de resolução de falhas" do agente Coordenador. Para um coordenador

preparar um plano, ele recebe uma falha identificada e consulta o repositório de conhecimento. Se a falha tem resolução, ele recupera as normas que orientam a resolução, elabora o plano de correção e informa o agente solucionador de falhas. Caso não exista solução, ele gera um alerta e notifica o agente responsável pela comunicação. Cada nó do diagrama de capacidades é um plano do agente, que pode ser especificado mais detalhadamente. Um plano também é especificado usando um diagrama de atividades para mostrar a seqüência de passos para execução do plano. A Figura 3-11 mostra o diagrama de planos para o plano "Registra episódio de falha e elabora plano de correção". A seqüência para realização deste plano é "aplicar as normas e políticas para o plano", "aplicar as restrições ao plano", "identificar o agente responsável pelo plano global" e "acionar wrapper para registro da resolução da falha".

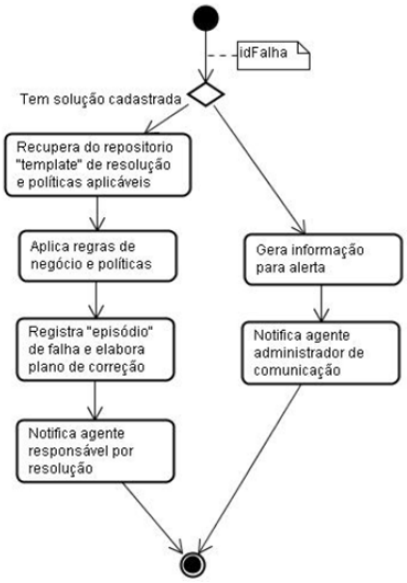


Figura 3-10 – Diagrama de capacidades

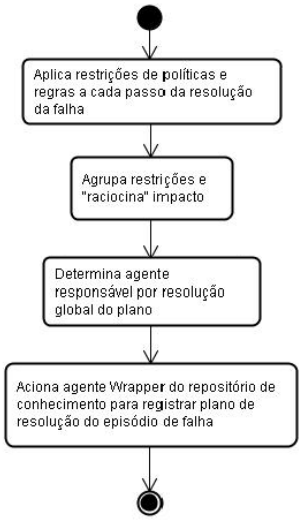


Figura 3-11 – Diagrama de planos

3.3.5 Implementação

Durante a implementação, a versão Tropos'02 inicialmente propôs o ambiente de desenvolvimento orientado a agentes, chamado JACK Intelligent Agents [JACK, 2007]. Este ambiente estende Java com o modelo teórico de agentes BDI (Belief Desire Intention). Os agentes de JACK podem ser considerados como componentes de software autônomo que têm metas a atingir, ou eventos para tratar (desejos). Os agentes são programados com um conjunto de planos (intenções). Para realizar um trabalho, o agente segue suas próprias metas (desejos), adota os planos apropriados (intenções) de acordo com seus dados atuais (crenças) sobre o estado do mundo. Para suportar a programação de agentes BDI, JACK oferece cinco

construtores de linguagem: agentes, capacidades, relações de banco de dados (armazenam crenças e dados dos agentes), eventos (mensagens trocadas pelos agentes) e planos (instruções que o agente segue para atingir suas metas).

3.4 Versões da metodologia Tropos

Conforme já mencionado anteriormente, os conceitos iniciais de Tropos foram desenvolvidos por Mylopoulos & Castro (2000) e evoluídas em duas versões, aqui chamadas de Tropos'02 [Castro, Kolp & Mylopoulos 2002] e Tropos'04 [Bresciani et al., 2004]. Estas duas propostas foram avaliadas durante as pesquisas para esta dissertação e algumas semelhanças e diferenças foram encontradas em suas atividades e artefatos produzidos [Silva, M. J. et al., 2007].

A versão Tropos'02 é uma versão revisada dos conceitos introduzidos na proposta original [Mylopoulos & Castro, 2000] acrescentando-se alguns novos conceitos. O uso do framework i* e as fases da metodologia são mantidos em conformidade com a proposta original. O trabalho integra resultados de diversos outros trabalhos na área e apresenta um estudo de caso completo para uma aplicação e-commerce que apresenta em detalhes as fases da metodologia. Os estilos arquiteturais são propostos a partir de estilos organizacionais e padrões sociais são usados para projetar a arquitetura. São usados extensões de UML para detalhar os modelos de projeto. São incluídas técnicas para geração de uma implementação do projeto detalhado. É adotada uma plataforma de programação orientada a agente chamada JACK, para os estudos de geração da implementação a partir do projeto detalhado. Além disto, esboça uma linguagem formal que fundamenta a metodologia.

A versão tropos'04 foi lançada com o objetivo de facilitar a adoção do Tropos na comunidade de desenvolvimento orientado a agentes. Esta versão também se baseia nos conceitos chaves da programação orientada a agentes e elementos intencionais, tais como atores, metas, planos, recursos, dependências, capacidades e crenças. As atividades são coordenadas a partir da evolução da modelagem destes elementos e segue a seqüência de modelagem de atores, modelagem de dependências, modelagem de metas, modelagem de planos e por fim modelagem de capacidades. Também é introduzido um estudo de caso real para a aplicação da metodologia. O estudo de caso foi desenvolvido para o governo da província de Trentino na Itália e prover informações culturais para os turistas. Esta versão propõe um novo metamodelo para os diagramas do i* [Susi; Perini & Mylopoulos, 2005], adicionando novos relacionamentos, que não existem na versão original do i* [Yu, 1995].

Adiciona também, um processo de desenvolvimento para cada fase, discutindo os modelos gerados em cada fase e os diagramas usados para descrever estes modelos. Uma linguagem de modelagem definida em termos de metamodelos UML é apresentada para descrever os conceitos usados da metodologia.

Na análise do Tropos'02 e Tropos'04, identificamos que as atividades e produtos gerados apresentam algumas diferenças (Tabela 3-2). Nas fases de requisitos iniciais e requisitos finais, Tropos'02 e Tropos'04 geram artefatos semelhantes: Lista de partes interessadas, Diagrama de Dependência Estratégica que é similar ao Diagrama de Atores e Diagrama de Raciocínio Estratégico que é similar ao Diagrama de Metas. A diferença entre as duas abordagens está na seqüência de atividades proposta. Tropos'04 constrói os modelos seguindo a análise de cada elemento intencional e aplicando quatro estágios de modelagem: modelagem de atores, modelagem de dependências, modelagem de metas e modelagem de planos. Em Tropos'01, a modelagem sugere apenas a elaboração dos dois diagramas SD e SR.

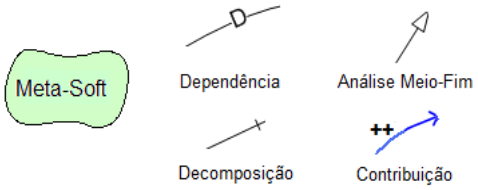
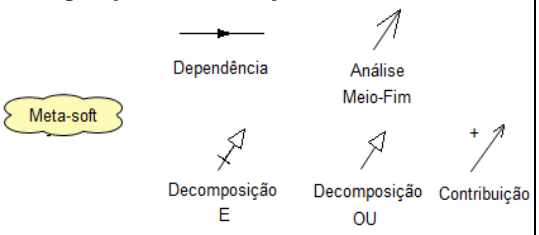
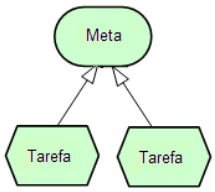
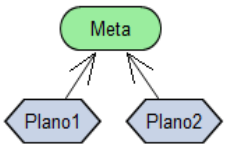
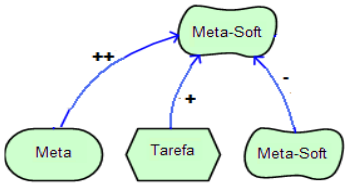
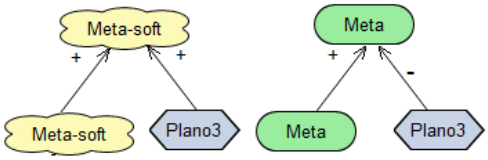
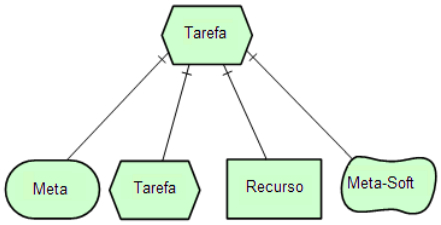
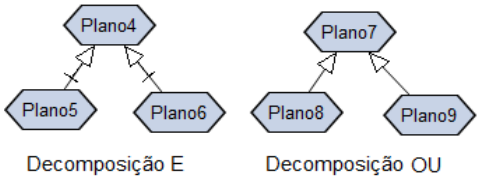
Outra diferença está na versão do framework utilizado. Tropos'02 utiliza a proposta original do i*, enquanto Tropos'04 criou um metamodelo específico acrescentando algumas mudanças, tais como: modificou o termo do modelo i* Tarefa para Plano, acrescentou um novo tipo de decomposição (decomposição-OR); acrescentou decomposição de metas em sub-metas e *metas-soft* em sub-*metas-soft*; a análise meio-fim sempre ocorre de tarefa para metas; e restringiu a decomposição de tarefas apenas a outras sub-tarefas.

Na fase projeto arquitetural, Tropos'01 primeiramente define o estilo arquitetural e a partir do estilo selecionado, atribui novos atores para atender as metas do sistema. As metas são preenchidas com respeito a padrões sociais analisando as capacidades destes padrões. Em Tropos'04, os atores são definidos através da análise das metas, depois o estilo arquitetural é selecionado e novos atores (sugeridos pelo estilo selecionado) são adicionados ao modelo arquitetural. As capacidades de cada ator são definidas a partir da análise dos planos que são gerados para atender as metas. Estas capacidades são classificadas e tipos de agentes são definidos com respeito aos padrões sociais. Os artefatos gerados são os mesmo, mas as duas versões seguem uma seqüência de atividades diferentes para geração do modelo arquitetural.

O projeto detalhado, na versão Tropos'04, gera um modelo de atividades para especificar as capacidades dos agentes, um modelo de atividades para especificar os planos dos agentes e um modelo de seqüência para especificar as interações entre os agentes. Tropos'02 gera um modelo de classes para representar as características do modelo BDI [Rao & Georgeff, 1991] para os agentes. Também usa diagramas de seqüência para modelar os protocolos de comunicação entre os agentes. Os planos dos agentes são modelados a partir de

um diagrama de transição de estados e especificações ConGolog [Lesperance et al., 1999]. A fase de implementação sugere a mesma ferramenta em ambas as versões do Tropos.

Tabela 3-2 – Diferenças entre o i* usado no Tropos'02 e Tropos'04

Diferenças	**Tropos'02**	**Tropos'04**
Nome de diagramas	Diagrama de dependência estratégica	Diagrama de ator
Nome de diagramas	Diagrama de raciocínio estratégico	Diagrama de metas
Versão i*	Usa a proposta original do i*	Usa extensão específica para o i*
Nome de elementos	Usa o termo Tarefa do i* original	Usa o termo Plano
Notação gráfica	Usa notação gráfica do i* original para a meta-soft, ligações de dependência, análise meio-fim, decomposição e contribuição. 	Usa notação gráfica específica para a meta-soft, ligações de dependência, análise meio-fim, decomposição e contribuição. 
Semântica da Análise Meio-Fim	Analise meio-fim: O fim é uma meta. Os meios são tarefas. 	Analise meio-fim: Tem uma meta como fim. Os meios são as metas, planos ou recursos. 
Semântica da Análise de contribuição	Análise de contribuição: ocorre entre uma tarefa e uma meta-soft; entre duas metas-soft; e/ou entre uma meta e uma meta-soft. 	Análise de contribuição: ocorre entre duas metas (ou meta-soft) ou entre um plano e uma meta (ou meta-soft). 
Semântica da decomposi-ção	Decomposição: ocorre entre uma tarefa que pode ser decomposta em subtarefas, metas, metas-soft e/ou recursos. 	Decomposição booleana: ocorre entre uma meta e submetas, *metas-soft* e *submetas-soft* e entre plano e subplanos; Existem dois tipos de decomposição: E e OU. 

3.5 Extensões às fases de Tropos

Vários grupos de trabalho em universidades e instituições de pesquisas distintas (Canadá, Brasil, Itália e Bélgica) têm investido em pesquisas para evoluir e melhorar as fases da proposta Tropos. Estes trabalhos resultaram em novos procedimentos que permitem o detalhamento de métodos usados: diretrizes para o uso do framework i*; refinamentos nas fases do Tropos, tais como o framework SIRA [Bastos, 2005], as visões refinadas do projeto detalhado [Silva, C.T.L.L., 2007a], especificações de capacidades de agentes [Penserini et al., 2006] e alguns trabalhos que envolvem a aplicação do gerenciamento de riscos [Asnar, Gorgini & Mylopoulos 2006], análise de requisitos de segurança [Giorgini & Mouratidis , 2005] ou gerenciamento de requisitos [Pinto, 2008]. Nesta seção descrevemos alguns destes avanços e os seus benefícios para os desenvolvedores que utilizam Tropos para análise e projeto dos seus sistemas.

3.5.1 Procedimentos para guiar a construção dos modelos i*

O framework i* é uma notação usada na modelagem de requisitos, pois permite uma visão explicita das metas dos atores de sistemas e os seus relacionamentos com outros atores [Yu, 1995]. A utilidade do i* tem sido destacada em muitos usos que os pesquisadores tem feito na modelagem de seus processos. Porém a construção de diagramas i* possui algumas dificuldades resultando em maus usos, variações no metamodelo inicial e complexidade nos modelos resultantes [Webster, Amaral & Cysneiros, 2005]. Estas dificuldades, muitas vezes, inibem o uso da metodologia Tropos que tem uma forte dependência dos modelos i* nas fases de requisitos. Alguns trabalhos têm sido realizados para minimizar estas dificuldades e para guiar a modelagem usando i*.

O RiSD [Grau et al., 2005] é uma metodologia para construir modelos SD reduzidos para sistemas de software. Possui um conjunto de atividades estruturadas em duas fases, uma para construir o sistema social (sem o software) e uma para construir o sistema socio-técnico (com o software). O RiSD propõe minimizar o tamanho dos modelos SD construídos fornecendo critérios de escolha para diferentes tipos de elementos intencionais quando existirem diversas opções. Para construir o sistema social, ele segue uma seqüência ordenada de atividades que são: a identificação do ator, o estabelecimento de dependências de metas, a classificação e a análise das dependências. Para construção do sistema sócio-técnico segue as atividades: adicionar o software ao modelo, identificar subsistemas no sistema de software, refinar as dependências do software, identificar dependências entre subsistemas, classificar e

analisar as dependências. A vantagem do RiSD é a seqüência ordenada de passos que auxilia na construção dos modelos SD, minimizando a subjetividade para os modeladores dos sistemas. Porém não prover um método para a construção de modelos SR.

O PRiM (*Process Reengineering i* Methodology*) [Grau; Franch & Maiden, 2005] é uma metodologia baseada no modelo de reengenharia de processo de negócio [Yu, 1995]. Utiliza processos onde a especificação de um novo sistema inicia da observação do processo corrente. A metodologia acrescenta fases: coleta de informações de domínio (especificação do sistema corrente), construção do modelo i* desta informação, raciocínio para descoberta de novas necessidades estratégicas, geração sistemática de alternativas estratégicas e proposta de um framework para direcionar a avaliação de alternativas. Estas contribuições têm o objetivo de construir modelos de processos de forma prescritiva (usando regras, diretrizes, padrões e questões que articulam um caminho bem definido) removendo a incerteza do raciocínio baseado no modelo i*. O PRiM é suportado por uma ferramenta chamada J-PRiM [Grau; Franch & Maiden, 2006]. O J-PRiM não utiliza a representação gráfica, mas uma abordagem em árvore hierárquica. Esta proposta permite melhor gerenciamento dos elementos, uma vez que os modelos gráficos podem se tornar complexos e lentos de gerir, apresentar dificuldades em localizar elementos e dificuldade de extensão. O PRiM faz uso de cenários (scripts DIS - Detailed Interaction Script) para coletar informações do domínio e gera os modelos i* a partir destas informações coletadas. Tem a vantagem de associar métodos de descrição de requisitos em linguagem natural com os modelos gráficos, além de apresentar a ferramenta J-PRiM como suporte. Além disto, apóia a construção tanto de modelos SD quanto modelos SR. Tem como desvantagem, o esforço adicional de especificação de scripts DIS, bem como o uso da ferramenta J-PRiM não gera os modelos gráficos o que pode gerar inconsistência no trabalho manual durante atualizações nos elementos.

SDSituation (Strategical Dependency Situation) [Oliveira; Padua & Cysneiros, 2006] é uma técnica para apoiar a elicitação de requisitos suportada pelo LEL (Lexicon extended language). SDSituation foi criada para documentar dependências estratégicas de requisitos e mostrar a cadeia de interdependências estratégicas que existem no ambiente organizacional. O processo de elicitação proposto é dividido em quatro fases: o primeiro é a definição dos símbolos do universo do discurso para elaborar o LEL. Baseado no LEL, são definidas as situações de dependência estratégica (*SDSituation).* As SDSituations são usadas para definir os cenários. Por fim os cenários serão usados para desenvolver os modelos i*. Esta proposta é semelhante aquela do PRiM, isto é a geração de modelos i* a partir da construção de cenários

abordando os diagramas SD e SR. A diferença está nas técnicas de elicitação de requisitos empregadas.

Cruz Neto (2008) apresenta uma solução para elicitação de requisitos baseada em um processo de pesquisa qualitativa que integra o uso de Grounded Theory [Strauss & Corbin, 1998], da Teoria da Atividade [Nardi, 1996] e da Técnica i* [Yu, 1995]. A proposta é centrada em três pontos principais: um processo indutivo de pesquisa qualitativa baseado nas etapas da Grounded Theory para construção de representações de práticas humanas baseadas na análise sistemática dos dados; uso de um framework analítico para tornar a interpretação qualitativa dos dados mais objetiva para os propósitos de concepção de sistemas de software; e guias para geração de modelos i* de dependências sociais estratégicas a partir das descrições de atividades. Desta forma, são propostas além de uma visão do contexto social, análises transformacionais que atendem às demandas dos engenheiros de software. Como resultado de todo o processo qualitativo é gerado um documento com descrições gerais das práticas que incluem representações de atividades e modelos i*. Tais gráficos são contextualizados no relatório através de uma narrativa que descreve uma história central do estudo realizado, citando as categorias, representações e modelos construídos a partir dos dados empíricos. Desta forma, pretende-se gerar documentos de descrição do contexto social que sejam tanto ricos e detalhados para um melhor entendimento das práticas humanas, como úteis para a natureza criativa da atividade de design de software.

3.5.2 Framework SIRA

O framework SIRA (Systematic Integration between Requirements and Architecture) [Bastos, 2005] descreve um sistema de software da perspectiva de uma organização no contexto do projeto Tropos atendendo as fases de requisitos finais e projeto arquitetural. Ele identifica e mapeia componentes chaves e interações baseadas nos requisitos de sistemas e conceitos organizacionais e ajuda a reduzir a lacuna entre modelos de requisitos para sistemas multiagentes e modelos arquiteturais. O framework SIRA propõe modelos adicionais para especificar propriedades de grupos organizacionais e ajudar a derivar as propriedades arquiteturais. Os dois modelos complementares são usados para separar os conceitos de organização social e organização arquitetural [Bastos; Castro & Mylopoulos, 2006]. No Modelo organizacional são identificados papéis, interações e o estilo arquitetural. No modelo de atribuição, papéis são agrupados em subgrupos que são mapeados para os componentes da arquitetura selecionada (Figura 3-12).

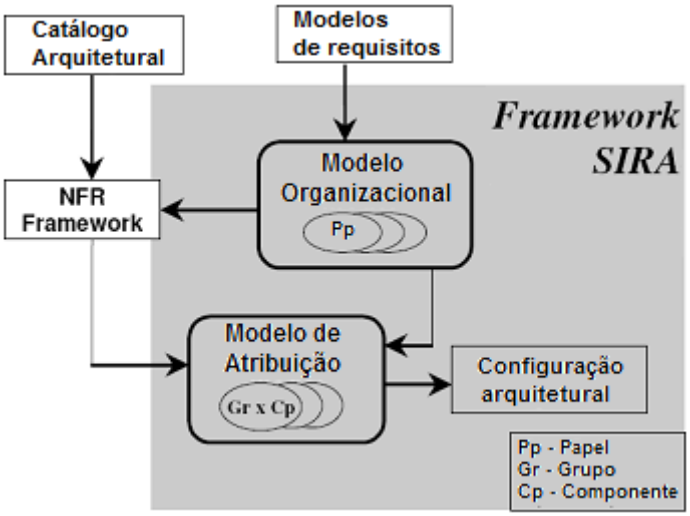


Figura 3-12 - O framework SIRA.

A Especificação do modelo organizacional envolve o mapeamento do modelo de requisitos i* no modelo organizacional SIRA. Metas, *metas-soft* e tarefas dos diagramas de dependência estratégica e diagramas de raciocínio estratégicos são a base para um contexto organizacional, e são usados para identificar um grupo social (subgrupos, papéis e interações) que os sistemas multiagentes devem fornecer. Além disto, é definida uma alternativa arquitetural para o sistema multiagentes. O modelo parte do entendimento que interações organizacionais têm como objetivo a obtenção de algumas metas globais desejadas e que os membros (atores e grupos) são autônomos e heterogêneos. A Especificação do modelo organizacional inclui três sub-atividades: refinamento das tarefas e metas; identificação de papéis e seleção arquitetural. Na sub-atividade refinamento de tarefas e metas, os atores são grupos organizacionais com responsabilidades a realizar. As responsabilidades do grupo são identificadas das dependências de metas (do modelo de raciocínio estratégico gerado na fase requisitos finais). As dependências de metas são decompostas através da decomposição de metas e a análise meio-fim é usada para identificar as tarefas. Este refinamento é realizado para identificar tarefas e subtarefas tal que estas exijam a cooperação de menos papéis.

O próximo passo é identificar os papéis e as interações. Na sub-atividade Identificação dos papéis, o interesse está na identificação da seqüência e tipos de tarefas que podem ser realizadas por um ator em colaboração com outros. Esta identificação envolve a distribuição de tarefas e interações para realizar cada meta do grupo de maneira coordenada. Os papéis podem ser identificados de um domínio específico. O próximo passo é especificar cada um destes papeis e os seus protocolos de interações. A especificação dos papéis envolve os conceitos de nome, objetivo, responsabilidades, colaboradores, habilidades, direitos, e

normas. Já a especificação de interações engloba os conceitos de nome, objetivo, iniciador, respondedor, entradas e saídas. Os padrões de interação que descrevem a estrutura social podem ser vistos como uma rede de relações. O Framework SIRA utiliza dois tipos de ferramentas matemáticas para representar interações entre papéis: matrizes e gráficos de rede social. Além das interações dos papéis, os padrões de comportamento são especificados através de metas que devem ser satisfeitas ou evitados pelos atores. Isto é representado como normas que devem ser impostas a um agente da missão.

O último passo para a construção do modelo organizacional é a análise das alternativas arquiteturais. Esta fase ocorre da mesma forma que na versão Tropos'02. São identificadas as *metas-soft* do sistema e analisadas em relação aos estilos organizacionais através do catálogo de estilos arquiteturais (Tabela 3-1), com o objetivo de selecionar um estilo para a arquitetura do sistema.

O modelo de atribuição SIRA propõe a análise da rede social [Hanneman & Riddle, 2005] como um método para ajustar e descrever os efeitos de características organizacionais e para relacionar sistemas multiagentes e arquiteturas organizacionais. O método é útil por que agrupa papéis em subgrupos medindo a força das relações entre membros no grupo. O modelo de atribuição associa subgrupos a um estilo arquitetural particular. Ele se propõe a responder duas questões: "Como agrupar papéis a subgrupos?" e "Como mapear subgrupos em componentes arquiteturais?". Um ator quando desempenha um papel é um sistema autônomo que pode ser descrito como um conjunto de interações. Estas interações são analisadas usando duas técnicas: a análise da centralidade e a análise da similaridade estrutural. A análise da centralidade indica quais são os papéis mais proeminentes e quais são os mais influentes do sistema. A análise da similaridade estrutural indica papéis com relacionamentos semelhantes. A integração do grupo e o grau de similaridade são calculados e, baseado nestes resultados, isto é, na identificação dos atores similares, os atores são agrupados em subgrupos. O mesmo processo é realizado para os componentes do estilo arquitetural. Estes resultados serão utilizados para realizar o mapeamento entre os subgrupos gerados a partir dos requisitos funcionais e os componentes do estilo arquitetural selecionado.

Em seguida realiza-se a análise da correlação entre subgrupos e componentes arquiteturais baseado na medida da centralidade. Esta medida pode revelar muito sobre a estrutura organizacional. Por exemplo, indica as relações entre os subgrupos e componentes como correlações fortes, correlações parciais e nenhuma correlação. Indica atores que dominam a estrutura organizacional. Indica se a arquitetura é muito centralizada e sujeita a falhas, caso algum ator dominante falhe. Após a análise de correlação da centralidade, os

padrões de relações do estilo organizacional selecionado são avaliados em termos de sua correlação de similaridade com os subgrupos SIRA.

É possível derivar a primeira configuração organizacional baseado nos resultados preliminares obtidos até aqui. Pela análise das correlações, o mapeamento dos subgrupos aos componentes da arquitetura é realizado. As correlações mais fortes estabelecem o mapeamento [Bastos & Castro, 2005]. A seção 6.3.3 apresenta um exemplo completo utilizando o framework SIRA como diretriz para guiar o projeto arquitetural a partir dos requisitos funcionais e não funcionais do sistema DBSitter-AS.

3.5.3 Projeto detalhado em UML

O trabalho de Silva, C.T.L.L. (2007a) propõe um novo conjunto de modelos para a fase de projeto detalhado. Esta tese defende que o uso de i* como uma linguagem de descrição arquitetural (ADL) não é apropriada, pois o i* tem limitações para expressar informações exigidas por arquiteturas de SMA, tais como portas, conectores, protocolos e interfaces. Por este motivo, é estudado o uso de UML 2.0 para descrever o projeto de um sistema. São propostas três contribuições para esta área: um metamodelo de agência, para definir os construtores exigidos para especificar características estruturais e dinâmicas de SMA de acordo com o modelo BDI e padrões FIPA; seis visões de projeto arquitetural modelados por diagramas UML; e um conjunto de diretrizes para ajudar a especificação de SMA de acordo com estes diagramas.

O metamodelo de agência estendeu a UML com os conceitos de Sistemas Multiagentes, utilizando estereótipos para representar estes conceitos. O metamodelo está organizado em duas categorias: intencional e interacional. Na categoria intencional (Figura 3-13) um SMA pode ser compreendido como uma organização que é composta de um número de papéis e outras organizações. Os papéis possuem metas, intenções, planos e crenças. A organização está inserida em um ambiente, que é composto de recursos com direitos de acesso definidos. Uma ação (complexa ou básica) determina os passos para realizar um plano. Um plano tem duas subclasses: um macro plano, se o plano é definido pelo papel do agente e um micro plano, se o plano é definido por um agente. O macro plano é parcial e é composto de ações complexas. O micro plano é o plano final e completo, composto de ações básicas. Crenças são as informações que o agente tem sobre si mesmo e sobre o ambiente. São precondições para executar os planos e organizar as ações dos planos.

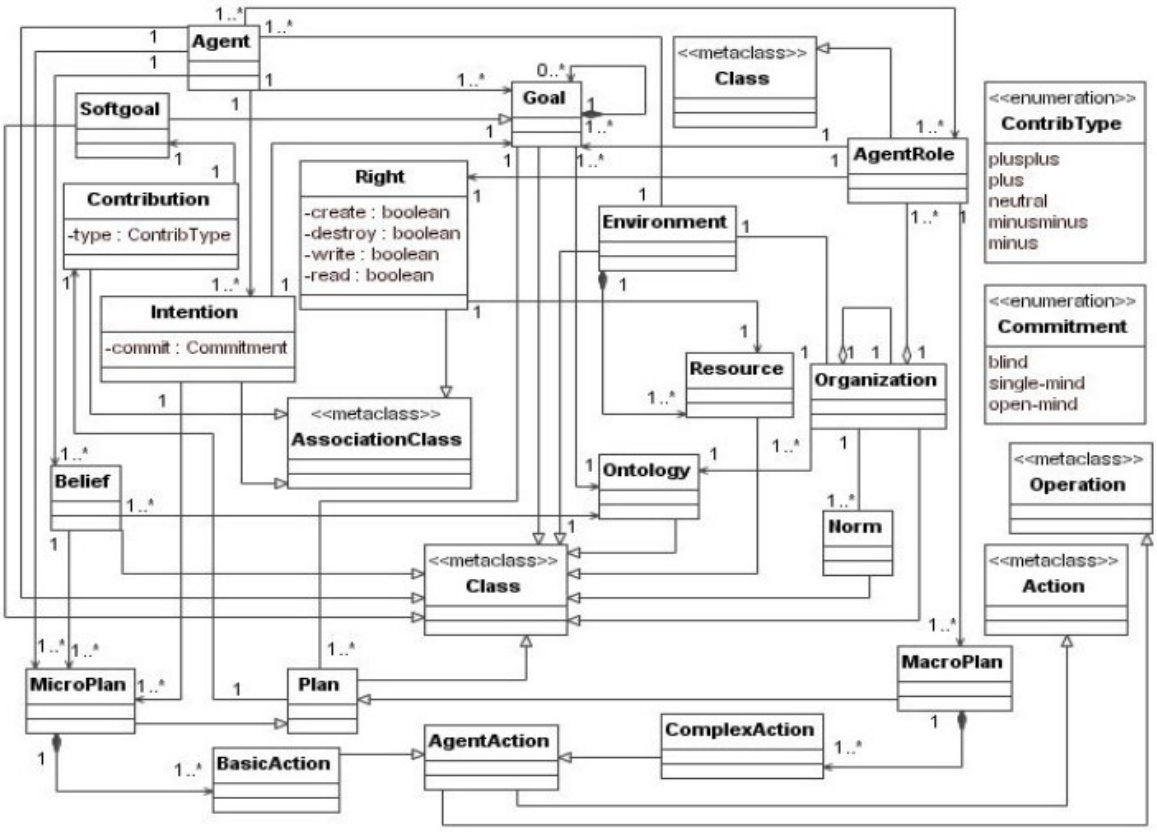


Figura 3-13 – Metamodelo de agência – Categoria intencional

Na categoria interacional, usam-se as seguintes características arquiteturais/ organizacionais: dependum, *dependee*, *depender,* conectores, portas organizacionais e conectores finais. Um *dependum* define um acordo de serviço entre dois papéis de agente (*dependee* e *depender*). O *dependee* é o papel do agente que prover serviços. O *depender* é o papel do agente que requer serviços. Um *dependum* pode ser de quatro tipos: metas, *metas-soft*, tarefas e recursos. Os papéis do agente trocam sinais através de um conector para executar o acordo contratual de prover serviços para a organização. Uma porta organizacional especifica um ponto de interação entre um papel de agente e seu ambiente. Um conector final liga os conectores a uma porta organizacional. Cada papel do agente pode interagir com outros de acordo com um protocolo de interação. Um protocolo de interação descreve uma seqüência de mensagens que podem ser enviadas ou recebidas por agentes que executam papéis. As mensagens podem ser de vários tipos: REQUEST, INFORM e REFUSE, entre outras definidas pela FIPA.

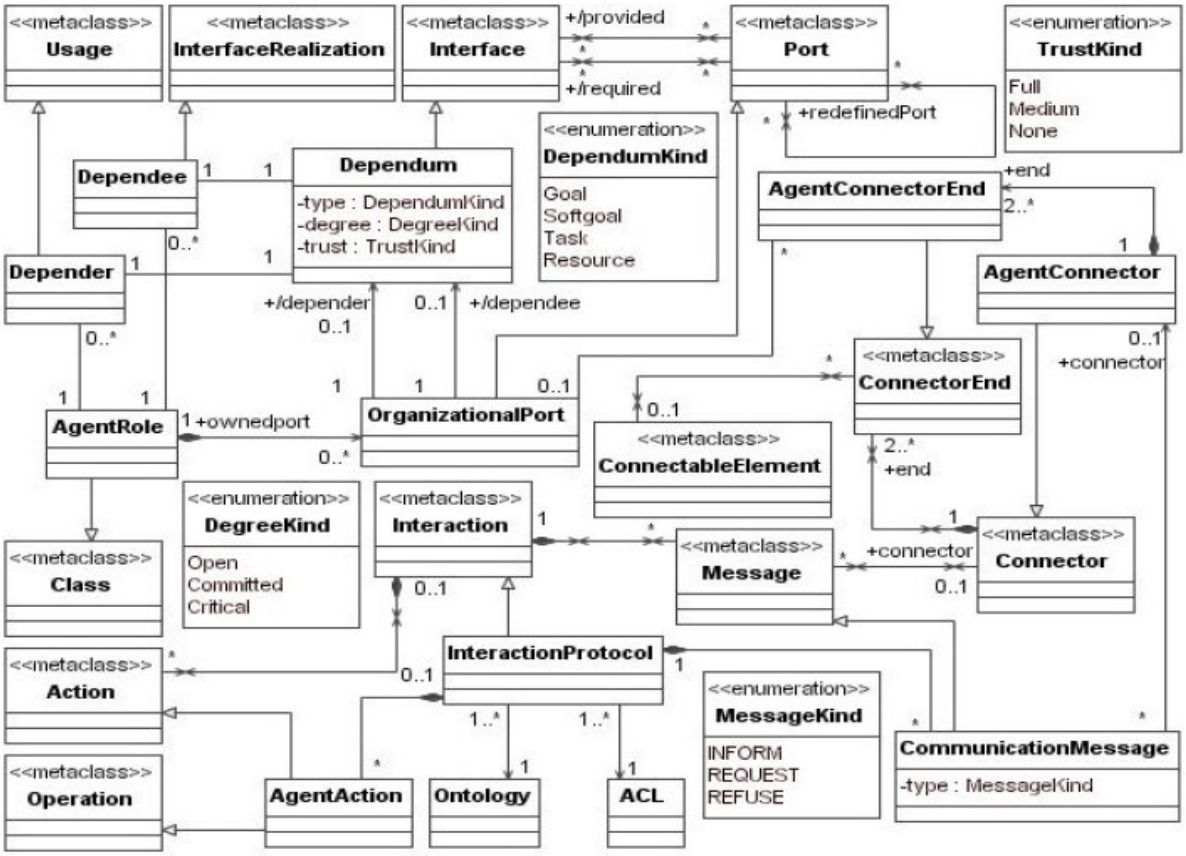


Figura 3-14 – Metamodelo de agência – Categoria interacional

Foram criados diagramas para modelar seis visões de projetos de SMA: diagrama arquitetural, diagrama de comunicação, diagrama ambiental, diagrama intencional, diagrama de raciocínio e diagrama de planos [Silva, C.T.L.L. et al., 2007b]. O diagrama arquitetural modela a visão estática de módulos que compõe um sistema. É definido em termos de papéis que possuem metas obtida através da execução de planos. O diagrama de comunicação é uma extensão do diagrama de seqüência da UML com mensagens assíncronas. É usado para modelar os padrões de comunicação de um conjunto de entidades à medida que eles interagem para implementar o comportamento. O diagrama ambiental estende o diagrama de classes para modelar papéis compreendendo uma organização que é situada em um ambiente. O ambiente é composto por recursos, que são acessados de acordo com seus direitos para executar suas responsabilidades. O diagrama intencional estende o diagrama de classes para modelar papéis, agentes, crenças, metas, planos e as normas e ontologias usadas na organização. O diagrama racional estende o diagrama de classes da UML para modelar metas-soft, contribuições de planos para satisfazer metas-soft, metas, planos e papéis dos agentes. O diagrama de plano estende o diagrama de atividades da UML para modelar a seqüência de ações que compõe os planos para obter as metas. Podem-se analisar as ações que são realizadas em paralelo e quais são dependentes de outras ações. Pode ser usado para modelar

a seqüência de ações em macro planos (ações complexas descritas pelo papel) ou micro planos (ações básicas definidas pelos agentes).

Para geração dos diagramas foi especificado um processo de projeto detalhado (Figura 3-15) em SPEM [SPEM, 2007] que introduz detalhes adicionais para a estrutura e comportamento de cada componente arquitetural de um sistema [Silva, C.T.L.L. et al., 2007c]. Este processo está agrupado em três definições de trabalho, que são o Projeto detalhado dos agentes, a Seleção dos padrões sociais e a Aplicação dos padrões sociais. O Projeto detalhado dos agentes é composto por oito atividades: realizar a análise meio-fim ajuda a identificar as razões/motivações associadas com cada dependência que um agente possui e produz o modelo SR de arquitetura; mapear i* para UML usa um conjunto de heurísticas para guiar o mapeamento das descrições i* para a notação baseada em UML e produz o documento de mapeamento. Como resultado da especificação do modelo arquitetural é produzido o modelo de arquitetura. Já a especificação do modelo racional produz o modelo racional. Por sua vez, a especificação do modelo de plano produz o modelo de planos. Enquanto a especificação do modelo ambiental produz o modelo ambiental. E a especificação do modelo de comunicação produz o modelo de comunicação. Por fim, a especificação do modelo intencional produz o modelo intencional. A definição de trabalho Seleção de padrões sociais é responsável pela análise das características de cada padrão social e pela produção do documento de decisões de projeto com os padrões selecionados. Já a definição de trabalho Aplicação dos padrões sociais é responsável pela aplicação do padrão social e atualização dos modelos baseados em UML.

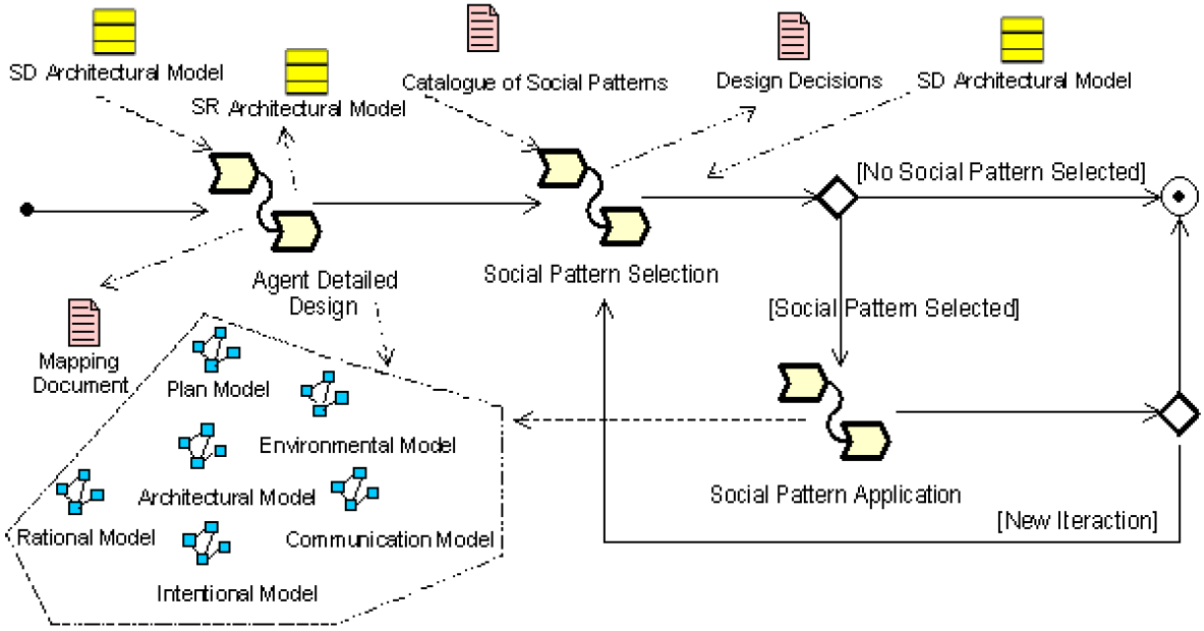


Figura 3-15 – Definições de trabalho do processo de projeto detalhado

3.6 Considerações Finais

O projeto Tropos surgiu como uma proposta inovadora na área de engenharia de software orientado a agentes, uma vez que é o único que incluí conceitos de requisitos iniciais em suas fases de desenvolvimento. As vantagens de usar o framework i*, que é uma ferramenta de modelagem poderosa baseada na modelagem social e intencional de atores também gerou grande interesse na proposta. Vários trabalhos surgiram com o objetivo de detalhar e evoluir o Tropos. Além disso, métodos para orientar a construção dos modelos i* surgiram para facilitar o seu uso.

Neste capítulo, foi descrita uma breve apresentação de algumas técnicas utilizadas pelo grupo de engenharia de requisitos local. Estas técnicas compreendem tanto trabalhos desenvolvidos por outros grupos de pesquisas (por exemplo, as técnicas que ajudam na geração dos modelos i* na elicitação de requisitos: PRIM, RiSD, SDSituation), quanto trabalhos desenvolvidos pelo grupo para as disciplinas de Projeto arquitetural (Framework SIRA) e Projeto detalhado (Visões de Projeto detalhado em UML).

O PRIM, RiSD, SDSituation e o "Estudo qualitativo para elicitação de requisitos" são algumas técnicas que permitem melhor detalhamento da elicitação dos requisitos e geração de diagramas do modelo i*. Este detalhamento pode ajudar a obter diagramas mais completos, menos complexos e mais coerentes. Além de prover documentos auxiliares, como os cenários, que documentam os elementos presentes nos diagramas. O framework SIRA é uma interessante contribuição para o Tropos, pois trata de lacunas existentes na análise dos requisitos e identificação dos componentes arquiteturais, fornecendo técnicas para gerar um projeto mais consistente com os modelos organizacionais de sistemas multiagentes. Os avanços no projeto detalhado permitiram obter visões mais detalhadas dos elementos conceituais dos agentes e suas intenções. Também descreveu melhor o ambiente em que estes os agentes estão inseridos e a sua interação com a organização.

Outros trabalhos têm surgido para auxiliar às áreas de gerenciamento dentro de um processo de desenvolvimento tais como: no gerenciamento de requisitos, o trabalho de Pinto (2008) propõe um modelo de referência e um processo para encontrar as informações de rastreabilidade relevantes durante as fases de requisito posterior e arquitetura do Tropos; o tratamento de aspectos de segurança de Giorgini & Mouratidis (2005) que introduz extensões ao Tropos para atender requisitos de segurança durante o desenvolvimento do sistema; e a análise de riscos de Asnar; Gorgini & Mylopoulos (2006) usa uma abordagem orientada a metas para modelar e analisar riscos nas fases de requisitos. Estes trabalhos são extensões às

fases originais da metodologia e podem também ser adicionados a atividades de suporte e gerenciamento do desenvolvimento do software.

Tropos provê um conjunto de atividades, que estão embutidas em sua descrição, ou seja, as atividades estão descritas em linguagem natural e algumas vezes a sua identificação não é muito clara devido a ambigüidades da linguagem natural. Em geral houve apenas uma preocupação com a natureza estrutural destas atividades. Os aspectos dinâmicos do ciclo de vida do processo de desenvolvimento não são mencionados. Alguns estudos [Wautelet; Kolp; Achbany, 2005] têm classificado Tropos como uma proposta orientada a um modelo de desenvolvimento em cascata, enquanto outros [Cernuzzi; Cossentino; Zambonelli, 2005] apontam Tropos como uma proposta de desenvolvimento iterativo. Nesta dissertação é proposto um modelo de processo iterativo e incremental para a metodologia Tropos. Contudo, antes de detalhar esta proposta, no próximo capítulo é apresentada uma breve introdução aos conceitos básicos relacionados aos modelos de processo mais utilizados e como a comunidade de desenvolvimento de sistemas multiagentes tem utilizado estes conceitos em seu trabalho.

Capítulo 4 – Processo de desenvolvimento de software

Neste capítulo são apresentados os conceitos e modelos de processos usados para o desenvolvimento de software. Atenção especial é dada aos modelos de processo para desenvolvimento de sistemas multiagentes. O SPEM, que é uma abordagem bastante popular para descrição de processos de desenvolvimento de software, também é apresentado. Por fim, é indicado como Tropos poderá ser descrito através de SPEM.

4.1 Introdução

Softwares são produtos complexos e difíceis de desenvolver e testar. Muitos softwares falham exibindo comportamentos inesperados causando enormes prejuízos aos usuários e desenvolvedores. Por este motivo pesquisadores e profissionais da área empregam esforços para entender e melhorar a qualidade do software que estava sendo desenvolvido [Fuggetta, 2000]. Um processo bem definido de software é um fator crítico para gerar sistemas com qualidade, pois ajuda a gerenciar e transformar as necessidades do usuário em um produto que realmente atenda estas necessidades.

Existem diversos conceitos relacionados aos processos de desenvolvimento [Cernuzzi; Cossentino & Zambonelli, 2005]. Os conceitos de processo, metodologia, métodos e modelos de processos são alguns exemplos. Um processo de desenvolvimento é um conjunto de passos ordenados que envolva atividades, restrições e recursos exigidos para produzir um conjunto de artefatos para satisfazer um conjunto de requisições. Estendendo este conceito para software, um processo de desenvolvimento de software é definido como um conjunto coerente de políticas, estruturas organizacionais, tecnologias, procedimentos e artefatos necessários a conceber, desenvolver, distribuir e manter um produto de software [Fuggetta, 2000]. Por esta definição o desenvolvimento de software engloba mais do que construir o software, mas envolve a concepção, distribuição e manutenção. Um modelo de processo de software descreve fases, iterações e coordenações sob as quais o desenvolvimento é organizado. Não se preocupam em definições, guias e estilo de modelagem, mas como estas podem ser mudadas e adaptadas em cada situação. Um método é uma maneira de realizar algum tipo de trabalho dentro de um processo para produzir uma saída específica. Uma metodologia é um conjunto de métodos que cobrem e conectam diferentes estágios em um processo.

Vários modelos de processos foram criados para o desenvolvimento de software, tais como o modelo em cascata [Royce, 1970], modelos evolucionários [May & Zimmer, 1996] e modelos iterativos [Basili & Turner, 1975]. Estudaremos estes modelos em mais detalhes nas seções seguintes. Desde que se iniciaram os estudos sobre melhorias de processos, várias formas de representação foram criadas e os processos passaram pelos mesmos problemas de falta de padrão que as linguagens de modelagem orientada a objetos passaram, até a definição da UML. Devido a isto, as companhias de software que fazem parte da OMG resolveram criar um metamodelo para especificação de processos, o SPEM [SPEM, 2007]. SPEM está sendo

muito usada hoje para descrição dos processos. A comunidade orientada a agentes tem se preocupado em criar métodos e metodologias para o desenvolvimento de softwares multiagentes e alguns esforços estão sendo empregados pela FIPA para criação e padronização de processos usando SPEM. Tropos também está especificada usando SPEM e um processo foi proposto usando o modelo de processo em espiral [WAUTELET; KOLP & ACHBANY, 2005].

Neste capítulo serão apresentados alguns modelos de processos de softwares mais conhecidos, a definição da linguagem de especificação SPEM e as especificações de processos para Tropos.

4.2 Modelos de processo de desenvolvimento de software

Um modelo de processo é uma estrutura sobre a qual definimos os estágios do desenvolvimento de um produto. Os primeiros softwares eram desenvolvidos de forma caótica. Uma especificação era criada e a codificação realizada sobre esta especificação. Se ocorresse algum erro ou necessidade de mudança, esta era realizada sob muito esforço de desenvolvimento. Muitas vezes o tempo de manutenção e correção era bem maior que o tempo de desenvolvimento. Com a necessidade de mais qualidade no produto final, foram criadas técnicas de melhorias e vários modelos surgiram.

O modelo em cascata [Royce, 1970] ficou conhecido nos anos setenta, com uma proposta para estruturar as fases de desenvolvimento de um sistema complexo. Este modelo de desenvolvimento inclui sete fases: os requisitos do sistema, requisitos do software, análise, desenho do programa, codificação, teste e operação. Cada fase é realizada de forma completa, antes de iniciar a próxima fase. O processo é demarcado com pontos de controle bem definidos para facilitar a gestão de projetos. Porém é um processo rígido, que exige que as fases de requisitos, análise e projeto estejam bem formadas, pois os testes e correções só ocorrem ao final do desenvolvimento. Portanto se ocorrerem erros nos estágios anteriores estes se propagam até o produto final. Uma variante permite a realimentação entre as fases permitindo a correção de uma fase para outra, diminuindo o risco de propagação de erros (Figura 4-1).

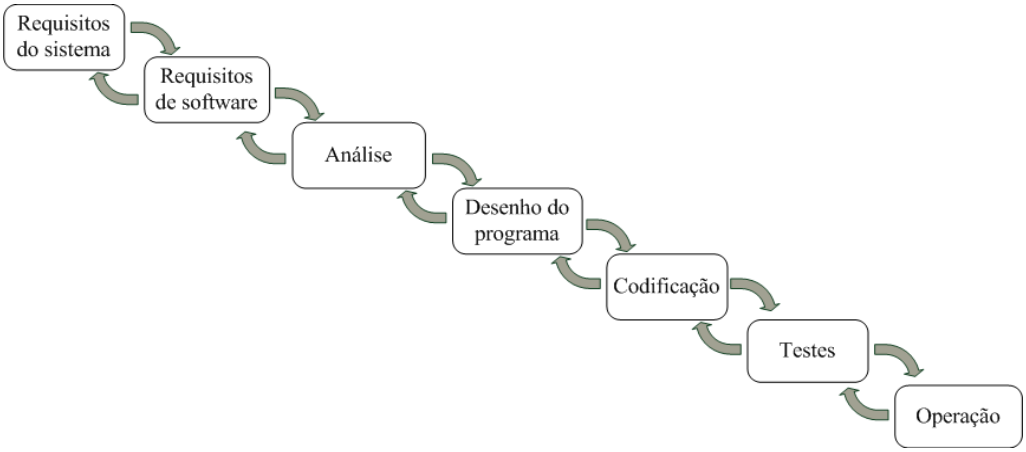


Figura 4-1 – Modelo Cascata

O desenvolvimento formal de sistemas ou modelo de transformação é uma abordagem que possui características semelhantes ao modelo em cascata, com a diferença que a especificação dos requisitos é expressa em notação matemática e o projeto, implementação e testes são substituídos por um processo transformacional da especificação em um produto executável. Este processo é adequado a sistemas que tenham rigorosas exigências de segurança, confiabilidade e garantia. Um exemplo clássico é o processo Cleanroom [Mills; Dyer & Linger, 1987].

O desenvolvimento evolucionário [Gilb, 1981] é realizado de forma incremental, através da liberação de versões que evoluem gradativamente de acordo com o feedback do usuário (Figura 4-2). O modelo ajuda os desenvolvedores dividindo o processo de desenvolvimento em subprojetos menores que contém todas as atividades desde o planejamento e coleta de requisitos até o projeto, implementação, testes e entrega. Cada subprojeto, chamado de ciclo evolucionário, é construído a partir dos novos requisitos dos clientes que foram levantados durante a validação de um produto entregue em um ciclo anterior. Existem dois tipos de desenvolvimento evolucionário. O primeiro é o desenvolvimento exploratório, onde o sistema é desenvolvido iniciando com os requisitos mais compreendidos pela equipe e vai evoluindo com novos requisitos até atingir o sistema completo. O segundo é o protótipo descartável, onde se utiliza a construção de protótipos do sistema para detalhar requisitos mal compreendidos. O processo evolucionário é mais eficaz do que o modelo em cascata uma vez que a especificação é desenvolvida gradativamente e é passível de adaptação a novos requisitos. Porém, torna o gerenciamento difícil, pois o processo não é visível ao gerente. Os sistemas podem ficar mal-estruturados, devido às freqüentes mudanças. E são necessárias ferramentas e técnicas especiais, onerando a equipe com habilidades específicas [Sommerville, 2005].

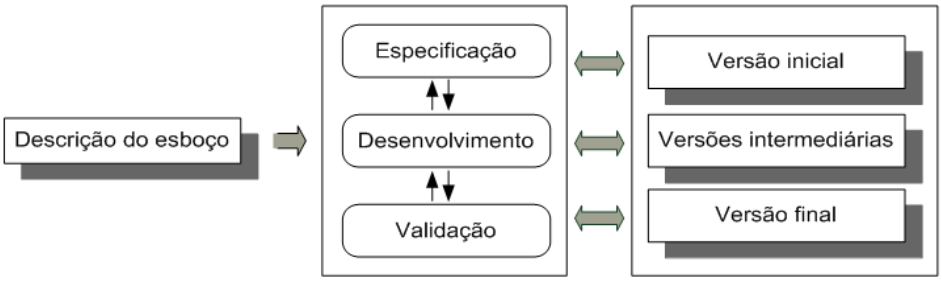


Figura 4-2 – Modelo evolucionário

Outras abordagens envolvem o desenvolvimento iterativo que combina vantagens dos modelos clássicos e desenvolve o sistema de forma iterativa, onde partes do processo são repetidas, à medida que o sistema é definido. O desenvolvimento incremental (Figura 4-3) é um modelo iterativo que combina as vantagens dos modelos em cascata e modelo evolucionário [Mills et al., 1980]. Neste modelo os clientes identificam as funções em um esboço do sistema determinando a prioridade entre estas. Em seguida os incrementos são definidos. Um incremento é um subconjunto das funcionalidades que serão entregues em estágios determinados, por ordem de maior prioridade. Uma vez que são determinados os incrementos, os requisitos são detalhados e desenvolvidos. O cliente recebe o incremento e coloca em operação para testes e validação dos usuários. À medida que novos incrementos são concluídos, estes são associados ao sistema já entregue aos usuários. Novas mudanças podem ser acrescentadas a um novo incremento ou postergadas para o fim do projeto, dependendo da negociação com os clientes. Cada incremento pode ser desenvolvido usando um modelo específico. Se os requisitos estão bem definidos, pode-se usar o modelo em cascata para o incremento, senão usa uma abordagem evolucionária.

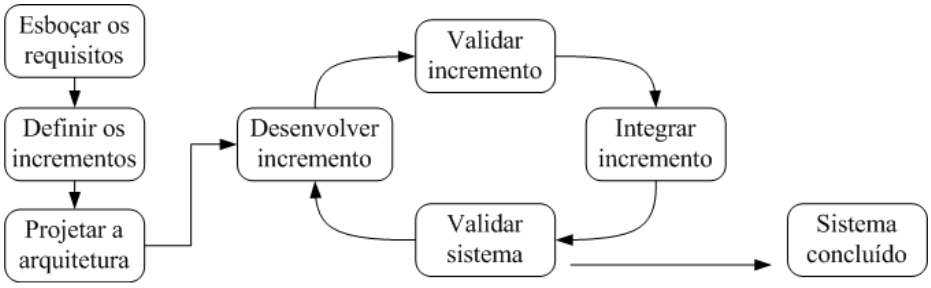


Figura 4-3 – Modelo de processo iterativo e incremental

As vantagens deste modelo são: o cliente pode começar a utilizar o sistema antes da conclusão de todo o projeto; o cliente fica mais maduro para definir os novos requisitos; o risco de fracasso do projeto completo diminui, já que menores incrementos podem ter mais sucesso; as funções mais prioritárias são mais testadas, já que são as primeiras a ser entregues. Uma desvantagem é a dificuldade em identificar os incrementos de forma que seja possível

caracterizar os requisitos básicos que gerem uma solução satisfatória para o cliente. Pois cada incremento deve ser pequeno, para garantir suas vantagens, mas não pode gerar a sensação de sistema incompleto e ineficiente.

Outro modelo iterativo bastante conhecido é o desenvolvimento em espiral [Boehm, 1987]. O processo é representado como uma espiral dividida em quatro setores (Figura 4-4): o setor de definição dos objetivos do sistema, onde as restrições do processo e produto são identificadas, o plano de projeto é preparado e os riscos são identificados; o setor Avaliação e análise de riscos, onde para cada risco identificado são definidas e aplicadas estratégias de resolução dos riscos; o setor de Desenvolvimento e verificação, onde é escolhido e aplicado um modelo de desenvolvimento de acordo com as características do sistema; e o setor de Planejamento, onde são revisados e definidos os planos do sistema. Em cada ciclo da espiral do processo é realizado um plano sob o qual o ciclo será gerenciado. Cada ciclo do processo representa uma fase de desenvolvimento do sistema. Assim um ciclo mais interno inicia pela análise dos riscos do projeto para determinar sua viabilidade. Se o projeto for viável, os conceitos da operação do sistema são definidos e o plano de requisitos é formulado. Outro ciclo inicia definindo-se os objetivos do ciclo de requisitos, a análise dos riscos deste ciclo, a elicitação e análise de requisitos e o plano para a próxima fase que seria o projeto do produto. E assim, para cada fase do projeto é gerado um novo ciclo. Dependendo do tipo de sistema, as atividades das fases podem ser modificadas, por exemplo, criando-se mais de um ciclo para a fase de requisitos. Um dos conceitos mais fortes do modelo em espiral é o tratamento dos riscos. O projeto é dirigido à identificação e resolução de riscos nos setores de planejamento, definição dos objetivos e análise de riscos. Esta é a principal característica que diferencia este modelo dos demais modelos iterativos.

No final dos anos 90, o interesse em modelos de processos de software cresceu de forma acelerada. Surgiram vários novos modelos de desenvolvimento iterativo e incremental. O desenvolvimento em cascata começou a ser bastante criticado e o modelo evolucionário encorajado. Modelos comerciais começaram a surgir como novas propostas de modelos iterativos e incrementais. Em 1994 surgiu o processo iterativo para suportar as práticas de desenvolvimento Rápido (RAD) [Stapleton, 1997]. Em meados de 1996 foi publicado o processo unificado da Rational [Kruchten, 1996], um dos modelos mais conhecidos e adotados nas empresas de desenvolvimento de software. Em 1996, também foram iniciadas as práticas do processo XP (Extreme Programming) que se popularizou devido à ênfase na simplicidade e agilidade do desenvolvimento dos produtos [Beck, 1999]. Em 1999 foi publicado o modelo Scrum [Beedle et al., 1999], que adotava desenvolvimento em iterações

de trinta dias. Em 2001 foi criada a aliança ágil que promoveu os processos ágeis [Cockburn, 2002] baseados em princípios e modelos simples de desenvolvimento (e.g XP e Scrum), em contrapartida aos modelos que tinham foco em uma grande quantidade de documentação (e.g RUP).

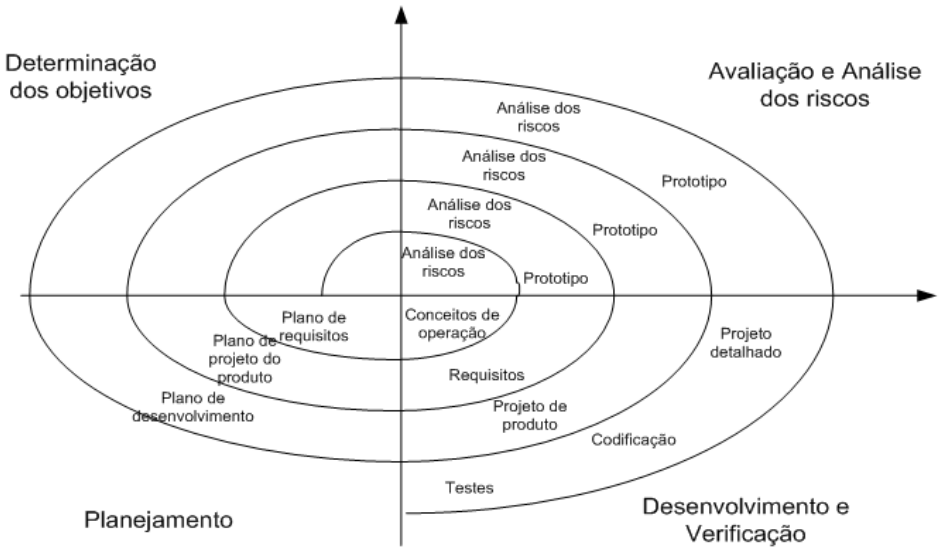


Figura 4-4 – Modelo de processo em espiral

4.3 Modelos de Processos para sistemas multiagentes

A engenharia de software orientada a agentes tem gasto um considerável esforço em criar metodologias para guiar o desenvolvimento. Estas metodologias foram criadas sem considerar um modelo de processo que possa ser usado na coordenação de projetos de desenvolvimento. Existem alguns esforços de pesquisadores para associar estas metodologias com modelos de processo através da análise das características apresentadas em suas descrições. Cernuzzi; Cossentino & Zambonelli (2005) analisa a relação entre as metodologias e os modelos de processo e identificam os modelos em cascata, evolucionários e incremental como os mais adotados pelas metodologias orientadas a agentes. GAIA e Prometheus usam o modelo em cascata. São consideradas assim, devido à seqüência rigorosa das fases da metodologia e não fazem menção a nenhuma iteração entre as fases. MaSE e AOR possuem características do modelo em cascata, embora tenham alguns estágios iterativos dentro das fases. OPM/MAS, MASSIVE, INGENIAS, Tropos e PASSI foram classificados como modelos evolucionários e incrementais. DESIRE é classificada como incremental e transformacional. MAS-commonKADS é citada como tendo a vantagem de mostrar que um modelo de processo em espiral pode ser usado no contexto de SMA. Esta análise é uma visão de Cernuzzi; Cossentino

& Zambonelli (2005) e nenhuma destas metodologias faz qualquer referência explícita a adoção de um modelo de processo.

Cossentino & Seidita (2004) propõe um novo processo de engenharia de software (SEP - Software engineering process) com o objetivo de reunir partes de processos, chamados fragmentos de métodos, de um repositório de métodos. Um fragmento de método é composto de uma porção do processo, artefatos de entrega, precondições, conceitos do metamodelo de SMA, guias de aplicação do fragmento, glossário e outras informações a respeito de ambiente, plataforma de desenvolvimento, etc. O processo inicia com a criação da base de fragmentos de métodos extraídos de metodologias já existentes. O projetista identifica o metamodelo que ele quer seguir e, usando uma ferramenta CAPE (Computer aided process engineering) ou CAME (Computer aided method engineering), seleciona os fragmentos desejados para compor o modelo de processo. Uma vez que o novo SEP tenha sido composto, uma ferramenta CASE é usada para projetar o sistema que resolverá um problema específico (Figura 4-5). Esta proposta sugere o uso de um processo iterativo e ágil seguindo as estratégias do manifesto ágil e as atividades do processo XP [Beck, 1999]. Esta proposta foi aplicada a metodologia PASSI e foi criado o Agile PASSI. Embora esta abordagem possa ser aplicada a qualquer metodologia.

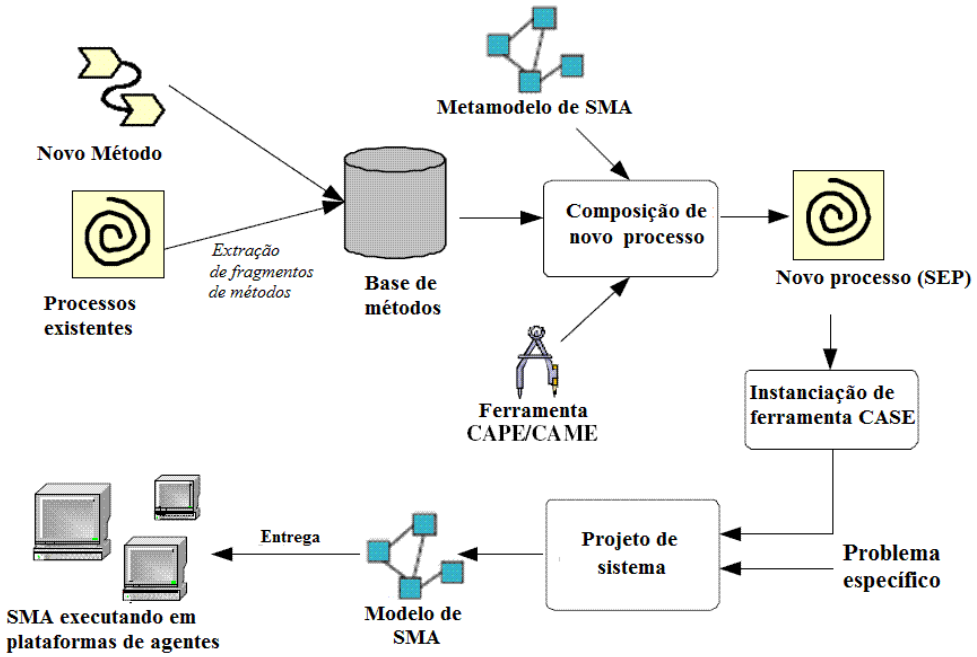


Figura 4-5 – SEP – Processo de engenharia de software

Esta iniciativa está sendo incentivada pelo comitê técnico da FIPA [FIPA, 2008] que está criando um repositório com as metodologias mais conhecidas. Foram definidos padrões para criação dos modelos de fragmentos do repositório tais como, o SPEM [SPEM, 2007]

para modelar processos, UML [UML, 2007] para modelar artefatos, FIPA [FIPA, 2007] como arquitetura de agentes e XML [XML, 2008] para representar os dados. A FIPA tem tido interesse em criar este repositório e algumas metodologias tais como GAIA, PASSI, ADELFE e inclusive Tropos foram modeladas em SPEM e adicionadas ao repositório.

Uma proposta semelhante é apresentada em Garcia-Odeja et al. (2007), onde é proposto o framework de processo de engenharia de sistemas multiagentes baseado em organização (O-MASE). Da mesma forma que no SEP, também é construído um processo usando fragmentos de métodos. O-MASE consiste de três estruturas básicas: um metamodelo pra definir os conceitos chaves de projeto e implementação de SMA; os fragmentos de métodos; e diretrizes para identificar como os fragmentos estão relacionados. O-MASE usa o Open Process Framework (OPF) [Firesmith & Henderson-Sellers, 2002] para descrever o processo, enquanto que o SEP usa o SPEM. O OPF é uma abordagem para produção de processos customizados que consiste de estágios, atividades, tarefas, técnicas, produtores, produtos de trabalho e linguagens.

A abordagem de Knublauch (2002) propõe a aplicação do modelo XP para desenvolver sistemas multiagentes. São proposto modelos e metamodelos simples, tal que interações de agentes possam ser modeladas, atualizadas e comunicadas rapidamente. São construídos e mantidos apenas dois modelos: o código fonte do agente executável (com os casos de testes) e um modelo de processo. O modelo de processo descreve cenários de aplicações de agentes como os cartões de estória de usuário (*story card*). Se o modelo inicial estiver incompleto ou errado, este é atualizado após o *feedback* da implementação gerando um modelo cíclico que passa pela implementação e a atualização do modelo.

4.4 SPEM - Software process engineering metamodel

O *Software Process Engineering Metamodel* (SPEM) é um metamodelo usado para descrever um processo de desenvolvimento de software concreto [SPEM, 2007]. Foi criado pela OMG (Object Management Group, Inc), uma organização internacional de fornecedores, desenvolvedores e usuários de software que tem como objetivo promover a teoria e prática da tecnologia orientada a objetos [OMG, 2007]. Os processos passaram pelos mesmos problemas da linguagem de modelagem de objetos, isto é, existiram várias propostas de especificação de processos diferentes, causando dificuldades nas pesquisas e desenvolvimento de produtos de software. SPEM foi criado para resolver este problema.

SPEM é um metamodelo que estende um subconjunto do metamodelo da UML, e está dividido em dois pacotes: o *SPEM_foundation* para definição de tipos e expressões, elementos estruturais, relacionamentos e dependências. E o *SPEM-Extensions* que define os elementos básicos, dependências, estrutura do processo, componentes do processo e ciclo de vida do processo. Os Elementos básicos de SPEM são informações apresentadas ao leitor do processo que contém descrições sobre o modelo utilizado. São eles a descrição externa e os guias do modelo. Os guias dos modelos podem ser diretrizes, técnicas, métricas, *profile UML*, listas de verificação, *tool mentors*, *templates* e *roadmaps* da tecnologia.

A Estrutura do processo define os elementos estruturais principais do qual uma descrição de processo é construída: o produto de trabalho (*WorkProduct*) ou um artefato é qualquer coisa produzida durante a realização das atividades. Um produto de trabalho está associado a um Tipo de produto de trabalho. A definição de trabalho (*WorkDefinition*) é um tipo de operação que descreve o trabalho executado no processo. A atividade (*Activity*) é a principal subclasse da definição de trabalho. Descreve as tarefas, operações e ações que são executadas por um papel do processo (*ProcessRole*). Uma atividade pode consistir de elementos atômicos chamados etapas. Um executor do processo (*ProcessPerformer*) define um executor para um conjunto de definição de trabalho em um processo. O papel (*ProcessRole*) é uma especialização do executor do processo e define responsabilidades sobre produtos de trabalho específicos, e define os papéis que executam e assistem atividades específicas.

Os Componentes do processo definem os elementos de alto nível que compõe um modelo de processo. São eles: pacote, componente de processo, processo e disciplina. Um pacote (*Packcage*) é um recipiente que possui e importa elementos da definição do processo. O pacote pode ser usado para executar a disposição geral de elementos da descrição do processo. Um Componente de processo (*ProcessComponent*) é um fragmento da descrição do processo que é internamente consistente e pode ser reusada com outros Componentes de processo para montar um processo completo. Um processo (*Process*) é um Componente de processo criado para ser um processo completo e único, fim-a-fim. Uma disciplina (*Discipline*) é uma especialização particular do pacote que divide as atividades dentro de um processo de acordo com um "tema comum".

O Ciclo de vida (*Life cicle*) do processo define o comportamento do processo sobre o tempo, e a estrutura do ciclo de vida em termos das fases e das iterações. Uma fase (*Phase*) é uma especialização da definição do trabalho, tal que sua condição prévia define os critérios da entrada da fase e seu objetivo (chamados freqüentemente um "marco" ou "*milestones"*) define

os critérios da saída da fase. Um ciclo de vida do processo é definido como uma seqüência das fases para atingir um objetivo específico. Uma iteração (*iteration*) é uma definição de trabalho composta com um marco menor. A cada definição de trabalho pode ser associada uma pré-condição e uma meta. As pré-condições e as metas são restrições expressas pelo estado dos produtos de trabalho, os quais são os parâmetros da definição de trabalho.

SPEM usa os diagramas básicos da UML para apresentar perspectivas distintas de um modelo de processo do software. São usados os seguintes diagramas: diagrama de pacotes, diagrama de classes, diagrama de atividades, diagrama de casos de uso, diagrama de seqüência e diagrama de estados. O diagrama de classe representa o relacionamento entre o executor do processo e produto de trabalho. Também representa a estrutura, decomposição e dependência de produtos de trabalho. Os diagramas de pacote permitem a representação do processo, componentes de processo, pacotes de processo e disciplinas. Os diagramas de caso do uso mostram o relacionamento entre papéis do processo e as principais definições do trabalho. Os diagramas de atividade permitem apresentar a seqüência das atividades com seus produtos do trabalho da entrada e da saída assim como estados do fluxo do objeto. Raias (*swimlane*) ou partições podem ser usadas para separar as responsabilidades de papéis diferentes do processo. Diagramas de estado e seqüência são também utilizados para ilustrar padrões de comportamento dos elementos do modelo em SPEM. São definidos ícones gráficos (Figura 4-6) específicos para modelagem dos elementos de SPEM nos diagramas da UML.

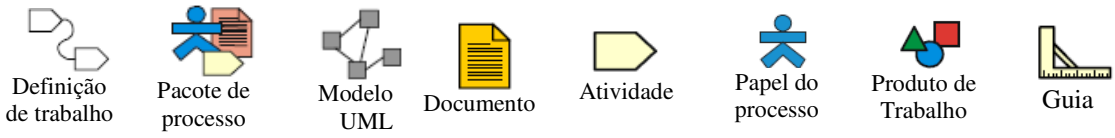


Figura 4-6 – Notação gráfica usada em SPEM

A Figura 4-7 apresenta alguns exemplos dos diagramas usados em SPEM com os estereótipos gráficos. O diagrama de classes representa a relação entre o Ator e o Produto de Trabalho. Significa que o Ator é responsável por este produto de Trabalho. E que o produto de Trabalho é composto de um Diagrama UML, um Documento e um Guia. O diagrama de Pacotes mostra uma visão geral do pacote de processo “Requisitos”. As atividades são realizadas por um papel de processo, que é o “Analista de Sistemas”. As atividades de Requisitos neste pacote são "Captura um vocabulário comum", "Gerencia dependências" e "Desenvolve a Visão". Existem quatro produtos de trabalho para os Requisitos: dois documentos (documento de visão e descrição de caso de uso), um diagrama da UML

(Diagrama de caso de uso) e um guia (Guia de casos de uso). O diagrama de Atividades representa o fluxo de atividades realizadas pelos papéis de processo Analista e Projetista de Interface. O analista inicia realizando a atividade "Definir Requisitos" que gera o produto de Trabalho "Documento de Visão". Em seguida realiza a atividade "Modelar os requisitos" que gera o diagrama de "Casos de uso". Em paralelo a esta atividade, o Projetista de Interface pode realizar a atividade "Esboça a Interface" tendo como entrada o Documento de Visão e como saída o “Documento de Interface Preliminar”. Após a conclusão destas duas últimas atividades, o Projetista de interface pode realizar a atividade "Desenha a interface final", que gera o documento "Interface do sistema".

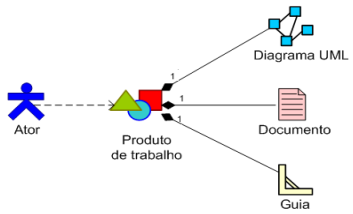


Diagrama de classes

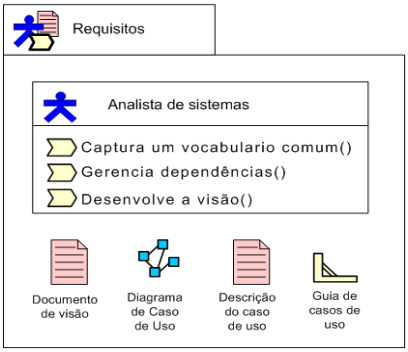


Diagrama de Pacotes

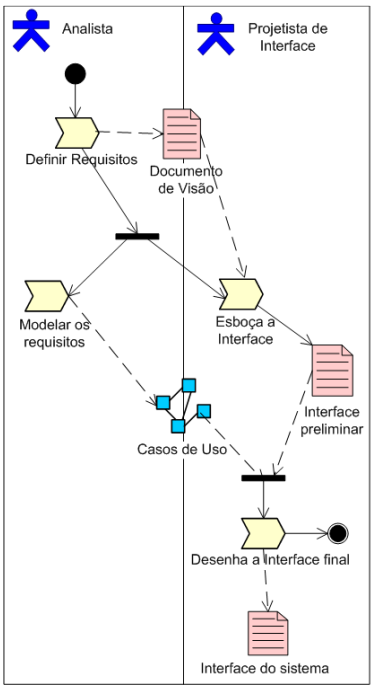


Diagrama de Atividades

Figura 4-7 – Exemplos de diagramas de elementos de processos em SPEM

Os pesquisadores de sistemas multiagentes estão empregando esforços para representar suas metodologias em um formato uniforme, tal que possam promover comparações, composições e reuso. O SPEM [SPEM, 2007] e o OPF [Firesmith & Henderson-Sellers, 2002] são abordagens usadas para padronizar descrições de processos. Nardini et al., (2008) usou SPEM para testar a especificação de uma metodologia para desenvolvimento de sistemas multiagentes. Algumas limitações foram identificadas. Por exemplo, diagramas UML podem se tornar ilegíveis para representação de metodologias complexas (como é o caso de algumas metodologias para SMA), mas isto pode ser resolvido com algumas extensões nos diagramas e elementos do metamodelo. SPEM também foi usado pelo comitê técnico da FIPA para especificar várias metodologias e compor um repositório de fragmentos. Nestes trabalhos

SPEM foi considerado um candidato natural para modelagem de processos de engenharia de software e pode ser aplicado a metodologias AOSE.

4.5 Especificações de Tropos usando SPEM

As duas versões de Tropos, descritas no Capítulo 3, foram modeladas usando a linguagem de especificação SPEM. Tropos'02 foi modelada no trabalho de Wautelet; Kolp & Achbany (2005) e apresenta uma proposta para um processo no modelo de processo em espiral, chamada S-Tropos. Tropos'04 foi modelada pelo comitê técnico de metodologia FIPA [FIPA, 2008].

O S-Tropos formaliza um processo genérico para a metodologia Tropos'02 usando o modelo de desenvolvimento em espiral. O objetivo deste processo é estender a metodologia Tropos, que se apresenta como um modelo em cascata, para incluir as vantagens do modelo de desenvolvimento em espiral, tais como gerenciamento de projeto de software, modelagem organizacional, levantamento de requisitos, implementação, teste e modularidade. O processo inclui atividades principais: requisitos iniciais, requisitos finais, projeto arquitetural, projeto detalhado, desenvolvimento, validação e distribuição. Há também uma atividade de apoio chamada gerenciamento de projetos e riscos. Estas atividades são denominadas *disciplinas* conforme a notação SPEM. O processo é dividido em iterações conforme o modelo do espiral SDLC [Boehm, 1987]. Cada iteração é dividida em quatro fases que são desenvolvidas seqüencialmente: configuração, prototipação, construção e produção. As atividades do processo são coletadas da seqüência proposta pela versão Tropos'02. Os produtos de trabalho são os artefatos gerados pela metodologia (e.g. modelo SD e modelo SR) e os guias são os "métodos" e "técnicas" utilizados pela metodologia Tropos tais como o framework i*, a linguagem AUML, linguagem de programação JACK e os conceitos do modelo BDI. As disciplinas validação, distribuição, gerenciamento de projetos e riscos são inseridas tomando como base as diretrizes do modelo espiral e técnicas da engenharia de software tradicional.

O trabalho para modelar a versão Tropos'04 está incluído na proposta adotada pelo comitê técnico de metodologia FIPA (*Foundation for Intelligent Physical Agents*) para criação de um repositório de métodos que permita aos desenvolvedores reutilizar fragmentos de diferentes metodologias para modelar sistemas multiagentes [FIPA, 2008]. Esta iniciativa gerou a modelagem de várias metodologias orientadas a agentes tais como GAIA, PASSI, Adelfe, entre outras. Tropos'04 fez parte desta proposta. A modelagem se deu traduzindo a metodologia tal como está descrita na versão textual em Bresciani et al. (2004) para a notação

SPEM com o acréscimo de conceitos padronizados usado no modelo de fragmentos criado pelo comitê da FIPA. Não existe a proposta de um ciclo de vida para o processo, apenas a descrição dos elementos estruturais, tais como as disciplinas (que são chamadas de fases), os papéis de processo, as definições de trabalho, as atividades e os produtos de trabalho.

4.6 Considerações Finais

A adoção de um modelo de processo de software ajuda a entender e melhorar a qualidade dos produtos de software desenvolvidos. A comunidade de desenvolvimento de sistemas orientado a agentes, por anos, tem empregado esforços para desenvolver metodologias que descrevem métodos para analisar, projetar e desenvolver sistemas multiagentes. Porém a preocupação em seguir modelos de processos tradicionais ou desenvolver novos modelos específicos para a área ainda está muito recente. Na análise destas metodologias, por Cernuzzi; Cossentino & Zambonelli (2005), foi estabelecida uma relação com modelos de processos tradicionais, observando que os mais utilizados são os modelos em cascata, modelos evolucionários e iterativos.

Algumas propostas de modelos de processo surgiram, tais como o SEP (*Software engineering process*) proposto por [Cossentino & Seidita, 2004], O-MASE [Garcia-Odeja et al., 2007] e [Knublauch, 2002]. Também foi observado que, o comitê técnico de metodologias da FIPA tem especificado metodologias para sistemas multiagentes usando a linguagem de especificação de processos SPEM, o que pode ser um passo importante na direção de produzir ou adotar modelos de processos que orientem os projetos de desenvolvimento. Inclusive, a metodologia Tropos foi especificada usando SPEM em duas propostas. O S-Tropos [Wautelet; Kolp & Achbany, 2005] propõe um modelo de processo em espiral para o Tropos'01 e o comitê técnico da FIPA [FIPA, 2008] propõe uma especificação em SPEM do Tropos'04. Analisando estes trabalhos observa-se que o SPEM tem recebido boa aceitação como linguagem de especificação de processos para as metodologias orientadas a agentes.

Embora o S-Tropos proponha o uso do modelo em espiral para Tropos, este não foi o modelo de processo selecionado nesta dissertação. A característica principal do modelo em espiral é que este é muito baseado no gerenciamento de riscos. Esta particularidade adiciona um subconjunto muito forte de atividades de identificação, análise, acompanhamento e mitigação de riscos, não sendo o foco deste trabalho. A abordagem de processos ágeis com o XP também não é adequada ao Tropos. Métodos ágeis investem fortemente na implementação, reduzindo atividades de análise e projeto do sistema, enquanto que Tropos

apresenta vários métodos de análise e projeto que são fundamentais para o desenvolvimento de SMA.

Por outro lado os métodos usados em Tropos são facilmente adaptados ao desenvolvimento incremental. Portanto, nesta dissertação foi selecionado o modelo de processo iterativo e incremental para orientar o desenvolvimento de um sistema multiagentes. No próximo capítulo será apresentada a principal contribuição desta dissertação que é a descrição do U-Tropos, um processo unificado que adota o modelo de processo iterativo e incremental para o Tropos.

Capítulo 5 – U-Tropos

Nesse capítulo é apresentada a principal contribuição desta dissertação, o processo unificado U-Tropos. O U-Tropos é uma proposta de processo unificado para apoiar o desenvolvimento de software orientado a agentes. É especificado em SPEM e integra diversas técnicas, métodos e diretrizes às atividades da abordagem Tropos.

5.1 Introdução

Tropos abrange os estágios de requisitos, projeto e implementação de sistemas e existem vários grupos realizando pesquisas para evoluir a abordagem e criar métodos para melhorias nas fases de desenvolvimento. Conforme exposto no Capítulo 3, Tropos está descrita em duas versões, que são utilizadas por grupos de trabalhos distintos. Embora várias melhorias tenham sido propostas, nenhum trabalho tem se preocupado em adicionar sistematicamente estas melhorias ao Tropos. Além disto, os métodos e técnicas foram criados dentro de uma visão particular da fase em que estão inseridos e utilizam notações sem nenhuma padronização. Esta dispersão dificulta a utilização de Tropos no desenvolvimento de sistemas multiagentes, pois os desenvolvedores precisam pesquisar e acessar muitas fontes de informação que, embora sejam ricas em detalhes, exigem um grande esforço de aprendizagem e interpretação.

Esta dissertação propõe analisar duas versões de Tropos e integrar as melhores práticas, técnicas e métodos, permitindo assim uma maior integração e melhor entendimento das suas fases. O resultado é apresentado em um processo unificado, iterativo e incremental descrito com a linguagem de especificação de processo SPEM. O processo unificado *U-Tropos* aborda os aspectos dinâmicos e estruturais do ciclo de vida de desenvolvimento. Primeiramente são propostas as fases e iterações do ciclo de vida do desenvolvimento ao longo do tempo. Em seguida são apresentados os aspectos estruturais que organizam o conteúdo do processo em termos de atividades, produtos de trabalho gerados e papéis envolvidos na execução das atividades.

Tropos'02 e Tropos'04 sugerem um conjunto de atividades para confecção dos produtos de trabalho gerados durante a elicitação e análise dos requisitos, projeto arquitetural, projeto detalhado e implementação. Como apresentado no Capítulo 3, a definição dos requisitos não apresentam grandes diferenças nas duas versões com exceção da variante do modelo i* adotado. Nesta dissertação é proposto que o U-Tropos adote um conjunto de atividades comuns entre as duas versões, sugerindo diretrizes para construção dos modelos i*. O projeto arquitetural tem uma mudança significante, comparada àquelas duas versões, pois o U-Tropos acrescenta as atividades sugeridas no framework SIRA [Bastos, 2005] para geração do modelo arquitetural. O projeto detalhado também segue um conjunto de atividades específicas, sendo orientado pelo processo de projeto detalhado proposto em Silva, C.T.L.L.

et al. (2007a). A implementação segue atividades padronizadas de codificação e integração de código.

Durante a análise dos requisitos, o framework i* é usado para representar os modelos de requisitos iniciais e finais. O Tropos'02 propõe a versão original do i* e o Tropos'04 propõe uma variante criada especialmente para uso no Tropos. Como já discutido no Capitulo 3, existem mudanças quanto aos nomes dos diagramas bem como na notação e semântica dos elementos do diagrama. Nesta dissertação, os termos adotados são aqueles usados na variante do i* proposto pela versão Tropos'04 para referenciar os diagramas do i* (diagrama de ator e diagrama de metas), devido a seleção da ferramenta de modelagem, o TAOM4E. Mas o usuário pode optar pela versão do i* original e utilizar os termos originais (diagrama de dependência estratégica e diagrama de raciocínio estratégico).

O processo é apresentado ordenado em três pacotes[8] principais seguindo a proposta de especificação do SPEM: o ciclo de vida do processo, os componentes do processo e a estrutura do processo. O ciclo de vida descreve como o projeto pode ser desenvolvido ao logo do tempo e está dividido em quatro fases (definição, projeto, construção e implantação). Os componentes e a estrutura do processo definem os elementos estruturais que descrevem o processo e é composto de oito disciplinas[8]: requisitos iniciais, requisitos finais, projeto arquitetural, projeto detalhado, implementação, gerenciamento, verificação e suporte (Figura 5-1). Nesta dissertação apenas a especificação das disciplinas que fazem parte da metodologia Tropos são detalhadas (Requisitos iniciais, Requisitos finais, Projeto arquitetural, Projeto detalhado, Implementação). As demais disciplinas serão desenvolvidas em trabalhos futuros.

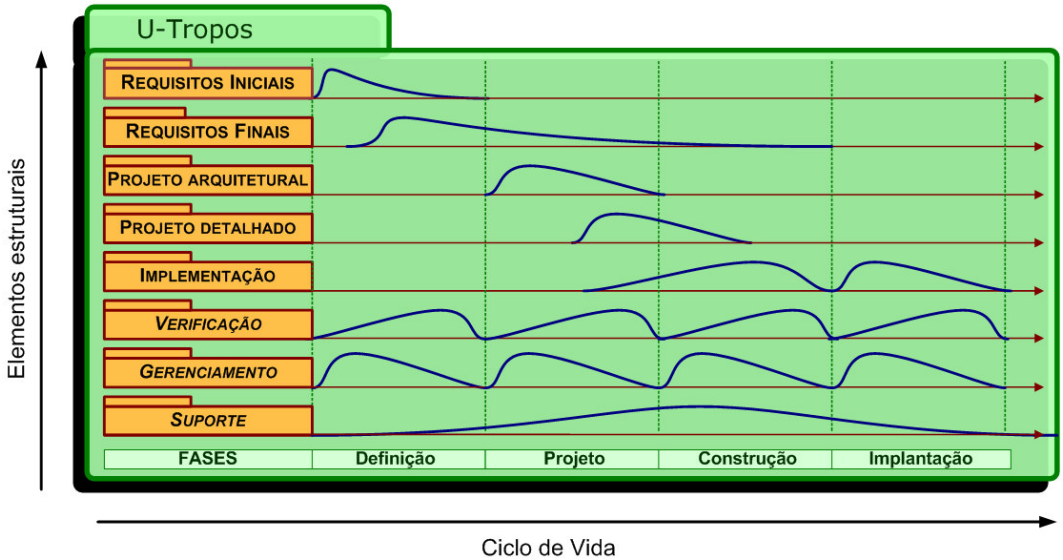


Figura 5-1– Elementos do U-Tropos

[8] Os termos pacotes e disciplinas fazem parte da especificação SPEM e estão descritos no Capítulo3

5.2 Ciclo de Vida do Processo

Um modelo de ciclo de vida é definido como um framework contendo os processos, atividades e tarefas envolvidas no desenvolvimento de um produto de software, englobando a existência do sistema desde a definição de seus requisitos até a finalização de seu uso. [ISO, 1994]. Atendendo esta definição, são acrescentados ao processo U-Tropos os elementos que definem o ciclo de vida de um processo, que são as fases do ciclo de vida e as iterações. Estes elementos fazem parte da especificação de processo em SPEM.

5.2.1 Fases do Ciclo de Vida

Um ciclo de vida do processo é definido como uma seqüência das fases para atingir um objetivo específico. Em Tropos, o termo *Fase* é usada para descrever as etapas da desenvolvimento dos produtos (requisitos iniciais e finais, projeto arquitetural, projeto detalhado e implementação), sugerindo a idéia de um modelo de processo em cascata [Royce, 1970]. Na especificação do U-Tropos, estas etapas são descritas como disciplinas, pois abrangem o conteúdo das atividades a serem executadas independentes do tempo (Seção 4.4.1). As fases definidas para o processo proposto descrevem o trabalho executado seqüencialmente através da propagação das datas pelo tempo decorrido. Estão delimitadas por uma condição prévia, que define os critérios de entrada, e um objetivo, que define os critérios de saída da fase (Figura 5-2). As fases propostas para o U-Tropos são definição, projeto, construção e implantação.

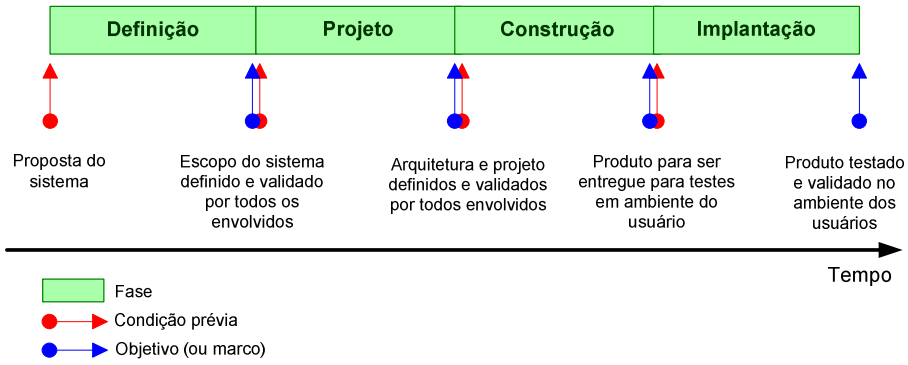


Figura 5-2– Fases do Ciclo de Vida do Processo

A fase de *Definição* é a fase inicial do ciclo de vida, onde o sistema é proposto, os objetivos e funcionalidades do sistema são elicitados, analisados, documentados e validados com todas as partes interessadas no sistema. A condição prévia para início desta fase é a proposta de desenvolvimento de um sistema. O objetivo desta fase é a definição do escopo do

sistema e sua validação por todos os envolvidos. A disciplina Requisitos iniciais deve ser iniciada e concluída nesta fase (Figura 5-1). O objetivo desta fase é definir um ponto de partida inicial para começar a definição do sistema. A definição do escopo não precisa incluir todos os requisitos do sistema, uma vez que requisitos são mutáveis e podem ser evoluídos durante o ciclo de vida do sistema. O escopo deve delimitar a abrangência do sistema dentro da organização e proporcionar um acordo entre os envolvidos sobre quais necessidades serão atendidas pelo uso do sistema.

A fase de *Projeto* define a arquitetura do sistema e detalha o projeto dos agentes que serão construídos. Estas definições partem de uma análise mais detalhada dos requisitos definidos na fase anterior para gerar uma solução de software que atenda a estes requisitos. Durante esta análise é possível a descoberta de novos requisitos ou mudanças em requisitos já definidos. A condição prévia para início desta fase é ter o escopo do software definido e validado por todos os envolvidos. O objetivo da fase é definir e testar a arquitetura e o projeto do sistema para iniciar a implementação do sistema.

Na fase de *Construção*, o desenvolvimento do sistema é concluído através da codificação baseada nos requisitos, arquitetura e projeto detalhado dos agentes definidos nas fases anteriores. Durante esta fase, ainda é possível a descoberta de novos requisitos ou mudanças em requisitos já definidos, embora o impacto de mudanças seja maior. A condição prévia para início desta fase é ter a arquitetura e o projeto definidos e validados por todos envolvidos. O objetivo da fase é gerar um produto (completo ou parcial) que possa ser entregue para operação em ambiente dos usuários.

Na fase de *Implantação*, o software deve estar disponível para seus usuários finais. O usuário é treinado no uso do software (completo ou parcial) e inicia o uso para identificar a necessidade de configurações, instalação, ajustes de funcionalidades e/ou de usabilidade. O retorno do usuário é indispensável nesta fase para permitir os ajustes necessários no software em uso. A condição prévia para início desta fase é ter disponível o produto (completo ou parcial) para o uso no ambiente dos usuários. O objetivo da fase é gerar um produto testado e validado no ambiente dos usuários.

5.2.2 Iterações

Uma iteração também descreve o trabalho executado seqüencialmente durante o tempo, mas composto com marcos[9] menores. Durante o ciclo de vida do processo, podem ocorrer diversas iterações e uma iteração pode ser composta de atividades de diversas disciplinas. Portanto, em uma iteração podem ser desenvolvidas atividades de várias disciplinas do processo. Estas atividades podem ser para a geração dos produtos ou para manutenção nos produtos criados conforme a evolução do desenvolvimento do sistema.

Uma fase do ciclo de vida pode passar por diversas iterações até a conclusão do desenvolvimento do software. Podem ser realizadas diversas iterações para cada fase. O projeto pode iniciar com uma iteração de *definição* do escopo do sistema onde são definidos os requisitos da organização e os requisitos gerais do sistema. A segunda iteração pode ser para projetar a arquitetura global do sistema. Caso seja necessário, a arquitetura pode ser testada incluindo a implementação de um protótipo executável. As próximas iterações podem ser iterações de construção, onde os requisitos gerais definidos na primeira iteração de definição são agrupados segundo a sua prioridade e cada subgrupo é analisado, projetado e implementado gerando versões executáveis do sistema. Iterações de entrega e operação do sistema podem ser criadas para acompanhamento do uso no ambiente de operação. Estas iterações podem gerar novos requisitos ou mudanças que serão incluídos nas iterações de construção. (Figura 5-3).

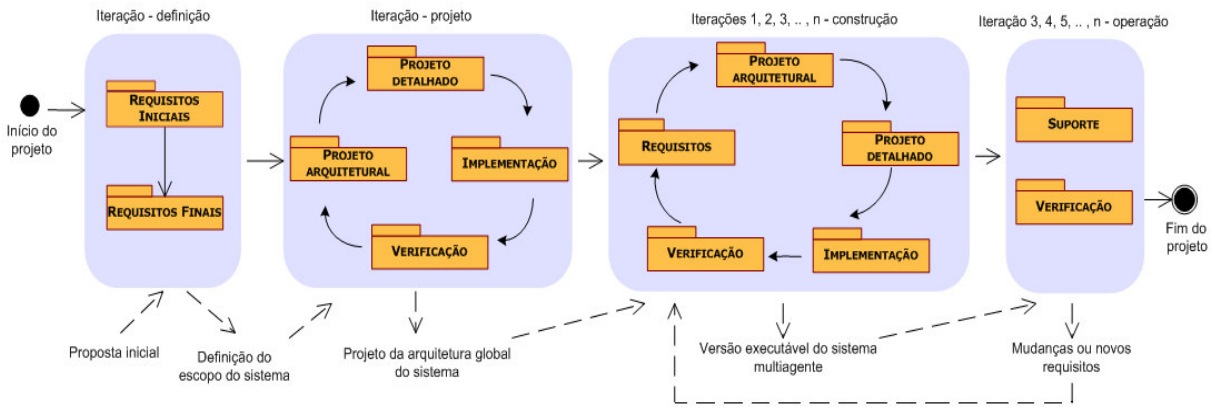


Figura 5-3– Iterações do Ciclo de Vida do Processo

[9] O termo marco faz parte da especificação SPEM e está descrito no Capítulo 3

5.3 Componentes do Processo

Os componentes do processo definem os elementos de alto nível que compõe um modelo de processo. Estes elementos foram descritos no capítulo 4. São eles: pacote, componente de processo, processo e disciplina. O U-Tropos é um componente de processo único que inclui o pacote de processo (que é dirigido pelo ciclo de vida descrito na seção 4.2) e o pacote de Disciplinas. Em Tropos são apresentados cinco grupos de atividades, denominadas "fases", que são os requisitos iniciais, requisitos finais, projeto arquitetural, projeto detalhado e implementação. Usando a terminologia de SPEM, estas "fases" são as disciplinas do processo, pois descrevem as atividades agrupadas por temas comuns. Além disso, estes grupos de atividades não estão associados ao conceito de evolução no tempo, o que definiria uma fase no SPEM.

O modelo de desenvolvimento de sistemas pode ser dividido em quatro categorias [CMMI, 2006]: gerenciamento de processo, gerenciamento de projeto, engenharia e suporte. As disciplinas da metodologia Tropos fazem parte da categoria *engenharia*. O processo proposto nesta dissertação sugere oito disciplinas para atender estas quatro categorias: requisitos iniciais, requisitos finais, projeto arquitetural, projeto detalhado, implementação, gerenciamento, verificação e suporte (Figura 5-1). As cinco primeiras disciplinas fazem parte do conjunto de atividades proposto por Tropos. As três últimas são sugeridas pelo U-Tropos, pois são essenciais para atender as necessidades básicas de um projeto de desenvolvimento de software.

A disciplina *Requisitos iniciais* é o entendimento do problema a partir da análise e estudo da configuração organizacional onde este está inserido. Esta disciplina envolve as atividades de análise das intenções dos atores sociais e seus relacionamentos com outros atores. As intenções são modeladas como metas, que a partir de uma análise orientada a metas geram os requisitos funcionais e não funcionais do sistema. O esforço maior empregado nesta disciplina ocorre na fase de definição do sistema, uma vez que é nesta fase que identificamos e documentamos quem são os atores envolvidos e suas necessidades para geração do escopo do sistema desenvolvido.

A disciplina *Requisitos finais* descreve o sistema dentro de seu ambiente operacional, a partir de requisitos funcionais relevantes e requisitos de qualidade (ou requisitos não-funcionais). O sistema é representado como um ou mais atores que participam de um modelo de dependência estratégica juntamente com outros atores do ambiente operacional do sistema. O esforço empregado nesta disciplina se estende desde a fase de definição onde são

identificados os primeiros requisitos do sistema até a fase de implementação, quando ainda podem ocorrer refinamentos nos requisitos estabelecidos ou descoberta de novos requisitos ou alterações decorrentes de solicitação dos usuários.

A disciplina *Projeto arquitetural* define a arquitetura global do sistema em termos de subsistemas e suas dependências. A arquitetura do sistema constitui um modelo de estrutura do sistema relativamente pequeno e intelectualmente gerenciável, que descreve como os componentes trabalham e como estão relacionados. O esforço empregado nesta disciplina inicia e finaliza na fase de projeto. É nesta fase que a arquitetura do sistema é definida e validada.

A disciplina *Projeto detalhado* especifica o comportamento de cada componente arquitetural, tal como a linguagem de comunicação entre agentes, os mecanismos de transporte de mensagens, o ambiente onde o agente está inserido, as ações tomadas pelos agentes para cumprir suas metas, o raciocínio interno de cada agente e sua estrutura arquitetural. O esforço empregado nesta disciplina inicia na fase de projeto e pode ser refinada na fase de construção do sistema, quando tem início a implementação dos agentes de software.

A disciplina *Implementação* contém as atividades de codificação do software. Estas são baseadas nas especificações geradas nas disciplinas anteriores e usam um suporte ferramental adequado a codificação. A implementação pode iniciar na fase de projeto com a construção do protótipo para o sistema e/ou protótipos para testes de arquiteturas alternativas. A fase onde se requer mais esforço de implementação é a construção do sistema. Após a entrega do sistema ao cliente, durante os testes do usuário no seu ambiente operacional, freqüentemente são necessários ajustes nas funcionalidades do sistema, ocorrendo, portanto um esforço considerável de implementação nesta fase.

A disciplina *Verificação* é responsável pela análise estática dos artefatos do sistema para descobrir problemas e pela observação do comportamento operacional do produto quando em operação no ambiente de usuário. As atividades desta disciplina, que incluem inspeções e testes, ocorrem durante todo ciclo de vida do sistema, iniciando com baixo grau de esforço em cada fase e aumentando gradualmente à medida que os produtos de trabalho são gerados.

A disciplina *Gerenciamento* envolve as atividades de planejamento, monitoramento e controle de elementos do projeto tais como recursos, tempo, custo, riscos, comunicação, qualidade e escopo do projeto. O esforço de gerenciamento é empregado durante todo o ciclo de vida do sistema. No início da fase de definição, o gerenciamento é iniciado com as

atividades de planejamento e aumenta gradualmente de acordo com as necessidades de monitoramento e controle do projeto de desenvolvimento do sistema. Ao final do ciclo de vida, o esforço diminui em decorrência da diminuição das atividades do desenvolvimento.

A disciplina *Suporte* inclui atividades que apóiam o desenvolvimento, entrega e manutenção do produto. Geralmente são atividades que atendem a organização como um todo e apóiam o projeto de desenvolvimento durante seu ciclo de vida. São exemplos de atividades: gerenciamento de configuração, gerenciamento de qualidade, análise e geração de métricas, instalação do produto no ambiente do cliente, treinamento, etc. As atividades da disciplina suporte ocorrem durante todo ciclo de vida do sistema. Assim como no gerenciamento, inicia na fase de definição e aumenta de acordo com as necessidades do projeto de desenvolvimento do sistema.

A Figura 5-1 é usada para ilustrar a relação das disciplinas e o seu grau de esforço em cada fase. As curvas de esforço não são geradas a partir de fórmulas matemáticas precisas, mas são usadas apenas para expressar de forma geral o esforço empregado ao longo de cada fase do processo. Esta mesma representação é usada em outros processos de desenvolvimentos tal como RUP - o Rational Unified Process [RUP, 2003].

5.4 Estrutura do processo

A estrutura do processo define os elementos estruturais principais com os quais uma descrição de processo é construída. Conforme vimos no capítulo 3, fazem parte da estrutura de um processo os seguintes elementos: as definições do trabalho, as atividades, os produtos de trabalho e os papéis do processo. Os diagramas básicos de UML são usados para apresentar perspectivas de um modelo de processo do software.

O U-Tropos usa os diagramas de pacotes, diagramas de atividades e diagramas de classes para apresentar os componentes da estrutura dos processos e o relacionamento entre eles. As definições de trabalho são refinadas em atividades que são executadas pelos papéis do processo. Para melhor ilustrar os modelos do U-Tropos, os diagramas de atividades usam partições (*swimlane*) para agrupar as responsabilidades dos papéis de processo. Também são usados quadros sombreados para agrupar as atividades por definições de trabalho. Cada definição de trabalho recebe um rótulo (por exemplo, na disciplina requisitos iniciais: RI1, RI2 e RI3) que identifica o quadro sombreado no diagrama de atividades. Será usada a notação gráfica do SPEM para representar os elementos do processo, incluindo uma notação específica para os modelos i* (Figura 5-4).

Figura 5-4 – Notação gráfica de SPEM

As próximas subseções apresentam a modelagem das cinco disciplinas da metodologia Tropos para o U-Tropos: requisitos iniciais, requisitos finais, projeto arquitetural, projeto detalhado e implementação. A descrição das outras disciplinas (Verificação, Gerenciamento e Suporte) não faz parte do escopo desta dissertação e será abordada em trabalhos futuros.

5.4.1 Disciplina Requisitos Iniciais

A análise dos requisitos iniciais concentra-se nas intenções das *partes interessadas* (do inglês Stakeholders). Estas intenções são metas a serem atingidas que, através de uma análise orientada a metas, produzem os requisitos do sistema que será construído. O framework i* é usado para modelar os conceitos analisados na disciplina de requisitos iniciais. As partes interessadas são modeladas como atores que dependem de outros para satisfazer suas metas, executar suas tarefas e fornecer os recursos. São gerados dois produtos de trabalho: o Modelo de dependência dos atores é usado para descrever a rede de relacionamentos entre os atores e o Modelo de raciocínio dos atores é usado para descrever e suportar o raciocínio que cada ator segue para se relacionar com outros atores.

A disciplina requisitos iniciais (Figura 5-5) envolve três papéis do processo (Engenheiro de requisitos, Analista de domínio e Partes interessadas), cinco produtos de trabalho (Modelo de dependência dos atores, Modelo de raciocínio dos atores, Modelo de requisitos iniciais e

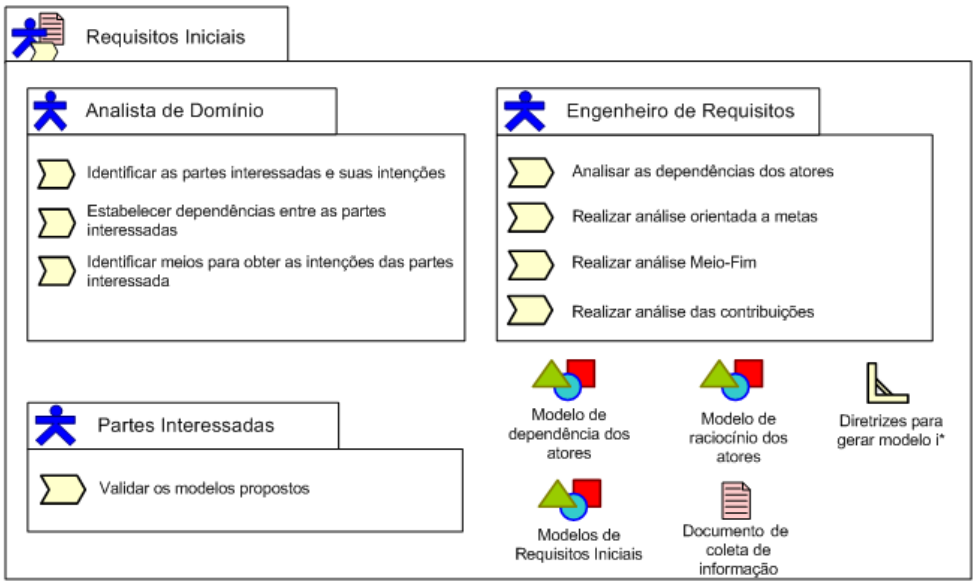


Figura 5-5 – Pacote da Disciplina Requisitos Iniciais

Documento de coleta de informação) e um guia (Diretrizes para gerar modelo i*). Cada Papel do Processo está responsável por um subconjunto de atividades.

O fluxo do trabalho nesta disciplina possui três definições de trabalho (Figura 5-6). A primeira definição de trabalho é a *RI1-Elicitação dos requisitos iniciais*, que compreende as atividades de identificação das partes interessadas e os seus relacionamentos no seu ambiente de trabalho. A segunda definição do trabalho é a *RI2-Análise e especificação dos requisitos iniciais* que compreende a modelagem dos requisitos do domínio. A terceira definição do trabalho é a *RI3 - Validação dos modelos propostos* por todos os envolvidos no processo. Os principais produtos de trabalho gerados nestas três definições de trabalho são o *Modelo de requisitos iniciais,* o *Modelo de dependência dos atores* e *Modelo de raciocínio dos atores.* O *Documento de coleta de informações* faz parte do *Modelo de Requisitos Iniciais*. O Guia *Diretrizes para gerar modelos i** é usado para apoiar as atividades de análise.

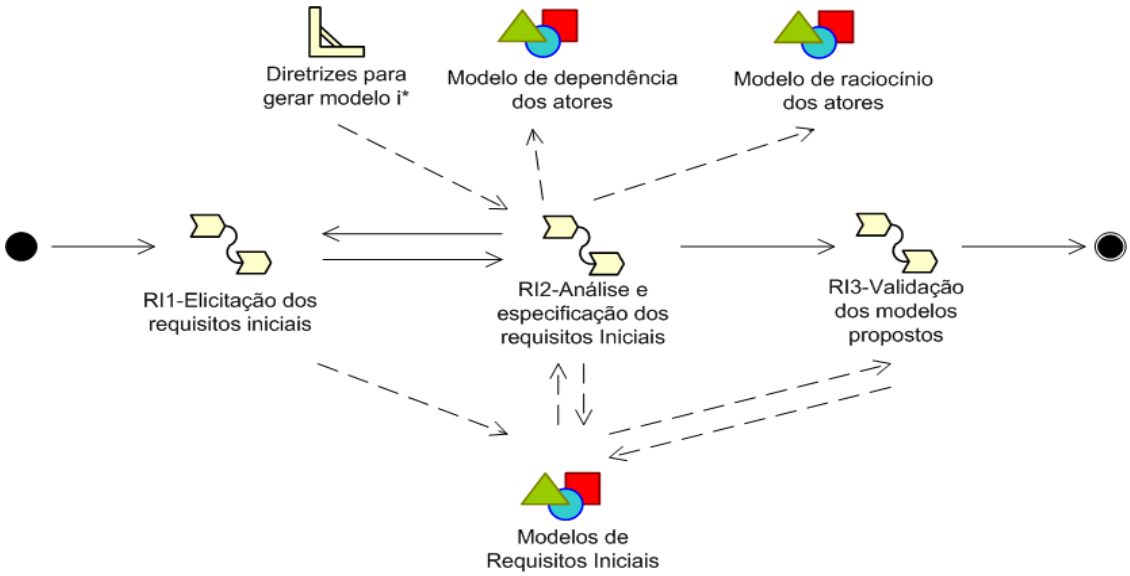


Figura 5-6– Definição do trabalho dos Requisitos Iniciais

O analista de domínio é o responsável pelas atividades de elicitação dos requisitos iniciais (RI1) (Figura 5-7). Ele faz isto através da execução das atividades: *1 - Identificar as partes interessadas e suas intenções,* que envolve a análise do ambiente para identificar quais as intenções ou necessidades de cada parte interessada no desenvolvimento do sistema dentro do seu ambiente organizacional; *2 - Identificar relacionamentos entre as partes interessadas*, que envolve a identificação das relações que existem entre as partes interessadas constituintes da organização; e *4 - Identificar as atividades detalhadas das partes interessadas, que* envolve o detalhamento das ações realizadas (e recursos utilizados) pelas partes interessadas

para cumprir com suas atividades e atingir suas metas. As informações identificadas são documentadas através do produto de trabalho *Documento de coleta de informações*.

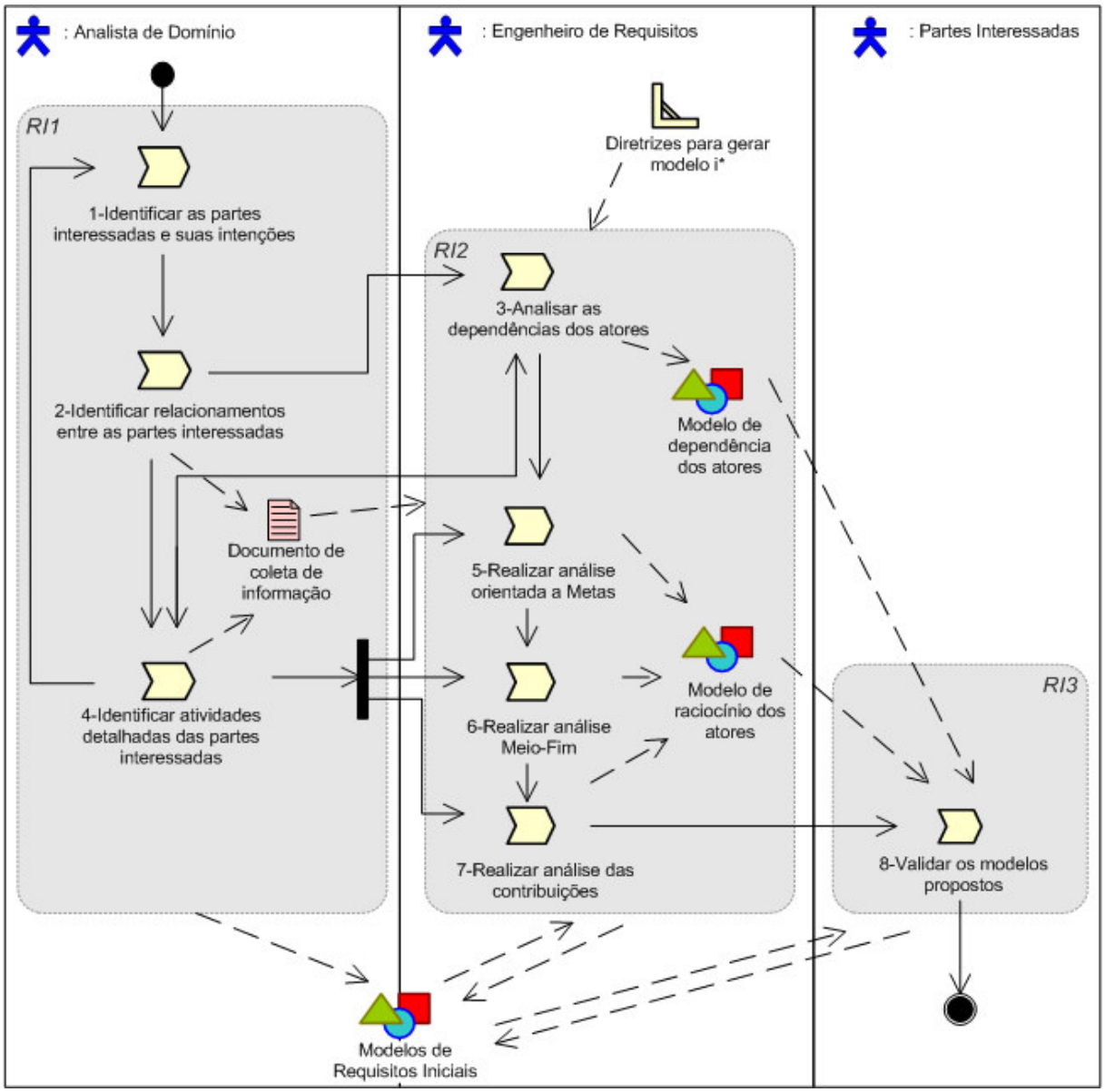


Figura 5-7 – Atividades da disciplina Requisitos Iniciais

O engenheiro de requisitos analisa as informações coletadas pelo analista de domínio e modela os requisitos iniciais realizando as atividades: *3 - Analisar as dependências estratégicas,* que envolve a análise individual das intenções/metas de cada parte interessada; *5 - Realizar análise orientada a metas,* que envolve o início da análise do raciocínio de cada parte interessada (modelada como um ator) para obtenção de suas metas; *6 - Realizar análise meio-fim, que* identifica e refina as tarefas que devem ser executadas para obtenção das metas; e *7 - Realizar a análise das contribuições,* que envolve a análise das tarefas identificadas na atividade anterior em relação à satisfação das metas. Estas quatro atividades geram os

produtos de trabalho *Modelo de dependência dos atores* e *Modelo de raciocínio dos atores* que contém diagramas do framework i*. Para auxiliar a modelagem são usadas como guias as *Diretrizes para gerar o modelo i**. Os modelos gerados são validados por todas as partes interessadas (que podem ser os usuários e gestores humanos da organização interessada no sistema, analistas de domínio e engenheiros de requisitos) durante a atividade *8 – Validar os modelos propostos*. Nesta atividade é gerada a versão validada dos requisitos iniciais do sistema que será usada como entrada para as outras disciplinas do sistema. A especificação completa das atividades dos Requisitos iniciais está descrita no Apêndice A.

Os principais produtos de trabalho gerados são o *Modelo de dependência de atores*, o *Modelo de raciocínio de atores* e o *Documento de coleta de informações* (Figura 5-8). Estes produtos de trabalho compõem o *Modelo de requisitos iniciais*. O *modelo de dependência de atores* é uma generalização para o diagrama i* que modela os atores e suas dependências. Dependendo da versão do i* selecionada, este diagrama é especializado com as características do *diagrama de dependência estratégica* do i* original ou do *diagrama de atores* do metamodelo do Tropos'04. Da mesma forma, o *modelo de raciocínio de atores* é uma generalização dos diagramas que modelam o raciocínio estratégico do ator. Estes podem ser o *diagrama de raciocínio estratégico* no i* original ou o *diagrama de metas* do metamodelo do Tropos'04. O *Modelo de requisitos* é o produto de trabalho final onde são descritas todas as informações relevantes elicitadas pelos analistas de domínio e engenheiros de requisitos para a documentação dos requisitos iniciais do sistema. O *modelo de dependência dos atores* e o *modelo de raciocínio dos atores* também são anexados ao documento de requisitos compondo um único artefato.

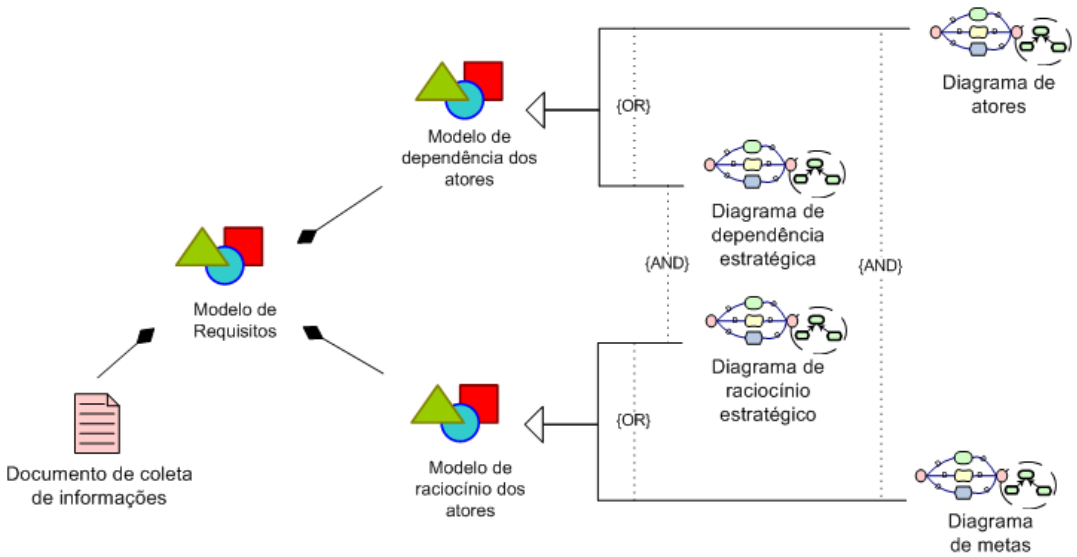


Figura 5-8 – Produtos de trabalho da disciplina Requisitos Iniciais

O documento de coleta de informações é formatado de acordo com o método adotado como guia (por exemplo, o PRIM registra as informações em scripts chamados Detailed Interaction Script (DIS); o SDSituation utiliza cenários; o EQER usa dados de ferramentas qualitativas e diagramas de atividades) . Os guias ou diretrizes para criação dos modelos são usados a partir da necessidade de modelagem do sistema que será desenvolvido:

- Se os modelos forem construídos em paralelo com a análise realizada, pode-se usar como guia o RISD [Grau et al., 2005], que é uma metodologia para construir modelos de dependência estratégica.
- O PRiM [Grau; Franch & Maiden, 2005] e o SDSituation [Oliveira; Padua & Cysneiros, 2006] utilizam cenários para registrar as informações coletadas e diretrizes para converter esse cenários em diagramas i*.
- O Estudo qualitativo para elicitação de requisitos [Cruz Neto, 2008] utiliza um processo completo para elicitação de requisitos através de etnografia e gera um modelo de atividades. Também apresenta um conjunto de diretrizes para gerar diagramas i* a partir do diagrama de atividades.
- Se os analistas optarem pela versão do i* original [Yu, 1997], o i* Guide [Grau, G. et al., 2008] apresenta sugestões de como utilizar corretamente o modelo i*.

5.4.2 Disciplina Requisitos Finais

A análise dos requisitos finais é a continuação da análise de requisitos iniciais com o acréscimo do sistema computacional que será desenvolvido. O sistema é descrito dentro de seu ambiente operacional, a partir de requisitos funcionais relevantes e requisitos de qualidade (ou não funcionais). O framework i* também é utilizado para modelar os requisitos funcionais e não funcionais do sistema.

O sistema é representado como um ou mais atores que participam de um modelo de dependência estratégica juntamente com outros atores do ambiente operacional do sistema. À medida que a análise prossegue, o sistema recebe responsabilidades adicionais e atende dependências de vários outros atores. Depois disto o sistema pode ser decomposto em vários sub-atores que assumem algumas destas responsabilidades. O modelo de dependências dos atores é usado nesta fase para representação do(s) ator (es) sistema(s) e seu relacionamento com os outros atores que representam as partes interessadas no sistema. O modelo de

raciocínio dos atores representa a modelagem dos requisitos funcionais e não funcionais do sistema, que são representados como metas, planos, recursos e metas-soft.

A disciplina requisitos finais envolve dois Papéis do processo (Engenheiro de requisitos, Partes Interessadas), quatro Produtos de trabalho (Modelo de dependência de atores, Modelo de raciocínio de atores, Documento de Coleta de informações e Modelo de requisitos finais) e um guia (Diretrizes para gerar modelo i*). Cada Papel do processo está responsável por um subconjunto de atividades (Figura 5-9).

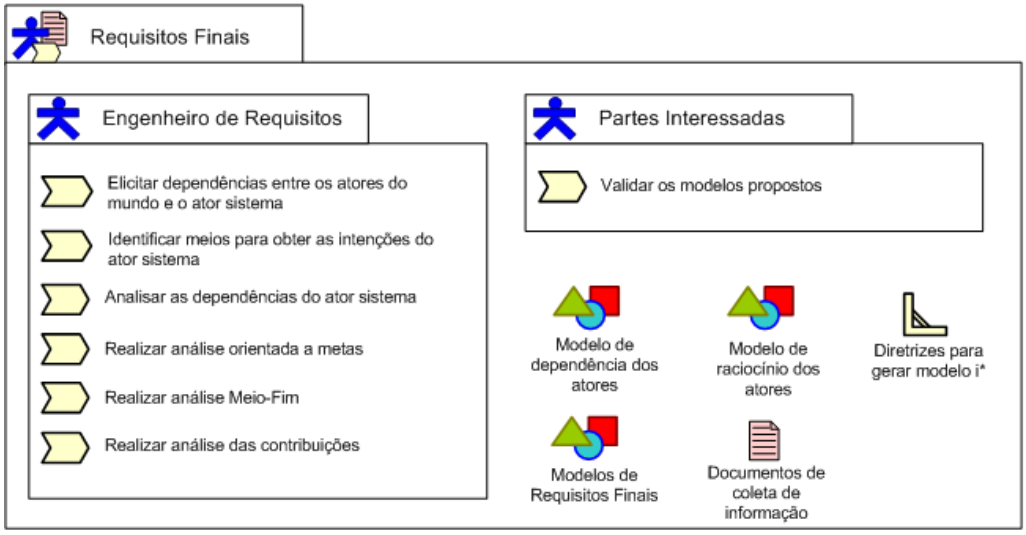


Figura 5-9 – Pacote da Disciplina Requisitos Finais

O fluxo de trabalho nesta disciplina possui três definições de trabalhos (Figura 5-10). A primeira é *RF1-Levantamento dos requisitos do sistema*, que compreende as atividades de elicitação dos requisitos do sistema com os seus usuários. A segunda é *RF2-Análise dos requisitos do sistema* que compreende a análise e modelagem dos requisitos funcionais e não funcionais do ator-sistema para obter as metas atribuídas a ele. E a terceira é *RF3-Validação dos modelos propostos* por todos os envolvidos no processo. *O Modelo de requisitos iniciais* é usado como fonte de informação para elicitação e análise dos requisitos finais. Os principais produtos de trabalho gerados são o *Modelo de requisitos finais, o Modelo de dependência dos atores e Modelo de raciocínio dos atores. O Documento de coleta de informações* é usado para registro das informações coletadas com os usuários / partes interessadas e pode mudar conforme técnica de coleta aplicada. Estes produtos de trabalho são uma evolução dos modelos gerados na disciplina requisitos iniciais. Os produtos de trabalho *Modelo de raciocínio de atores* e *Modelo de dependência de atores* modelam os requisitos do sistema computacional através de diagramas i*.

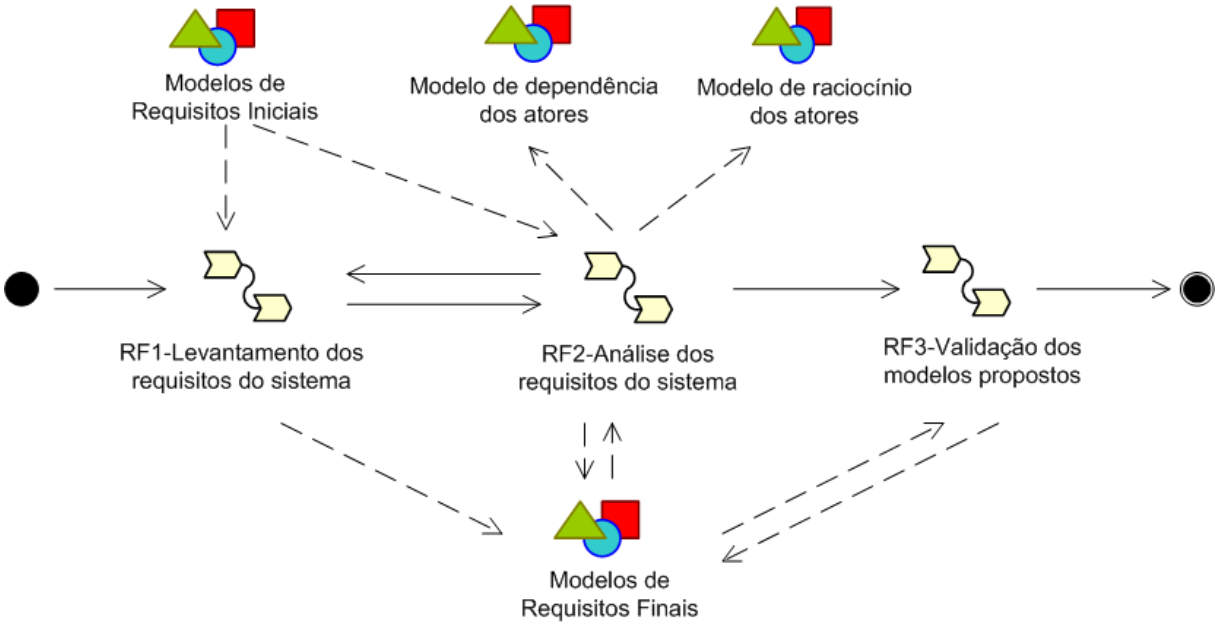


Figura 5-10 – Definição do trabalho dos requisitos finais

A identificação e detalhamento das atividades da disciplina Requisitos finais seguem um raciocínio semelhante ao da análise de requisitos iniciais, com a diferença que a análise é elaborada sobre os requisitos do sistema (Figura 5-11). O engenheiro de requisitos é o responsável pelas atividades de levantamento dos requisitos de sistema. Ele faz isto através da execução das atividades: *1 - Elicitar dependências dos atores do mundo com o ator sistema,* que envolve a identificação do(s) sistema(s) que será(ão) desenvolvido para atender as necessidades dos usuários e das partes interessadas no sistema (ou atores do mundo) e *3 - Identificar meios para obter as intenções do ator-sistema,* que envolve o levantamento detalhado com os usuários, de todas as informações para o sistema com o objetivo de descobrir como as metas desejadas para o sistema podem ser atingidas através das tarefas executadas e dos recursos utilizados. Estas atividades envolvem a participação dos usuários e partes interessadas que informam as funcionalidades requeridas para o sistema. As informações contidas no *Modelo de requisitos iniciais* são base para identificar os atores humanos e suas relações dentro da organização. As informações identificadas são registradas no *Documento de coleta de informação*.

O engenheiro de requisitos analisa as informações coletadas com os usuários e modela os requisitos finais durante a execução das atividades: *2- Analisar as dependências estratégicas do ator sistema,* que envolve a análise das metas das partes interessadas, com a finalidade de identificar como o sistema que será desenvolvido poderá obter estas metas; *4 - Realizar análise orientada a metas,* que envolve a análise detalhada das metas desejadas para

o sistema e a sua modelagem utilizando o framework i*, com o objetivo de identificar os requisitos funcionais e não-funcionais do sistema; *5 – Realizar Analise Meio-Fim,* que envolve a definição das tarefas e recursos para obtenção de cada meta através da análise meio-fim e de decomposição de tarefas (do framework i*); e *6 – Realizar análise das contribuições*, que envolve a análise de cada tarefa de acordo com as suas contribuições para os atributos de qualidade do sistema, ou seja, as *metas-soft*. A especificação completa das atividades dos Requisitos finais está descrita no apêndice A.

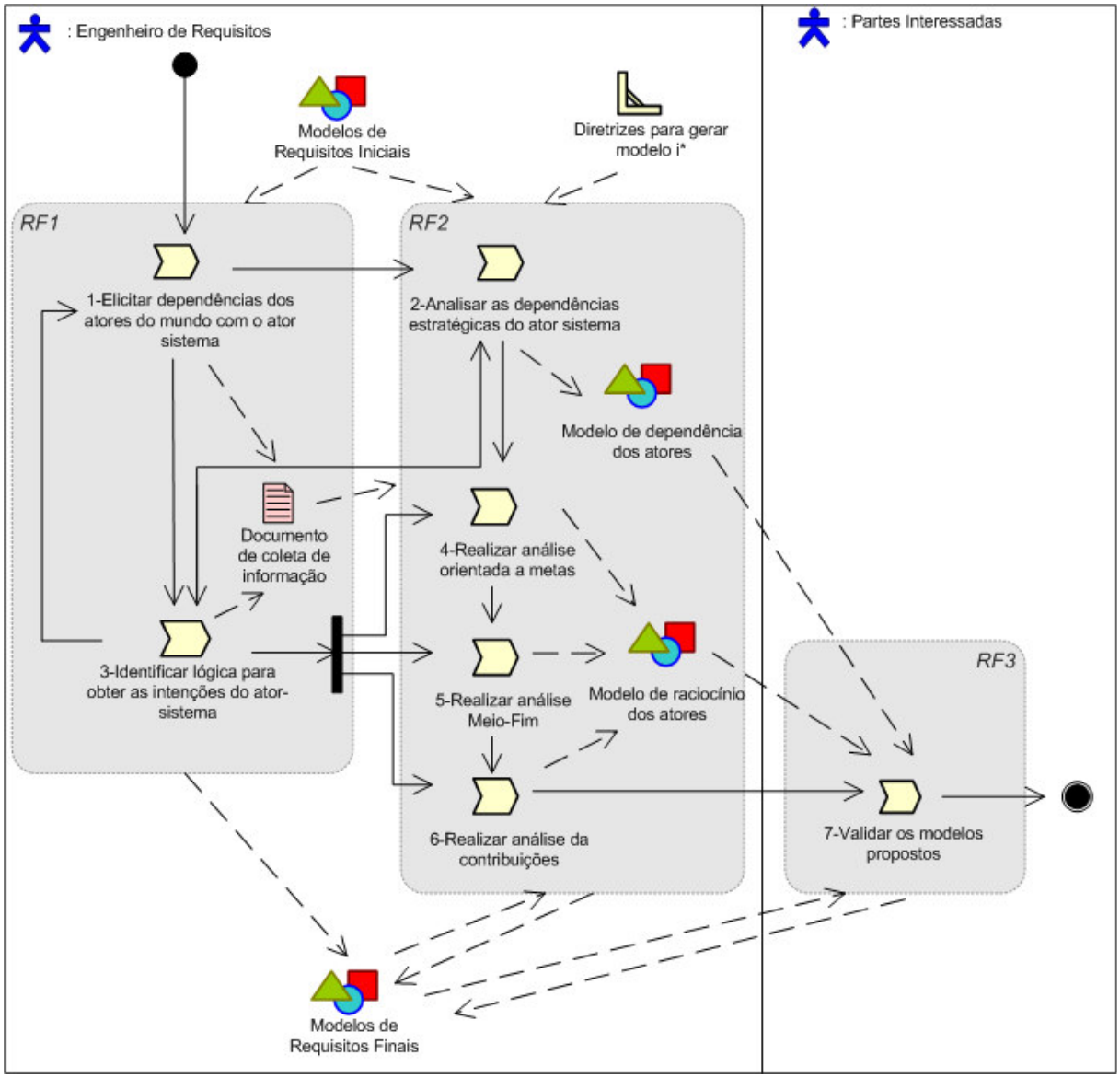


Figura 5-11 – Atividades da disciplina Requisitos Finais

Estas atividades geram os produtos de trabalho *Modelo de dependência dos atores* e *Modelo de raciocínio dos atores* para o ator sistema. Para auxiliar a modelagem são usadas como guias as *Diretrizes para gerar o modelo i**. Os modelos gerados são validados por todas

as partes interessadas (que são os usuários e gestores humanos da organização interessada no sistema, analistas de domínio e engenheiros de requisitos) através da atividade *7 – Validar os modelos propostos* e é gerada uma versão dos requisitos finais do sistema que será usada como entrada para as outras disciplinas do sistema.

O engenheiro de requisitos analisa e modela as informações levantadas na disciplina Requisitos iniciais para identificar os elementos relevantes para os requisitos finais do sistema. Aplicando técnicas de Engenharia de Requisitos, tais como entrevistas, questionários, etnografia, entre outras, faz a elicitação dos requisitos do sistema com os usuários. Após isto modela os requisitos do sistema usando o *Modelo de dependência dos atores* e o *Modelo de raciocínio dos atores*. As partes interessadas podem ser tanto usuários e membros da organização como especialistas em sistemas multiagentes que provêm informações para identificação dos requisitos destes sistemas.

O *Modelo de dependência de atores* é a complementação do *Modelo de dependência de atores* iniciado na disciplina Requisitos iniciais, acrescentando-se o ator sistema e as suas dependências. O *Modelo de raciocínio de atores* é a complementação do *modelo de raciocínio de atores* iniciado na disciplina Requisitos Iniciais acrescentando-se a análise das metas na perspectiva do ator sistema. Os elementos do modelo i* são os mesmos para as duas disciplinas de requisitos iniciais e finais, mudando respectivamente os significados aplicados aos conceitos de modelagem organizacional e modelagem funcional. A Tabela 5-1 apresenta um resumo destes conceitos em cada disciplina.

Tabela 5-1– Elementos dos modelos i* nas disciplinas de requisitos

	Requisitos Iniciais	***Requisitos Finais***
Elementos i*	**Conceitos na modelagem organizacional**	**Conceitos na modelagem funcional**
Ator	Partes interessadas no ambiente organizacional onde será inserido o sistema. Exemplos: Cargos ou papéis assumidos pelas pessoas; Organizações; Divisões das organizações; Sistemas legados da empresa, etc.	É o sistema que será desenvolvido. Ex: Sistema de comércio eletrônico, sistema de notícias eletrônicas, sistema autônomo de administração de SGBD, etc.
Metas	Objetivos, intenções ou estados que atores desejam atingir para cumprir com o seu papel no contexto onde está inserido.	Requisitos funcionais de usuário descrito como objetivos, intenções ou estados que o sistema deve atingir.
Metas-soft	Objetivos de qualidade que atores desejam atingir para satisfazer suas metas dentro do contexto do ambiente onde está inserido.	Requisitos não funcionais do sistema, descritos como os atributos de qualidade que o sistema deve atender.
Tarefas	Forma de executar uma atividade para obtenção de uma meta dentro do contexto onde está inserido.	Requisitos funcionais do sistema. São as funcionalidades que serão desenvolvidas para atender aos requisitos funcionais do usuário.
Recursos	Informações trocadas no contexto do ambiente onde estão inseridas para obter as metas propostas.	Dados utilizados pelo sistema para operacionalização de suas tarefas e/ou dados trocados entre o sistema e o ambiente para obter suas metas.

Os guias e diretrizes utilizados na disciplina *Requisitos Iniciais* podem também ser utilizados nesta disciplina para apoio na geração dos *Modelos de dependências dos atores* e *Modelo de raciocínio de atores.*

5.4.3 Disciplina Projeto Arquitetural

O projeto arquitetural é a definição global da arquitetura do software em termos de subsistemas e suas dependências. A arquitetura do sistema constitui um modelo de estrutura do sistema relativamente pequeno e intelectualmente gerenciável, que descreve como os componentes trabalham. O framework i* também pode ser utilizado para modelar inicialmente os componentes do sistema e suas interações. O arquiteto de software é responsável por todas as atividades do projeto arquitetural. Nas etapas iniciais, o arquiteto pode contar com o apoio do engenheiro de requisitos para analisar as informações levantadas na fase de requisitos finais e identificar o modelo organizacional e estilo arquitetural. Então, o arquiteto aplica as técnicas de análise da rede social definida pelo framework SIRA [Bastos, 2005] e define o modelo de atribuição e a configuração arquitetural do sistema.

O Framework SIRA é usado como guia para detalhar os procedimentos de análise dos requisitos (funcionais e não funcionais) e a geração dos modelos que compõe o projeto arquitetural. O projeto arquitetural é constituído de quatro modelos principais: o modelo organizacional, o estilo arquitetural selecionado, o modelo de atribuição e o modelo de arquitetura. O modelo organizacional apresenta os componentes do sistema identificados dos requisitos funcionais e o estilo organizacional apresenta os componentes identificados dos requisitos não-funcionais. O modelo de atribuição contém uma correspondência matemática dos componentes do modelo organizacional com os componentes do estilo arquitetural. O Modelo de arquitetura apresenta a configuração arquitetural gerada a partir do modelo de atribuição.

A disciplina projeto arquitetural envolve dois Papéis do Processo (Arquiteto de software e Partes interessadas), seis produtos de trabalho (Documento de requisitos finais, Catalogo arquitetural do framework NFR, Estilo arquitetural selecionado, Modelo organizacional, Modelo de atribuição e Modelo da arquitetura) e dois guias (Framework SIRA e Framework NFR). Cada Papel do Processo está responsável por um subconjunto de atividades (Figura 5-12).

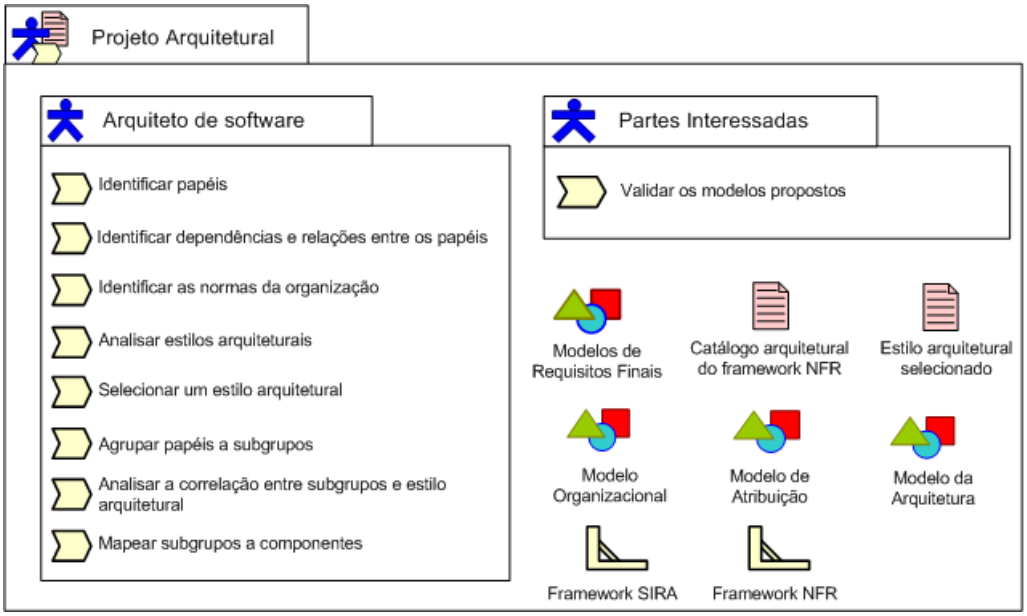


Figura 5-12 – Pacote da Disciplina Projeto Arquitetural

O fluxo do trabalho nesta disciplina possui cinco definições de trabalho (Figura 5-13). A *PA1-Especificar o modelo organizacional* compreende as atividades de definição de grupos sociais que os sistemas multiagentes devem fornecer. A *PA2-Selecionar um estilo arquitetural* compreende a análise e seleção de alternativas arquiteturais a partir dos estilos organizacionais. Estas duas primeiras definições de trabalho são realizadas em paralelo e geram os produtos de trabalho *Modelo organizacional* e *Estilo arquitetural selecionado*, que são usados para gerar o modelo de arquitetura. Em seguida a definição do trabalho *PA3-Definir o modelo de atribuição* compreende uma análise da rede social para ajustar e descrever características organizacionais e relacionar sistemas multiagentes e estilos arquiteturais selecionados. Depois, há a definição de trabalho *PA4-Configurar a arquitetura* que compreende o mapeamento dos componentes do sistema multiagente aos componentes do estilo arquitetural selecionado aplicando o modelo de atribuição definido. Por último, é realizado a *PA5-Validar os modelos propostos* por todos os envolvidos no processo. Cada uma destas tem como entrada produtos de trabalhos para uso na elaboração do projeto arquitetural e geram produtos de saída que são resultados do trabalho realizado.

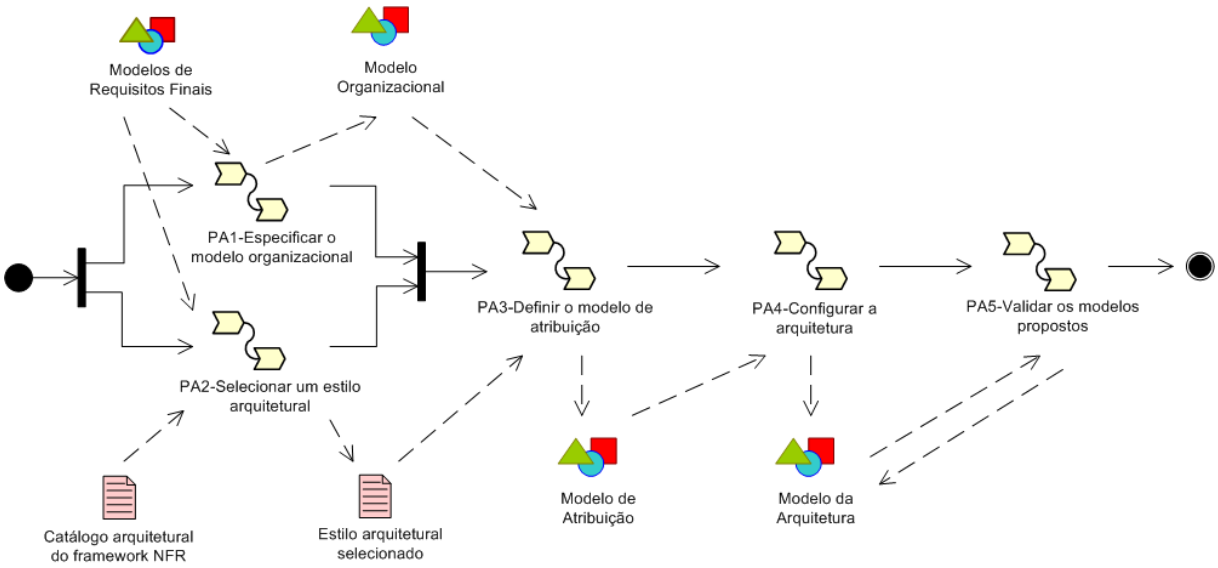


Figura 5-13 – Definição do trabalho do projeto arquitetural

A identificação e detalhamento das atividades para a geração dos modelos de arquitetura do sistema seguem a proposta do framework SIRA para detalhar a integração entre o modelo de requisitos e o modelo arquitetural (Figura 5-14). O arquiteto de software analisa os modelos de requisitos finais para gerar o *Modelo organizacional* da arquitetura. Para isto ele realiza as atividades: *1 - Identificar papéis,* que envolve a identificação e especificação de papéis com responsabilidades no sistema a partir da análise de metas e tarefas dos modelos de requisitos e/ou de informações do domínio da aplicação; *2 - Identificar dependências e relações entre os papéis,* que compreende a identificação das interações que ocorrem entre os papéis da organização; *3 – Identificar as normas da organização,* que envolve a identificação das normas que devem ser impostas aos agentes do sistema/organização.

Em paralelo outro grupo de arquitetos de software selecionam um estilo arquitetural que se adéqüe à organização do sistema através da realização das atividades: *4 - Analisar estilos arquiteturais*, que envolve a análise das propriedades não-funcionais relevantes de arquiteturas multiagentes (e.g *Previsibilidade, Segurança, Adaptabilidade, Coordenação, Cooperação, Disponibilidade, Integridade, Modularidade, Agregabilidade*, etc.) [Kolp; Giorgini & Mylopoulos, 2006] e as alternativas arquiteturais que existem para cada uma destas propriedades (e.g. *flat structure, pyramid, joint venture, structure-in-5, takeover, arm's length, vertical integration, co-optation, bidding, etc.*); *5 - Selecionar um estilo arquitetural,* que envolve a seleção de um estilo organizacional apropriado para a arquitetura do sistema a partir dos resultados da atividade anterior. Estas atividades geram o *Estilo arquitetural*

selecionado, que em conjunto com o *Modelo organizacional* será usado para gerar o *Modelo de atribuição*.

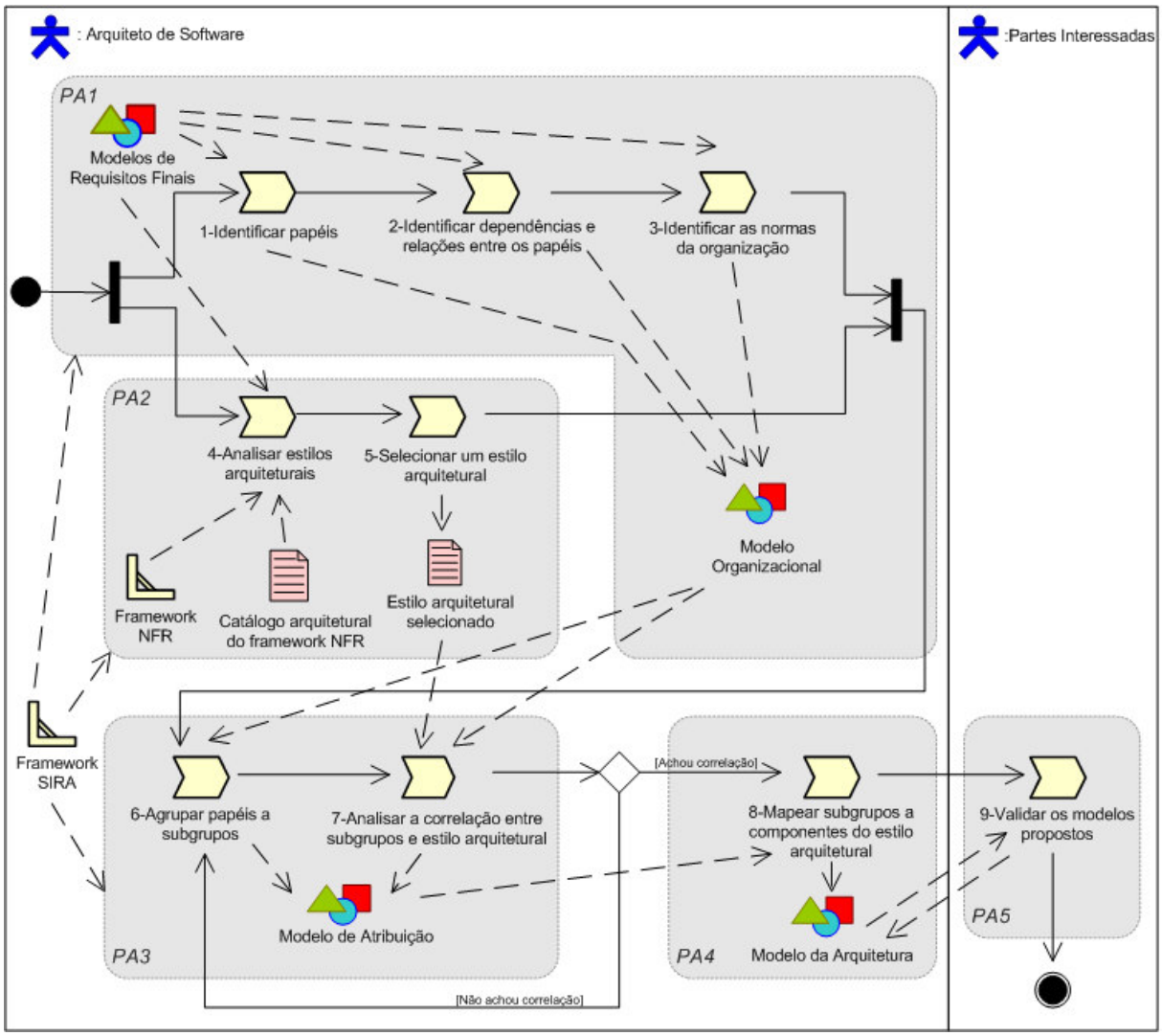


Figura 5-14 – Atividades da disciplina Projeto arquitetural

Este modelo é gerado pela realização das atividades: *6 - Agrupar papéis a subgrupos*, que envolve a análise dos papéis da organização e sua classificação em subgrupos autônomos através da análise da centralidade e da similaridade dos papéis; *7 - Analisar a correlação entre subgrupos e estilo arquitetural,* que envolve a análise e comparação do comportamento dos subgrupos (identificados na atividade *6 - Agrupar papéis a subgrupos)* com os componentes do estilo arquitetural (selecionado na atividade *5 - Selecionar um estilo arquitetural)* para indicar o grau de associação entre estes.

O *modelo de Atribuição* contém dados que permitem ao arquiteto de software definir a configuração inicial da arquitetura do sistema. Ele faz isto através da atividade *8 - Mapear subgrupos a componentes do estilo arquitetural,* que envolve o mapeamento entre os

subgrupos identificados pelos requisitos do sistema e os componentes do estilo arquitetural, para derivar a primeira configuração organizacional e gera o Modelo de Arquitetura.

Por fim é executada a atividade *9 - Validar os modelos propostos,* onde os modelos gerados são validados por todas as partes interessadas (que são os engenheiros de requisitos, especialistas em organização de agentes, arquitetos de software e desenvolvedores do sistema) e é gerada a arquitetura global do sistema. O framework SIRA é usado como diretriz para realização de todas estas atividades. A especificação completa das atividades do Projeto arquitetural está descrita no apêndice A.

O arquiteto de software é responsável por todas as atividades do projeto arquitetural. Nas etapas iniciais desta disciplina, ele pode contar com o apoio do engenheiro de requisitos para analisar as informações levantadas na fase de requisitos finais e identificar o modelo organizacional e estilo arquitetural. Então, o arquiteto aplica as técnicas de análise da rede social definida pelo framework SIRA e define o modelo de atribuição e a configuração arquitetural do sistema.

Os principais produtos de trabalho gerados são o *modelo organizacional,* o *estilo arquitetural selecionado,* o *modelo de atribuição* e o *modelo de arquitetura* (Figura 5-15). O *modelo organizacional* apresenta a identificação dos papéis e suas interações por meio de grafos e tabelas de interação. O *estilo arquitetural selecionado* inclui os papéis e interações do estilo arquitetural selecionado e o método de seleção. Estes dois modelos são gerados para identificar as duas visões de componentes do sistema: os componentes identificados dos requisitos funcionais e os componentes identificados dos requisitos não-funcionais. O *Framework NFR* é usado como guia para auxiliar na seleção do estilo arquitetural. O *modelo de atribuição* é um conjunto de análises dos dois produtos citados anteriormente que tem por fim a criação de uma correspondência entre eles. É composto de tabelas com medidas do grau de centralidade e a proximidade da centralidade e cálculo da similaridade estruturada dos papéis do subgrupo e papéis do estilo arquitetural selecionado. Também é apresentada a matriz de correlação entre os papéis do subgrupo com os papéis do estilo arquitetural selecionado. O *Modelo de arquitetura* apresenta a análise da força das correlações dos papéis e o diagrama de dependência estratégica em i* para a arquitetura do sistema.

Os guias ou diretrizes são as técnicas usadas para definição dos modelos. O Framework SIRA contém o detalhamento da geração dos modelos e exemplos ilustrativos de como utilizar os procedimentos para associar os requisitos funcionais e não funcionas do sistema ao modelo arquitetural. O Framework NFR é uma técnica de análise refinada dos requisitos não

funcionais e auxilia na identificação das *metas-soft* do sistema em relação aos atributos de qualidade do catálogo de correlação dos estilos arquiteturais.

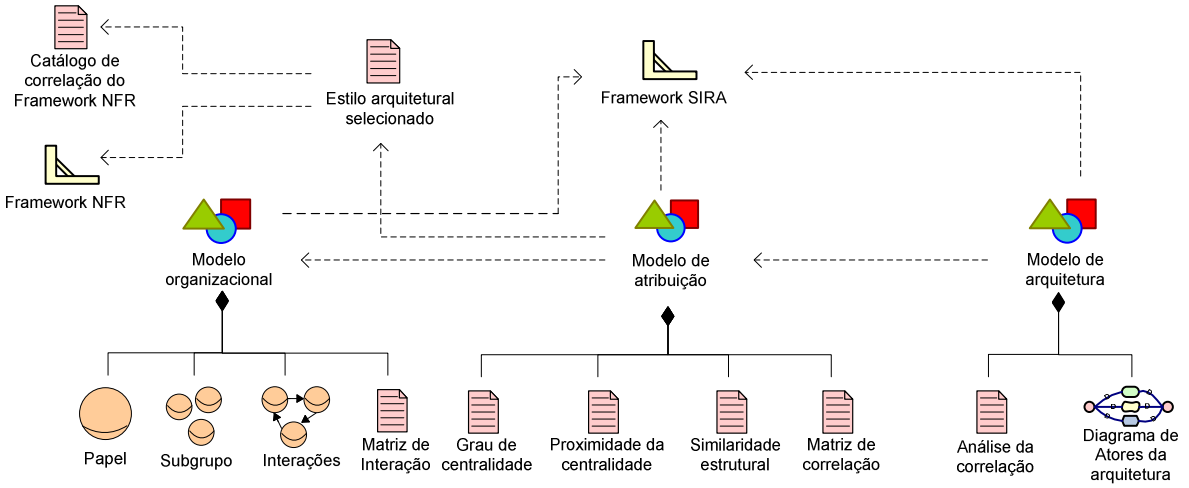


Figura 5-15 – Produtos de trabalho da disciplina Projeto Arquitetural

5.4.4 Disciplina Projeto Detalhado

O objetivo da disciplina de projeto detalhado é introduzir detalhes adicionais sobre a estrutura e comportamento de cada componente arquitetural do sistema. Consiste em definir como as metas atribuídas a cada ator do modelo arquitetural serão preenchidas pelos agentes. Os modelos gerados são diretrizes para a implementação dos agentes e suas atividades. Através destes modelos são analisadas seis visões de projetos multiagentes: a estrutura arquitetural, a comunicação entre os agentes, o ambiente onde os agentes estão inseridos, os elementos intencionais como crenças e planos dos agentes, o raciocínio para operacionalizar as tarefas dos agentes e as ações realizadas pelos agentes para executar as suas tarefas/planos [Silva, C.T.L.L., 2007a].

O modelo arquitetural que é projetado com um diagrama de dependências estratégicas do framework i* é refinado e mapeado para diagramas UML. Isto é feito para facilitar o uso de padrões sociais de sistemas multiagentes e permitir a modelagem de visões diferentes do sistema. Assim o sistema pode ser implementado em qualquer plataforma FIPA [FIPA, 2004] que dê suporte a modelos de agentes BDI [Rao & Georgeff, 1991].

A disciplina projeto detalhado envolve três papéis do processo (arquiteto de software, projetista de sistema e partes interessadas), dez produtos de trabalho (documento de mapeamento, catálogo de padrões sociais, decisões de projeto, diagrama de arquitetura, diagrama de comunicação, diagrama de intenção, diagrama de planos, diagrama de raciocínio, diagrama do ambiente e projeto detalhado dos agentes) e um guia (heurísticas de mapeamento

i* para UML). Cada papel do processo está responsável por um subconjunto de atividades (Figura 5-16).

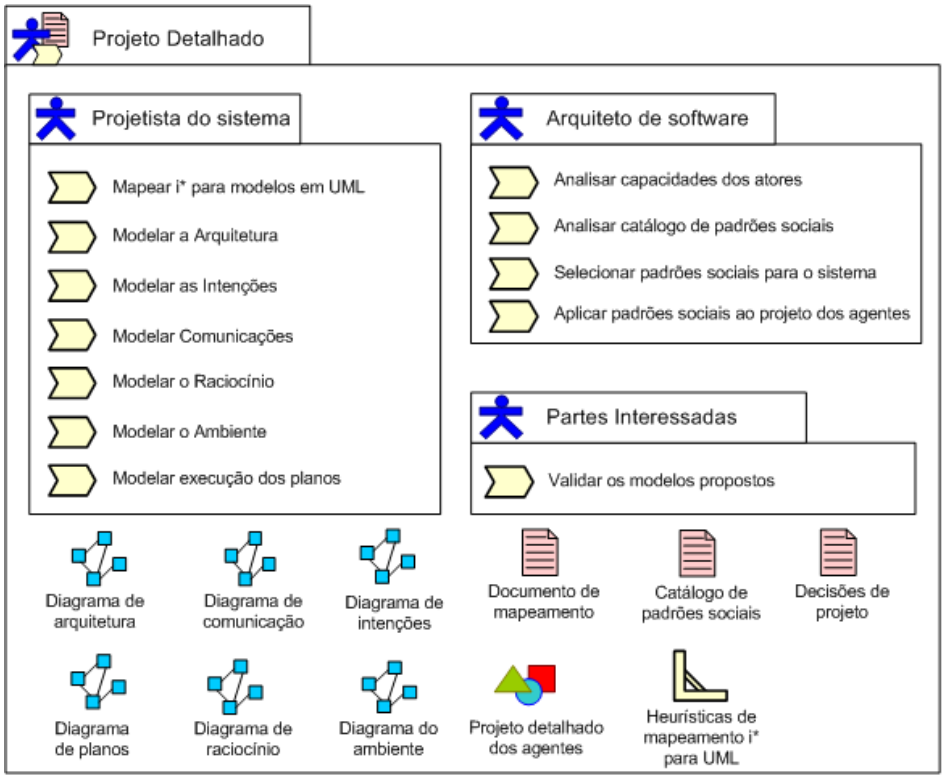


Figura 5-16 - Pacote da Disciplina Projeto Detalhado

O fluxo do trabalho nesta disciplina possui três definições de trabalho (Figura 5-17). A primeira definição de trabalho é a *PD1-Refinamento do projeto da arquitetura,* que compreende o mapeamento do modelo da arquitetura que está representada como um modelo i* em seis diagramas da UML que representam diferentes visões da arquitetura. Estes seis diagramas compõem o produto de trabalho de saída chamado *Projeto detalhado dos agentes*. A segunda definição do trabalho é a *PD2-Análise do uso de padrões sociais* que compreende uma análise de como as metas atribuídas a cada ator são preenchidas por agentes com respeito a padrões sociais. Estes padrões são definidos em um catálogo de padrões sociais, que é entrada desta definição de trabalho. A aplicação dos padrões sociais gera uma atualização no produto de trabalho *Projeto detalhado de agentes*. A terceira definição do trabalho é a *PD3-Validação dos modelos propostos* por todos os envolvidos no processo.

As atividades para a geração dos modelos de projeto detalhado de agentes seguem a proposta de Silva, C.T.L.L. (2007a), que partindo do modelo de arquitetura, gera seis visões detalhadas dos agentes e o ambiente onde está inserido (Figura 5-18). O Projetista de agentes inicia realizando a atividade: *1 - Mapear i* para modelos em UML*, que envolve o mapeamento do *Modelo de arquitetura* modelados na notação i* para diagramas na notação

UML usando como guia o documento *Heurísticas de mapeamento i* para UML* e gerando o *Documento* de *mapeamento*. Este documento é usado para orientar a execução das atividades de modelagem dos diagramas UML que representam as visões do projeto detalhado do sistema.

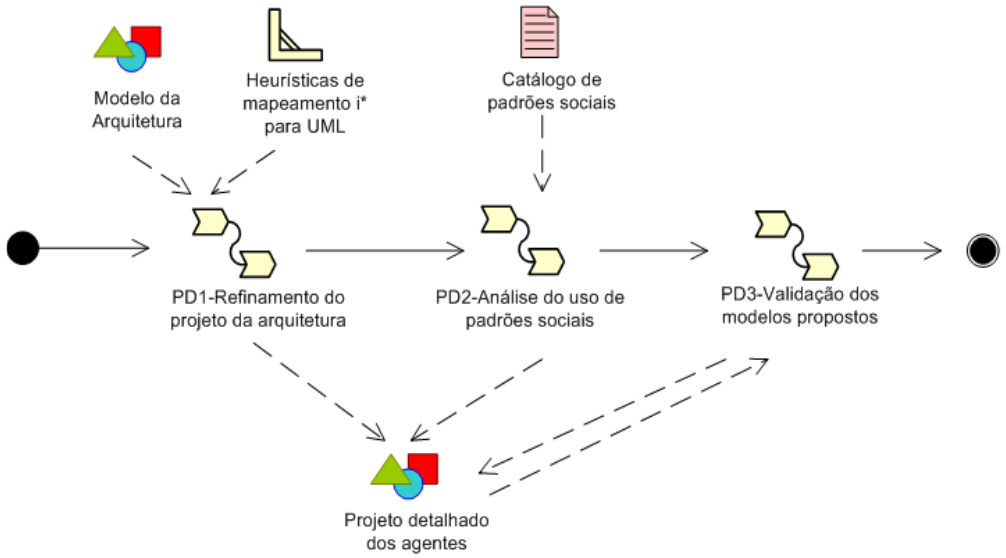


Figura 5-17 – Fluxo de trabalho do projeto detalhado

Estas visões são desenvolvidas pelo Projetista de agentes através das atividades: 2 - *Modelar a Arquitetura*, que envolve a modelagem do sistema multiagente como um diagrama de classes da UML gerado a partir do *Modelo Arquitetural* e do *Documento de Mapeamento; 3 - Modelar Comunicações,* que envolve a modelagem das interações entre os papéis e agentes do sistema multiagente através de um diagrama de seqüência da UML; *4 - Modelar as Intenções,* que envolve a modelagem, em um diagrama de classes da UML, dos elementos intencionais dos agentes, ou seja, os papéis executados, suas crenças, metas, planos, normas e ontologias usadas na organização; *5 - Modelar o Ambiente,* que envolve a modelagem, em um diagrama de classes da UML, dos recursos do ambiente e dos direitos de acesso dos papéis e agentes de uma organização; *6 - Modelar o raciocínio,* que envolve a modelagem, em um diagrama de classes da UML, dos elementos racionais dos agentes, ou seja, especifica a influência das tarefas para a satisfação das metas-soft com o objetivo de ajudar o agente a escolher entre tarefas alternativas, quais obtém uma dada meta; *7 – Modelar a execução dos planos,* que envolve a modelagem, em um diagrama de atividades da UML, da seqüência de ações que compõe os planos para obter as metas. Estas atividades geram respectivamente os *diagrama de arquitetura*, *diagrama* de *comunicação*, *diagrama de intenções*, *diagrama de ambiente*, *diagrama de* raciocínio e *diagrama de planos*.

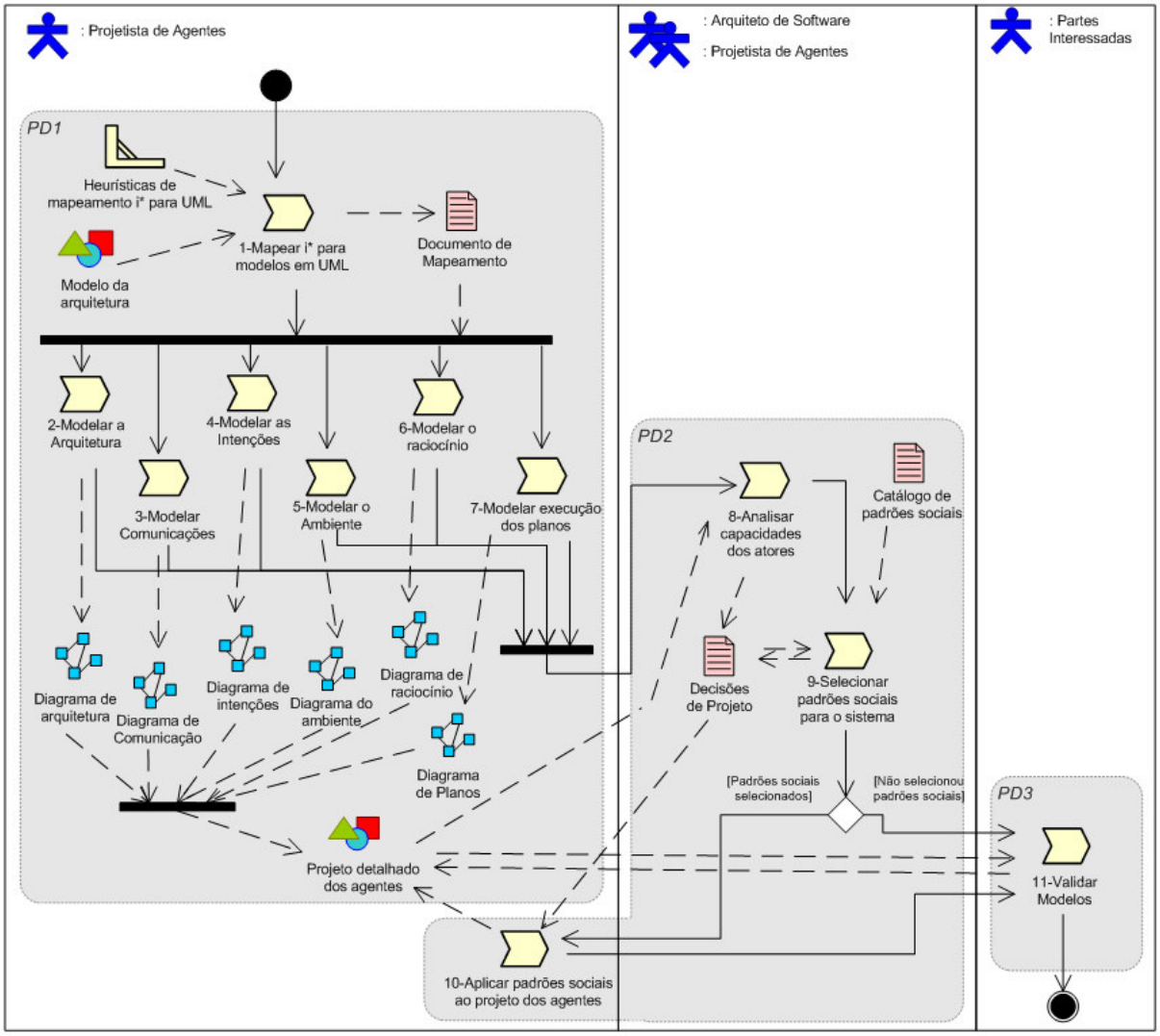


Figura 5-18 – Atividades da disciplina Projeto detalhado

Após isto, os Arquitetos de software executam a atividade *8 - Analisar capacidades dos atores*, que envolve a identificação das capacidades dos agentes, ou seja, a habilidade de um ator em definir, escolher e executar um plano para atendimento de uma meta, mediante certas condições ambientais e na presença de um evento específico. Após isto é realizada a atividade *9 - Selecionar padrões sociais para o sistema* usando o catálogo de padrões sociais e o documento *Decisões de projeto* para identificar padrões sociais para o sistema. As decisões são documentadas no documento de *Decisões do projeto*. Se existirem padrões sociais selecionados, a atividade *10 - Aplicar padrões sociais ao projeto de agentes* é executada. Se não, a atividade *11- Validar Modelos* é realizada e finaliza o fluxo de atividades desta disciplina. Após a realização da atividade *10 - Aplicar padrões sociais ao projeto de agentes*, os diagramas UML definidos no projeto detalhado são atualizados e validados na execução da atividade *11 - Validar Modelos*. Esta última atividade é executada pelas Partes Interessadas e

gera atualizações no Projeto detalhado dos agentes. Este produto de trabalho é união dos seis diagramas UML descritos anteriormente. A especificação completa das atividades do Projeto detalhado está descrita no apêndice A.

O engenheiro de requisitos tem um papel na primeira parte do projeto detalhado, pois juntamente com o arquiteto de software analisa as metas dos componentes da arquitetura. Após isto o projetista de agentes modela o projeto detalhado dos agentes baseado no projeto detalhado da arquitetura. O projetista de sistemas pode usar padrões sociais de agentes. Esta decisão é tomada juntamente com o arquiteto de software. Estes dois papéis de processo analisam os padrões sociais em contrapartida com os papéis de agentes do sistema e identificam se existem padrões sociais que podem ser utilizados no sistema multiagentes. As partes interessadas são todos os envolvidos no projeto detalhado dos agentes, incluindo representantes técnicos do cliente, especialistas em agentes, arquitetos de software, projetistas de agentes e/ou desenvolvedores de agentes.

Os principais produtos de trabalho gerados são o *Projeto detalhado da arquitetura,* o *Documento de mapeamento,* as *Decisões de projeto* e o *Projeto detalhado de agentes* (Figura 5-19).

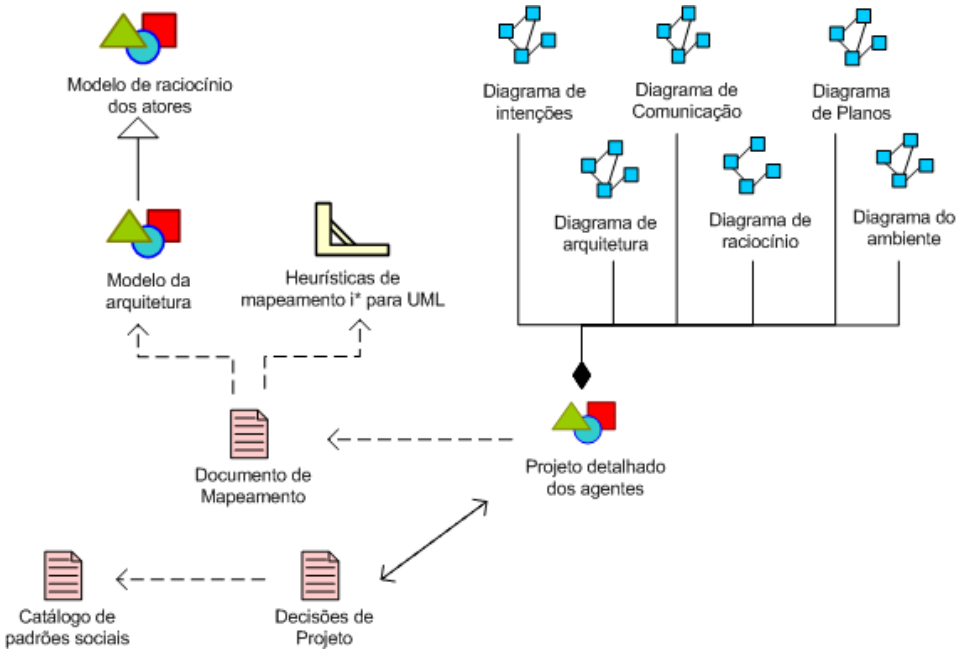


Figura 5-19 – Produtos de trabalho do Projeto detalhado

O *Documento de mapeamento* contém tabelas com instruções de derivação do *Modelo da arquitetura* no *Projeto detalhado do agente* seguindo as diretrizes descritas no guia *Heurísticas de mapeamento i* para UML*. O Projeto detalhado é composto de seis modelos

UML contendo as seis visões do projeto detalhado dos agentes, que são a visão estrutural modelada no *Diagrama arquitetural*, a visão de interação dos agentes modelada no *Diagrama de comunicação*, a visão das intenções dos agentes modelada no *Diagrama de intenções*, a visão racional dos agentes modelada no *Diagrama de raciocínio* e a visão das ações realizadas pelos agentes modelada no *Diagrama de planos*. As *Decisões* de projeto incluem a análise dos padrões sociais e as decisões tomadas quanto a sua aplicação no projeto detalhado dos agentes.

5.4.5 Disciplina Implementação

Durante a implementação, a especificação do projeto arquitetural é utilizada para codificar o sistema multiagente. Podem-se usar os ambientes de desenvolvimento de sistema tais como JADEX [Pokahr; Braubach & Lamersdorf, 2005] ou JACK [JACK, 2007]. Estes ambientes simplificam a codificação oferecendo um framework para implementar os agentes, seu comportamento e as interações com outros agentes.

A disciplina *Implementação* envolve um Papel do processo (Programador), cinco produtos de trabalho (*Projeto detalhado dos agentes, Ambiente de desenvolvimento, Incrementos de software, o sistema multiagente (SMA) e a Documentação* do SMA) e um guia (*Guia do programador*). O *Programador* é responsável por todas as atividades de implementação (Figura 5-20). As atividades de testes e verificação não fazem parte desta disciplina, mas da disciplina *Verificação*.

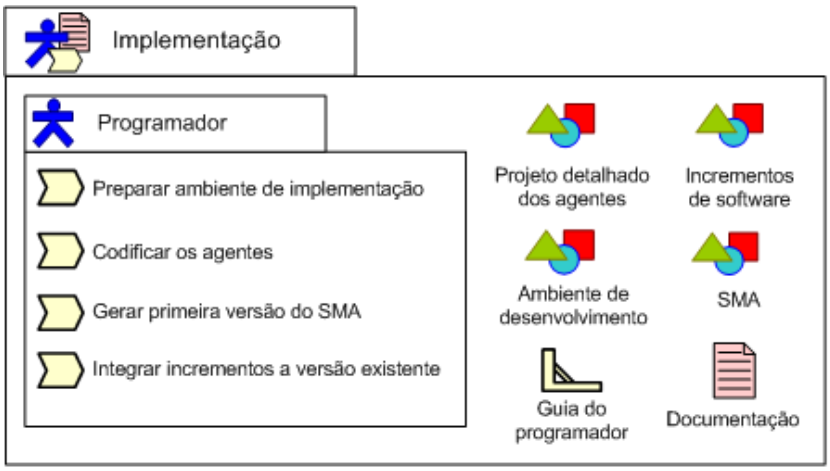


Figura 5-20 – Pacote da disciplina Implementação

O fluxo do trabalho nesta disciplina possui duas definições de trabalho (Figura 5-21). A primeira definição de trabalho é a *IM1-Codificação* que compreende as atividades de

preparação do ambiente de implementação e a codificação dos requisitos selecionados para a iteração. Nesta definição de trabalho, o projeto detalhado de agentes é convertido em código implementável e aplicado a um framework de desenvolvimento (e.g. JADEX ou JACK) para gerar os incrementos do software. A segunda definição do trabalho é a *IM2 - Integração* que compreende as atividades de integração dos incrementos do software à versão do sistema já codificada em iterações anteriores. Esta definição de trabalho gera o sistema multiagente executável pronto pra ser entregue ao cliente.

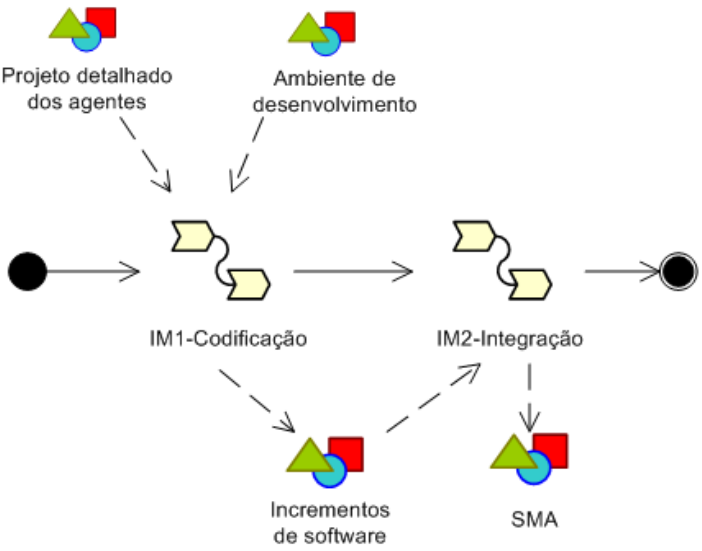


Figura 5-21 – Definição de trabalho da Implementação

O programador é o papel de processo responsável pelas atividades de *Codificação* (IM1). Ele faz isto através da execução das atividades: *1 - Preparar ambiente de implementação* e *2 - Codificar os agentes*. A primeira atividade envolve a configuração do framework de desenvolvimento para iniciar o desenvolvimento a partir do projeto detalhado. A segunda atividade é a implementação dos requisitos definidos na iteração. Após a codificação dos requisitos definidos para a iteração, é gerada uma versão do Sistema multiagente para avaliação do usuário. Esta versão é gerada fazendo a integração do código (IM2) através das atividades *3 - Gerar primeira versão do SMA* ou *4 - Integrar incrementos a versão existente*. A atividade 3 é realizada se for a primeira versão do sistema gerada. Se já existir uma versão anterior que foi entregue ao usuário, é realizada a atividade 4 para integrar o código que foi desenvolvido à versão atual (Figura 5-22). A especificação completa das atividades da Implementação está descrita no apêndice A.

O programador desempenha o papel de processo, sendo responsável por todas as atividades de implementação. Seu trabalho é apoiado pelo projetista de agentes para garantir a implementação de acordo com o projeto e identificar possíveis alterações no projeto a partir de necessidades de implementação.

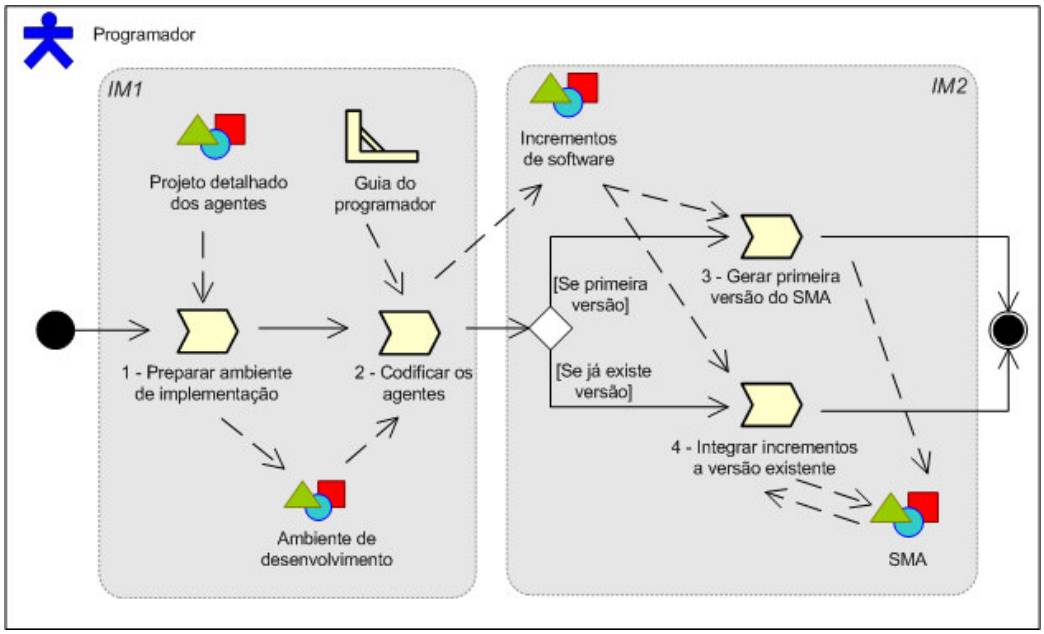


Figura 5-22 – Atividades da disciplina Implementação

Os principais produtos de trabalho gerados são o *Ambiente de desenvolvimento*, os *Incrementos de* software, *o Sistema multiagentes (SMA) e a Documentação* do SMA (Figura 5-23). O SMA é composto pelos *Incrementos de software* que são integrados e pela *Documentação do código*. Os incrementos de softwares são gerados a partir de informações descritas no Projeto detalhado de agentes e nas características do Ambiente de desenvolvimento adotado.

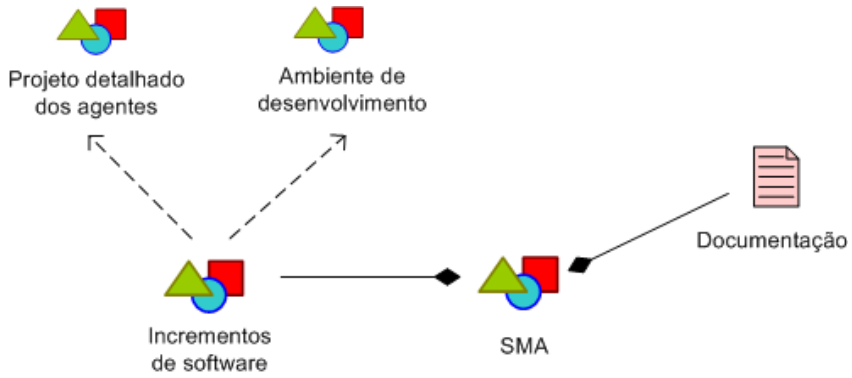


Figura 5-23 – Produtos de trabalho da disciplina Implementação

5.5 Considerações Finais

Neste capítulo é proposto o U-Tropos: um novo processo unificado para sistemas multiagentes conforme Tropos. Esta proposta agrega técnicas recentes que proporcionam melhorias ao Tropos. São definidas atividades e especificado um processo iterativo usando a linguagem de especificação de processos SPEM. O U-Tropos inclui um modelo de ciclo de vida que abrange todo o processo de desenvolvimento. Este modelo inclui as fases de definição, projeto, construção e implantação e as disciplinas requisitos iniciais, requisitos finais, projeto arquitetural, projeto detalhado, implementação, gerenciamento, verificação e suporte. As fases são desenvolvidas de forma iterativa ao longo do tempo. Cada iteração inclui um conjunto de atividades das disciplinas com o objetivo de produzir um produto de trabalho para atender alguma meta específica do projeto de desenvolvimento do software.

Nesta dissertação foram detalhadas as seguintes disciplinas: os requisitos iniciais e finais, o projeto arquitetural, projeto detalhado e implementação. As demais disciplinas (gerenciamento, verificação e suporte) serão desenvolvidas em trabalhos futuros, pois não faziam parte do escopo da dissertação. As cinco disciplinas apresentadas foram detalhadas com a descrição dos elementos conforme proposto no metamodelo SPEM. Estes elementos são o fluxo de trabalho realizado, as atividades que compõe o fluxo de trabalho, os papéis executores do processo, os produtos de entrada e produtos de saída das atividades e os guias usados como diretrizes para realização das atividades.

As disciplinas de *Requisitos iniciais e finais* foram definidas seguindo as recomendações de levantamento e análise de requisitos propostas nas versões Tropos'02 [Castro; Kolp & Mylopoulos, 2002] e Tropos'04 [Bresciani et al., 2004]. Como visto no capítulo 3 existem duas nomenclaturas para nomear os produtos de trabalho gerados. Tropos'02 usa os termos originais usados no framework i* (Modelos de dependência estratégica e Modelo de raciocínio estratégico) e o Tropos 04 usa uma denominação própria (Modelo de atores e Modelo de metas). Os usuários podem escolher a notação de sua preferência. Nesta dissertação, foram acrescentadas atividades para levantamento de requisitos que não foram incluídas nas propostas Tropos'02 e Tropos'04, que abrangem apenas a modelagem. Também, foram sugeridas algumas técnicas para guiar a criação de modelos i* a partir do levantamento dos requisitos. As técnicas sugeridas foram o RISD [Grau et al., 2005], PRIM [Grau; Franch & Maiden, 2005], SDSituation [Oliveira; Padua & Cysneiros, 2006] e o Estudo qualitativo para elicitação de requisitos [Cruz Neto, 2008]. O

guia do i* [Grau et al., 2008] também tem sido muito útil para gerar modelos i* corretos, portanto foi acrescentado como guia nestas disciplinas.

A disciplina de *Projeto arquitetural* foi acrescida da proposta do framework SIRA [Bastos, 2005] para gerar uma arquitetura a partir dos requisitos funcionais e não funcionais do sistema. Nas versões anteriores de Tropos, o projeto arquitetural envolvia as atividades de geração do modelo arquitetural e definição dos padrões sociais de agentes. No processo U-Tropos, o projeto arquitetural finaliza com a definição do modelo arquitetural e os padrões sociais de agentes são definidos no projeto detalhado.

A disciplina *Projeto detalhado* segue o modelo apresentado no trabalho de Silva, C.T.L.L. (2007a), que detalhou os agentes em seis visões de projeto. Estas visões são o modelo arquitetural, modelo ambiental, modelo de comunicação, modelo racional, modelo intencional e modelo de planos. Estas modelos usam uma versão estendida da UML como linguagem de modelagem. Após este refinamento dos agentes, os padrões sociais são selecionados. As capacidades dos agentes são identificadas a partir dos diagramas do projeto detalhado e os padrões sociais são definidos.

Os trabalhos de Wautelet, Kolp & Achbany (2005) e FIPA (2008) também apresentam uma modelagem da metodologia Tropos usando SPEM. Mas eles se detêm a modelar o que foi proposto nas metodologias sem acrescentar outros recentes trabalhos, o que é objetivo desta proposta. O S-Tropos [Wautelet; Kolp & Achbany, 2005] acrescenta outras disciplinas (Validação, Distribuição e Gerenciamento de projetos e riscos), mas não adiciona melhorias às disciplinas da metodologia original.

Neste capítulo, foi descrito apenas a definição genérica do processo U-Tropos. Este processo pode ser customizado para cada projeto específico, dependendo dos guias selecionados para orientar o desenvolvimento. Isto ocorre principalmente nas fases de requisitos iniciais e finais, onde foram sugeridos diversos guias (PRIM, RISD, SD-Situation, Estudos qualitativos de elicitação de requisitos) e o uso da versão do i* (i* original ou i* do Tropos). O próximo capítulo fornece um exemplo de utilização do processo proposto para desenvolver um sistema de software multiagentes para uma aplicação de resolução de falhas autônomas em banco de dados.

Capítulo 6 – Exemplo de aplicação do processo

Esse capítulo tem o objetivo de apresentar um exemplo para uso prático da proposta desta dissertação. Será utilizado o desenvolvimento do sistema multiagentes DBSitter-AS, que tem como objetivo a detecção e correção autônoma de falhas em banco de dados. Nesse contexto, é apresentado um guia para a aplicação do processo dividido em iterações que geram os produtos de trabalho das atividades de requisitos, projeto arquitetural e projeto detalhado.

6.1 Introdução

Com o objetivo de comparar as duas versões da metodologia [Silva, M.J. et al., 2007], no estágio inicial desta dissertação foi escolhido um sistema multiagente que estava em desenvolvimento para ser modelado em Tropos Este sistema também será usado para exemplificar a aplicação do U-Tropos. Contudo, foram feitas adaptações à modelagem inicial do sistema DBSitter-AS, seguindo as disciplinas e guias propostos no U-Tropos.

Neste capítulo é apresentado o sistema DBSitter-AS e suas características principais. Em seguida é proposta uma estrutura de iterações a ser seguidas no ciclo de vida do desenvolvimento do sistema. Algumas iterações são apresentadas demonstrando o uso das atividades e o desenvolvimento dos produtos de trabalhos.

6.2 O framework DBSitter-AS

O Framework DBSitter-AS [Maciel, 2007] é uma especificação arquitetural que define como construir componentes de software que forneçam autonomia de gerenciamento para Sistemas Gerenciadores de Banco de Dados (SGBD). Organizada como um sistema multiagentes (SMA), esta especificação arquitetural é compatível com os princípios da computação autônoma e provê uma forma para desenvolvimento de características flexíveis de prevenção e resolução de falhas em SGBD tradicionais.

Os componentes desenvolvidos seguindo a especificação DBSitter-AS são localizados externamente ao SGBD. Com o auxílio desses componentes, o administrador de banco de dados é ser poupado de tarefas rotineiras e tem tempo para se dedicar ao planejamento estratégico da gestão dos dados. O objetivo principal da proposta é manter um SGBD em funcionamento normal, de modo mais autônomo possível, realizando prevenção, resolução de falhas e emissão de alertas. Além disso, os componentes autônomos podem aprender com as ações realizadas mediante análise do *feedback* recebido do usuário.

O DBSitter-AS foi concebido como uma abstração de uma organização administradora de SGBD, formada por agentes e composta por três sub-organizações: *Detecção, prevenção e correção de falhas, Notificações e registros e Aprendizagem e sugestão*. O ambiente onde o SMA de arquitetura DBSitter-AS atua, é composto pelo SGBD alvo de gerenciamento (e.g. Oracle, PostgreSQL, SQL Server), o sistema operacional (SO) do servidor onde o SGBD é executado e uma base de conhecimento e registro. A sub-organização para detecção, prevenção e correção de falhas atua no SGBD e no SO, assim como a sub-organização para

notificação e registros. As três sub-organizações utilizam-se da base de conhecimento e registro. Desta forma há uma separação funcional entre os agentes que farão parte das sub-organizações.

Em particular, a sub-organização *Detecção, prevenção e correção de problemas* por sua complexidade e variedade de problemas existentes em bancos de dados, é subdividida em mais duas sub-organizações: *Detecção e correção de problemas de SGBD* e *Detecção e correção de problemas de infra-estrutura*. A classificação feita busca caracterizar as diversas falhas que podem acontecer em um SGBD. A sub-organização *Detecção e correção de problemas de SGBD* foi dividida em três novas sub-organizações de gerenciamento: *Gerenciamento de estruturas de controle, Gerenciamento de estruturas físicas e Gerenciamento de estruturas lógicas*. Estruturas de controle são usadas comumente pelos SGBD para definir seus comportamentos, como arquivos de inicialização, parâmetros para definição de linguagem ou nível de acesso dos usuários. Estruturas físicas dizem respeito aos arquivos físicos utilizados pelos SGBD para armazenar os dados. Estruturas lógicas são os objetos manipuláveis para tratamento dos dados, como tabelas, índices, catálogos e stored procedures. A sub-organização *Detecção e correção de problemas de infra-estrutura* foi dividida em *Gerenciamento de sistema operacional (SO)* e *Gerenciamento de rede*, referentes a ambientes onde também podem ocorrer erros que afetem o desempenho do SGBD. Em relação ao SO podem ocorrer picos de processamento que identificados ajudem no diagnóstico de perdas de desempenho ou mesmo a necessidade de interrupção de processos. Em relação à rede, há a necessidade do controle de estruturas responsáveis pela comunicação do SGBD com o mundo exterior, como portas de comunicação ou processos “Listeners” que precisem ser iniciados ou cancelados.

Para cada sub-organização identificada há grupos de agentes com perfis, habilidades e responsabilidades semelhantes, que interagem com os das outras sub-organizações através de solicitações de serviços e informações para atingir de forma harmônica, os objetivos globais da organização DBSitter-AS.

6.3 O processo de análise e projeto para o DBSitter-AS

Um sistema complexo será mais bem sucedido se implementado em pequenos passos [Gilb,1981]. Um passo chave no processo é iniciar uma implementação de um subconjunto dos requisitos do software e iterativamente aprimorar a seqüência de versões até que o sistema completo esteja implementado. A cada iteração, modificações de projeto são realizadas

adicionando-se novas capacidades funcionais. Desta forma é possível receber *feedback* do mundo real antes de usar todos os recursos planejados para o sistema, possibilitando a correção de erros de projeto.

A iteratividade tem uma relação estreita com o desenvolvimento de sistemas multiagentes, que se classifica como um sistema complexo e subdividido em sub-sistemas representados pelos agentes que interagem em um determinado ambiente. As iterações podem ser definidas para desenvolvimento de um agente específico ou um subgrupo de agentes que fazem parte da organização completa. O DBSitter-AS será desenvolvido seguindo iterações planejadas acompanhando as fases do processo U-Tropos.

Como descrito no Capítulo 5, o U-Tropos está dividido em quatro fases: definição, projeto, construção e implantação. A fase de definição tem como condição prévia a proposta do sistema e como objetivo o escopo definido e validado por todos os envolvidos. A descrição do DBSitter-AS na Seção 6.2 exemplifica a proposta inicial do sistema. A definição e validação do escopo do sistema é dscrito nos produtos gerados nas disciplinas de requisitos iniciais e requisitos finais. A fase de projeto inicia com a condição prévia de ter definido e validado o escopo do sistema. Como o processo é iterativo, esta condição se resume a ter o escopo da iteração de definição bem definida e validada pelas partes interessadas. O objetivo é a arquitetura e projeto definidos e validados por todos os envolvidos. Este objetivo é alcançado com a execução das atividades das disciplinas projeto arquitetural e projeto detalhado. Por fim, o sistema é implementado de acordo com o projeto detalhado em sucessivas iterações de construção. Estas fases são divididas em iterações. A Tabela 6-1 sugere um conjunto de cinco iterações que podem ser usadas para desenvolver um sistema seguindo as fases e atividades das disciplinas definidas no U-Tropos.

A iteração *1-Definição do sistema* envolve a definição dos requisitos que definem o escopo do sistema, isto é, onde o sistema estará inserido dentro da organização e que metas das partes interessadas estará atendendo. Esta é uma iteração da fase de *Definição* e envolve todas as atividades da disciplina *Requisitos Iniciais* (RI1, RI2, RI3) e apenas as atividades de *Requisitos Finais* que incluem as dependências dos atores (partes interessadas) com o ator sistema (RF1.1, RF2.2 e RF3.7). A saída resultante é a identificação das partes interessadas no sistema e as dependências que estes têm do sistema.

A iteração *2 - Análise dos requisitos* contém as atividades de levantamento e análise dos requisitos que serão usados no projeto e implementação do primeiro protótipo operacional. Para isto, é selecionado um subconjunto prioritário de metas atribuídas ao ator sistema e estas são analisadas para obter os requisitos funcionais e não funcionais que serão implementados.

Esta é uma iteração da fase de *Definição,* pois os requisitos estão sendo definidos. E envolve atividades de levantamento (RF1.3) e análise das metas do ator sistema (RF2) na disciplina *Requisitos Finais*. Cada iteração é concluída com uma atividade de validação (RF3).

Tabela 6-1 Iterações definidas para o desenvolvimento do DBSitter-AS

Disci-plinas	*Atividades*
Iteração 1 - Definição do sistema	
RI	**RI1-Elicitação dos requisitos iniciais**
RI	1-Identificar partes interessadas e suas intenções
RI	2-Identificar relacionamentos entre as partes interessadas
RI	4-Identificar atividades detalhadas das partes interessadas
RI	**RI2-Análise e especificação dos requisitos iniciais**
RI	3-Analisar as dependências dos atores
RI	5-Realizar Análise orientada a Metas
RI	6-Realizar Análise Meio-Fim
RI	7-Realizar Análise das contribuições
RI	**RI3-Validação dos modelos propostos**
RI	8-Validar os modelos propostos
RF	**RF1-Levantamento dos requisitos do sistema**
RF	1-Elicitar dependências dos atores e ator-sistema
RF	**RF2-Análise dos requisitos do sistema**
RF	2-Analisar as dependências do ator-sistema
RF	**RF3-Validação dos modelos propostos**
RF	7-Validar os modelos propostos
Iteração 2 - Análise dos requisitos do sistema	
RF	**RF1-Levantamento dos requisitos do sistema**
RF	3-Identificar lógica p/ obter intenções do ator
RF	**RF2-Análise dos requisitos do sistema**
RF	4-Realizar análise orientada a metas
RF	5-Realizar análise Meio-Fim
RF	6-Realizar análise das contribuições
RF	**RF3-Validação dos modelos propostos**
RF	7-Validar os modelos propostos

Disci-plinas	*Atividades*
Iteração 3 - Projeto da arquitetura global do sistema	
PA	**PA1-Especificar o modelo organizacional**
PA	1-Identificar papéis
PA	2-Identificar dependências entre os papéis
PA	3-Identificar as normas da organização
PA	**PA2-Selecionar um estilo arquitetural**
PA	4-Analisar estilos arquiteturais
PA	5-Selecionar um estilo arquitetural
PA	**PA3-Definir o modelo de atribuição**
PA	6-Agrupar papéis a subgrupos
PA	7-Analisar a correlação entre subgrupos e estilo arquitetural
PA	**PA4-Configurar a arquitetura**
PA	8-Mapear subgrupos a componentes do estilo arquitetural
PA	**PA5-Validar os modelos propostos**
PA	9-Validar os modelos propostos
Iteração 4 - Projeto detalhado do primeiro protótipo do sistema	
RF	**RF2-Análise dos requisitos do sistema (componentes)**
RF	4-Realizar Análise orientada a Metas
PD	**PD1-Refinamento do projeto arquitetura**
PD	1-Mapear i* para modelos em UML
PD	2-Modelar a Arquitetura
PD	3-Modelar comunicações
PD	4- Modelar as intenções
PD	5-Modelar o ambiente
PD	6-Modelar o raciocínio
	7-Modelar execução dos planos
PD	**PD3-Validação dos modelos propostos**
PD	11-Validar Modelos
Iteração 5 – Implementação do primeiro protótipo do sistema	
IM	**IM1-Codificação**
IM	1-Preparar ambiente de desenvolvimento
IM	2-Codificar os agentes
IM	3-Gerar primeira versão do SMA

Legendas:

RI-Requisitos Iniciais; RF-Requisitos Finais;
PA-Projeto Arquitetural;
PD-Projeto detalhado; IM-Implementação

A iteração *3 – Projeto e arquitetura global do sistema* inclui atividades de análise de um subconjunto de metas para definir a arquitetura do sistema. Esta é a primeira iteração da

fase de *Projeto* e envolve todas as atividades da disciplina *Projeto arquitetural* para produzir a primeira configuração arquitetural do sistema.

A iteração *4 – Projeto detalhado do primeiro protótipo do sistema* envolve as atividades de projeto detalhado e construção do conjunto de requisitos analisados na iteração 2. Esta é uma iteração da fase de *Projeto* e envolve atividades da disciplina *Requisitos Finais* (RF2) e todas as atividades da disciplina *Projeto detalhado* (PD1, PD2 e PD3). As atividades da disciplina *requisitos finais* são necessárias para revisar a análise das metas dos papéis de agentes do sistema definidos no *Projeto arquitetural*. Após isto, as atividades de projeto detalhado são realizadas produzindo os modelos UML (PD1) e identificando padrões sociais (PD2). Por fim os modelos são validados (PD3).

A iteração *5- Implementação do primeiro protótipo do sistema* é composta das atividades de implementação, onde os agentes são codificados gerando a primeira versão do sistema multiagente. Esta é uma iteração da fase de *construção* e envolve apenas atividades da disciplina *Implementação*.

Estas iterações geram um primeiro protótipo do sistema, com um subconjunto de requisitos do sistema. Dependendo do tamanho e complexidade do sistema, as iterações 2, 3, 4 e 5 se repetem e versões do software são produzidas até a conclusão do projeto de desenvolvimento. Ao final de todas as iterações é gerado um sistema completo. A divisão das iterações é uma decisão de projeto e deve ser definida de acordo com as necessidades.

O exemplo do desenvolvimento do sistema DBSitter-AS é organizado seguindo as atividades das quatro primeiras iterações definidas na Tabela 6-1. Nas próximas seções estão apresentados os produtos de trabalho gerados pela realização das atividades destas iterações.

6.3.1 Iteração 1 – Definição do sistema

O projeto do sistema inicia com uma iteração da fase *Definição*. Nesta iteração, são identificadas as informações globais do sistema e da organização onde este será inserido. As principais atividades pertencem às disciplinas de *Requisitos iniciais* e *finais*. Serão desenvolvidas seis definições de trabalho conforme a Tabela 6-1. A identificação das partes interessadas no sistema e seus relacionamentos é realizada pelas atividades 1 e 2 da definição de trabalho *RI1- Elicitação dos requisitos iniciais.* As ações executadas pelas partes interessadas são identificadas através da atividade 4. Estas informações são analisadas usando modelos i* durante a realização das atividades 3, 5, 6 e 7 da definição de trabalho *RI2 – Análise e especificação dos requisitos iniciais*. Os modelos são validados pela realização da atividade 8 da definição de trabalho *RI3 – Validação dos modelos propostos*.

Com a finalidade de definir o escopo dos requisitos finais do sistema nesta iteração de *Definição*, são realizadas atividades da disciplina Requisitos Finais. A atividade 1 da definição de trabalho RF1 – Levantamento dos requisitos do sistema identifica quais são as necessidades das partes interessadas em relação ao sistema. Estas necessidades são analisadas por meio de modelos i* na atividade 2 da definição de trabalho *RF2-análise dos requisitos do sistema.* Os modelos são validados pela realização da atividade 7 da definição de trabalho *RF3 – Validação dos modelos propostos*. Através da execução destas atividades, o sistema DBSitter-AS é definido estabelecendo-se quais são suas metas e relacionamentos dentro da organização que será inserido.

Na disciplina de requisitos, podem ser usados métodos de apoio a elicitação e construção dos modelos i*. Neste exemplo, o PRIM (Process Reengineering i* Methodology) foi selecionado como guia para documentação dos requisitos. O PRIM é dividido em cinco fases. A primeira é uma fase de coleta de informações da organização (fase1). As próximas fases sugerem um modelo de construção do i*, acrescentando diretrizes de construção do modelo a partir das informações coletadas: fase2 - Construção do modelo i*; fase3 - Análise do raciocínio estratégico; fase4 - avaliação de alternativas estratégicas e fase5 - especificação do novo sistema. A fase2 do PRIM propõe o uso de cenários chamados de Roteiros de interação detalhados ou DIS (Detailed Interaction Script). O DIS organiza informações para facilitar construções do modelo i*. Documenta metas, atores, precondições, eventos, ações e pós-condições para que a atividade aconteça. Também são apresentadas regras para construção do modelo i* operacional a partir das informações coletadas no DIS (Tabela 6-2). O PRIM usa diagramas i* da versão original do i*. Este exemplo usa a versão do i* proposta pelo Tropos'04. As diferenças, neste caso se refletem a notação utilizada. Portanto foram feitas algumas adaptações as regras apenas mudando os termos do i* original: Diagrama SD para Diagrama de Ator; Diagrama SR para Diagrama de Metas; Tarefa para planos.

Tabela 6-2 – Regras para construção do modelo i* propostas pelo PRIM

Código	*Descrição*
R1	Cada atividade do ator é definida como um Plano do Diagrama de Ator. Este Plano é relacionado à meta principal do ator com uma ligação meio-fim.
R2	Cada Plano é decomposta nas ações do DIS, usando a ligação de decomposição.
R3	Se houver uma interação com troca de recursos, existe uma dependência entre os atores. O recurso é o *dependum*. O ator destinatário é o ator *depender* e o iniciador é o *dependee*.
R4	Um recurso consumido por uma ação deve ter sido produzido por outra. Assim este recurso é gerado por um Plano em uma dependência.
R5	O evento de início é modelado como dependência de Metas. O ator que inicia o Plano é o depender e o ator que toma o evento iniciante para si é o dependee. Precondições e pós-condições são adicionadas como metas na decomposição do Plano.

O exemplo apresentado neste capítulo segue as diretrizes de coleta de informações para elicitação dos requisitos (definições de trabalho RI1 e RF1) e as regras de construção do modelo i* para a análise e especificação dos requisitos (definições de trabalho RI2). A seguir são apresentados os resultados de execução de cada atividade realizada.

RI1-1: Identificação das partes interessadas e suas intenções: Os requisitos iniciais foram elicitados com o idealizador do DBSitter-AS. Estas informações foram documentadas na dissertação de mestrado de definição do sistema DBSitter-AS [Maciel, 2007] e artigos relacionados [Silva, M.J et al., 2007]. Estas duas fontes são usadas para iniciar a definição do sistema. A primeira etapa é identificar as partes interessadas, que chamaremos de atores a partir deste ponto. São identificados na documentação do sistema os atores Administrador de BD e Clientes. Em seguida é necessário identificar as intenções, que também estão relacionadas na documentação. Os atores e intenções são catalogados na Lista de partes interessadas que faz parte do documento de requisitos. A Tabela 6-3 relaciona estas intenções.

Tabela 6-3 – Lista de partes interessadas

Ator	***Intenções***
Administrador de BD	Realizar atividades de administração de bancos de dados; Liberar-se de tarefas rotineiras; Ter auxílio em planejamento de correção de falhas; Ter alertas em tempo real; Ter mais tempo para se dedicar a outras tarefas.
Cliente	Ter uma administração de dados mais eficiente a um custo menor; Melhorar alocação de recursos humanos na empresa; Ter um mecanismo de prevenção de falhas automático; Ter maior continuidade de serviços.

RI1-2: Identificar relacionamentos entre as partes interessadas: A primeira etapa desta atividade é selecionar um ator para identificar os relacionamentos. O ator Administrador de BD é o primeiro selecionado. Além das intenções (Tabela 6-3) são identificadas as suas atividades técnicas relacionadas na Tabela 6-4. Cada uma destas atividades pode ser analisada para identificar possíveis relacionamentos. Por exemplo, a atividade Fornecer suporte técnico aos bancos de dados existentes é uma atividade realizada para atender os Clientes. A atividade Supervisionar a instalação de novos bancos de dados é realizada com interações com os clientes e a equipe de suporte técnico. Assim, identificamos outros atores com a análise das atividades: equipe de suporte técnico. As atividades do ator Cliente estão associadas ao seu

relacionamento com o Administrador de BD e incluem a solicitação, avaliação e pagamento dos serviços.

Tabela 6-4 – Atividades dos atores

Ator	*Atividades*	*Relacionamentos*
Administrador de BD	Fornecer suporte técnico aos bancos de dados existentes	Clientes
	Personalizar bancos de dados comerciais para necessidades específicas	Clientes
	Solucionar problemas para atender às necessidades dos clientes	Clientes
	Programar bancos de dados para uma ampla variedade de aplicações	Clientes
	Supervisionar a instalação de novos bancos de dados	Equipe de suporte técnico, Clientes
	Treinar a equipe das empresas clientes no uso de bancos de dados novos e existentes	Clientes
Clientes	Solicitar Serviços	Administrador de BD
	Avaliar Serviços	Administrador de BD
	Realizar Pagamentos	Administrador de BD

RI1-4: Identificar atividades detalhadas das partes interessadas: Esta atividade é realizada com apoio dos cenários (DIS) propostos pelo PRIM, que contém informações sobre as atividades dos atores e seus relacionamentos. Primeiramente, são identificadas as atividades dos atores e depois para cada atividade é criado um DIS (Tabela 6-5). As ações para realizar esta atividade são listadas. Para cada ação é identificado o ator que inicia a ação, uma curta descrição da ação e os recursos consumidos ou produzidos pela ação. Se a ação requer uma interação com outro ator, o destinatário e os recursos providos também são identificados. O DBSitter-AS atua diretamente na atividade "Solução de problemas para atender as necessidades dos clientes", portanto esta é a primeira atividade especificada no DIS (Tabela 6-5). O ator Administrador de BD realiza as ações *Analisar solicitação, Investigar estado no SGBD, Investigar estado no ambiente de redes, Investigar estado no sistema operacional, Realizar diagnóstico, Pesquisar solução, Executar solução*, *Notificar solução e Cobrar serviço executado*. Nas ações de Investigar estado no SGBD na rede ou no Sistema operacional, ele interage com atores sistemas que são o SGBD e o Sistema Operacional, usando as Solicitações de atendimento e produzindo Relatórios de análise. Na ação de *Enviar o diagnóstico*, ele interage com o Cliente enviando o diagnóstico a este ator. Ele também interage com o Cliente em outras duas ações *Notificar solução e Cobrar serviço executado* enviando a este os relatórios necessários. O cliente inicia uma única ação que é *Autorizar o pagamento*. Esta atividade é iniciada com a pré-condição de ter ocorrido uma *Solicitação de atendimento* e tem como pós-condição a *Solicitação de atendimento concluída*.

Tabela 6-5 – DIS para a atividade Solucionar problemas para atender às necessidades dos clientes

	DIS: Solucionar problemas para atender às necessidades dos clientes					
Fonte	Atividades de elicitação de requisitos iniciais					
Atores	Administrador de BD, Cliente, SGBD, Sistema Operacional					
Precondições	Solicitação de atendimento recebida					
Evento de início	--					
Ações	**Iniciador**	**Ação**	**Recursos consumidos**	**Recursos produzidos**	**Destinatário**	**Recursos fornecidos**
	Administrador de BD	Analisar solicitação	Solicitação de atendimento			
	Administrador de BD	Investigar estado no SGBD	Solicitação de atendimento	Relatório de análise do SGBD	SGBD	
	Administrador de BD	Investigar estado no ambiente de redes e sistema operacional	Solicitação de atendimento	Relatório de análise de redes e SO	Sistema Operacional	
	Administrador de BD	Realizar diagnóstico	Relatório de análise		Cliente	Diagnóstico
	Administrador de BD	Executar solução	Diagnóstico	Solução		
	Administrador de BD	Notificar solução			Cliente	Relatório solução do problema
	Administrador de BD	Cobrar serviço executado			Cliente	
	Cliente	Autorizar pagamento	Relatório solução do problema		Administrador de BD	Pagamento
Pós-condições	Solicitação de atendimento concluída					

RI2-3: Analisar as dependências dos atores: Nesta atividade o Diagrama de ator é usado na análise. A primeira etapa é identificar os atores e suas intenções relacionadas na *Lista de partes interessadas* (Tabela 6-3). Estas intenções podem ser analisadas como metas ou *metas-soft* no Diagrama de ator. Metas são objetivos concretos que podem ser atingidos pelo ator, enquanto as metas-soft são atributos de qualidade desejados pelos atores. A intenção da Lista de Partes Interessadas *Realizar atividades de administração de dados* é uma meta do Administrador de BD que foi reescrita como *Atividades de Administração de BD realizadas* no diagrama de atores (Figura 6-1). Pela análise foi identificado que esta é uma meta inicial do ator Cliente, mas este não pode obtê-la sozinho, dependendo do ator Administrador de BD para alcançá-la. Por isto, esta meta é modelada no diagrama de ator como uma dependência entre estes dois atores. A intenção *Ter uma administração de dados mais eficiente a um custo menor* da Lista de partes interessadas está identificada como sendo do ator *Cliente*, mas é modelada no diagrama de atores como a dependência de meta-soft *Administração de dados eficientes*. Esta é uma meta-soft, pois está associada a eficiência que é um atributo de

qualidade. Embora seja uma intenção do *Cliente*, este não pode atingi-la sozinho, dependendo do *Administrador de BD* para alcançá-la, por isto foi modelada como uma dependência. O mesmo raciocínio é usado para analisar a intenção *Ter mais tempo para se dedicar a outras tarefas*. Esta é modelada como a dependência de meta-soft *Tempo liberado para outras tarefas.* As intenções *Liberar-se de tarefas rotineiras, Ter auxílio em planejamento de correção de falhas e ter alerta em tempo real* são analisadas respectivamente como as metas do Administrador de BD: *Tarefas rotineiras delegadas, Planejamento de correção de falhas automatizadas* e *Monitoramento automatizado.*As intenções do ator Cliente *Melhorar alocação de recursos humanos na empresa* e *Ter maior continuidade de serviços* são analisadas como as metas-soft *Continuidade de serviços garantidas* e *Alocação de RH melhorada.* E a intenção *Ter um mecanismo de prevenção de falhas automático* é analisada como a meta *Prevenção de falhas automatizada.*

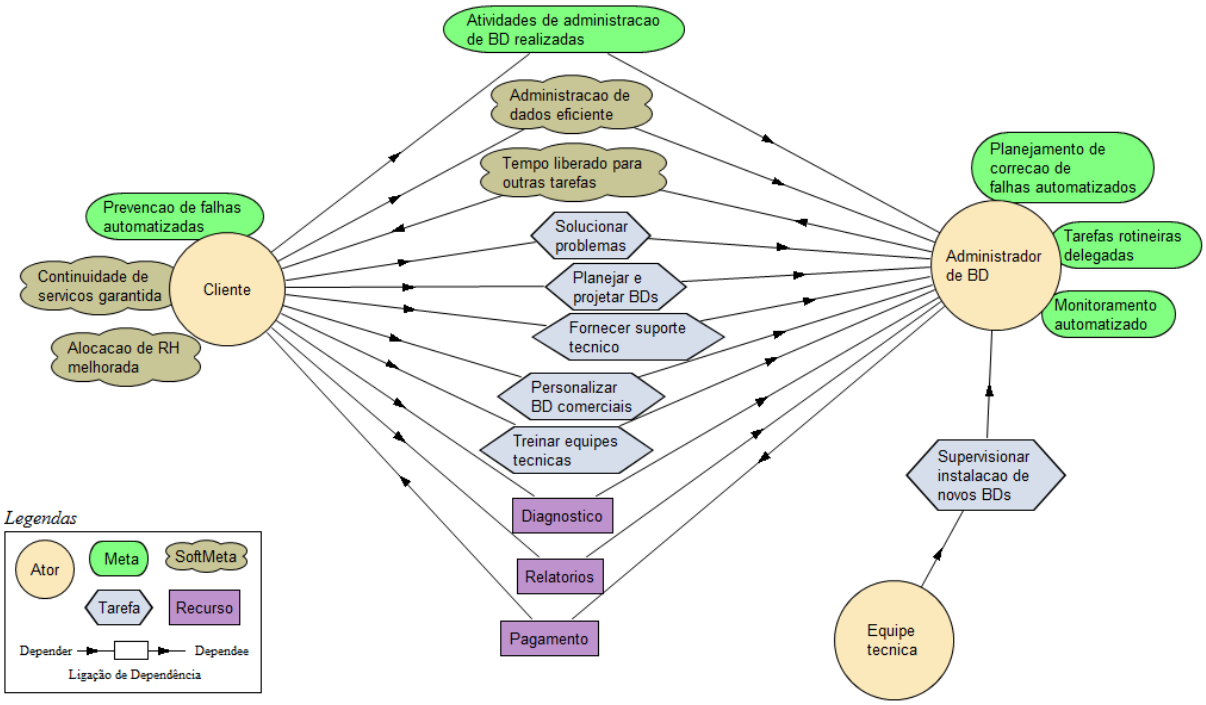


Figura 6-1 – Diagrama de ator – Requisitos Iniciais

Para identificar outros elementos da análise aplicam-se as regras R1, R3 e R5 do PRIM. Segundo a regra R1, as atividades do ator Administrador de BD (listadas na Tabela 6-4) são modeladas como dependências de planos. A atividade *Supervisionar a instalação de novos Bancos de dados* é modelada como a dependência de plano *Supervisionar a instalação de novos Bancos de dados* entre o ator Equipe Técnica e o ator Administrador de BD. As demais atividades são modeladas como dependências de planos entre o cliente e o Administrador de

BD. A regra R3 define as dependências de Recursos. Os recursos *Relatórios de solução de problemas, Diagnóstico e Pagamento*, que foram definidos no DIS (Tabela 6-5), nas interações entre cliente e Administrador de BD, se tornam dependências entre estes atores. Outras dependências entre o ator Cliente e o ator Administrador de BD não foram modeladas, para simplificar o diagrama de ator (Figura 6-1) neste exemplo.

RI2-5: Realizar análise orientada a metas: Nesta atividade inicia-se a construção do diagrama de metas que representa a lógica de execução de tarefas dos atores para atingir suas metas. O primeiro passo é selecionar um ator e depois refinar as metas através de decomposições booleanas. Neste exemplo, foi selecionado o ator Administrador de BD. Na atividade anterior, foram identificadas quatro metas e uma meta-soft para este ator. Estas metas são analisadas para verificar se é necessária a decomposição. A meta *Atividades de administração de BD realizadas* é a principal meta do ator. Esta meta pode ser decomposta em *Resolução de problemas rotineiros automatizada* e *Atividades realizadas de forma convencional*. A primeira meta ainda pode ser refinada em *Monitoramento Automatizado* e *Planejamento de correção de falhas automatizado*. Já a meta do ator *Tarefas rotineiras delegadas* não é decomposta nesta análise. Esta análise é modelada no diagrama de metas do ator Administrador de BD (Figura 6-2).

RI2-6: Realizar análise meio-fim: Nesta atividade, são analisadas as tarefas dos atores e identificado se estas estão associadas a alguma meta. Também são identificadas novas tarefas e recursos. As regras do PRIM auxiliam nesta atividade para coletar estas informações do DIS. Pela regra R1, as atividades, que são tarefas do modelo i*, estão relacionadas a uma meta principal. Realizando esta análise no ator *Administrador de BD*, a meta principal é *Atividades de administração de BD realizadas*. Esta meta está decomposta em duas submetas, que são a *Resolução de problemas rotineiros automatizada* e *Atividades realizadas de forma convencional*. O ator realiza atividades operacionais para atingir a segunda meta. Estas atividades são modeladas como Tarefas do modelo i*. Portanto, as tarefas *Personalizar BD comerciais, Treinar equipes técnicas, Fornecer suporte técnico, Solucionar problemas, Planejar e projetar BDs* e *Supervisionar instalação de novos BDs* são meios para atingir a submeta *Atividades Realizadas de forma convencional*. Pela regra R2, a tarefa *Solucionar problemas* é decomposta nas ações identificadas no DIS. Pela regra R4 o recurso *Relatório* está relacionado às tarefas *Realizar diagnóstico* e *Autorizar pagamento*. Pela regra R5, identificam-se as metas *Solicitação de atendimento requerida* advinda da precondição e *Solicitação concluída* advinda da pós-condição (Figura 6-2).

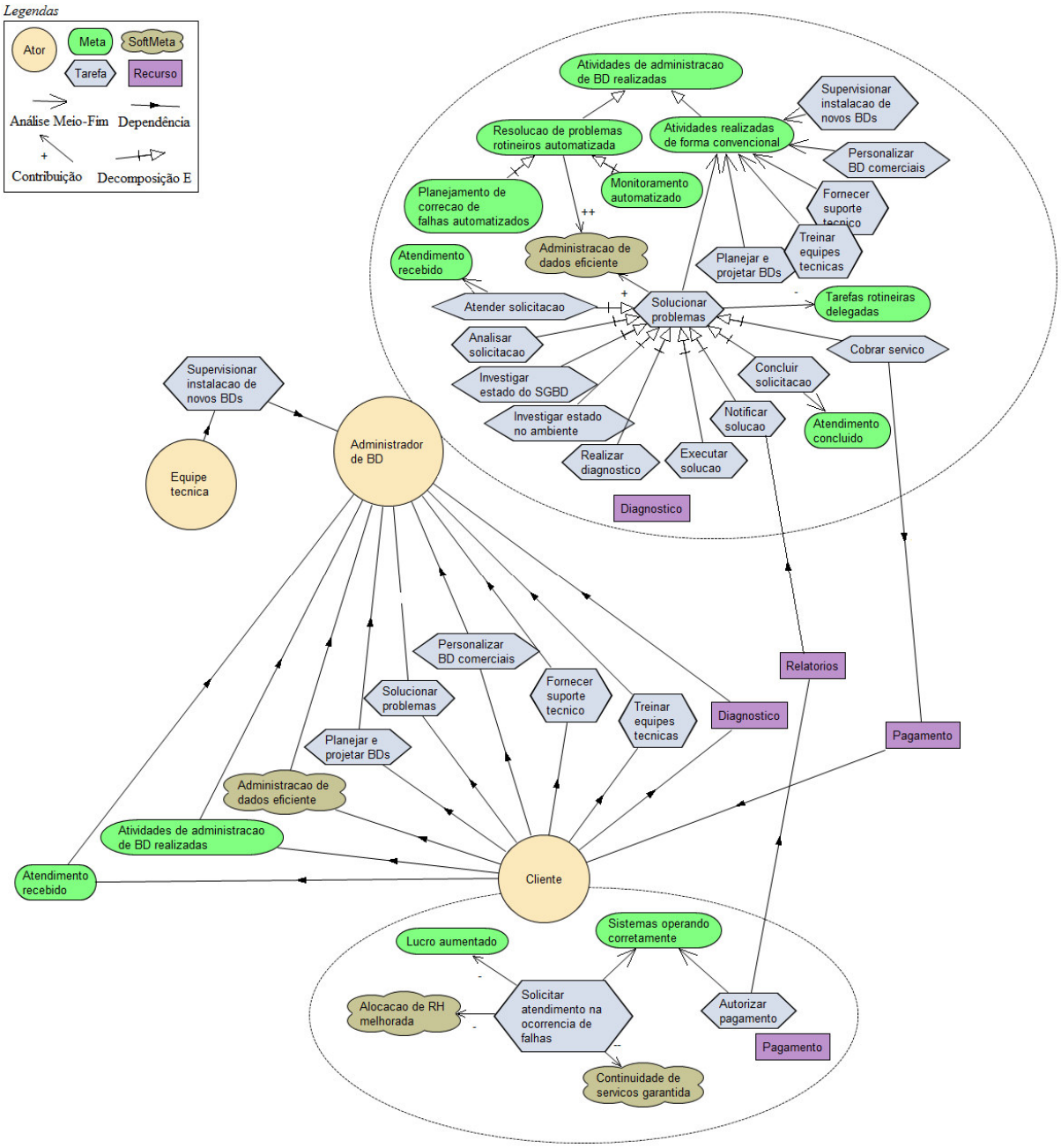


Figura 6-2 – Diagrama de Metas – Requisitos Iniciais

RI2-6: Realizar análise das contribuições: Nesta atividade, cada plano é analisado para identificar se estes contribuem para a obtenção das metas do ator. A seqüência de sub-tarefas para a tarefa *Solucionar problemas* contribui positivamente para a meta-soft *Administração de dados eficiente*, mas contribui negativamente para a meta *Tarefas rotineiras delegadas*. A obtenção da meta *Resolução de problemas rotineiros automatizada* contribui fortemente para obtenção da meta-soft *Administração de dados eficiente* (Figura 6-2).

RI3-8: Validar modelos propostos: A validação ode ocorrer pela apresentação dos modelos às partes interessadas e analistas de domínio. Por exemplo, os analistas podem checar o

modelo i* pelas diretrizes de checagem fornecida pelo PRIM, que confirma se todas as informações coletadas no DIS foram representadas no modelo i* (Tabela 6-6). Se os modelos forem aprovados, a análise prossegue, senão os ajustes são executados antes dos próximos passos.

Tabela 6-6 – Checagem da transformação do DIS no modelo i*

	Checagem	***Avaliação***
1	Toda atividade no DIS é modelada como uma tarefa.	✓
2	Cada meta principal de um ator é decomposta pela análise meio-fim nas tarefas onde o ator realiza as ações.	✓
3	Cada ação é modelada como uma subtarefa que é decomposta das tarefas pela decomposição-E no modelo SR do ator iniciante.	✓
4	Cada recurso envolvido nas interações se torna uma dependência de recurso, onde o depender é o ator que produz o recurso e o dependee o ator que consome o recurso.	✓
5	Precondições e pós-condição são modeladas como metas no diagrama SR e evento de inicio como dependência de metas.	✓
6	Cada tarefa é meio de uma análise meio-fim para uma meta (que é o fim), que pode ser refinada em outras metas.	✓
7	Algumas restrições não funcionais são estabelecidas sobre os recursos e as tarefas. Estas restrições são as metas-soft.	✓

RF1-1: Elicitar dependências dos atores e ator-sistema: Nesta atividade, a análise dos atores sociais onde o sistema será inserido já foi realizada e inicia o processo de levantamento dos requisitos do sistema. Nesta iteração, serão definidos apenas os requisitos que definem o escopo do sistema. Isto é realizado, por exemplo, através de entrevistas com os partes interessadas (identificadas nos requisitos iniciais) e análise da documentação do sistema DBSitter-AS. O primeiro passo é analisar as necessidades das partes interessadas. A maior necessidade das partes interessadas está na resolução automática de problemas rotineiros em Banco de dados para reduzir as dependências do cliente em relação ao Administrador de BD e para liberar o Administrador de BD para outras tarefas. O próximo passo é identificar sistemas que interagem com o DBSitter-AS. As interações serão com o SGBD e com os serviços de sistema operacional. Após isto, identificam-se ações para o sistema que atendam as metas das partes interessadas. A Tabela 6-7 lista estas informações em um cenário DIS. Os atributos de qualidade estão associados a como o resultado das ações devem atender aos envolvidos com o sistema. No levantamento realizado, as partes interessadas esperam que o sistema seja confiável, seguro, fácil de manter, flexível e fácil de interagir. Estes atributos são documentados como Pós-condição no cenário DIS.

Tabela 6-7 – DIS para delimitar ações do sistema DBSitter-AS

	DIS: Resolução automática de problemas rotineiros em Banco de dados					
Fonte	Entrevistas com partes interessadas, documentação do sistema anterior.					
Atores	Administrador de BD, DBSitter-AS, SGBD, SO, Servidor de e-mail Base de conhecimento					
Precondições	Resolução autônoma de problemas rotineiros					
Evento início						
Ações	**Iniciador da ação**	**Ação**	**Recursos consumidos**	**Recursos produzidos**	**Destinatário da ação**	**Recursos fornecidos**
	DBSitter-AS	Executar serviços para resolução e prevenção de falhas			SGBD, SO	
	DBSitter-AS	Notificar ação executada			Servidor de E-mail	Notificações
	DBSitter-AS	Manter base de conhecimento			Base de conhecimento	Informações
Pós-condições	Problemas resolvidos de forma autônoma As ações realizadas devem ser executadas com Confiabilidade e Segurança. O sistema deve ser fácil de manter, flexível e de fácil interação com os usuários.					

RF2-2: Analisar as dependências dos atores: Aplicando as regras do PRIM, a partir das informações coletadas no cenário DIS, identifica-se que o DBSitter-AS se relaciona com os atores *Administrador de BD, SGBD, Sistema Operacional (SO), Servidor de e-mail* e *Base de conhecimento*. A precondição para que este sistema exista é a necessidade de *Resolução de problemas rotineiros de forma autônoma*, que é analisada como uma meta do Administrador de BD no diagrama de ator para o sistema DBSitter-AS (Figura 6-3). Portanto, esta passa a ser uma dependência que este ator tem do sistema. As ações listadas no DIS (Tabela 6-7) são modeladas como dependências de planos no diagrama de ator. Assim são modeladas as dependências de planos entre o DBSitter-AS e outros atores: *Executar resolução ou prevenção de falhas no SO* com o ator *Sistema operacional; Executar resolução ou prevenção de falhas* com o ator *SGBD; Notificar ação realizada* com o ator *Servidor de e-mail* e *Manter informações e conhecimento* com o ator *Base de conhecimento*. O recurso *Notificações* se torna uma dependência de recurso que o ator Servidor de e-mail tem do DBSitter-AS e o recurso *Informações* é uma dependência que o ator *Base de conhecimento* tem do DBSitter-AS. (Figura 6-3).

RF3-7: Validar os modelos propostos: A validação ocorre, por exemplo, pela apresentação dos modelos às partes interessadas e analistas de domínio. A mesma avaliação do modelo i* realizada para os modelos das atividades de requisitos iniciais pode ser realizada aqui (Tabela 6-6). Se os modelos forem aprovados, a análise prossegue, senão os ajustes são executados antes dos próximos passos.

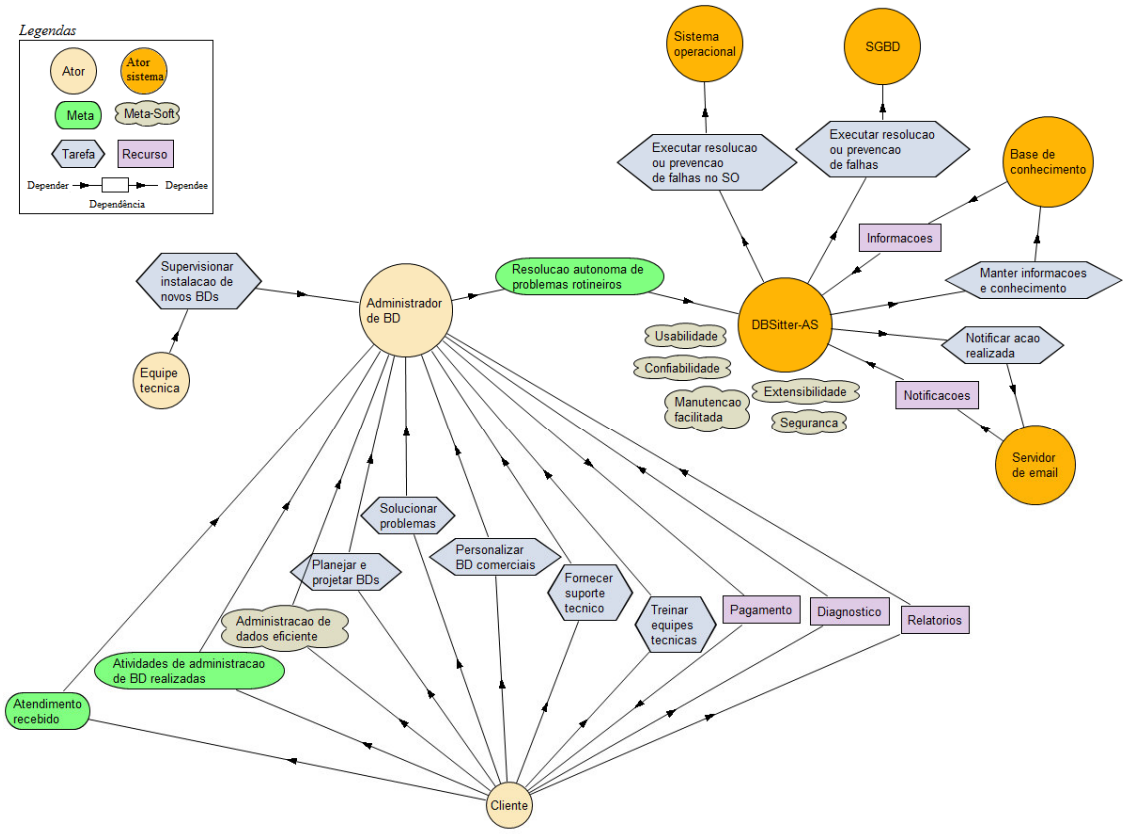


Figura 6-3 – Diagrama de atores – Requisitos finais

Este subconjunto de atividades define a primeira visão do sistema que será construído. Os produtos de trabalho *Diagrama de atores e Diagrama de metas* são criados durante a análise dos requisitos iniciais e finais. O *Modelo de requisitos iniciais*, neste caso, contém a lista das partes interessadas, as atividades de todos os envolvidos, os cenários do DIS para estas atividades e os diagramas i*. Neste exemplo foram apresentadas apenas as atividades do ator Administrador de BD e o DIS para a atividade *Solucionar problemas para atender às necessidades dos clientes*. Mas em um estudo de caso completo é necessário à criação de cenários para todas as atividades das partes interessadas. O Modelo de requisitos finais contém nesta versão preliminar, o DIS para o ator sistema e os diagramas gerados. Sendo assim a iteração *1 – Definição de sistema* é concluída. Esta teve como objetivo definir o escopo dos requisitos que serão desenvolvidos. Pode-se agora prosseguir para a próxima iteração.

6.3.2 Iteração 2 – Análise dos requisitos do sistema

Nesta iteração, as metas do sistema, que representam os requisitos de usuário, são analisadas para definir o primeiro conjunto de requisitos. Estes serão projetados e implementados nas próximas iterações. É importante que as partes interessadas definam qual(is) meta(s) tem

maior prioridade para definir os requisitos desta iteração. No plano de iterações (Tabela 6-1) gerado para este exemplo foram definidas três definições de trabalho e cinco atividades. As definições de trabalho *RF1- Levantamento dos requisitos do sistema* e *RF2- Análise dos requisitos do sistema e RF3-Validação dos modelos propostos*. A atividade 3 da primeira definição de trabalho é executada para identificar o raciocínio lógico para cada meta do sistema. Na segunda definição de trabalho são realizadas as atividades 4,5 e 6 para analisar e modelar o raciocínio estratégico do sistema que atenda as metas selecionadas. Por último os modelos são validados na atividade 7.

RF2-3 Identificar lógica p/ obter as intenções do ator-sistema: Esta atividade inicia com o engenheiro de requisitos entrevistando as partes interessadas e especialistas em agentes para definir as atividades que o sistema deve executar. Ele seleciona um subconjunto de metas para o detalhamento e então valida e refina estas metas. A meta identificada para o DBSitter-AS foi *Resolução autônoma de problemas rotineiros*. Esta meta pode ser refinada em *Falhas prevenidas, Falhas resolvidas e BD Monitorado*. Outras metas que são identificadas podem ser *Ação realizada notificada* e *Informações e conhecimento mantido* que são executadas pelas tarefas delegadas aos atores *Servidor de email* e *Base de conhecimento* respectivamente. Falhas podem ser classificadas como *Falhas no SGBD* e *Falhas de infra-estrutura*, isto é em processos do sistema operacional ou em serviços de conectividade. As falhas em SGBD podem ser subdivididas em três categorias: *falhas em estruturas de controle, falhas em estruturas físicas e falhas em estruturas lógicas*. A resolução de falhas para todas as classificações seguem um conjunto de ações semelhantes, diferindo apenas no tipo de problema realizado e nas interações com outros atores.

O DIS pode ser usado para documentar as atividades e ações para atingir as metas. Será usado para demonstração das informações levantadas e documentadas no DIS as atividades para alcançar as metas *Resolução de falhas em objetos lógico e Notificação de ações realizadas.* A Tabela 6-8 apresenta o DIS com o detalhamento da *Resolução de falhas em objetos lógico*. E a Tabela 6-9 apresenta o DIS com o detalhamento de *Notificação de ações realizadas*.

São identificadas nove ações que são executadas pelo DBSitter-AS para realizar a atividade descrita no DIS *Resolver falhas em objetos lógicos* (Tabela 6-8). Observa-se que a ação *Identificar natureza do problema* pode ser classificada em seis tipos de problemas diferentes. Ocorrem duas interações com ações de retorno entre O DBSitter-As e outros atores. A primeira é a ação *Consultar tipos de soluções* na Base de Conhecimento fornecendo

o recurso *Tipo de Falha.* A base de conhecimento por sua vez retorna com a execução da ação *Fornecer tipos de solução para o problema* repassando o recurso *Tipos de soluções*. A outra ação é *Solicitar execução de serviços para correção de Falhas em objetos lógicos (FOL)* que é destinada ao SGBD. Este retorna executando a ação *Executar serviços.* As outras ações são todas executadas pelo DBSitter-AS.

Tabela 6-8 – DIS para Resolver falhas em objetos lógico

	DIS: Resolver falhas em objetos lógicos					
Fonte	Elicitação de Requisitos					
Atores	DBSitter-AS, Base de conhecimento, SGBD, BD, Servidor de e-mail					
Precondições	Falhas de objetos lógicos detectadas					
Evento de início						
Ações	**Iniciador da ação**	**Ação**	**Recursos consumidos**	**Recursos produzidos**	**Destinatário da ação**	**Recursos fornecidos**
	DBSitter-AS	Detectar falhas de objetos lógicos (OL)			BD	
	DBSitter-AS	Identificar natureza do problema: Resolver problema de desempenho Redimensionar tabelas, índices e áreas lógicas Consertar tabelas e índices corrompidos Redimensionar partições de áreas de armazenamento Consertar fragmentação de tabelas e índices Alertar violação de segurança				
	DBSitter-AS	Consultar tipos de solução			Base de conhecimento	Tipo de falha
	Base de conhecimento	Fornecer tipos de solução para problema			DBSitter-AS	Tipos de solução
	DBSitter-AS	Decidir tipo de solução a aplicar	Tipos de solução	Solução selecionada		
	DBSitter-AS	Decidir tipo de ação: Sugerir ou executar a correção				
	DBSitter-AS	Solicitar execução de serviços para correção de falhas em objetos lógicos (FOL)			SGBD	Solução
	SGBD	Executar serviços				
	DBSitter-AS	Identificar estado da ação de correção			BD	
	DBSitter-AS	Registrar solução			Base de conhecimento	Solução
	DBSitter-AS	Notificar Ação			Servidor de e-mail	Solução
Pós-condições	Falhas de objetos lógicos resolvidas					

O DIS *Notificar ação executada* (Tabela 6-9) relaciona quatro ações realizadas pelo DBSitter-AS. A ação *Identificar tipo de notificação* pode ser classificada em três tipos distintos: *e-mail*, cujo destinatário é o *Servidor de e-mail*; *alerta em interface do usuário*, cujo destinatário é o *sistema operacional*; *relatório*, cujo destinatário é a *impressora*.

Tabela 6-9 – DIS para Notificar ação executada

	DIS: Notificar ação executada					
Fonte	Elicitação de Requisitos					
Atores	DBSitter-AS					
Precondições	Execução de uma ação					
Evento de início						
Ações	**Iniciador da ação**	**Ação**	**Recursos consumidos**	**Recursos produzidos**	**Destinatário da ação**	**Recursos fornecidos**
	DBSitter-AS	Identificar ação executada	Ações			
	DBSitter-AS	Identificar o destinatário				
	DBSitter-AS	Identificar tipo de notificação:				
		E-mail			Servidor de e-mail	Mensagem
		Alerta em interface do usuário			Sistema operacional	Alerta
		Relatório			Impressora	Relatório
	DBSitter-AS	Notificar Ação				
Pós-condições	Notificação realizada					

RF2-4 Realizar Análise orientada a Metas: Nesta atividade as metas levantadas com as partes interessadas são modeladas para compor o modelo de metas do DBSitter-AS (Figura 6-4). A meta principal do sistema é *Resolução autônoma de problemas rotineiros* que é decomposta em *BD Monitorado, Falhas prevenidas, Falhas resolvidas, Ação notificada* e *Informações e conhecimento mantidos*. A meta *Falhas resolvidas* pode ainda ser mais refinada em *Falhas de SGBD resolvidas* e *Falhas de infra-estrutura resolvidas*. Por sua vez, a meta *Falhas de SGBD resolvidas* pode ainda ser refinada em *Falhas em objetos físicos resolvidas, Falhas em objetos lógicos resolvidas* e *Falhas em estruturas de controle resolvidas*. Já a meta *Falhas de infra-estrutura resolvidas* pode ser refinada em *Falhas no SO resolvidas* e *Falhas de conectividade resolvidas.* Estes refinamentos são realizados por decomposição-E. As metas-soft também são analisadas e refinadas. *Disponibilidade* e *Confidencialidade* são refinamentos de *Segurança*. O Acesso autorizado gera *Confidencialidade*. Enquanto *Confiabilidade* é um refinamento de *disponibilidade*. Por sua vez, a *Usabilidade* pode ser refinada em *Operabilidade* e *Flexibilidade*. A manutenção facilitada significa que o sistema deve ser facilmente extensível e *Extensibilidade* é um refinamento de *Adaptabilidade*.

RF2-5: Realizar a análise meio-fim: Nesta atividade as metas identificadas para o sistema são detalhadas através de análise meio-fim e decomposições E/OU sendo documentadas no diagrama de metas (Figura 6-4). As regras do PRIM para conversão do DIS para o diagrama

i* são usadas para identificar as tarefas e recursos durante a análise. A atividade especificada no DIS *Resolver Falhas em objetos lógicos (OL)* é um meio para atingir a meta *Falhas em objetos lógicos resolvidos*. Esta atividade é executada através de ações que são analisadas como sub-tarefas da tarefa principal. As sub-tarefas são as seguintes: *Detectar falhas de OL, Identificar natureza do problema, Consultar tipos de solução, Decidir tipo de solução a aplicar, Decidir tipo de ação, Solicitar execução de serviços para correção de Falhas de OL, Identificar estado da ação de correção, Registrar solução* e *Notificar Ação.* Fazendo uma análise mais refinada das ações, é identificado que *Decidir o tipo de ação* é decomposto em *Sugerir ação* ou *Executar ação*. A execução da ação é realizada por *Solicitação de execução de serviços para correção de Falhas de OL.* A natureza do problema identificado relaciona os tipos de resolução de serviços. Desta forma a sub-tarefa *Solicitação de execução de serviços para correção de Falhas de OL* é decomposta em: *Redimensionar partições de áreas de armazenamento; Redimensionar tabelas, índices e áreas lógicas; Consertar tabelas e índices corrompidos; Consertar fragmentação de tabelas e índices; Resolver problema de desempenho ou Alertar violação de segurança.*

RF2-6: Realizar análise das contribuições: Nesta atividade as tarefas são analisadas para identificar suas contribuições às metas-soft. *Notificar Solução e Registrar solução* contribuem para satisfação da meta-soft *Manutenção facilitada*, pois a análise das soluções possibilita aos usuários identificar informações para executar manutenções e ajustes no sistema. *Registrar solução* também contribui para a meta-soft *Aprendizagem*, pois gera uma base de dados com instruções de como resolver problemas. As subtarefas *Decidir tipo de ação* e *Decidir tipo de solução a aplicar* contribuem positivamente para *Autonomia*, pois o sistema deve ter a capacidade de tomar decisões. *Sugerir ação* contribui positivamente para a *Confiabilidade*, pois, se o sistema identifica uma nova solução, é mais confiável que a primeira execução desta solução seja autorizada por um ator humano. Porém esta ação contribui negativamente para a *Autonomia* do sistema. Esta análise também está modelada no diagrama de metas do DBSitter-AS (Figura 6-4 – Diagrama de Metas – Requisitos Finais).

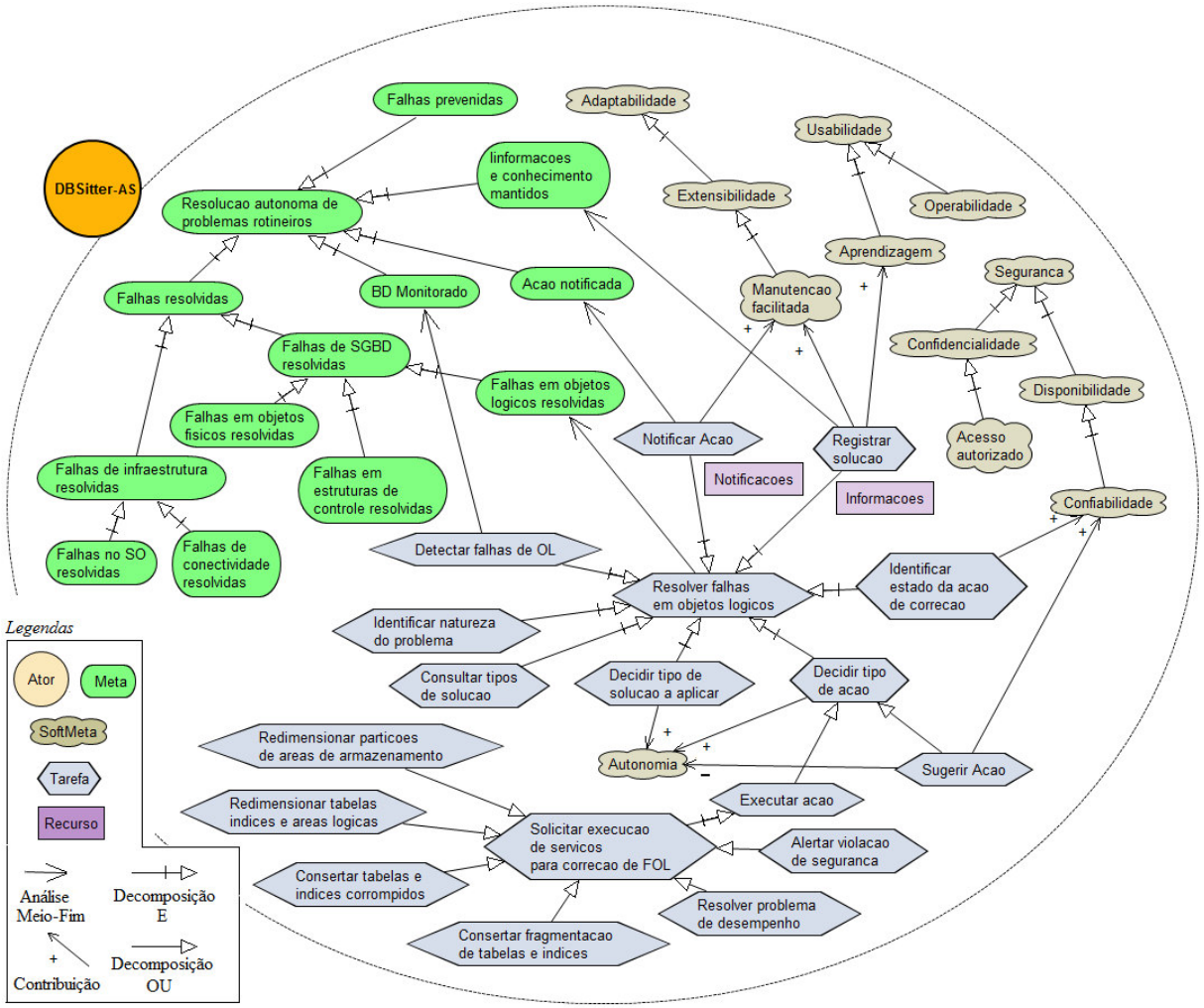


Figura 6-4 – Diagrama de Metas – Requisitos Finais

RF3-7: Validar os modelos propostos: A validação ocorre pela apresentação dos modelos às partes interessadas. A mesma avaliação do modelo i* realizada para os modelos das atividades de requisitos iniciais pode ser realizada aqui (Tabela 6-6). Se os modelos forem aprovados, a análise prossegue, senão os ajustes são executados antes dos próximos passos.

Uma vez que um conjunto de requisitos tenha sido analisado e detalhado, o projeto da arquitetura do sistema é iniciado. A seção seguinte exemplifica a próxima iteração, que trata da elaboração do projeto arquitetural do sistema DBSitter-AS.

6.3.3 Iteração 3 – Projeto da arquitetura global do sistema

Nesta iteração, é definida a arquitetura global do sistema multiagentes. A pré-condição para iniciar esta iteração é que o escopo do sistema esteja definido, o que já ocorreu nas iterações de definição e análise dos requisitos. A partir dos produtos de trabalho gerados nas atividades de requisitos iniciais e finais, é possível gerar uma configuração organizacional, onde possam

ser inseridos os agentes do sistema multiagentes. No plano de iterações (Tabela 6-1) gerado para este exemplo foi definido o fluxo de trabalho a adotar. Todas as cinco definições de trabalho da disciplina Projeto arquitetural são realizadas para compor a primeira configuração arquitetural que atendem os requisitos analisados, totalizando nove atividades.

Primeiramente são realizadas as atividades 1, 2 e 3 da definição de trabalho *PA1-Especificar o modelo organizacional,* com o objetivo de definir os papéis que compõe o modelo organizacional derivados dos requisitos funcionais do sistema (metas e planos). Em seguida, são realizadas as atividades 4 e 5 da definição de trabalho *PA2-Selecionar um estilo arquitetural.* Estas atividades definem qual estilo arquitetural será utilizado pelo sistema a partir da análise dos requisitos não funcionais (metas-soft). Após isto, são realizadas as atividades 6 e 7 da definição de trabalho *PA3 – Definir o modelo de atribuição* que faz uma correlação entre os papéis do modelo organizacional e os componentes do estilo arquitetural. Então, através da atividade 8 da *PA4 – Configurar a arquitetura* aqueles dois modelos são mapeados em um único modelo arquitetural, finalizando o projeto da arquitetura. Por último, os modelos são validados durante a execução da atividade 9 da definição de trabalho *PA5-Validar os modelos propostos.* A seguir, estas atividades são detalhadas.

PA1.1 Identificar papéis: Os papéis representam a divisão do trabalho entre os membros da organização. O grupo organizacional é composto dos papéis e de suas interações. Os papéis são capturados em um domínio específico e podem ser identificados das metas presentes no Modelo de requisitos. As atividades de resolução de falhas são da mesma natureza, diferenciando-se apenas o tipo das falhas corrigidas. Desta forma identifica-se um papel *Administrador de resolução de falhas* que é responsável pelas tarefas de administração das resoluções de falhas e papéis responsáveis por correções específicas, tais como o *Administrador de falhas em SGBD* e o *Administrador de falhas de infra-estrutura*. Por ser um agente de software, as tarefas de monitorar e agir devem se encontrar no mesmo agente, portanto a meta *falhas monitoradas* também faz parte do papel *Administrador de resolução de falhas*. Outros papéis identificados a partir das metas do sistema são: o *Notificador de ações* que é responsável por comunicar ações aos atores externos à organização; o *Administrador de conhecimento* é responsável por recuperar e manter o conhecimento sobre o ambiente e as ações executadas; e o *Preventor de falhas* é responsável por prevenir a ocorrência de falhas previsíveis. Estes papéis são especificados identificando-se seus objetivos, responsabilidades, habilidades, colaboradores e normas organizacionais (Tabela 6-10).

Tabela 6-10 – Especificação dos papéis

Administrador de resolução de falhas	
Objetivos:	Falhas resolvidas
Responsabilidades:	Administrar falhas em SGBD e Infra-estrutura
Colaboradores:	Notificador de Falhas, Administrador de conhecimento, Administrador de falhas em SGBD, Administrador de falhas de Infra-estrutura
Habilidades:	Conhecimento sobre os planos de resolução de falhas
Normas:	Não desativar o BD quando este estiver em uso; Não executar operações que comprometa a segurança das informações; As operações devem ser transparente ao ator humano Administrador de BD

Notificador de ações	
Objetivos:	Ações notificadas
Responsabilidades:	Notificar atores externos ao sistema sobre as ações executadas
Colaboradores:	Administrador de resolução de falhas
Habilidades:	Conhecimento sobre meios de comunicação com o ambiente externo
Normas:	Notificar todas as ações recebidas

Administrador de conhecimento	
Objetivos:	Conhecimento mantido
Responsabilidades:	Manter a base de conhecimento
Colaboradores:	Administrador de resolução de falhas
Habilidades:	Conhecimento sobre manutenção de dados
Normas:	Atender solicitações do Administrador de conhecimento; Não modificar os dados sem autorização

Administrador de falhas em SGBD	
Objetivos:	Falhas de SGBD resolvidas
Responsabilidades:	Administrar resolução de falhas em objetos lógicos, objetos físicos e estruturas de BD
Colaboradores:	Administrador de resolução de Falhas, SGBD
Habilidades:	Conhecimento sobre os serviços de resolução de falhas em SGBD
Normas:	Não desativar o BD quando este estiver em uso; Não executar operações que comprometa a segurança das informações; As operações devem ser transparente ao ator humano Administrador de BD

Administrador de falhas em infra-estrutura	
Objetivos:	Falhas de infra-estrutura resolvidas
Responsabilidades:	Administrar resolução de falhas de sistemas operacionais e conectividade
Colaboradores:	Administrador de resolução de Falhas, Sistema operacional
Habilidades:	Conhecimento sobre os serviços de resolução de falhas em sistemas operacionais e conectividade
Normas:	Não desativar processos enquanto o sistema estiver em uso por outros sistemas; Não executar operações que comprometa a segurança das informações; As operações devem ser transparente ao ator humano Administrador de segurança.

PA1.2 Identificar dependências entre os papéis: Esta atividade define a rede de relacionamentos entre os papéis identificados. As interações são identificadas a partir dos colaboradores identificados. Por exemplo, o administrador de resolução de falhas interage com o *Notificador de Falhas*, *Administrador de conhecimento, Administrador de falhas em*

SGBD e Administrador de falhas de Infra-estrutura. A interação pode ser identificada a partir da análise das tarefas que executam as metas do sistema. Por exemplo, a tarefa *Notificar ação* é um refinamento da tarefa *Resolver falhas em OL* e também é um meio para obter a meta *Ação Notificada*. Portanto, *Notificar ação* é uma interação entre o *Administrador de resolução falhas* e o *Notificador de falhas*. Outras interações identificadas são: *Registrar solução* e *Consultar tipos de solução* entre o *Administrador de resolução falhas* e o *Administrador do conhecimento*. A Figura 6-5 apresenta um grafo com a rede de interações entre os papéis identificados. Os círculos representam os papéis e as setas as interações. As setas com uma única direção indicam uma interação unidirecional, significando que um papel apenas envia mensagem a outro. As setas bidirecionais indicam que há uma troca de mensagens e recursos entre os papéis.

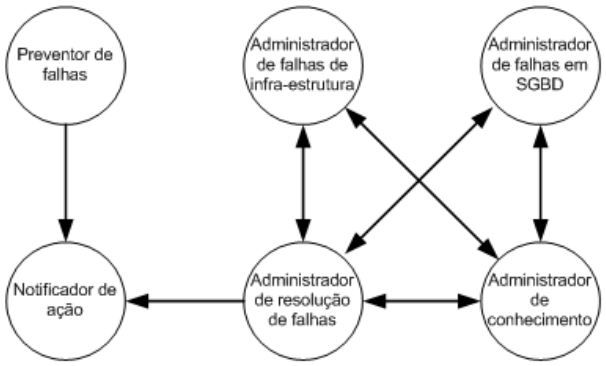


Figura 6-5 – Interações entre os papéis

As interações também podem ser representadas em uma matriz de interação (Tabela 6-11). Se a interação for um zero "0", significa que não há interação entre o papel disposto na linha e o papel disposto na coluna. Se for um número um "1", significa que há uma interação.

Tabela 6-11 – Matriz de interações

	Notificador de ação	*Adm. de resolução de falhas*	*Adm. de conhecimento*	*Adm. de falhas em infra-estrutura*	*Adm. de falhas em SGBD*
Notificador de ação	--	0	0	0	0
Adm. de resolução de falhas	1	--	1	1	1
Adm. de conhecimento	0	1	--	1	1
Adm. de falhas em infra-estrutura	0	1	1	--	0
Adm. de falhas em SGBD	0	1	1	0	--

PA1.3-Identificar as normas da organização: As normas organizacionais especificam o comportamento dos membros da organização. Da especificação dos papéis, apresentados na

Tabela 6-10, são identificadas algumas normas. Por exemplo, o Administrador de resolução de falhas, para resolver uma falha, deve seguir as normas *Não desativar o BD quando este estiver em uso; Não executar operações que comprometa a segurança das informações; As operações devem ser transparente ao ator humano Administrador de BD.* Nesta tabela, cada papel está sujeito a um conjunto de normas para realizar suas ações.

PA2.4 Analisar estilos arquiteturais: Os estilos arquiteturais são baseados em conceitos e alternativas de projetos provenientes de pesquisas em gestão organizacional. *Flat strucuture, Pyramid, Joint-venture, Structure-in-5, Takeover,* entre outros são exemplos destes estilos [Kolp; Giorgini & Mylopoulos, 2006]. Os estilos organizacionais são avaliados usando os atributos não-funcionais de qualidade de software, identificados para arquiteturas envolvendo componentes autônomos e coordenados, tais como *previsibilidade (1), segurança (2), adaptabilidade (3), coordenação (4), cooperação (5), disponibilidade (6), integridade (7), modularização (8) ou agregação (9).* Esta avaliação é documentada no catálogo de correlações (Tabela 3-1). As relações no catálogo são identificadas como contribuições parcial/positivo (+), suficiente/positivo (++), parcial/negativo (-) e suficiente/negativo (--).

Os critérios de qualidade do software representados como metas-soft no diagrama de metas (Figura 6-4) são analisados em relação àqueles atributos de qualidade. As metas-soft apresentadas no diagrama de metas do DBSitter-AS são Adaptabilidade (que é refinada em Extensibilidade), Usabilidade (que é refinada em Aprendizagem e Operabilidade), Segurança (refinada em Confidencialidade e Disponibilidade; Confidencialidade por sua vez é refinada em Acesso autorizado e Disponibilidade é refinada em Confiabilidade) e autonomia. Para análise no catálogo de correlações, são identificadas como atributos para avaliação dos estilos organizacionais as metas-soft *Segurança, Disponibilidade e Adaptabilidade*. Embora disponibilidade seja um refinamento de segurança, é um ponto forte na avaliação dos estilos arquiteturais, por isto fez parte da análise. Também é um requisito das partes interessadas que o sistema seja flexível e pouco centralizado.

PA2.5-Selecionar um estilo arquitetural: Uma vez definidos os atributos de qualidade mais relevantes para o DBSitter-AS, podemos comparar com os critérios de qualidade dos estilos organizacionais através do catálogo de correlação. Podemos observar que os estilos que mais se adaptam ao DBSitter-AS são o *Pyramid, Joint-venture e Takeover* (Figura 6-6). Esta seleção é baseada na existência de correlações suficientes/positivas e ausência de correlações negativas entre os as metas-soft do DBSitter-AS e os tipos de estilos arquiteturais. O estilo *Joint-venture* é o que mais possui correlações suficientes/positivas, portanto é o selecionado.

	Segurança	Adaptabilidade	Disponibilidade
Pyramid	++	+	+
Joint-Venture	+	++	++
Bidding	– –	++	–
Takeover	++	–	+

Figura 6-6 – Seleção de estilo arquitetural

O próximo passo nesta atividade é analisar as interações do estilo arquitetural selecionado. O *Joint-venture* possui um papel *Joint management*, papéis representando parceiros principais e papéis representando parceiros secundários. O *Joint Management* troca informações com os parceiros principais. Os parceiros principais trocam mensagens entre si e solicitam serviços aos parceiros secundários (Figura 6-7).

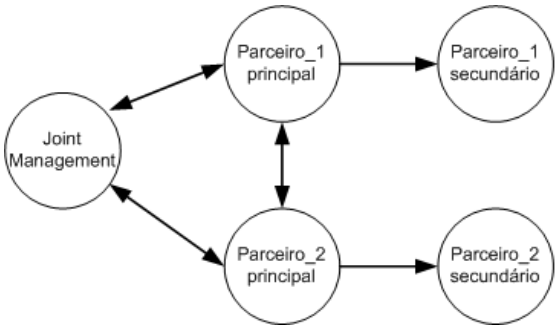


Figura 6-7 – Interações do estilo arquitetural Joint-Venture

PA3.6 Agrupar papéis a subgrupos: Nesta atividade os papéis identificados são analisados para identificar similaridades e a composição em subgrupos. A primeira etapa é analisar o grau de centralidade e grau de proximidade dos papéis. O grau de centralidade é medido analisando a quantidade de saídas e entradas nos grafos de interação. Por exemplo, o papel *Administrador de falhas de infra-estrutura* tem uma interação de saída (grau de saída) e duas de entrada (grau de entrada). Estes valores são normalizados, de forma a comparar com outras redes de tamanhos diferentes. Os valores de entrada e saída normalizados são calculados avaliando os percentuais em relação ao tamanho da rede, isto é a quantidade de papéis. Se existem cinco papéis na rede, o percentual de normalização é calculado dividindo o grau de entrada ou saída pela quantidade de papéis subtraindo um que é o próprio papel analisado. A Tabela 6-12 apresenta o grau de centralidade para os papéis analisados nesta iteração [Bastos, 2005]. A medida *Grau de saída* representa quão influente é um papel e a medida *Grau de entrada* representa quão proeminente um papel é no grupo. O papel *Administrador de Conhecimento* e o *Administrador de resolução de falhas* são os papéis mais proeminentes e

influentes do grupo, possuindo o maior grau de entrada e saída. *O Notificador de falhas* não tem nenhuma influência no grupo e tem pouca proeminência. Observa-se também, nesta tabela, que o *Administrador de falhas de infra-estrutura* e o *Administrador de falhas em SGBD* têm o mesmo padrão de interações assim como o *Administrador de conhecimento* e o *Administrador de resolução de falhas.*

Tabela 6-12 – Medida do grau de centralidade

Papéis	*Grau de saída*	*Grau de entrada*	*Saída Normalizada*	*Entrada Normalizada*
Administrador de falhas de infra-estrutura	2	2	50	50
Administrador de falhas em SGBD	2	2	50	50
Administrador de conhecimento	3	3	75	75
Administrador de resolução de falhas	3	3	75	75
Notificador de ação	0	1	0	25

O próximo passo é calcular o grau de proximidade que mede quão próximo um papel está dos outros. Analisam-se as medidas da soma da distância geodésica entre um papel e todos os outros [Bastos, 2006]. Por exemplo, no grafo de interações entre os papéis (Figura 6-5), o *Administrador de conhecimento* está associado diretamente aos atores *Administrador de falhas de infra-estrutura, Administrador de falhas em SGBD e Administrador de resolução de falhas*, portanto sua distância até eles é igual a um (1). E para chegar ao *Notificador de ação* precisa passar por pelo menos outro papel, então a distancia é igual a dois (2). Portanto a distância do Administrador de conhecimento é obtida adicionando *um* para os três primeiros papéis e *dois* para este último o resultado é cinco (5). O *Administrador de resolução de falhas* é o papel mais central, pois têm o menor valor de Distância geodésica com outros papéis enquanto o *Notificador de ação* é o mais periférico. A Proximidade é calculada calculando o inverso do valor da distância (1 / distância) normalizada em relação ao valor da distância do papel mais central. A Tabela 6-13 – Medida do grau de proximidade dos papéis mostra as medidas de distância e Proximidade calculadas para os papéis do sistema DBSitter-AS. Observa-se que o *Administrador de falhas de infra-estrutura* e o *Administrador de falhas em SGBD* têm padrões de medidas semelhantes.

Tabela 6-13 – Medida do grau de proximidade dos papéis

Papéis	*Distância*	*Proximidade*
Administrador de falhas de infra-estrutura	6	66,67
Administrador de falhas em SGBD	6	66,67
Administrador de conhecimento	5	80,00
Administrador de resolução de falhas	4	100,00
Notificador de ação	7	57,14

O próximo passo é medir a similaridade estrutural entre os papéis identificados através do cálculo dos coeficientes de correlação (r) usando a fórmula de Pearson (1).

$$r = \frac{n\sum XY - \sum X \sum Y}{\sqrt{\left(n\sum X^2 - \left(\sum X\right)^2\right)\left(n\sum Y^2 - \left(\sum Y\right)^2\right)}} \qquad (1)$$

Onde n é a quantidade de interações e as variáveis X e Y são a existência de correlações de dois papeis, expressos pelos números um e zero (1 e 0). Por exemplo, na Tabela 6-11, se avaliar a similaridade entre os papéis *Administrador de resolução de falhas e Administrador de conhecimento*, os valores de X serão {1,1,1,1} e de Y serão {0,1,1,1}. O valor de n é igual a cinco, por existirem cinco papéis na tabela que são analisados. As medidas do grau de similaridade dos papéis analisados nesta iteração estão apresentadas na Tabela 6-14.

Analisando os valores do grau de similaridades entre os papeis identifica-se que o *Notificador de falhas* não é similar a nenhum outro papel (r=0). O *Administrador de falhas em SGBD e o Administrador de falhas em infra-estrutura* são fortemente similares (r=1). E os demais papeis têm similaridades parciais (0,3 > r < 0,7).

Tabela 6-14 – Grau de similaridade dos papéis

	1	*2*	*3*	*4*	*5*
1. Notificador de ação	--				
2. Administrador de resolução de falhas	0	--			
3. Administrador de conhecimento	0	0,612	--		
4. Administrador de falhas em infra-estrutura	0	0,408	0,667	--	
5. Administrador de falhas em SGBD	0	0,408	0,667	1	--

Obs.: Os números 1, 2, 3, 4 e 5 das colunas da tabela correspondem aos números que identificam os papéis nas linhas. Apresentado assim por questão de espaço.

Pela análise do grau de centralidade é observado que os papeis *Administrador de falhas de infra-estrutura e Administrador de falhas em SGBD* tem o mesmo padrão de interações. A análise do grau de proximidade também identifica este mesmo comportamento. Pela análise do grau de similaridade é confirmado que estes papéis pertencem ao mesmo conjunto de papéis similares. Os demais papéis não apresentam comportamento semelhante. Desta forma conclui-se que o sistema DBSitter-AS possui quatro subgrupos de papéis: *Notificador de ação* para executar tarefas de comunicação com atores externos; *Administrador de resolução* de falhas para coordenar atividades de resolução de falhas; *Administrador de conhecimento* para gerenciar troca de informações entre os papéis; e o *Administrador de falhas específicas* para executar tarefas de resolução de falhas.

PA3.7-Analisar a correlação entre subgrupos e estilo arquitetural : Esta atividade analisa os graus de centralidade e proximidade dos componentes do estilo arquitetural selecionado e compara com os mesmos valores para os subgrupos identificados na atividade anterior. A Tabela 6-15 apresenta a medida do grau de centralidade e a Tabela 6-16 apresenta a medida do grau de proximidade do estilo arquitetural *joint-venture*.

Tabela 6-15 - Medida do grau de centralidade do estilo arquitetural joint-venture

Componentes	*Grau de saída*	*Grau de entrada*	*Saída Normalizada*	*Entrada Normalizada*
Joint Management	2	2	50	50
Parceiro_1 principal	3	2	75	50
Parceiro_1 secundário	0	1	0	25
Parceiro_2 principal	3	2	75	50
Parceiro_2 secundário	0	1	0	25

Tabela 6-16 - Medida do grau de proximidade do estilo arquitetural Joint-venture

Componentes	*Distância*	*Proximidade*
Joint Management	6	66,67
Parceiro_1 principal	5	80
Parceiro_1 secundário	8	50
Parceiro_2 principal	5	80
Parceiro_2 secundário	8	50

A correlação entre os subgrupos identificados dos requisitos e os componentes arquiteturais é baseada na comparação das medidas de centralidade e proximidade. O *Notificador de ações* e o *Parceiro_Secundário* têm medidas semelhantes para o grau de centralidade (0 e 25). O grau de proximidade também é similar, isto é 50 para o *Notificador de ações* e 57,14 para o Parceiro_Secundário. O subgrupo A*dministrador de falhas específicas* e o componente *Joint Management* têm graus de centralidade de entrada (50) e saída (50) e graus de proximidade (66,67) semelhantes. O *Administrador de conhecimento* tem medidas semelhantes a um *Parceiro_principal* para o grau de centralidade de saída (75) e o grau de proximidade (80).

O próximo passo é analisar a similaridade estrutural entre a organização de subgrupos e o estilo arquitetural para identificar se as organizações contém semelhanças e portanto podem ter seus componentes mapeados. Os grupos são analisados a partir do ator mais central. O subgrupo *Administrador de resolução de falhas* é o mais central da organização de subgrupos, pois tem grau de proximidade 100 e graus de centralidade de entrada e saída igual a 75. No

estilo arquitetural *Joint-venture* o ator mais central é o *parceiro_principal* com grau de proximidade 80 e grau de centralidade de entrada 75 e grau de centralidade de saída 50. As correlações entre os subgrupos e componentes do estilo arquitetural podem ser avaliadas como fortes (++), parcialmente fortes (+) e nenhum. Os pares *Notificador de ações / Parceiro_secundário* e *Administrador de falhas específicas / Joint Management* têm correlações de centralidade forte, pois têm as mesmas medidas. O *Administrador de conhecimento e Parceiro_principal* têm correlação de centralidade parcial, pois tem a medida do grau de centralidade de saída diferente (Tabela 6-17). A pontuação dos dois grupos pode ser considerada fortemente centralizada (igual a 1), uma vez que a média dos valores é maior que 50.

Tabela 6-17 – Correlação de centralidade entre subgrupos do DBSitter-AS e o Joint-Venture

	Saída Normalizada	*Entrada Normalizada*	*Proximidade*	*Pontuação do grau*	*Ator central*
Joint-Venture	50	75	80	1	Parceiro_principal
Grupo organizacional do DBSitter-AS	75	75	100	1	Administrador de resolução de falhas

	Componentes do Joint-Venture		
	Joint Management	*Parceiro_1 principal*	*Parceiro_1 secundário*
Administrador de falhas específicas	++		
Administrador de conhecimento		+	
Administrador de resolução de falhas			
Notificador de ação			++

A similaridade estrutural entre os subgrupos e componentes do estilo arquitetural completa a análise da correlação. Primeiramente calcula-se a similaridade pelo grau de centralidade de entrada e depois calcula a similaridade pelo grau de centralidade de saída (Tabela 6-18). Lembrando que a similaridade estrutural é calculada pela aplicação da Fórmula de Pearson (explicada na atividade PA3.6.) usando os valores das matrizes de integração (Tabela 6-11).

Tabela 6-18 – Similaridade estrutural entre os subgrupos e o estilo arquitetural

	Valores usados	*Parceiro_1 secundário*	*Parceiro_1 principal*	*Joint Management*
Notificador de ação	Grau de entrada	0,5774	-0,5774	-0,5774
	Grau de saída	1	0,3333	-1
Administrador de resolução de falhas	Grau de entrada	-0,3333	1	1
	Grau de saída	0,25	0,6123	0,6123
Administrador de conhecimento	Grau de entrada	-0,3333	1	1
	Grau de saída	-1	-0,3333	1
Administrador de falhas especificas	Grau de entrada	-0,3333	1	1
	Grau de saída	-1	-0,3333	1

Pela análise, o *Notificador de ação* tem forte correlação com o *Parceiro_1 secundário* (a similaridade estrutural para o grau de entrada e saída são positivos e maiores que 0,3) e não tem correlação com os outros (similaridades negativas ou menores que 0,3). O *Administrador de resolução de falhas* tem forte correlação com *Parceiro_1 principal* e *Joint Management* (a similaridade estrutural para os graus de entrada e saída são positivos e maiores que 0,3). O *Administrador de conhecimento* tem forte similaridade com o *Joint Management* (similaridade estrutural igual a 1). Também existe alguma correlação entre o *Administrador de falhas específicas* e os *Parceiro_1 principal* e *Joint Management* (similaridade variando entre -0,3 e 03). Como já identificado, existe uma forte correlação entre *Administrador de falhas especificas* e o *Joint Management*. A Tabela 6-19 resume esta análise. Com este resultado a análise de correlações finaliza e é possível derivar a primeira configuração organizacional para o sistema.

Tabela 6-19 – Correlação de similaridade do exemplo DBSitter-AS

	Parceiro_1 secundário	*Parceiro_1 principal*	*Joint Management*
Notificador de ação	**++**	**-**	--
Administrador de resolução de falhas	**-**	**++**	++
Administrador de conhecimento	**--**	**+**	**++**
Administrador de falhas específicas	**-**	**+**	**++**

PA4.8 Mapear subgrupos a componentes do estilo arquitetural: Esta atividade define o mapeamento entre os subgrupos e os componentes do estilo arquitetural. Primeiramente identifica-se as correlações mais fortes e relaciona os componentes do estilo arquitetural assumidos pelos subgrupos. Pela Tabela 6-19, podemos concluir que o *Notificador de ação* assume claramente o papel do componente *Parceiro_1 secundário* e tem a responsabilidade de Notificar as ações recebidas de componentes do tipo *Parceiro_1 Principal*. No estilo arquitetural *Joint-venture* deve existir apenas um *Joint Management*, portanto deve ser selecionado entre os três candidatos o que mais se adéqua a este componente e os demais assumem o papel de *Parceiro_1 principal*. O *Administrador de resolução de falhas* tem a mesma correlação com os dois componentes do *Joint-venture*, portanto é selecionado para assumir o papel de *Parceiro_1 principal*. Sua responsabilidade é a administração da resolução de falhas globais do sistema. Ele depende do *Notificador de ações* para se comunicar com os atores externos ao sistema. O papel de *Joint Management* é atribuído ao *Administrador de conhecimento*, pois os dois têm em comum a meta de administrar conhecimento estratégico da

organização trocando informações com os parceiros principais. Por sua vez, o *Administrador de falhas específicas* assume o papel do componente *Parceiro_1 principal*, pois tem responsabilidades específicas dentro da organização. O modelo de arquitetura resultante é apresentado na Figura 6-8.

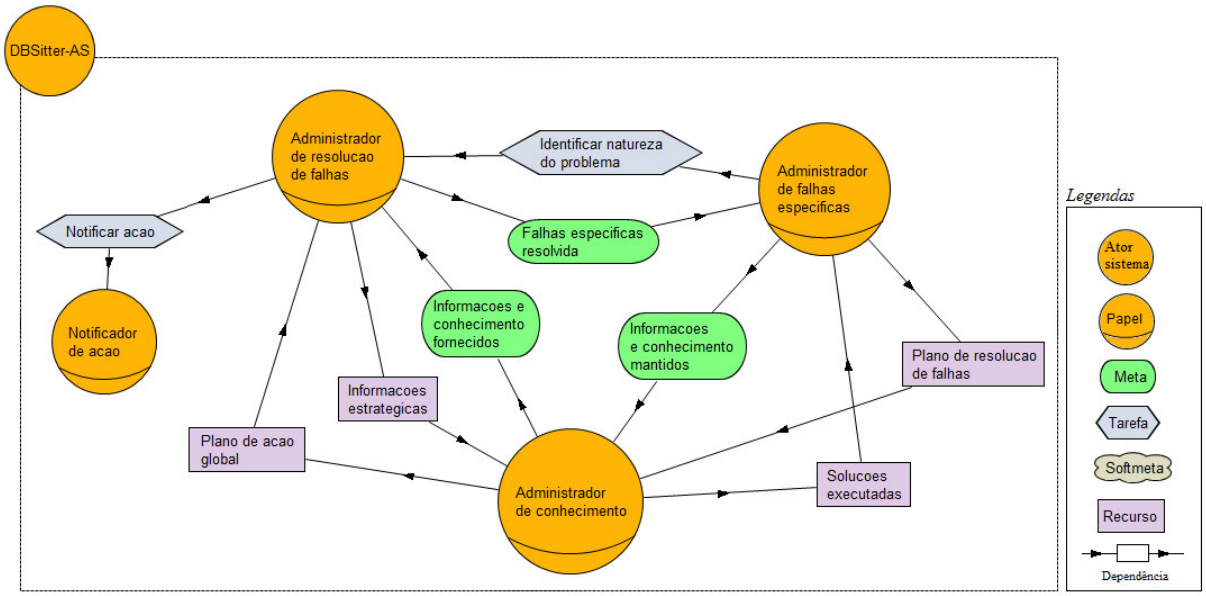


Figura 6-8 – Modelo arquitetural do DBSitter-AS

PA5.9 Validar os modelos propostos: A validação ocorre pela apresentação dos modelos às partes interessadas. Neste caso envolve os engenheiros de requisitos, desenvolvedores, arquitetos e projetistas de sistemas multiagentes. Se os modelos forem aprovados, a análise prossegue, senão os ajustes são executados antes dos próximos passos.

Com o modelo arquitetural projetado, os projetistas podem iniciar o processo de analisar as estruturas internas dos componentes e projetar a construção dos agentes que executarão papéis dentro da organização do sistema. A próxima seção exemplifica uma iteração de projeto e construção de agentes do DBSitter-AS.

6.3.4 Iteração 4 – Projeto detalhado do primeiro protótipo do sistema

Esta iteração tem como objetivo projetar e construir os agentes prioritários do sistema. O foco está no projeto interno dos agentes que executam os papéis definidos na primeira configuração arquitetural do sistema. Por isto, esta iteração inicia com atividades de análise de requisitos (Tabela 6-1). A primeira definição de trabalho é *RF2-Análise dos requisitos do sistema*. As atividades 4,5 e 6 desta definição de trabalho se concentram em analisar os requisitos intencionais dos componentes arquiteturais, que inclui tanto metas como ligações meio-fim. A próxima definição de trabalho é *PD1-Refinamento do projeto da arquitetura*. A

atividade 1 é responsável pelo mapeamento do modelo arquitetural em i* para modelos UML que representam as visões do projeto detalhado de agentes (Arquitetura, comunicação, Intenção, Ambiente, Raciocínio e Planos). As atividades de 2 a 7 apresentam a especificação destes modelos. Este exemplo finaliza demonstrando as atividades citadas até este ponto. As demais atividades e iterações serão apresentadas em trabalhos futuros. A seguir são exemplificadas as atividades desta iteração, aplicadas ao projeto detalhado do sistema DBSitter-AS.

RF2-4 Realizar análise orientada a Metas: Nesta atividade as metas identificadas no modelo de requisitos, mais especificamente no diagrama de metas do DBSitter-AS são redistribuídas entre os componentes da arquitetura. Novas metas podem surgir da análise realizada. Por exemplo, a meta *Informações e conhecimentos fornecidos* foi identificada como uma dependência entre o *Administrador de conhecimento* e o *Administrador de resolução de falhas*, portanto se tornou uma nova meta a atingir. Metas dos papéis da arquitetura também são atribuídas aos subgrupos. A meta *Coordenação* foi atribuída ao Administrador de conhecimento que assumiu o papel do *Joint management*. Esta atividade é realizada em equipe pelo Engenheiro de requisitos e Arquiteto do software, uma vez que está sendo realizada uma análise das metas e planos já levantados na fase de requisitos finais, mas com a intenção de distribuí-los para os componentes do projeto arquitetural definido. Esta análise já iniciou na *iteração 3 – Projeto da arquitetura global do sistema* na definição de trabalho *PA1-Especificar o modelo organizacional*, quando os papéis foram definidos a partir das metas e tarefas. Esta atividade apenas conclui a análise e modela o diagrama de metas para a arquitetura (Figura 6-9).

RF2-5-Realizar análise Meio-Fim: A análise meio-fim implica em uma revisão dos planos que executam as metas dos componentes da arquitetura. Nesta iteração não houveram mudanças de requisitos ou solicitações de usuários, uma vez que é a primeira iteração de Projeto detalhado. Mas alguns planos foram adicionados como refinamentos de planos para atender necessidades do projeto arquitetural definido. Por exemplo, os planos *Recuperar Planos de ação* e *Atualizar informação e conhecimento* foram adicionados ao componente *Administrador de conhecimento*. Novos planos também foram inseridos para atender novas metas definidas no projeto arquitetural, por exemplo, *Coordenar tarefas* para o componente *Administrador de conhecimento.*

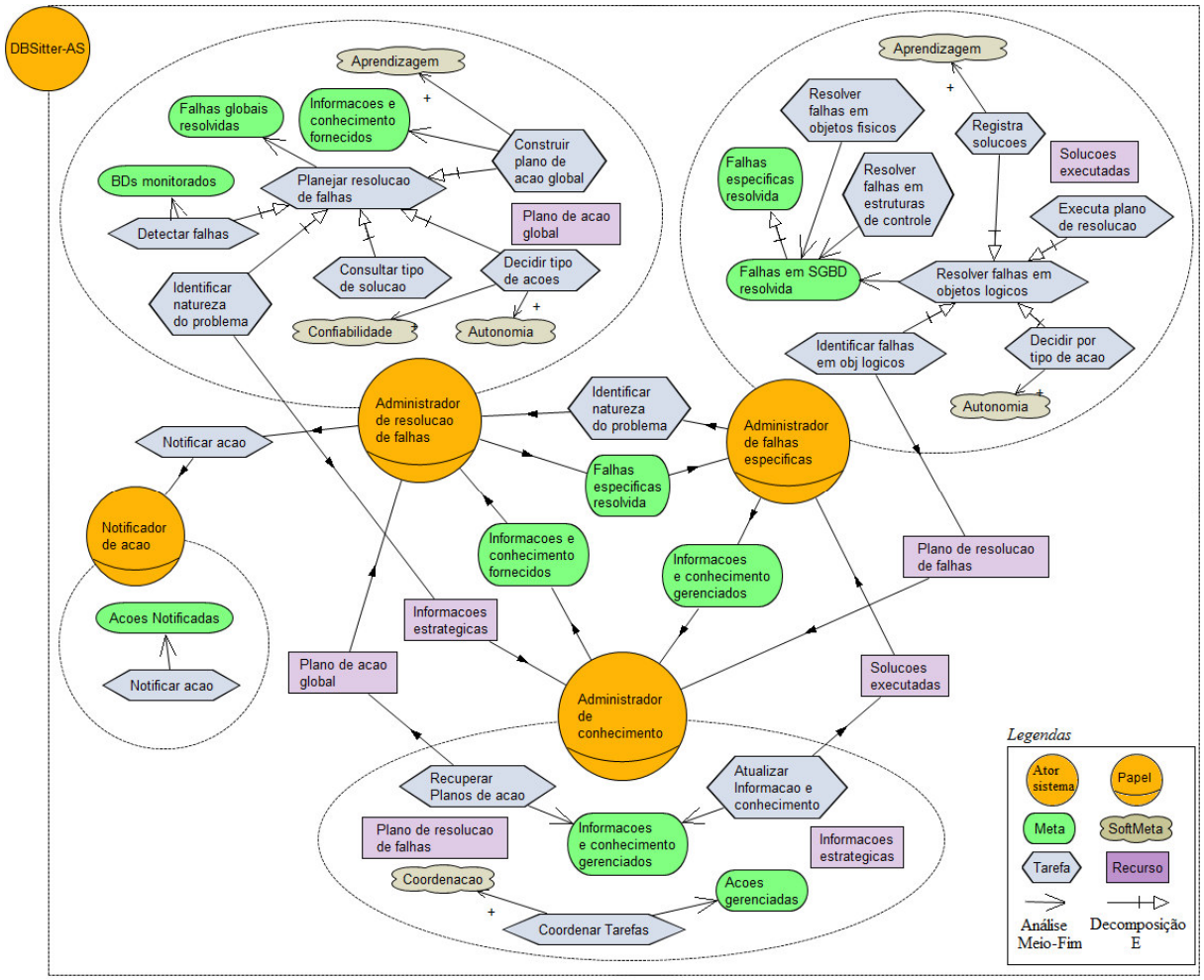


Figura 6-9 – Diagrama de metas para os componentes da arquitetura

RF2-6 Realizar Análise das contribuições: Nesta atividade, as metas-soft também são distribuídas por componentes a partir da influência dos planos. A meta-soft *Coordenação* é inerente ao componente *Joint management*, portanto deve ser satisfeita pelo *Administrador de conhecimento* que assumiu esta responsabilidade. Alguns planos e metas-soft que estavam no diagrama de metas do DBSitter-AS (Figura 6-4 – Diagrama de Metas – Requisitos Finais) não foram apresentadas neste fragmento de diagrama. À medida que forem sendo selecionadas novas metas a ser analisadas e desenvolvidas em iterações posteriores, serão realizadas novas análises e todos os requisitos (metas e planos) do diagrama de metas serão acrescentados ao diagrama de arquitetura.

PD1-1-Mapear i* para modelos em UML: Esta atividade realiza uma análise nos elementos do modelo i* e gera um mapeamento para os diagramas da UML que compõe o projeto detalhado. Usa como guia as heurísticas de mapeamento proposto em [Silva, C.T.L.L., 2007a]. O mapeamento é apresentado na Tabela 6-20.

Tabela 6-20 – Mapeamento dos elementos do diagrama da Figura 6-9 em i* para UML

Heurísticas	*Mapeamento*
H1. Cada papel no modelo i* se torna uma classe «AgentRole» no diagrama arquitetural	São <<AgentRoles>>: Administrador de resolução de falhas, Notificador de ação, Administrador de falhas especificas e Administrador de conhecimento
H2. Cada agente no modelo i* se torna uma classe «Agent» no diagrama intencional	Não se aplica ao exemplo
H3. Cada relacionamento *play* entre um agente e um papel no modelo i* se torna um relacionamento da classe associativa «Intention» entre o «Agent» e «AgentRole» no diagrama intencional	Não se aplica ao exemplo
H4. Cada dependum no i* se torna uma interface «Dependum» no diagrama Arquitetural	São <<Dependum>>: Notificar ação, Plano de ação global, Informações estratégicas, Informações e conhecimento fornecidos, Falhas específicas resolvidas, Identificar natureza do problema, Informações e conhecimento gerenciados, Soluções executadas, Plano de resolução de falhas
H5. As dependências do tipo (o-open, "" - commit, x-critica) são capturadas pela propriedade degree de «Dependum»	São todas do tipo Commit
H6. Cada Depender se torna um «Depender» no diagrama Arquitetural	São <<Depender>> as dependências de: Administrador de resolução de falhas, Administrador de falhas especificas e Administrador de conhecimento
H7. Cada Dependee se torna a realização de uma interface «Dependee» no diagrama Arquitetural	São <<Dependee>> as dependências de: Administrador de resolução de falhas, Notificador de ação, Administrador de falhas especificas e Administrador de conhecimento
H8. Cada dependência (depender -> dependum -> dependee) se torna uma associação «Connector» no diagrama Arquitetural	Todos os elementos mapeados na H4
H9. Cada recurso se torna um «Resource» no diagrama ambiental	São <<Resource>>: Plano de ação global, Informações estratégicas, Plano de resolução de falhas
H10. Cada meta (ou meta-soft) no modelo i* que não é decomposta se torna um «Goal» em ambos os diagramas arquiteturais e intencionais	São <<goal>>: BDs monitorados, Falhas globais resolvidas, Informações e conhecimento fornecidos, Falhas especificas resolvida, Informações e conhecimento gerenciados, Ações gerenciadas, Ações Notificadas, Aprendizagem, Confiabilidade, Autonomia e Coordenação
H11. Cada meta-soft que não é decomposto se torna um «Softgoal» no diagrama racional	São <<Softgoal>>Aprendizagem, Confiabilidade, Autonomia e Coordenação
H12. Cada link de contribuição entre uma tarefa e uma meta-soft se torna um relacionamento de classe associativa «Contribution» entre o «MacroPlan» e o «Softgoal» no diagrama racional	Existem 6 links de contribuição no diagrama i*
H13. Cada tarefa nos modelos i* se torna um «MacroPlan» nos diagramas arquiteturais e intencionais	São <<MacroPlan>> Planejar resolução de falhas, Resolver falhas em objetos físicos, Resolver falhas em estruturas de controle, Resolver falhas em objetos lógicos, Recuperar Planos de ação, Atualizar Informação e conhecimento e Coordenar Tarefas
H14. Cada sub-tarefa nos modelos i* se torna um «ComplexAction» no diagrama arquitetural. Ela representa cada passo que compõe um «MacroPlan»	São <<ComplexAction>> do <<MacroPlan>> Planejar resolução de falhas: Detectar falhas, Identificar natureza do problema, Consultar tipo de solução, Decidir tipo de ações, Construir plano de ação global e são <<ComplexAction>> do <<MacroPlan>> Resolver falhas em objetos lógicos: Identificar falhas em objetos lógicos, Decidir por tipo de ação, Executa plano de resolução e Registra soluções.
H15. Uma «Belief» é alguma condição para executar uma tarefa (i.e. um «Plan» ou um «AgentAction»).	Não se aplica ao exemplo
H16. Uma «Organization» é composta de papéis e outras organizações	O DBSitter-AS é uma <<organizacao>> composta dos papeis citados na H1.
H17. Cada (soft)meta que é parte de uma tarefa se torna um «Goal» e cria um «ComplexAction» responsável por gerar aquele meta no «MacroPlan» correspondente a tarefa	Não se aplica ao exemplo

PD1-2-Modelar a Arquitetura: O diagrama arquitetural (Figura 6-10) modela a estrutura do sistema em termos de papéis de agentes que dependem de outros para atender suas metas. É representado por um diagrama de classes estendido da UML. As heurísticas apresentadas na atividade anterior definem os elementos do modelo arquitetural que são mapeados do diagrama i*.

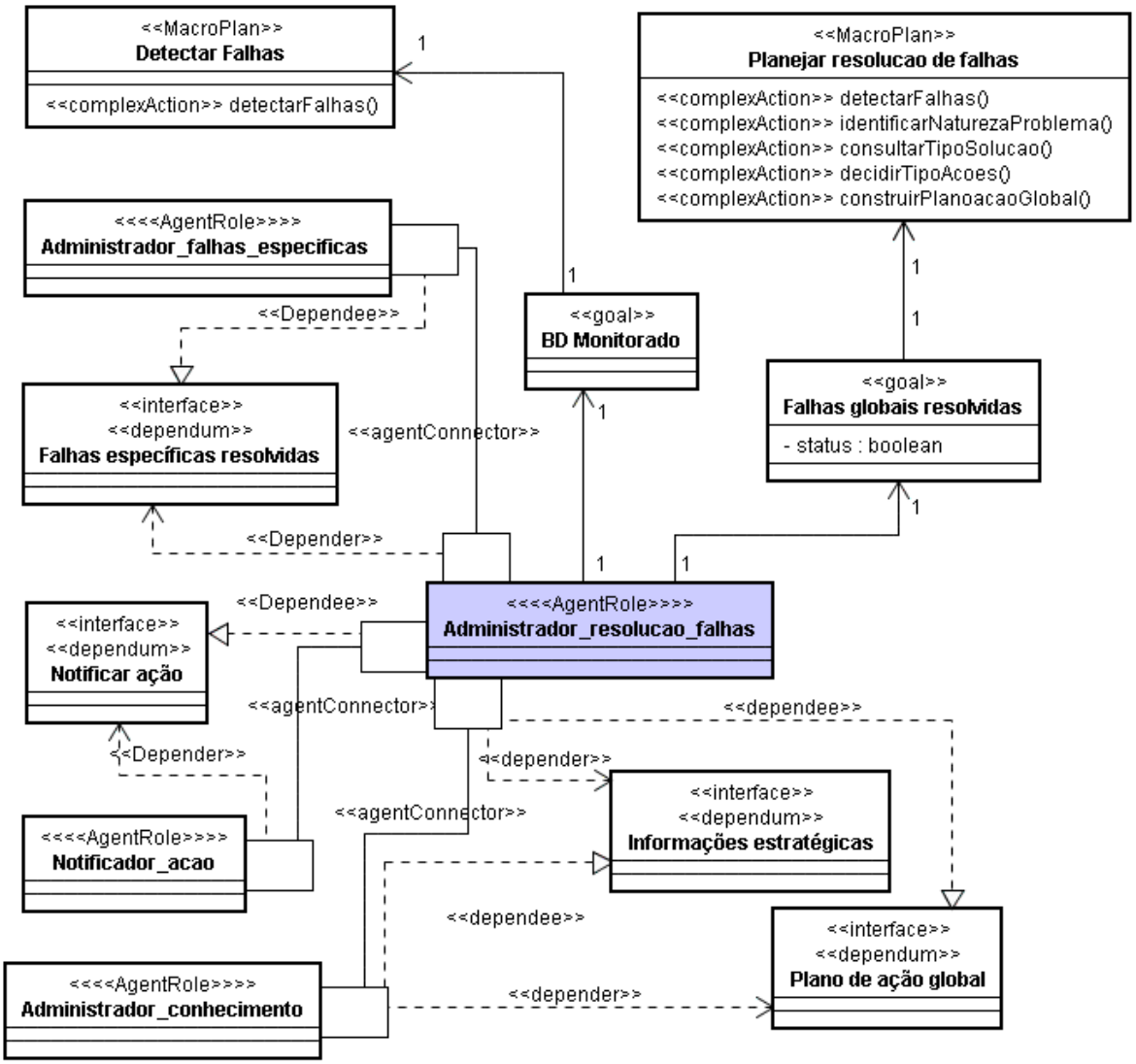


Figura 6-10 – Diagrama arquitetural

Por exemplo, o componente *Administrador de resolução de falhas* foi mapeado para a classe *Administrador_resolucao_Falhas* que tem como estereótipo <<AgentRole>>. Ele tem três dependências representadas pelos conectores <<agentConnector>>. Estes conectores são associados aos outros componentes também representados como classes com estereótipo <<AgentRole>>: *Administrador_falhas_especificas, Notificador_Acao* e *Administrador_ conhecimento.* As dependências identificam quem é o depender pelo relacionamento <<depender>> e o dependee pelo relacionamento <<dependee>>. O dependum, que é

representado como uma classe do tipo interface, é identificado pelo estereótipo <<dependum>>. Assim, identifica-se o dependum *Falhas específicas resolvidas* que tem como dependee o *Administrador_falhas_especificas* e como depender o *Administrador_resolucao_falhas*. As metas de um papel são representadas como classes com o estereótipo <<goal>> e estão relacionadas com o papel por uma associação. No diagrama, para a classe <<AgentRole>> *Administrador_resolucao_falhas*, são identificadas as metas *BD monitorado* e *Falhas globais resolvidas*. As tarefas associadas à meta por ligações meio-fim são representados como uma classe com estereótipo <<MacroPlan>>. *Planejar Resolução de falhas* é um <<MacroPlan>> associado a classe *Falhas globais resolvidas*. As decomposições das tarefas são representadas como operações da classe <<MacroPlan>> e recebem como estereótipo <<complexAction>>. No diagrama são identificadas as operações da classe Planejar resolução de falhas: *detectarFalhas, identificarNaturezaProblema, consultarTipoSolucao, decidirTipoAcoes e construirPlanoAcaoGlobal.*

PD1-3 Modelar Comunicações: O modelo de comunicação é representado por um diagrama de seqüência que apresenta as interações entre agentes que executam os papéis representados na arquitetura. Primeiro identifica-se os agentes, depois os protocolos de comunicação e as mensagens trocadas. No exemplo, são usados quatro agentes (*AdmSGBD, AdmConhecimento, AdmFalhas, Notificador*) que se comunicam para executar uma tarefa. As mensagens trocadas são para preparação do plano de resolução de falhas. São usados os protocolos REQUEST para solicitar uma ação ou informação a outro agente, por exemplo o *AdmSGBD* solicita um Plano ao *AdmConhecimento* através do protocolo REQUEST passando como parâmetros o recurso *Plano Resolução Falhas* solicitado. O protocolo INFORM é usado para responder a solicitação, enviando o recurso solicitado.

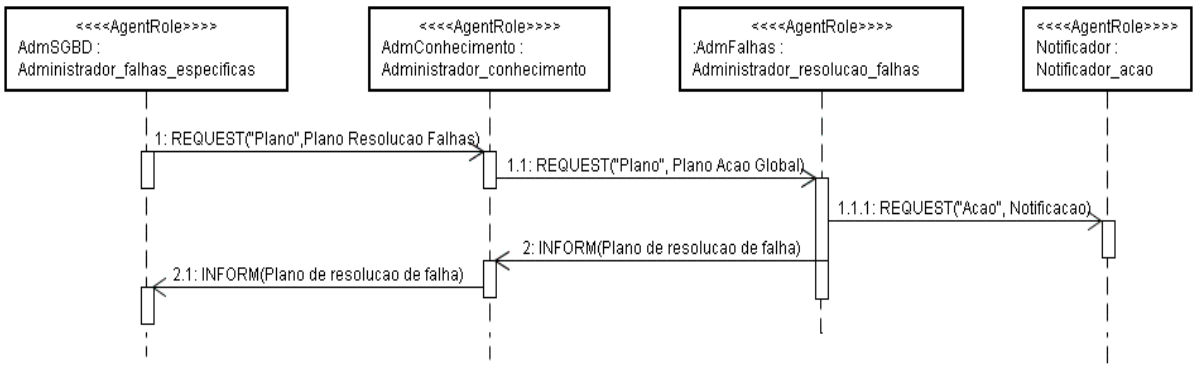


Figura 6-11 – Diagrama de comunicação

PD1-4 Modelar as Intenções: Esta atividade modela os elementos intencionais do sistema (Figura 6-12). Primeiro identifica-se os papéis da organização, o ambiente e os elementos

intencionais. Os papéis modelados (classes <<agentRoles>>) são *Administrador-resolucao_falhas* e *Administrador_falhas_especificas*. Os elementos intencionais são as metas dos papéis (classes identificadas pelo estereótipo <<goal>>), as Tarefas para obter as metas (classes identificadas pelo estereótipo <<macroPlan>>), os agentes (classes identificadas pelo estereótipo <<agent>>) que executam os papéis, suas intenções para obter as metas (classes identificadas pelo estereótipo <<intention>>) e as Crenças do agente no ambiente (classes identificadas pelo estereótipo <<belief>>). Os papéis fazem parte da organização DBSitter-AS (classes identificadas pelo estereótipo <<organization>>) e são executados por agentes dentro do ambiente de execução do sistema (classe identificadas pelo estereótipo <<environment>>).

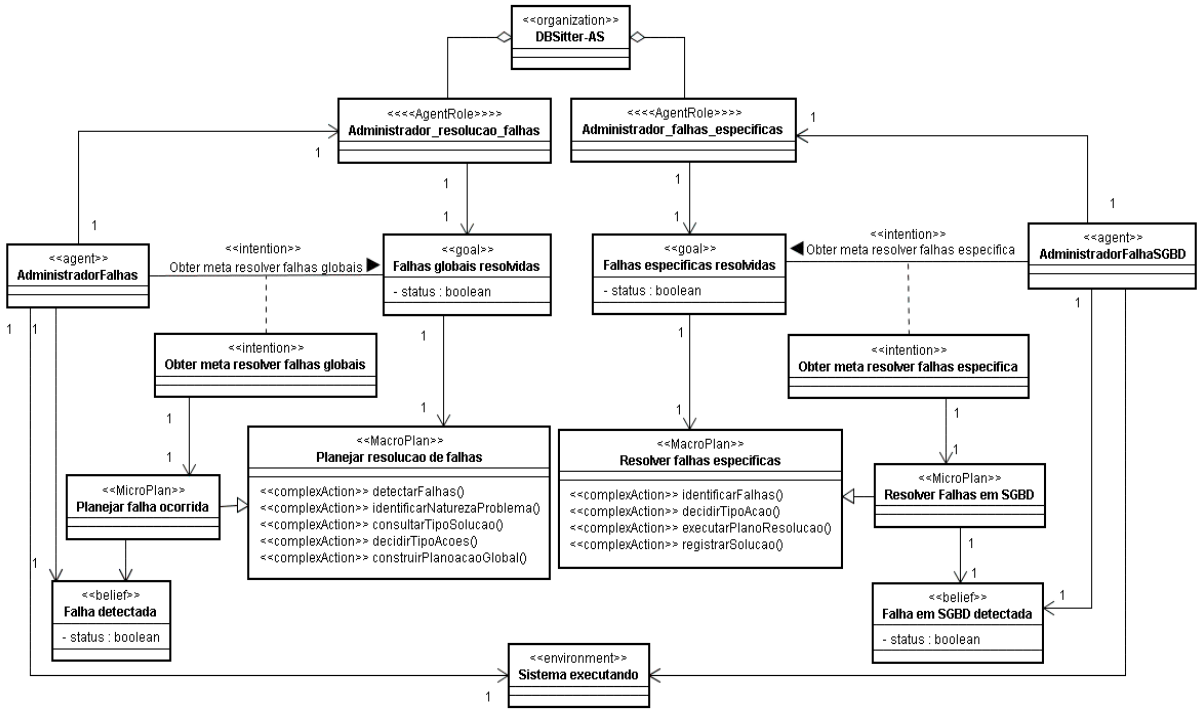


Figura 6-12 – Diagrama intencional

PD1-5 Modelar o Ambiente: Esta atividade modela o ambiente no qual o sistema está inserido (Figura 6-13). Primeiro é necessário identificar a organização (classes DBSitter-AS identificada pelo estereótipo <<organization>>) e os papéis (classes identificada pelo estereótipo <<agentRole>>) que a compõe. Depois, identifica-se o ambiente (classe identificadas pelo estereótipo <<environment>>) e os seus recursos (classes identificadas pelo estereótipo <<resource>>). Os recursos são mapeados do diagrama i*. São eles *Notificações*, *Plano de ação global, Informação estratégica, Registro de soluções executadas* e *Plano de resolução de Falhas.* Estes são associados aos papéis que produzem ou consomem os recursos. A cada papel é atribuído o direito de acesso (classe associativa com estereótipo

<<Right>>). Por exemplo, o papel *Notificador_Acao* (<<AgentRole>>), tem direito de leitura do recurso *Notificações* (<<resource>>). O papel *Administrador_resolucão_falhas* (<<AgentRole>>), tem direito de criar, editar e destruir o recurso *Notificação*. Nenhum outro papel tem direito sobre este recurso.

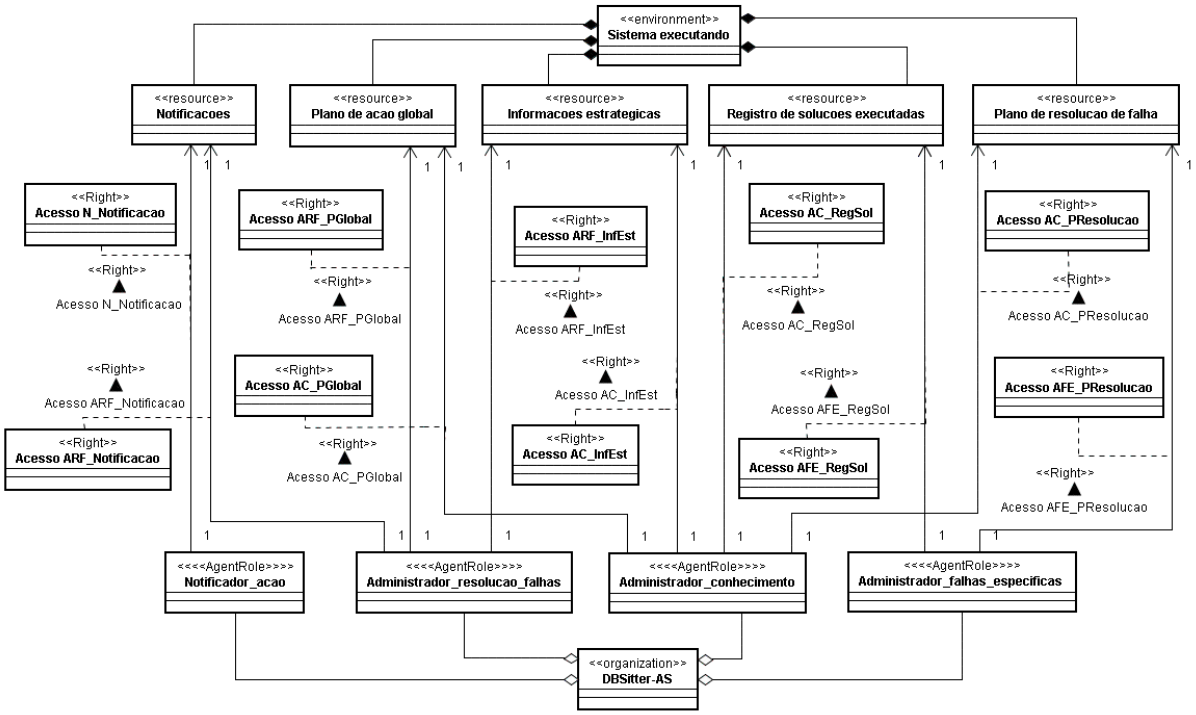


Figura 6-13 – Diagrama de ambiente

PD1-6 Modelar o raciocínio: Nesta atividade são modelados os elementos racionais dos papéis (Figura 6-14). Primeiro identificam-se os Papéis (classes identificada pelo estereótipo <<agentRole>>), Tarefas (classes identificada pelo estereótipo <<MacroPlan>>) e Metas-soft (classes identificada pelo estereótipo <<softgoal>>). Depois, modelam-se os seus relacionamentos e as contribuições das tarefas para as metas-soft por classes associativas. Por exemplo, o *Administrador_falhas_especificas* (<<agentRole>>) tem as metas-soft *Autonomia* (<<softgoal>>) e *Aprendizagem* (<<softgoal>>) a satisfazer. A tarefa *Planejar resolução de falhas* (<<MacroPlan>>) tem influência positiva para a meta-soft *Autonomia* (classe associativa PRF-At-influencia3) e para a meta-soft *Aprendizagem* (classe associativa PRF-Ap-influencia1).

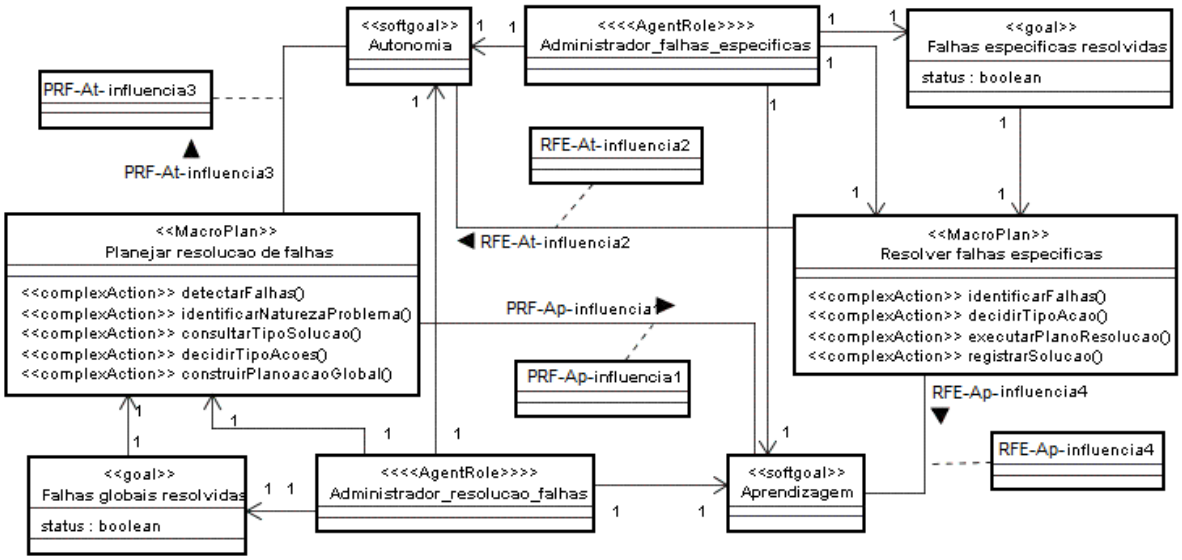


Figura 6-14 – Diagrama racional

PD1-7-Modelar execução dos planos: Nesta atividade é modelada a seqüência de ações executadas para realizar um plano do papel ou do agente (Figura 6-15). Primeiro identifica-se qual a tarefa (<<MacroPlan>> ou <<MicroPlan>>) que se quer modelar, depois as ações complexas desta tarefa (operações <<complexAction>>). Por exemplo, a seqüência de ações da tarefa *Planejar resolução de falhas* (<<MacroPlan>>) do papel *Administrador_resolucão _falhas* (<<AgentRole>>) é modelada pela seqüência de suas ações complexas: *detectarFalhas, identificarNaturezaProblema, consultarTipoSolucao, decidirTipoAcoes e construirPlanoAcaoGlobal*.

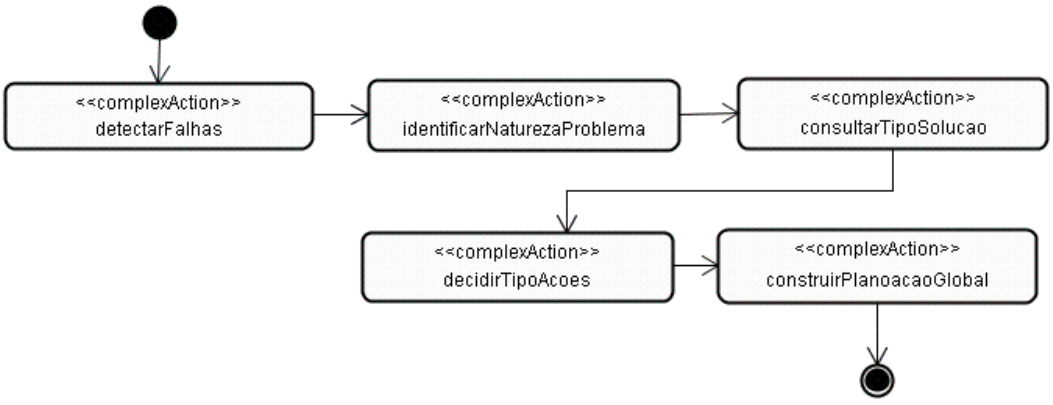


Figura 6-15 – Diagrama de planos

O exemplo de demonstração do processo finaliza nesta atividade. As próximas atividades envolveriam a análise das capacidades dos agentes e seleção de padrões sociais. Caso fosse selecionado algum padrão social necessário ao sistema DBSitter-AS, estes seriam incorporados aos diagramas do projeto detalhado apresentados nesta iteração. E o

desenvolvimento prosseguiria com a implementação dos agentes. Para o exemplo do DBSitter-AS, estas atividades serão apresentadas em trabalhos futuros.

6.4 Considerações Finais

Neste capítulo foi apresentado um exemplo de uso do processo U_Tropos. Este não foi um estudo de caso completo, pois a validação de um processo é um trabalho que exige um tempo de preparação maior e não foi possível realizar durante esta dissertação. Foi utilizado como exemplo o sistema multiagente DBSitter-AS, que já tinha sido modelado anteriormente com a metodologia Tropos.

O exemplo foi dividido em um projeto com no mínimo cinco iterações: definição do sistema, análise dos requisitos, projeto da arquitetura global do sistema, projeto detalhado do sistema e construção do primeiro protótipo. Cada iteração foi, ou iniciada com atividades de requisitos ou foi precedida por outra iteração de requisitos. Isto garante que o sistema possa ser constantemente atualizado com requisitos que não foram elicitados completamente nas iterações anteriores. Estas iterações tiveram como objetivo apresentar um exemplo prático de um subconjunto de atividades do U-Tropos. Em um estudo de caso real existiriam outras iterações para análise de outros requisitos, refinamento do projeto arquitetural, projeto detalhado destes novos requisitos e implementação. Estas iterações poderiam ser desenvolvidas até a conclusão do desenvolvimento do sistema completo.

Os produtos de trabalho gerados para o sistema DBSitter-AS tiveram diferenças em relação a versão anterior [Silva, M. J. et al., 2007]. Uma justificativa para a diferença foi a aplicação de técnicas de elicitação e modelagem dos diagramas i*. Na versão de Silva, M. J. et al. (2007), não foi usada nenhuma técnica específica de análise dos requisitos projeto arquitetural. Nesta dissertação foi usado o PRIM para auxiliar a elicitação e análise dos requisitos (metas, planos, recursos) do sistema. Foram gerados três artefatos para elicitação de requisitos: A lista de partes interessadas, a lista de atividades e o cenário DIS. A versão anterior [Silva, M. J. et al., 2007] utilizou apenas as listas de partes interessadas e já iniciava a análise modelando os diagramas i*. Os documentos de elicitação permitiram maior detalhamento dos atores e de suas atividades gerando modelos de dependência de atores e modelos de raciocínio de atores mais detalhados.

O produto de trabalho Diagrama Arquitetural também teve uma configuração diferente da versão anterior. A técnica usada, baseada no framework SIRA, gerou o mesmo estilo

organizacional o joint-venture, mas o mapeamento dos componentes da arquitetura ficou diferente. Por exemplo, na primeira versão [Silva, M. J. et al., 2007] foi inserido, de forma *ad-hoc*, um componente somente para fazer o papel de gerenciamento das correções de falhas. Usando o processo U-Tropos, que segue as diretrizes do framework SIRA, foi identificado que existia um papel que poderia assumir esta responsabilidade, apenas acrescentando uma nova meta em suas metas iniciais (o Administrador de conhecimento).

Na iteração de projeto detalhado do protótipo do sistema, o diagrama arquitetural modelado em i* foi mapeado para diagramas em UML. Foram geradas seis visões do projeto do sistema multiagente que permitiu aos desenvolvedores identificar todas as características dos agentes e do ambiente em que eles estão inseridos.

Para modelagem destes diagramas foi utilizada a ferramenta TAOM4E [TAOM4E, 2007] para modelar os diagramas i*. Foi escolhido o TAOM4E por ser uma ferramenta livre, prática e que apresenta facilidades em seu uso. Além de está sendo muito usada pelos usuários da metodologia Tropos. Para geração dos modelos UML foi utilizado o Jude UML Modeling Tool [JUDE, 2007].

Com este exemplo, foi possível ter informações preliminares sobre a viabilidade de uso das recomendações do processo U-Tropos para desenvolvimento de sistemas multiagentes. Em particular, com o acréscimo das novas técnicas foi observado que os produtos de trabalho foram mais completos e precisos. Mais completo, por que as técnicas de elicitação sugerem meios de levantar informações que não eram claramente especificadas nas versões anteriores. Além disso, o projeto detalhado engloba uma visão mais completa dos conceitos de agentes e organizações. Mais preciso, por que com o acréscimo do framework SIRA, foi possível definir uma arquitetura baseada em uma técnica formal de análise de grupos sociais. Anteriormente a definição da arquitetura se baseava no conhecimento do arquiteto sobre o sistema desenvolvido.

Algumas técnicas podem ser custosas de desenvolver, tais como os métodos do framework SIRA. Mas este conjunto de métodos pode ser facilmente automatizado. Para o exemplo desta dissertação, todos os cálculos foram realizados usando planilhas, o que facilitou a geração dos resultados das matrizes usadas.

Na próxima seção são apresentadas as considerações finais desta dissertação e os trabalhos futuros e novas direções sobre esta pesquisa.

Capítulo 7 – Conclusões e Trabalhos Futuros

Neste capítulo são apresentadas as considerações finais sobre o desenvolvimento deste trabalho, assim como as principais contribuições, abrangência e limitações encontradas. Serão mostrados alguns trabalhos futuros que complementem esta proposta e que possam direcionar futuras pesquisas nesta área.

7.1 Conclusão

Nesta dissertação foi apresentado o U-Tropos, um processo iterativo e incremental, que propõe um conjunto de atividades para guiar o desenvolvimento de sistemas multiagentes usando a abordagem Tropos. Este processo foi especificado usando o SPEM, que é uma linguagem de especificação muito utilizada para modelar processos do paradigma orientados a objetos e tem sido adotada pela comunidade de sistemas orientado a agentes para modelar suas metodologias. Conforme recomendado pela especificação de processo do SPEM, o U-Tropos abordou os aspectos dinâmicos do ciclo de vida do desenvolvimento do software e os aspectos estruturais que descrevem o conteúdo metodológico.

Na dimensão dinâmica ou temporal foi proposto um modelo de processo iterativo, que divide as fases do desenvolvimento do software em iterações que possibilitam a construção incremental dos artefatos usados. Na dimensão estrutural, foram acrescentadas à metodologia Tropos novas técnicas, guias e modelos. Os guias permitem melhorar o entendimento do que já é usado na abordagem e as novas técnicas cobrem lacunas existentes na proposta atual. Como guias, foram acrescentados métodos para direcionar a construção do modelo i* nas elicitação de requisitos. Algumas novas técnicas foram incorporadas: o framework SIRA [Bastos, 2005] para possibilitar a derivação do projeto arquitetural a partir dos requisitos do sistema; e um conjunto de modelos da UML para especificar as visões de projeto detalhado [Silva, C.T.L.L., 2007a], que são a arquitetura, o ambiente, as intenções dos agentes, as interações entre os agentes, o raciocínio lógico usado pelos agentes para atingir suas metas e os seus planos de ação.

Ainda na dimensão estrutural, o U-Tropos foi dividido em oito disciplinas que agrupam as características do processo por temas comuns. As fases da abordagem Tropos foram traduzidas em disciplinas: requisitos iniciais, requisitos finais, projeto arquitetural, projeto detalhado e implementação. Outras três disciplinas que fazem parte da estrutura de desenvolvimento de softwares foram acrescentadas: o gerenciamento, verificação e suporte. Estas três últimas não tiveram o conteúdo detalhado nesta dissertação, pois envolvem pesquisas fora do escopo desta dissertação, tais como: gestão de projeto, testes e verificação de software multiagentes e atividades de suporte ao desenvolvimento, como medições, gerenciamento de configuração e ambiente, entre outras.

Para ilustrar o uso do processo, o U-Tropos foi aplicado em um exemplo para construir os modelos de desenvolvimento de um sistema multiagentes. O DBSitter-As foi modelado originalmente com uma versão híbrida de atividades das versões da metodologia Tropos'02 e Tropos'04. Com este exemplo, foi possível identificar a divisão do desenvolvimento em iterações, a aplicação das atividades propostas no processo e os produtos de trabalho gerados. Observou-se que os produtos de trabalho gerados nas disciplinas de requisitos e projeto sofreram alterações. Isto ocorreu devido à aplicação das novas técnicas, que permitem um maior detalhamento dos modelos construídos.

7.2 Contribuições

As principais contribuições desta dissertação podem ser resumidas como:

- Foi realizada uma análise das versões da abordagem Tropos e sua comparação em termos de seqüência de atividades e produtos de trabalho gerados.
- Foi realizado um levantamento e análise de modelos de processos aplicados a metodologias orientadas a agentes, para ilustrar o interesse que a comunidade de desenvolvimento de sistemas multiagentes tem tido por este assunto.
- Foram analisadas técnicas recentes de melhorias para a abordagem Tropos cobrindo lacunas existentes nos requisitos, projeto arquitetural e projeto detalhado.
- Foram identificados guias e diretrizes para auxiliar a elicitação e documentação de requisitos em Tropos usando o i*.
- Foi proposta a especificação em SPEM de um processo unificado para Tropos, abordando não somente as questões estruturais da metodologia, mas o ciclo de vida de desenvolvimento de um sistema multiagente.
- Foi apresentado um exemplo de uso e aplicação do processo U-Tropos para desenvolvimento de um sistema multiagentes.

7.3 Considerações finais e Trabalhos Futuros

Entretanto algumas limitações e restrições do processo U-Tropos podem ser citadas. Um dos pontos principais é que o processo proposto não foi validado em um estudo de caso real, onde possam ser definidas iterações reais e avaliado se as atividades propostas estão seguindo uma

seqüência natural e adequada ao desenvolvimento de sistemas multiagentes. A opinião do usuário final e da equipe de desenvolvimento seria de imenso valor para validação do processo. Outra questão está associada ao não detalhamento das disciplinas de gerenciamento, verificação e suporte. Técnicas tradicionais de gerenciamento, testes e suporte de sistemas orientados a objetos que usam o modelo iterativo e incremental podem ser aplicadas ao processo, porém é necessário um estudo sobre as melhores técnicas aplicadas ao desenvolvimento de sistemas multiagentes.

Durante as pesquisas realizadas nesta dissertação, foram identificadas outras técnicas que não foram acrescidas nesta versão do U-Tropos. O estudo das capacidades dos agentes [Penserini et al., 2006] é um bom exemplo que poderia ser acrescentada ao U-Tropos. Não foi incluído por que seriam necessários mais estudos sobre a integração e unificação com as técnicas de Silva, C.T.T.L. (2007). Estes estudos serão realizados em trabalhos futuros. Um ponto menos impactante, mas que também têm importância no processo se refere à versão das ferramentas usadas. Foi usada a versão 1.4 do SPEM para descrever o processo. Em 2008, foi publicada uma nova versão [SPEM, 2008] que pode conter novas características importantes a descrição de um processo.

Como trabalho futuro, terá prioridade o desenvolvimento das disciplinas de gerenciamento, verificação e suporte. O gerenciamento é uma disciplina de grande alcance e existem algumas pesquisas realizadas neste sentido. Os trabalhos já citados anteriormente, sobre riscos [Asnar; Gorgini & Mylopoulos, 2006], segurança [Giorgini & Mouratidis, 2005] e gerenciamento de requisitos [Pinto, 2008] aplicados a metodologia Tropos serão os pontos de partida para as novas pesquisas.

Outro foco de trabalho será a aplicação do U-Tropos em alguns estudos de casos reais que possam demonstrar os benefícios do uso do processo e as adaptações necessárias a desenvolvimento de sistemas multiagentes.

O desenvolvimento de uma ferramenta de apoio à conversão dos modelos de requisitos para o projeto arquitetural, seguindo o procedimento do framework SIRA, é necessário para facilitar o uso deste framework e gerar um modelo interessante à equipe de desenvolvimento.

A disciplina de implementação pode ser mais detalhada, fazendo uma análise dos frameworks de desenvolvimento de sistemas multiagentes. Técnicas de transformação do projeto detalhado para o framework de implementação podem ser acrescentadas.

Dessa forma, será possível fornecer um processo completo e unificado para guiar o desenvolvimento de sistemas multiagentes, usando os conceitos de Tropos.

Referências

AGILE Manifesto. Disponível em < http://agilemanifesto.org>. Acesso em agosto de 2008.

AMBLER, S. “Agile Modelling: The Official Agile Modelling (AM) Site.” Disponível em <http://www.agilemodeling.com>. Acesso em agosto de 2008.

ASNAR, Y.; GIORGINI, P.; MYLOPOULOS, J. “Risk Modelling and Reasoning in Goal Models”. Technical Report DIT-06-008. DIT - University of Trento. 2006.

BECK, K., “Test-Driven Development by Example.”. Addison Wesley. 2003.

BASILI, V. R.; TURNER, A. J. “Iterative Enhancement: A Practical Technique for Software Development”. IEEE Trans. Software Engineering, 1,4, 390-396. 1975.

BASTOS, L. “Integration of System Requirements and Multi-Agent Software Architecture”. Tese (Doutorado em Ciência da Computação). Universidade Federal de Pernambuco, Centro de Informática, Recife-PE. 2005.

BASTOS, L.; CASTRO, J. "Systematic Integration between Requirements and Architecture". In: Ricardo Choren; Alessandro Garcia; Carlos Lucena; Alexander Romanovsky. (Org.). Software Engineering for Multi-Agent Systems III: Research Issues and Practical Applications. : Springer Verlag. Lecture Notes in Computer Science LNCS 3390, v. 3390, p. 85-103. 2005.

BASTOS, L.; CASTRO, J.; MYLOPOULOS, J. "Deriving Multi-Agent Organizational Architectures from Requirements". In Proc. of the Second Workshop on Software Engineering for Agent-oriented Systems - SEAS´06, p. 13-24. Florianopolis. 2006.

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.

BECK, K. “Extreme Programming Explained: Embrace Change”. Addison-Wesley. 1999.

M. BEEDLE et al. “SCRUM: An Extension Pattern Language for Hyperproductive Software Development”. Pattern Languages of Program Design, vol. 4, pp. 637-651. 1999.

BELLIFEMINE, F.; CAIRE, G.; POGGI, A.; RIMASSA, G. “JADE - A White Paper”. In: Special issue on JADE of the TILAB Journal EXP. 2003.

BELINA, F.; HOGREFE, D. "The CCITT-Specification and Description Language SDL". Computer Networks and ISDN Systems 16 (1988/89) 311-341, North-Holland. 1988.

BERNON, C.; GLEIZES, M.P.; PEYRUQUEOU, S.; PICARD, G. "ADELFE, a Methodology for Adaptive Multi-Agent Systems Engineering". Third International Workshop "Engineering Societies in the Agents World" (ESAW-2002), 16-17. 2002.

BOEHM, B. “A Spiral Model of Software Development and Enhancement”. Computer, 20(9), 61-72. 1987.

BRESCIANI, P.; GIORGINI, P.; GIUNCHIGLIA, F.; MYLOPOULOS, J.; PERINI, A. “Tropos: An Agent-Oriented Software Development Methodology”. Autonomous Agents and Multi-Agent Systems, 8(3):203–236. 2004.

CAIRE, G.; CHAINHO, P.; EVANS, R.; GARIJO, F.; GOMEZ SANZ, J.; KEAMEY, P.; LEAL, F.; MASSONET, P.; PAVON, J.; STARK, J. "Agent Oriented Analysis using MESSAGE/UML". In Proc. of the Second International Workshop on Agent-Oriented Software Engineering (AOSE-2001). Montreal, Canada. 2001.

CASTRO, J.; KOLP, M.; MYLOPOULOS, J. “Towards Requirements-Driven Information Systems Engineering: The Tropos Project”. Information Systems Journal, Elsevier, Vol 27: 365-89. 2002.

CERNUZZI, L.; COSSENTINO, M.; ZAMBONELLI, F. “Process Model for agent-based development”. Engineering Application of Artificial Intelligence, vol. 18 205-222. 2005.

CHUNG, L.; NIXON, B.; YU, E.; MYLOPOULOS, J. “Non-Functional Requirements in Software Engineering”. Kluwer Publishing. 2000.

CHELLA, A.; COSSENTINO, M.; SABATUCCI, L.; V. SEIDITA. "From passi to agile passi: tailoring a design process to meet new needs". In IEEE/WIC/ACM Conference on Intelligent Agent Technology (IAT 2004), Beijing - China, 20-24. 2004.

CLYNCH, N.; COLLIER, R., “SADAAM: Software Agent Development – An Agile Methodology”. In Proceedings of the Workshop of Languages, methodologies, and Development tools for multi-agent systems (LADS’007), Durham, U.K. 2007.

CMMI® for Development, Version 1.2, Carnegie Mellon University. CMU/SEI-2006-TR-008. Pittsburgh, PA 15213-3890. 2006.

COCKBURN, A. “Agile Software Development”. Addison-Wesley. 2000.

COLEMAN, D.; ARNOLD, P.; BODOFF, S.; DOLLIN, C.; GILCHRIST, H.; HAYES, F.; JEREMAES, P. "Object-Oriented Development: The Fusion Method", Englewood Cliffs, NJ, Prentice Hall. 1994.

COLLINOT, A.; DROGOUL, A.; BENHAMOU, P. "Agent oriented design of a soccer robot team". In Proc. of the Second International Conference on MultiAgent Systems (ICMAS-96), Kyoto, Japan, pages 41--47. 1996.

COSSENTINO, M.; POTTS, C. "A CASE tool supported methodology for the design of multi-agent systems". The 2002 International Conference on Software Engineering Research and Practice (SERP'02). Las Vegas (NV), USA. 2002.

COSSENTINO M.; SEIDITA V. “Composition of a New Process to Meet Agile Needs Using Method Engineering”. In: Software Engineering for Large Multi-Agent Systems vol. III, Lecture Notes in Computer Science 3390, Springer-Verlag, pp.36-51. 2004.

CRUZ NETO, G. G. "Estudos qualitativos para elicitação de requisitos: uma abordagem que integra análise sócio-cultural e modelagem organizacional". Tese (Doutorado em Ciência da Computação). Universidade Federal de Pernambuco, Centro de Informática, Recife-PE. 2008.

DEBENHAM, J.K.; HENDERSON-SELLERS, B. “Full lifecycle methodologies for agent-oriented systems – the extended OPEN Process Framework”. In Proceedings Agent-Oriented Information Systems (eds. P. Giorgini, Y. Lesperance, G. Wagner and E. Yu), Toronto, 87-101. 2002.

DELOACH, S.A. "Engineering Organization-Based Multiagent Systems". A. Garcia et al. (Eds.): SELMAS 2005, LNCS 3914, pp. 109 – 125. 2006.

FERBER, J. “Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence”. Addison-Wesley. 1999.

FIPA - Foundation for Intelligent Physical Agents. Disponível em < http://www.fipa.org>. Último acesso em Novembro de 2007.

FIPA Methodologies. Disponível em <http://www.pa.icar.cnr.it/~cossentino/FIPAmeth/>. Último acesso em Janeiro de 2008.

FIRESMITH, D.G.; HENDERSON-SELLERS, B. “The OPEN Process Framework: an Introduction”. Addison-Wesley, Harlow–England. 2002.

FUGGETTA, A. “Software Process: A Roadmap”. In Proc. of the Future of Software Engineering, ICSE’2000, Limerick, Ireland. 2000.

FUXMAN, A.; GIORGINI, P.; KOLP, M.; MYLOPOULOS, J. “Information systems as social structures”. In Proc. of the 2nd Int. Conf. on Formal Ontologies for Information Systems, FOIS’01, Ogunquit, USA. 2001.

GARCIA-OJEDA, J.C.; DELOACH, S.A.; ROBBY; OYENAN, W.H.; VALENZUELA, J. “O-MaSE: A Customizable Approach to Developing Multiagent Development Processes”. In 8th International Workshop on Agent Oriented Software Engineering, Honolulu HI. 2007.

GILB, T. “Evolutionary Development,” ACM Software Eng. Notes, p. 17. 1981

GIORGINI, P.; MOURATIDIS, H. “Secure Tropos: A Security-Oriented Extension of the Tropos Methodology”. In Journal of Autonomous Agents and Mult-Agent Systems. Kluwer Academic Publisher. 2005.

GRAHAM, I.; HENDERSON-SELLERS, B.; YOUNESSI, H. "The OPEN Process Specification". Harlow, UK: Addison-Wesley, 314pp. 1997.

GRAU, G.; FRANCH, X.; MAYOL, E.; AYALA, C.; CARES, C.; HAYA, M.; NAVARRETE, F.; BOTELLA, P.; QUER, C. “RiSD: A Methodology for Building i* Strategic Dependency Models”. In 17th International Conference on Software Engineering and Knowledge Engineering. SEKE'05. Howard International House, Taipei, Taiwan. 2005.

GRAU, G.; FRANCH, X.; MAIDEN, N. “A Goal-Based Round-Trip Method for System Development”. In 11th International Workshop on Requirements Engineering: Foundations for Software Quality (REFSQ'05). ISBN:3-922602-98-3. 2005.

GRAU, G.; FRANCH, X.; AVILA, S. “J-PRiM: A Java Tool for a Process Reengineering i* Methodology”. In 14th IEEE International Requirements Engineering Conference (RE'06). 2006.

GRAU, G.; HORKOFF, J.; YU, E.; ABDULHADI, S. “i* Guide”. Disponível em <http://istar.rwth-aachen.de/tiki-view_articles.php>. Último acesso em Julho 2008.

HENDERSON-SELLERS, B.; GIORGINI, P. “Agent-Oriented Methodologies”. Published by: Idea Group, Inc. 2005.

HANNEMAN, R. A.; RIDDLE, M. “Introduction to social network methods”. Riverside, CA: University of California, Riverside (published in digital form at http://faculty.ucr.edu/ ~hanneman/). 2005.

HAYDEN, S.; CARRICK, C.; YANG, Q. “Architectural design patterns for multiagent coordination”. In Proc. of the 3rd Int. Conf. on Autonomous Agents, Agents’99, Seattle. 1999.

HUGET, M.P., “Nemo: An Agent-Oriented Software Engineering Methodology”. In Proceedings of OOPSLA Workshop on Agent-Oriented Methodologies, John Debenham, Brian Henderson-Sellers, NicholasJennings and James Odell, Seattle, USA. 2002.

IGLESIAS, C.; GARIJO, M.; GONZALEZ, J. C.; VELASCO, J. R. "Analysis and design of multiagent systems using MAS-CommonKADS". In AAAI'97 Workshop on Agent Theories, Architectures and Languages, Providence, RI. 1997.

ISO/IEC 12207: 1994: Information Technology Software Life Cycle Processes. ISO/IEC. 1994.

JACK Intelligent Agents. Disponível em <http://www.agent-software.com>. Último acesso em Julho de 2007.

JUDE/Community - Free UML Modeling Tool, Version 3.0.1. Disponível em <http://jude.change-vision.com/jude-web/index.html>. Último acesso em Julho 2007.

KENDALL, E.; MALKOUN, M.; JIANG, C. “A Methodology for Developing Agent Based Systems”. DAI 1995: 85-99. 1995.

KINNY, D.; GEORGEFF, M.; RAO, A. "A Methodology and Modelling Technique for Systems of BDI Agents". In Van Der Velde, W., Perram, J. (eds.): Agents Breaking Away: Proceedings of MAAMAW’96, (LNAI Vol. 1038). Springer-Verlag: Heidelberg, Germany. 1996.

KNUBLAUCH, H. “Extreme Programming of MultiAgent Systems”. In Proc. of the First International Joint Conference on Autonomous Agents & MultiAgent Systems (AAMAS 2002), Bologna, Italy. 2002

KOLP, M.; CASTRO, J.; MYLOPOULOS, J. “A social organization perspective on software architectures”. In Proc. of the 1st Int. Workshop from Software Requirements to Architectures, STRAW’01, pages 5–12, Toronto, Canada. 2001.

KOLP, M.; GIORGINI, P.; MYLOPOULOS, J. “Multi-Agents Architectures as Organizational Structures”. In Journal of Autonomous Agents and Multi-Agent (JAAMAS), 13(1):3-25, Springer. 2006.

KRUCHTEN, P. “Rational Development Process,” Crosstalk: J. Defense Software Eng., July 1996; < http://www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1996/07/>. 1996.

LESPERANCE, Y.; KELLEY, T.; MYLOPOULOS, J.; Yu, E. “Modeling dynamic domains with ConGolog”. In Proc. of the 11th Int. Conf. on Advanced Information Systems Engineering CAiSE’99, pages 108–123, Heidelberg, Germany. 1999.

LIND, J. "The Massive Development Method for Multiagent Systems". In J. Bradshaw and G. Arnold, editors, Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 2000). Manchester, UK. 2000.

MACIEL, P. R. “DBSitter-AS: um Framework Orientado a Agentes para Construção de Componentes de Gerenciamento Autônomo para SGBD”. Dissertação (Mestrado em ciência da computação), Universidade Federal de Pernambuco, Centro de Informática, Recife-PE. 2007.

MAY E.L.; ZIMMER, B. A. “The Evolutionary Development Model for Software”. In proc. of the Hewlett-Packard Journal. 1996.

MILLS, H.; O'NEILL, D. “The management of Software Engineering”. IBM Systems J., vol. 24, no. 2, pp. 414-477. 1980.

MILLS, H.D.; DYER, M.; LINGER, R.C. “Cleanroom Software Engineering, IEEE Software, 4, 5, 19-25. 1987.

MYLOPOULOS, J.; CASTRO, J. F. B. “Tropos: A Framework for Requirements-Driven Software Development”. Brinkkemper, J. and Solvberg, A. (eds.), Information Systems Engineering: State of the Art and Research Themes, Springer-Verlag, pages 261-273. 2000.

NARDI, B. A. “Context and Consciousness: Activity Theory and Human-Computer Interaction”. London: MIT Press. 1996.

NARDINI, E.; MOLESINI, A.; OMICINI, A.; DENTI, E. “SPEM on Test: the SODA Case Study”, 23th ACM Symposium on Applied Computing (SAC 2008), 16-20. 2008.

ODELL, J.; VAN DYKE, P.; BAUER, B. "Extending UML for agents". In proc. of the 2nd International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS’00). Austin, USA. 2000.

OLIVEIRA, A.; PADUA A.; CYSNEIROS, L. M. “Defining Strategic Dependency Situations in Requirements Elicitation”. IX Workshop on Requirements Engineering. Rio de Janeiro, Brazil. 2006.

OMICINI, A. "Soda: Societies and infrastructures in the analysis and design of agent based systems". In P. Ciancarini and M. Wooldridge, editors, Agent-Oriented Software Engineering. Proceedings of the First International Workshop (AOSE-2000). Springer-Verlag: Berlin, Germany. 2000.

Object Management Group. Disponível em <http://www.omg.org>. Último acesso em November 2007.

PADGHAM, L.; WINIKOFF, M. "Prometheus: A Methodology for Developing Intelligent Agents". Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002). Bologna, Italy. 2002.

PEARSON - Pearson Correlation Coefficient. Disponível em <http://www.cmh.edu/stats/definitions/ correlation.htm>. Último acesso em Julho de 2007.

PENSERINI, L.; PERINI, A.; SUSI, A.; MYLOPOULOS, J. “From Stakeholder Intentions to Software Agent Implementations”. In 18th Conference on Advanced Information Systems Engineering (CAiSE’06). LNCS, Springer-Verlag. 2006.

PINTO, R.C. “Improving Traceability in Agent Oriented Development”, Tese de doutorado, Universidade Federal de Pernambuco, Centro de Informática. 2008.

POKAHR, A.; BRAUBACH, L.; LAMERSDORF, W. "Jadex: A bdi reasoning engine". In: Bordini, J.D.R., Dastani, M., Seghrouchni, A.E.F. (eds.) Multi-Agent Programming,pp. 149–174. Springer Science, Business Media Inc. 2005.

RUMBAUGH et al. "Object oriented modeling and design". Prentice Hall. 1991.

RAO, A. S., GEORGEFF, M. P. “Modelling rational agents within a BDI-architecture”, in Knowledge Representation and Reasoning (KRR-91) Conference, San Mateo CA. 1991.

ROYCE, W.W.: “Managing the Development of Large Software Systems: concepts and techniques”. In Proc. IEEE WESTCON, Los Angeles, CA, Cap 3. 1970.

RUDOLPH, E., GRABOWSKI, J.; GRAUBMANN, P. "Tutorial on message sequence charts (MSC)". In Proceedings of FORTE/PSTV’96 Conference, October. 1996.

RUP - Rational Unified Process. Disponível em <http://www-128.ibm.com/ developerworks/ rational/library/253.html>. Último acesso em novembro de 2006.

SCHREIBER, A.; WIELINGA, B.; AKKERMANS, J.M.; VAN DE VELDE, W.; DE HOOG R. “CommonKADS. A Comprehensive Methodology for KBS Development”. IEEE Expert, 9(6): 28--37. 1994.

SILVA, M. J.; MACIEL, P. R.; PINTO, R. C.; ALENCAR, F.; TEDESCO, P.; CASTRO, J. “Extracting the Best Features of Two Tropos Approaches for the Efficient Design of MAS”. In proc. of the IDEAS’07, Isla de Margarita, Venezuela. 2007.

SILVA, C. T. L. L. “Separating Crosscutting Concerns in Agent Oriented Detailed Design: The Social Patterns Case”. Tese (Doutorado em Ciência da Computação). Universidade Federal de Pernambuco, Centro de Informática. 2007. (2007a)

SILVA, C. T. L. L.; ARAUJO, J.; MOREIRA, A.; CASTRO, J. "Designing Social Patterns using Advanced Separation of Concerns". In: CAISE'07 - 19th Conference on Advanced Information Systems Engineering, 2007, Trondheim, Norway. Proceedings. Heidelberg : Springer Verlag - LNCS 4495, v. 4495. p. 309-323. 2007. (2007b).

SILVA, C. T. L. L; CASTRO, J.; TEDESCO, P.; ARAUJO, J.; MOREIRA, A.; MYLOPOULOS, J. “Improving Multi-Agent Architectural Design”. In: Choren, R.; Garcia, A.; Giese, H.; Leung, H.-f.; Lucena, C.; Romanovsky, A.. (Org.). Software Engineering for

Multi-Agent Systems V: Research Issues and Practical Applications. : Springer Verlag - LNCS 4408, v. 4408, p. 165-184. 2007. (2007c).

SOMMERVILLE, I. “Engenharia de Software”, Pearson Addison Wesley. 6th Ed. 2005.

SPEM - Object Management Group: Software Process Engineering Metamodel (SPEM) Specification, Version 1.1. Disponível em <http://www.omg.org/docs/formal/05-01-06.pdf>. Último acesso em Dezembro de 2007.

STAPLETON, J. “DSDM: Dynamic Systems Development Method”. Addison-Wesley. 1997.

STRAUSS; CORBIN. “Basics of Qualitative Research: Techniques and procedures for developing grounded theory”. Sage. 1998.

SUSI, A.; PERINI, A.; MYLOPOULOS J. “The Tropos Metamodel and its Use”. Informatica, 29(4):401–408. 2005.

TAOM4E - Tool for Agent Oriented Modeling, Version 0.5. Disponível em <http://sra.itc.it/tools/taom4e/>. Último acesso em Outubro 2007.

UML - Object Management Group: Unified Modeling Language (UML): Superstructure, Version 2.0. Disponível em <http://www.omg.org/docs/formal/05-07-04.pdf>. Último acesso em Novembro 2007.

WAUTELET Y.; KOLP M.; ACHBANY Y. “S-Tropos, an Iterative SPEM-Centric Software Project Management Process”, Working Paper IAG. 2005.

WEBSTER, I.; AMARAL, J.; CYSNEIROS, L.M. "A Survey of Good Practices and Misuses for Modelling with i* Framework". In proceedings of the Workshop em Engenharia de Requisitos (WER2005) 148-160, Porto, Portugal. 2005.

WHITESTEIN Information Technology Group AG. Disponível em <http://www.whitestein.com>. Último acesso em janeiro de 2008.

WIRFS-BROCK, R.; WILKERSON, B.; WIENER, L. "Designing Object-Oriented Software". Prentice-Hall. 1990.

WOOD, M.; DELOACH, S., “An Overview of the Multiagent Systems Engineering Methodology”, In P. Ciancarini and M. Wooldridge, editors, Agent-Oriented Software Engineering - First International Workshop (AOSE), Limerick, Ireland, June 10, 2000. LNCS. Vol. 1957, Springer Verlag, Berlin. 2001.

WOOLDRIDGE, M. “An Introduction to MultiAgent Systems”. John Wiley and sons, LTD, Chichester, England. 2002.

XML - Extensible Markup Language. Disponível em <http://www.w3.org/XML/>. Último acesso em fevereiro de 2008.

YU, E. “Modelling Strategic Relationships for Process Reengineering”. PhD thesis. University of Toronto, Department of Computer Science. 1995.

ZAMBONELLI, F.; JENNINGS, N. R.; WOOLDRIDGE, M. “Developing Multiagent Systems: the Gaia Methodology”, in ACM Trans on Software Engineering and Methodology, 12(3): 417-470. 2003.

Apêndice A

Descrição detalhada das atividades das disciplinas Requisitos iniciais, Requisitos Finais, Projeto arquitetural, Projeto detalhado e Implementação do processo U-Tropos.

Disciplina Requisitos Iniciais

A seguir as atividades da disciplina Requisitos iniciais (Figura A-1) são detalhadas apresentando uma breve descrição, a relação das etapas sugeridas para realização do trabalho, a identificação dos produtos de entrada e saída, os guias usados e o papel de processo responsável pela execução da atividade. Os guias são escolhidos, conforme o contexto do sistema que é desenvolvido e serão mais detalhados no final desta seção.

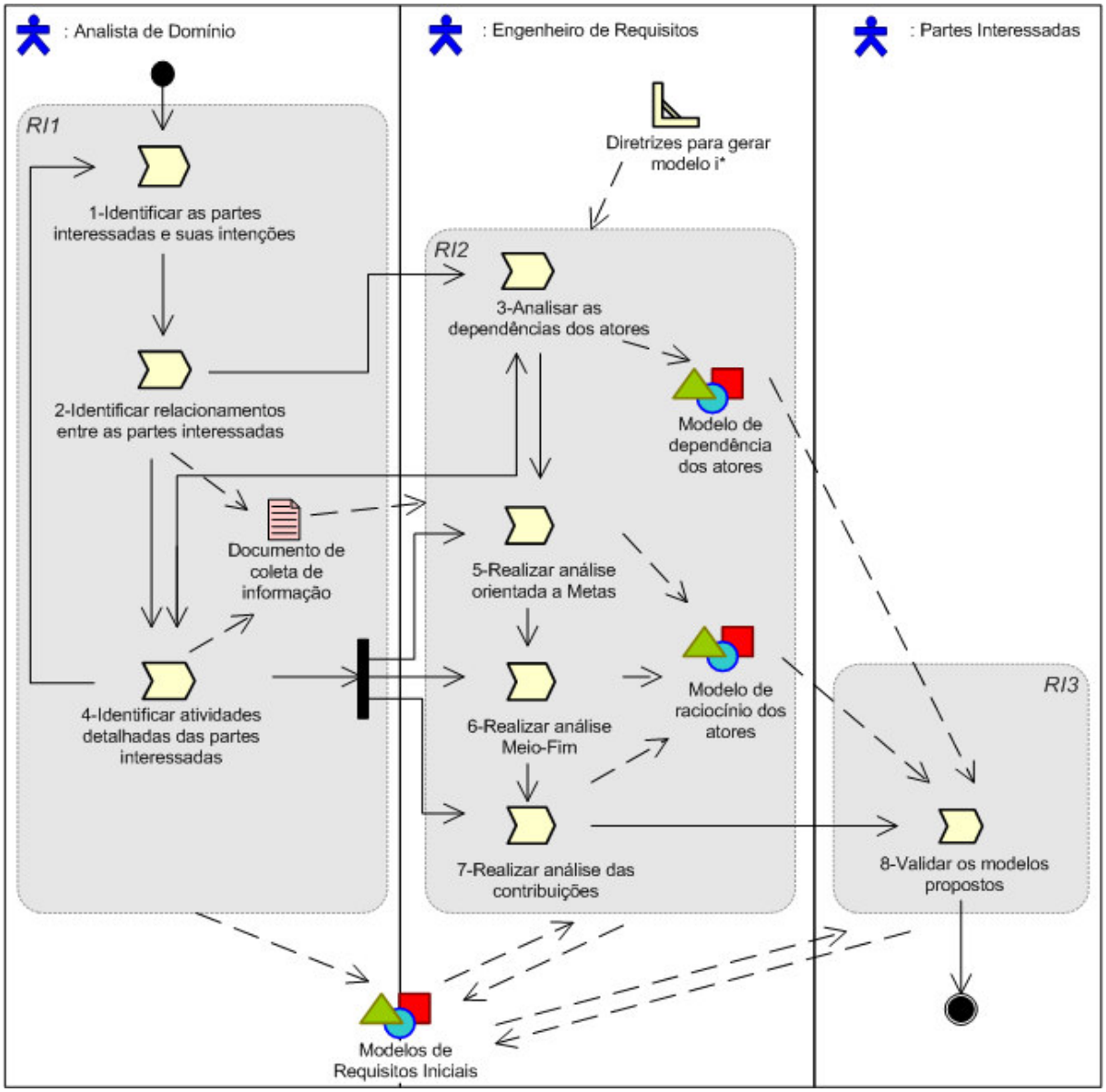


Figura A-1 – Atividades da disciplina Requisitos Iniciais

1 - Identificar as partes interessadas e suas intenções: Esta atividade envolve a análise do ambiente onde o sistema será utilizado que é composto das partes interessadas no desenvolvimento do sistema. Esta análise identifica quais as intenções ou necessidades de cada parte interessada dentro do seu ambiente organizacional, independente do sistema. Podem-se utilizar com guias para esta atividade os métodos PRiM , SDSituation ou os Estudos qualitativos para elicitação de Requisitos (EQER) [Cruz Neto, 2008] para coletar e registrar as informações. As informações são registradas no Documento de coleta de informações, que pode conter as listas de atores, suas atividades e interações. A Tabela A-1 apresenta o detalhamento destas atividades.

Tabela A-1 - Detalhamento da atividade *Identificar as partes interessadas e suas intenções*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Identificar as partes interessadas. 2. Elicitar suas intenções e/ou necessidades. 3. Documentar os modelos de requisitos iniciais.	*Guias:* • PRiM, SDSituation ou EQER.	• Documento de coleta de Informações	Analista de Domínio

2 – Identificar relacionamentos entre as partes interessadas: Esta atividade envolve a identificação das relações que existem entre as partes interessadas constituintes da organização. Estas relações são identificadas como dependências que cada membro da organização tem em relação aos outros membros para cumprirem com suas atividades. Podem-se utilizar como guias para esta atividade os métodos PRiM, SDSituation ou os Estudos qualitativos para elicitação de Requisitos (EQER) [Cruz Neto, 2008] para coletar e registrar as informações. No Documento de coleta de informações são registradas as informações conforme o guia selecionado. A Tabela A-2 apresenta o detalhamento destas atividades.

Tabela A-2 - Detalhamento da atividade *Identificar relacionamentos entre as partes interessadas*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Identificar os outros membros que dependem de uma determinada parte interessada. 2. Identificar os outros membros dos quais uma determinada parte interessada depende. 3. Identificar as atividades ou informações que geram as dependências entre eles. 4. Documentar nos Modelos de Requisitos Iniciais.	*Produtos de Trabalho:* • PRiM, SDSituation ou EQER.	• Documento de coleta de Informações	Analista de Domínio

3 – Analisar as dependências estratégicas: Esta atividade envolve a análise individual das intenções/metas de cada parte interessada e a modelagem utilizando o framework i*. Como pré-requisito para início desta atividade, é necessária a seleção da versão do i* adotado, ou

seja o engenheiro de requisitos pode selecionar usar o i* original ou o i* do metamodelo do Tropos'04. As partes interessadas são modeladas como atores e as suas intenções são modeladas como metas a serem atingidas. Os relacionamentos são analisados e modelados como dependências entre os atores. Os mesmos guias das atividades anteriores podem ser usados, com acréscimo do RISD, que é específico para construção de diagramas SD. Neste processo podem ser identificadas novas metas que não foram estabelecidas na atividade anterior. A Tabela A-3 apresenta o detalhamento destas atividades.

Tabela A-3- Detalhamento da atividade *Analisar as dependências estratégicas*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Selecionar versão do i* para modelagem dos requisitos. 2. Modelar as partes interessadas e suas intenções como atores, metas e metas-soft segundo notação do framework i*. 3. Modelar cada dependência de um ator com outros atores. 4. Modelar cada dependência de outros atores com o ator analisado.	*Produtos de Trabalho:* • Documento de coleta de Informações. *Guias:* • Guia do i* • SDSituation, PRIM, RISD, EQER.	• Modelo de dependência de atores.	Engenheiro de Requisitos.

4 – Identificar as atividades detalhadas das partes interessadas: Esta atividade envolve o levantamento detalhado das atividades realizadas (e recursos utilizados) pelas partes interessadas para atingir suas metas. É desenvolvida a partir de entrevistas, identificação de documentação organizacional ou observação das atividades dos membros da organização. São usados os mesmos guias das atividades 1 e 2. A Tabela A-4 apresenta o detalhamento destas atividades.

Tabela A-4 - Detalhamento da atividade *Identificar as atividades detalhadas das partes interessadas*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Selecionar uma parte interessada (ou grupos que se relacionam) para detalhamento. 2. Identificar as atividades detalhadas realizadas pela(s) parte(s) interessada(s) selecionada(s). 3. Identificar os recursos utilizados pela(s) parte(s) interessada(s) selecionada(s). 4. Documentar no Documento de Requisitos Iniciais.	*Produtos de Trabalho:* • Documento de coleta de Informações. *Guias:* • SDSituation, PRIM ou EQER.	• Documento de coleta de Informações.	Analista de Domínio.

5 - Realizar análise orientada a metas: Esta atividade envolve o início da análise do raciocínio de cada parte interessada (modelada como um ator) para obtenção de suas metas. As metas identificadas são refinadas através de decomposição das metas em submetas até que

estas não possam mais ser refinadas. A Tabela A-5 apresenta o detalhamento destas atividades.

Tabela A-5 - Detalhamento da atividade *Realizar análise orientada a metas*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Selecionar uma parte interessada (ou grupos que se relacionam) para análise. 2. Refinar cada Meta em submetas através da decomposição (E/OU). 3. Refinar cada *Meta-soft* em *submetasofts* através da decomposição (E/OU). 4. Modelar o Modelo de raciocínio dos atores. OBS: Se for selecionada a versão do i* original [Yu,1995] para modelagem, esta atividade não é realizada. Esta é uma característica da análise usando a variante do i* usado no Tropos'04.	*Produtos de Trabalho:* • Modelo de dependência dos atores. • Documento de coleta de Informações.	• Modelo de raciocínio dos atores. • Modelos de Requisitos Iniciais (parcial)	Engenheiro de Requisitos

6 - Realizar análise meio-fim: Esta atividade envolve a análise meio-fim de cada meta (se realizada a atividade 5, são analisadas as metas do último nível de refinamento, isto é, as folhas das árvores de refinamento). Através da análise meio-fim serão identificadas as tarefas que devem ser executadas para obtenção das metas. Cada tarefa é refinada através de decomposições e novas tarefas são identificadas. Novos recursos usados para execução das atividades também são identificados aqui. A Tabela A-6 apresenta o detalhamento destas atividades.

Tabela A-6 - Detalhamento da atividade *Realizar análise meio-fim*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Selecionar uma meta para análise. 2. Aplicar análise meio-fim para identificar os planos/tarefas que excutam estas metas. 3. Refinar cada Plano/Tarefa em Subplanos/Subtarefas através da decomposição. 4. Refinar cada Plano/Tarefa em recursos usados através da decomposição. 5. Atualizar o Modelo de raciocínio dos atores. OBS: O Guia do i* é somente indicado, se selecionar a versão do i* original [Yu,1995].	*Produtos de Trabalho:* • Modelo de dependência dos atores. • Modelo de raciocínio dos atores. • Documento de coleta de Informações. *Guias:* • Guia do i* • SDSituation, PRIM, RISD ou EQER.	• Modelo de raciocínio dos atores. • Modelo de Requisitos Iniciais (parcial)	Engenheiro de Requisitos

7 - Realizar a análise das contribuições: Esta atividade envolve a análise das tarefas identificadas na atividade anterior em relação à satisfação das metas-soft. Cada tarefa realizada pode contribuir de forma positiva ou negativa para as metas-soft dos atores (partes

interessadas) analisados. Esta análise identifica quatro tipos de contribuições, que são: Contribuição positiva (++), Contribuição parcialmente positiva (+), Contribuição negativa (--) e Contribuição parcialmente negativa (-). Com esta análise podemos descobrir quais as tarefas que devem ser executadas pelos atores envolvidos. A Tabela A-7 apresenta o detalhamento destas atividades.

Tabela A-7 - Detalhamento da atividade *Realizar a análise da influência das tarefas/planos*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Selecionar uma meta para análise. 2. Analisar cada tarefa identificada e as suas contribuições (positivas, parcialmente positivas, parcialmente negativas, negativas) para as metas-soft. 3. Atualizar o Modelo de raciocínio dos atores. OBS: O Guia do i* é somente indicado, se selecionar a versão do i* original [Yu,1995].	<u>*Produtos de Trabalho:*</u> • Modelo de dependência dos atores. • Modelo de raciocínio dos atores. • Documento de coleta de Informações. <u>*Guias:*</u> • Guia do i*	• Modelo de raciocínio dos atores. • Modelos de Requisitos Iniciais (parcial)	Engenheiro de Requisitos

8 - Validar os modelos propostos: Após a análise das partes interessadas e de suas intenções e a geração dos produtos de trabalho da disciplina requisitos iniciais, estes produtos são apresentados a todas as partes interessadas no desenvolvimento do sistema para validação e refinamento da versão final dos requisitos iniciais do sistema. A Tabela A-8 apresenta o detalhamento destas atividades.

Tabela A-8 - Detalhamento da atividade *Validar os modelos propostos*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Apresentar os produtos de trabalho para todas as partes interessadas. 2. Coletar e analisar correções solicitadas. 3. Atualizar os produtos de trabalho acrescentando as correções sugeridas. 4. Validar as versões finais que serão usadas como base para a identificação dos requisitos funcionais e não funcionais. 5. Concluir o documento de requisitos iniciais.	<u>*Produtos de Trabalho:*</u> • Modelo de dependência dos atores. • Modelo de raciocínio dos atores. • Modelo de Requisitos Iniciais	• Modelo de dependência dos atores. • Modelo de raciocínio dos atores. • Modelo de Requisitos Iniciais	• Engenheiro de Requisitos • Analista de Domínio • Partes Interessadas

Disciplina Requisitos Finais

As atividades da disciplina Requisitos Finais (Figura A-2) são detalhadas apresentando uma breve descrição da atividade, a relação das etapas sugeridas para realização do trabalho, a identificação dos produtos de entrada e saída, os guias usados e o papel de processo responsável pela execução da atividade.

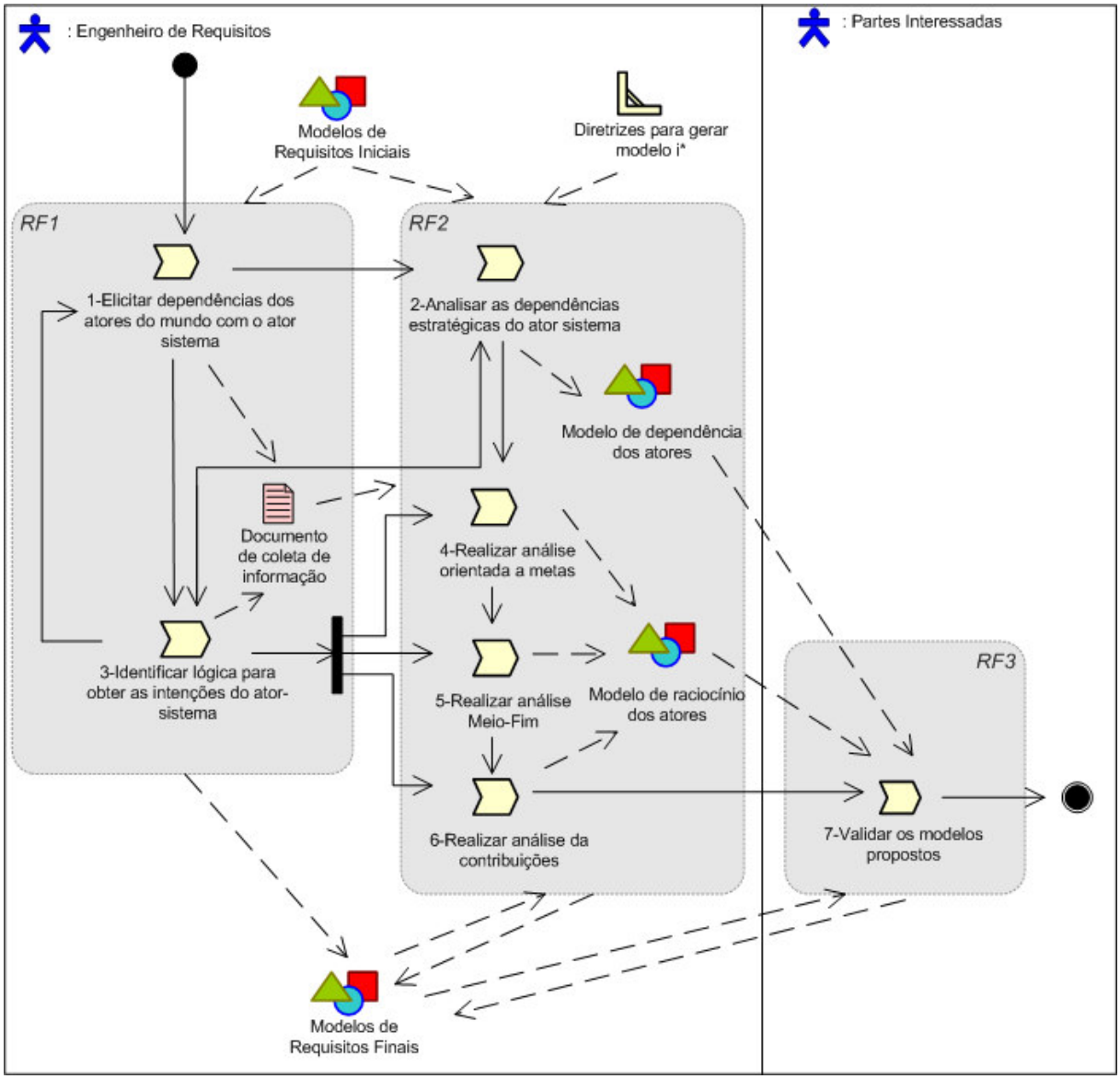


Figura A-2 – Atividades da disciplina Requisitos Finais

1 - Elicitar dependências dos atores do mundo com o ator sistema: Esta atividade envolve a identificação do(s) sistema(s) que será(ão) desenvolvido para atender as necessidades dos usuários e das partes interessadas no sistema (ou atores do mundo). Os relacionamentos destes atores, bem como as metas que eles esperam que o sistema atenda são identificados. Também

são identificados outros sistemas legados que podem se relacionar com o sistema a ser desenvolvido. As informações são registradas no Documento de coleta de informações. Podem-se utilizar com guias para esta atividade os métodos PRiM, SDSituation ou os Estudos qualitativos para elicitação de Requisitos (EQER) [Cruz Neto, 2008] para coletar e registrar as informações. A Tabela A-9 apresenta o detalhamento destas atividades.

Tabela A-9 – Detalhamento da atividade *Elicitar dependências dos atores do mundo com o sistema*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Analisar as necessidades das partes interessadas e identificar quais sistemas podem atender estas necessidades. 2. Identificar se existem outros sistemas legados que podem depender ou fornecer informações para o sistema. 3. Identificar as metas que as partes interessadas esperam obter do(s) sistema(s). 4. Identificar os atributos de qualidade desejados para o sistema. 5. Identificar tarefas e operações que as partes interessadas esperam que seja executado pelo sistema. 6. Identificar as informações que as partes interessadas e/ou sistemas legados trocam com o sistema. 7. Documentar no documento de requisitos finais.	*Produtos de Trabalho:* • Documento de Requisitos Iniciais. *Guias:* • PRiM, SDSituation ou EQER.	• Documento de coleta de Informações	• Engenheiro de Requisitos • Partes Interessadas (usuários)

2- Analisar as dependências estratégicas do ator sistema: Esta atividade envolve a análise e modelagem do sistema como um ator no framework i*. Primeiro é realizada a análise das metas que, os atores que representam as partes envolvidas esperam obter com o sistema e em seguida estas metas são modeladas como dependências com o ator sistema. A Tabela A-10 apresenta o detalhamento destas atividades.

Tabela A-10 – Detalhamento da atividade *Analisar as dependências estratégicas do ator sistema*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Analisar e modelar usando o framework i* o ator sistema e as metas que este deve atingir. 2. Modelar novos atores sistemas (legados) que se relacionam com o ator sistema. 3. Modelar as metas das partes interessadas / sistemas legados como dependências de metas. 4. Modelar os atributos de qualidade identificados como dependências de metas-soft. 5. Modelar as tarefas e operações identificadas como dependências de tarefas (ou planos). 6. Modelar as informações trocadas com o sistema como dependências de recursos. 7. Validar os modelos com as partes interessadas.	*Produtos de Trabalho:* • Documento de Requisitos Iniciais • Documento de coleta de Informações *Guias:* • i* guide • RISD, PRIM, SDSituation, EQER.	• Modelo de dependência dos atores (para ator sistema).	• Engenheiro de Requisitos

3 - Identificar meios para obter as intenções do ator-sistema: Esta atividade envolve a análise detalhada com os usuários, de todas as informações levantadas para o sistema. Esta

análise envolve a descoberta de como as metas desejadas para o sistema podem ser atingidas através das tarefas que são executadas e dos recursos utilizados. As atividades dos usuários são detalhadas de forma que sejam identificados todos os detalhes que envolvam as funcionalidades do sistema. São utilizadas técnicas de elicitação de requisitos, tais como entrevistas, observação do trabalho, análise de documentação, entre outros. Podem ser usados os mesmos guias sugeridos na atividade 1. A Tabela A-11 apresenta o detalhamento destas atividades.

Tabela A-11 – Detalhamento da atividade *Identificar meios para obter as intenções do ator-sistema*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Aplicar técnicas de elicitação de requisitos, tais como entrevistas, workshops, etnografia, Teoria da atividade para identificar as funcionalidades do sistema. 2. Validar as metas já definidas para o sistema. 3. Identificar como os usuários executariam os procedimentos para realizar atividades para atingir estas metas. 4. Identificar os recursos usados nestas atividades. 5. Identificar quais as exigências e restrições quanto à qualidade dos procedimentos realizados. 6. Documentar no documento de requisitos finais.	*Produtos de Trabalho:* • Documento de Requisitos Iniciais • Documento de coleta de Informações *Guias:* • PRIM, EQER, SDSituation	• Documento de coleta de Informações.	• Engenheiro de Requisitos • Partes Interessadas (usuários)

4 - Realizar análise orientada a metas: Esta atividade envolve a análise detalhada das metas desejadas para o sistema e a sua modelagem utilizando o framework i*. As metas podem ser refinadas através da análise orientada a metas usando decomposição *booleana*, até chegar às metas elementares que o sistema deve atender. Os atributos de qualidade também são analisados e modelados como metas-soft e refinados usando técnicas de análise do framework NFR [Chung et al., 2000]. A Tabela A-12 apresenta o detalhamento destas atividades.

Tabela A-12 – Detalhamento da atividade *Realizar análise orientada a metas*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Analisar cada meta e/ou meta-soft que o sistema deve atender. 2. Refinar as metas através de uma análise orientada a metas usando decomposição (E/OU). 3. Refinar as metas-soft através de uma análise de metas-soft usando decomposição (E/OU). 4. Modelo o diagrama de raciocínio estratégico para o ator sistema. 5. - Validar o modelo proposto com as partes interessadas. Obs.: Se selecionar a versão do i* original para modelagem, esta atividade não é realizada. Esta é uma característica da análise usando a variante do i* usado no Tropos'04.	*Produtos de Trabalho:* • Documento de coleta de Informações *Guias:* • Framework NFR • Guia do i* • RISD, PRIM, SDSituation, EQER.	• Modelo de raciocínio dos atores (para ator sistema)	• Engenheiro de Requisitos

5 – Realizar Analise Meio-Fim*:* Esta atividade envolve a definição das tarefas para obtenção de cada meta. Isto é feito através de uma análise meio-fim (do framework i*), onde são analisados tarefas (ou planos) alternativos para a obtenção de cada meta. Após a identificação das tarefas (dos planos), estes podem ser refinados através da decomposição. As tarefas que estiverem no último nível de refinamento são os requisitos detalhados que indicam o que poderá ser desenvolvido no sistema. A Tabela A-13 apresenta o detalhamento destas atividades.

Tabela A-13 – Detalhamento da atividade *Realizar análise meio-fim*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Realizar a análise meio-fim de cada meta do sistema para obtenção das tarefas (planos) alternativas para operacionalizar as metas. 2. Analisar as tarefas (planos) fornecidas pelos atores *partes interessadas* ou *sistemas legados* para atendimento de alguma meta. 3. Refinar as tarefas (planos), através de decomposição até que sejam obtidos elementos indivisíveis que representem os requisitos que devem ser desenvolvidos. 4. Analisar cada tarefa (plano) para identificar os recursos recebidos e fornecidos pelos atores (partes interessadas e sistemas legados). 5. Validar o modelo proposto com as partes interessadas.	*Produtos de Trabalho:* • Documento de coleta de Informações *Guias:* • i* guide • RISD, PRIM, SDSituation , EQER.	• Modelo de raciocínio dos atores (para ator sistema)	• Engenheiro de Requisitos

6 – Realizar análise das contribuições: Esta atividade envolve a análise de cada tarefa (ou plano) de acordo com as suas contribuições para os atributos de qualidade do sistema, ou seja, as *metas-soft*. As tarefas (ou planos) podem contribuir de forma positiva ou negativa para obtenção das metas-soft. Esta análise indica quais tarefas (ou planos) podem ser operacionalizados pelo sistema e qual o seu impacto no atendimento das necessidades das partes interessadas. A Tabela A-14 apresenta o detalhamento destas atividades.

Tabela A-14 – Detalhamento da atividade *Realizar análise da influência dos planos*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Identificar as tarefas (ou planos) e metas-soft do sistema. 2. Analisar cada tarefa (ou plano) em relação às metas-soft e identificar qual tipo de contribuições: positiva e/ou negativa. 3. Validar o modelo proposto com as partes interessadas.	*Produtos de Trabalho:* • Documento de Requisitos Finais (parcial) • Modelo de raciocínio dos atores (sistema) *Guias:* • Guia do i* • RISD, PRIM, SDSituation ou EQER.	• Modelo de raciocínio dos atores (sistema)	• Engenheiro de Requisitos

7 - Validar os modelos propostos: Após a análise das partes interessadas e de suas intenções e a geração dos produtos de trabalho de cada atividade, estes produtos são apresentados a todas as partes interessadas no desenvolvimento do sistema para validação e refinamento da versão final dos requisitos finais do sistema. Uma reunião de validação do modelo final a ser trabalhado nas próximas iterações também deve ser realizada. A Tabela A-15 apresenta o detalhamento destas atividades.

Tabela A-15 - Detalhamento da atividade *Validar os modelos propostos*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Apresentar os produtos de trabalho para todas as partes interessadas. 2. Coletar e analisar mudanças. 3. Atualizar os produtos de trabalho acrescentando as mudanças sugeridas 4. Validar as versões finais que serão usadas como base para o desenvolvimento do sistema. 5. Concluir o documento de requisitos finais.	*Produtos de Trabalho:* • Modelo de dependência de atores. • Modelo de raciocínio de atores. • Modelo de Requisitos Finais.	• Modelo de dependência de atores. • Modelo de raciocínio de atores. • Modelo de Requisitos Finais.	• Engenheiro de Requisitos • Partes Interessadas • Usuários

Disciplina Projeto Arquitetural

As atividades da disciplina Projeto arquitetural (Figura A-3) são detalhadas apresentando uma breve descrição da atividade, a relação das etapas sugeridas para realização do trabalho, a identificação dos produtos de entrada e saída, os guias usados e o papel de processo responsável pela execução da atividade.

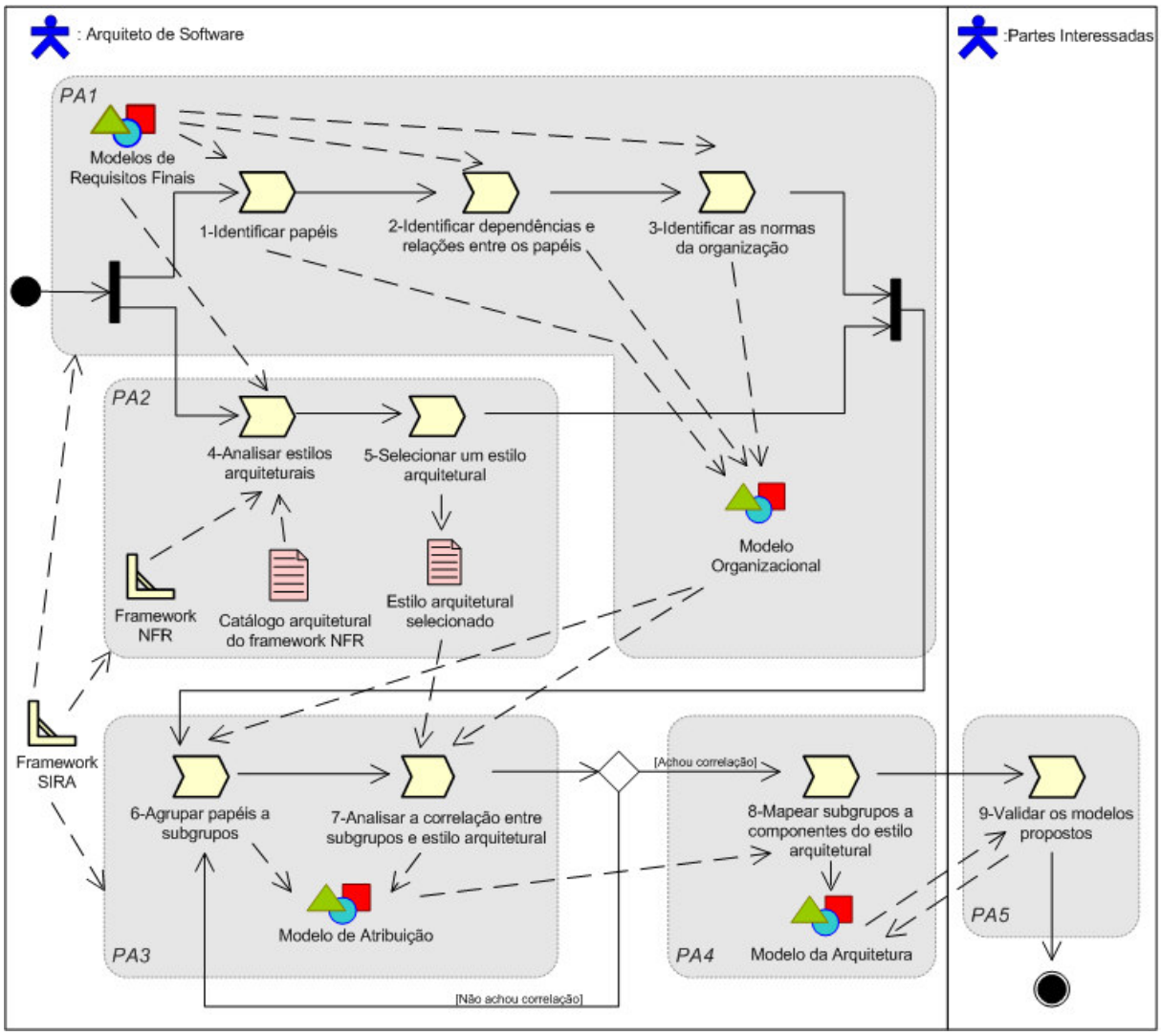


Figura A-3 – Atividades da disciplina Projeto arquitetural

1 - Identificar papéis: Esta atividade envolve a identificação e especificação de papéis com responsabilidades no sistema. Inicia com a identificação das responsabilidades a partir da análise de metas e tarefas (ou planos) dos modelos de requisitos. As responsabilidades também podem ser identificadas do domínio da aplicação. Os papéis são identificados pela distribuição de responsabilidades do grupo de maneira coordenada. Após a identificação, os papéis são especificados. A Tabela A-16 apresenta o detalhamento desta atividade.

Tabela A-16- Detalhamento da atividade *Identificar papéis*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Identificar as responsabilidades pela análise das metas e tarefas (ou planos) dos modelos de requisitos. 2. Identificar papéis que podem assumir as responsabilidades. 3. Especificar os papéis identificados. 4. Documentar no Modelo organizacional	*Produtos de Trabalho:* • Modelo de Requisitos finais *Guia:* • Framework SIRA	• Modelo organizacional (parcial): – Especificação de papéis e protocolos	• Arquiteto de software

2 - Identificar dependências e relações entre os papéis: Esta atividade compreende a identificação das interações que ocorrem entre os papéis da organização. Os padrões de interação que descrevem a estrutura social podem ser vistos como uma rede de relações. São usados dois tipos de ferramentas matemáticas para representar interações entre papéis: matrizes e gráficos de rede social. A Tabela A-17 apresenta o detalhamento desta atividade.

Tabela A-17- Detalhamento da atividade *Identificar dependências e relações entre os papéis*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Analisa as dependências entre os papéis identificados, através da análise das informações que podem ser trocadas entre os papéis para cumprir suas metas. 2. Modela os papéis e suas dependências como um grafo direcionado e/ou uma matriz de interação. 3. Documentar no Modelo organizacional.	*Produtos de Trabalho:* • Modelo de Requisitos finais • Modelo organizacional (parcial) *Guia:* • Framework SIRA	• Modelo organizacional (parcial): – Matriz de interação – Grafo de interação	• Arquiteto de software

3 – Identificar as normas da organização: Esta atividade envolve a identificação das normas da organização. Além das interações dos papéis, os padrões de comportamento são especificados através normas, ou seja, metas que devem ser satisfeitas ou evitadas pelos atores. Para cada papel identificado, as normas são identificadas e documentadas. A Tabela A-18 apresenta o detalhamento desta atividade.

Tabela A-18- Detalhamento da atividade *Identificar normas da organização*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Analisa as interações entre os papéis da organização. 2. Identificar padrões de comportamento que devem ser satisfeitos pelos papéis, para cumprir suas metas dentro da organização. 3. Identificar padrões de comportamento que devem ser evitados pelos papéis, para cumprir suas metas dentro da organização. 4. Documentar no modelo organizacional.	*Produtos de Trabalho:* • Modelo de Requisitos finais • Modelo organizacional (parcial) *Guia:* • Framework SIRA	• Modelo organizacional (parcial): – Relação as normas da organização	• Arquiteto de software

4 - Analisar estilos arquiteturais: Esta atividade envolve a análise dos estilos organizacionais que são aplicados a arquitetura multiagentes de sistemas. Para relacionar os requisitos do sistema e a arquitetura organizacional, é necessário conhecer as propriedades não-funcionais relevantes de arquiteturas multiagentes (e.g *Previsibilidade, Segurança, Adaptabilidade, Coordenação, Cooperação, Disponibilidade, Integridade, Modularidade, Agregabilidade*, etc.) e quais alternativas arquiteturais existem para cada uma destas propriedades (e.g. *flat structure, pyramid, joint venture, structure-in-5, takeover, arm's length, vertical integration, co-optation, bidding, etc.*) [Kolp; Giorgini; Mylopoulos, 2006]. A Tabela A-19 apresenta o detalhamento desta atividade.

Tabela A-19- Detalhamento da atividade *analisar estilos arquiteturais*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Identificar as metas-soft que representam os requisitos não funcionais de um SMA. 2. Analisar estas metas-soft e compará-las as propriedades não funcionais das arquiteturas multiagentes usando o framework NFR. 3. Identifica os estilos arquiteturais que podem ser aplicados ao domínio da aplicação do SMA 4. Documentar no documento Estilo arquitetural selecionado	*Produtos de Trabalho:* • Modelo de Requisitos finais *Guia:* • Framework NFR	• Estilo arquitetural selecionado (parcial)	• Arquiteto de software

5 - Selecionar um estilo arquitetural: Esta atividade envolve a seleção de um estilo organizacional apropriado para a arquitetura do sistema. O estilo arquitetural é selecionado utilizando o catálogo de correlações que avalia as propriedades não-funcionais (representadas como metas-soft no diagrama de raciocínio dos atores) com os padrões organizacionais (e.g. flat structure, pyramid, joint venture, structure-in-5, takeover, arm's length, vertical integration, co-optation, bidding, etc.). A Tabela A-20 apresenta o detalhamento desta atividade.

Tabela A-20 - Detalhamento da atividade *Selecionar um estilo arquitetural*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Comparar as metas-soft identificadas com os estilos arquiteturais baseado no catálogo de correlações do framework NFR. 2. Identificar estilos arquiteturais que melhor se ajustem às metas-soft do sistema. 3. Se houver mais de um estilo selecionado, fazer a seleção de acordo com a prioridade das metas-soft mais relevantes para o domínio de aplicação do sistema. 4. Representa o estilo arquitetural selecionado com um grafo direcionado e/ou uma matriz de interação. 5. Documentar no documento Estilo arquitetural selecionado.	*Produtos de Trabalho:* • Estilo arquitetural selecionado (parcial) *Guia:* • Catálogo de correlação do Framework NFR	• Estilo arquitetural selecionado – Grafo de interação – Matriz de interação	• Arquiteto de software

6 - Agrupar papéis a subgrupos: Esta atividade envolve a análise dos papéis da organização e sua classificação em subgrupos autônomos. Esta classificação é realizada através da *análise da centralidade* e *da similaridade dos papéis*. A *análise da centralidade* indica os papéis que têm o mesmo padrão de interações. Já a *similaridade dos papéis* indica se existem papéis semelhantes que podem ser agrupados. A análise da centralidade é feita calculando o grau de centralidade e grau de proximidade a partir da quantidade de saídas e entradas nos grafos de interação (criado na atividade *2 – Identificar dependências e relações entre os papéis*). A similaridade é calculada através do uso da fórmula de correlação de Pearson [Pearson, 2007]. Estes cálculos estão descritos no framework SIRA e exemplificados no capítulo 6. Baseado nos resultados obtidos, isto é na identificação dos atores similares, os papéis são agrupados em subgrupos. A Tabela A-21 apresenta o detalhamento desta atividade.

Tabela A-21- Detalhamento da atividade *Agrupar papéis a subgrupos*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Calcular o grau de centralidade dos papéis. 2. Calcular o grau de proximidade dos papéis. 3. Calcular o coeficiente de similaridade dos papéis. 4. Identificar os atores similares pela análise dos resultados dos passos 1,2 e 3. 5. Agrupa os atores similares em subgrupos. 6. Analisar as interações para os subgrupos através das matrizes de interação e/ou grafo de interação 7. Documentar no Modelo de Atribuição.	<u>*Produtos de Trabalho:*</u> • Modelo organizacional <u>*Guia:*</u> • Framework SIRA	• Modelo de Atribuição – Matriz de centralidade dos papéis – Matriz de proximidade dos papéis – Matriz de similaridade dos papéis – Matriz / grafo de interação dos subgrupos	• Arquiteto de software

7 - Analisar a correlação entre subgrupos e estilo arquitetural: Esta atividade envolve a análise e comparação do comportamento dos subgrupos identificados na atividade anterior com os componentes do estilo arquitetural selecionado. São analisadas e medidas a centralidade e a similaridade entre os subgrupos e os componentes do estilo arquitetural. Estas análises são comparadas e resultam em coeficientes de correlação que indicam associações positivas fortes (++) e parcialmente fortes (+). Bem como associações negativas fortes (--) e parcialmente fortes (-). A Tabela A-22 apresenta o detalhamento desta atividade.

Tabela A-22 - Detalhamento da atividade *Analisar a correlação entre subgrupos e estilo arquitetural*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Realizar a medição do grau de centralidade dos subgrupos. 2. Calcular o coeficiente de similaridade dos subgrupos. 3. Realizar a medição do grau de centralidade dos componentes do estilo arquitetural selecionado. 4. Calcular o coeficiente de similaridade dos componentes do estilo arquitetural selecionado. 5. Avalia a similaridade estrutural entre o grupo e o estilo arquitetural. 6. Avalia a similaridade estrutural entre os subgrupos e os componentes do estilo arquitetural. 7. Se não identificar similaridades, recalcular incluindo outros subgrupos de outras metas. 8. Documentar no Modelo de Atribuição.	*Produtos de Trabalho:* • Modelo organizacional • Estilo arquitetural selecionado • Modelo de atribuição (parcial) *Guia:* • Framework SIRA	• Modelo de Atribuição: – Matriz de centralidade dos subgrupos – Matriz de similaridade dos subgrupos – Matriz de centralidade do estilo organizacional – Matriz de similaridade do estilo organizacional – Matriz de correlação	• Arquiteto de software

8 - Mapear subgrupos a componentes do estilo arquitetural: Esta atividade envolve o mapeamento entre os subgrupos identificados pelos requisitos do sistema e os componentes do estilo arquitetural. Assim, será possível derivar a primeira configuração organizacional baseado nos resultados preliminares obtidos até aqui. Pela análise das correlações, o mapeamento dos subgrupos aos componentes da arquitetura é realizado. As correlações mais fortes estabelecem o mapeamento. A Tabela A-23 apresenta o detalhamento desta atividade.

Tabela A-23 - Detalhamento da atividade *Mapear subgrupos a componentes do estilo arquitetural*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Identifica as correlações mais fortes entre os subgrupos e componentes do estilo arquitetural. 2. Relaciona quais componentes do estilo arquitetural são assumidos pelos subgrupos. 3. Modela a arquitetura como um modelo de dependência de atores em i*.	*Produtos de Trabalho:* • Modelo de Atribuição *Guia:* • Framework SIRA	• Modelo de Arquitetura	• Arquiteto de software

9 - Validar os modelos propostos: Após a geração do projeto de arquitetura, os modelos são apresentados a todas as partes interessadas no desenvolvimento do sistema para validação e refinamento da versão final da arquitetura do sistema. A Tabela A-24 apresenta o detalhamento desta atividade.

Tabela A-24 - Detalhamento da atividade *Validar os modelos propostos*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Apresentar os modelos de arquitetura para todas as partes interessadas. 2. Coletar e analisar mudanças com as partes interessadas. 3. Refinar os modelos acrescentando as mudanças sugeridas. 4. Validar as versões finais que serão usadas como base para o projeto detalhado e implementação do sistema. 5. Concluir a documentação com as informações analisadas nesta disciplina.	*Produtos de Trabalho:* • Modelo organizacional • Modelo de Atribuição • Modelo de Arquitetura *Guia:* • Framework SIRA	• Modelo de Arquitetura	• Arquiteto de software • Partes Interessadas

Disciplina Projeto Detalhado

As atividades da disciplina Projeto detalhado (Figura A-4) são detalhadas apresentando uma breve descrição, a relação das etapas sugeridas para realização do trabalho, a identificação dos produtos de entrada e saída, os guias usados e o papel de processo responsável pela execução da atividade.

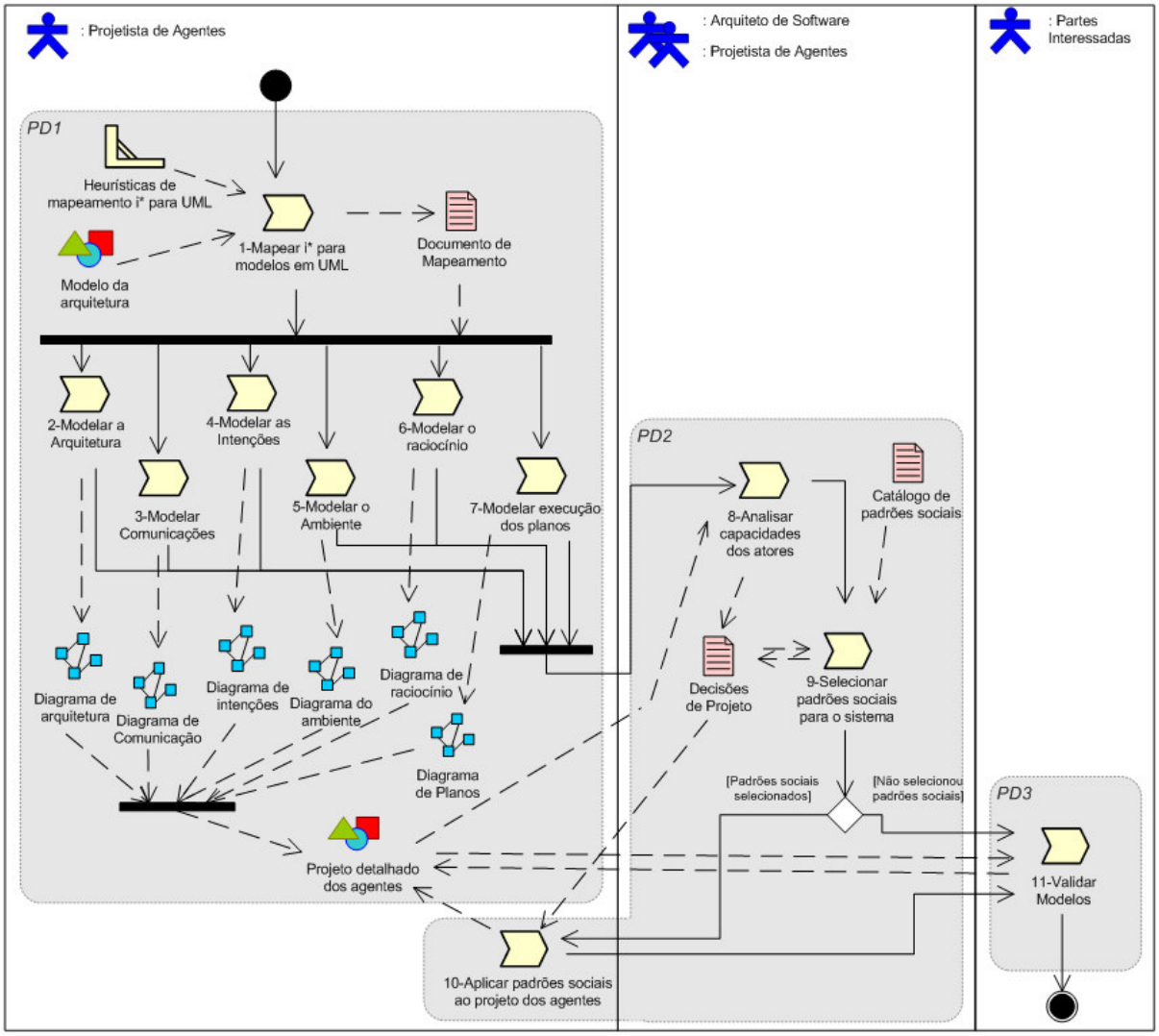


Figura A-4 – Atividades da disciplina Projeto detalhado

1 - Mapear i* para modelos em UML: esta atividade envolve o mapeamento de diagramas de arquitetura de sistemas multiagentes modelados na notação i* para diagramas na notação UML usando um conjunto de heurísticas descritas no guia *Heurísticas de mapeamento i* para UML*. É produzido um produto de trabalho chamado *Documento de Mapeamento* que

servirá de diretriz para as demais atividades. A Tabela A-25 apresenta o detalhamento destas atividades.

Tabela A-25 – Detalhamento da atividade *Mapear i* para modelos em UML*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Identificar os elementos do diagrama i* e associá-los com as classes e diagramas dos modelos UML. 2. Identificar as informações que não estão diretamente modeladas nos diagrama i* tais como restrições, direitos de acesso, crenças, planos de ação dos agentes. 3. Documentar no Documento de mapeamento.	*Produtos de Trabalho:* • Modelo detalhado da arquitetura *Guias:* • Heurísticas de mapeamento i* para UML	• Documento de Mapeamento	• Projetista de agentes

2 - ***Modelar a Arquitetura:*** esta atividade envolve a modelagem do sistema multiagente como um diagrama estendido de classes da UML gerado a partir do *Modelo Arquitetural* e do *Documento de Mapeamento*. O produto gerado é o *Diagrama Arquitetural* que modela a visão estática de módulos que compõe um sistema e reflete o padrão cliente-servidor adaptado para sistemas multiagentes. A Tabela A-26 apresenta o detalhamento destas atividades.

Tabela A-26 - Detalhamento da atividade *Modelar a arquitetura*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Identificar as classes e relacionamentos do diagrama de arquitetura mapeados no documento de mapeamento. 2. Modelar o diagrama de arquitetura.	*Produtos de Trabalho:* • Documento de Mapeamento *Guias:* • Heurísticas de mapeamento i* para UML	• Diagrama de arquitetura	• Projetista de agentes

3 - Modelar Comunicações: esta atividade envolve a modelagem das interações entre os papéis e agentes do sistema multiagente. O produto gerado é o Diagrama de comunicação (Diagrama de seqüência da UML) que é usado para modelar as mensagens trocadas entre os agentes que executam um papel, à medida que eles interagem para implementar o seu comportamento. A Tabela A-27 apresenta o detalhamento destas atividades.

Tabela A-27 - Detalhamento da atividade *Modelar a comunicação dos atores*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Identificar os agentes e papéis que foram mapeados no documento de mapeamento. 2. Identificar as interações que ocorrem entre os agentes a partir do documento de mapeamento. 3. Identifica os protocolos destas comunicações. 4. Modelar o diagrama de comunicação.	*Produtos de Trabalho:* • Documento de Mapeamento *Guias:* • Heurísticas de mapeamento i* para UML	• Diagrama de comunicação	• Projetista de agentes

4 - Modelar as Intenções: esta atividade envolve a modelagem dos elementos intencionais dos agentes do sistema multiagente. O produto gerado é o *Diagrama de intenções* que é usado para modelar os papéis executados pelos agentes, suas crenças, metas, planos, normas e ontologias usadas na organização. A Tabela A-28 apresenta o detalhamento destas atividades.

Tabela A-28 – Detalhamento da atividade *Modelar as intenções*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Identificar os agentes e papéis que foram mapeados no documento de mapeamento. 2. Identificar os elementos intencionais do documento de mapeamento: metas, planos, crenças e normas. 3. Modelar o diagrama de intenções.	*Produtos de Trabalho:* • Documento de Mapeamento *Guias:* • Heurísticas de mapeamento i* para UML	• Diagrama de intenções	• Projetista de agentes

5 - Modelar o Ambiente: esta atividade envolve a modelagem do ambiente onde estão inseridos os agentes do sistema multiagente. O produto gerado é o *Diagrama de ambiente* que é usado para modelar os papéis e agentes que formam a organização que está situada em um ambiente. O ambiente é composto por recursos que são acessados de acordo com direitos dos papéis e agentes para executar suas responsabilidades. A Tabela A-29 apresenta o detalhamento destas atividades.

Tabela A-29 – Detalhamento da atividade *Modelar o ambiente*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Identificar a organização, os agentes e papéis mapeados no documento de mapeamento. 2. Identificar o ambiente e seus recursos e os direitos de acesso de cada agente e/ou papel a partir do documento de mapeamento. 3. Modelar o diagrama de Ambiente.	*Produtos de Trabalho:* • Documento de Mapeamento *Guias:* • Heurísticas de mapeamento i* para UML	• Diagrama de ambiente	• Projetista de agentes

6 - Modelar o raciocínio: esta atividade envolve a modelagem dos elementos racionais dos agentes que compõem o sistema multiagentes. O produto gerado é o *Diagrama de raciocínio* que é usado capturar a influência dos planos para a satisfação das metas-soft. Esta análise da influência ajuda o agente a escolher os planos apropriados entre alternativas para obter uma dada meta. A Tabela A-30 apresenta o detalhamento desta atividade.

Tabela A-30 - Detalhamento da atividade *Modelar o raciocínio para os atores*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Identificar a seus agentes e papéis mapeados no documento de mapeamento. 2. Identificar as metas-soft e planos destes agentes. 3. Identificar as contribuições dos planos para as metas-soft. 4. Modelar o diagrama de raciocínio.	*Produtos de Trabalho:* • Documento de Mapeamento *Guias:* • Heurísticas de mapeamento i* para UML	• Diagrama de raciocínio	• Projetista de agentes

7 – Modelar a execução dos planos: esta atividade envolve a modelagem dos planos executados pelos agentes que compõem o sistema multiagentes para atingir uma meta. O produto gerado é o *Diagrama de planos* que é usado para modelar a seqüência de ações que compõe os planos para obter as metas. Podem-se analisar as ações que são realizadas em paralelo e quais são dependentes de outras ações. A Tabela A-31 apresenta o detalhamento desta atividade.

Tabela A-31- Detalhamento da atividade *Modelar a execução dos planos*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Identificar os agentes e papéis mapeados no documento de mapeamento. 2. Identificar a seqüência de ação para execução dos macro planos / micro planos dos agentes. 3. Modelar os diagramas de planos.	*Produtos de Trabalho:* • Documento de Mapeamento *Guias:* • Heurísticas de mapeamento i* para UML	• Diagrama de planos	• Projetista de agentes

8 - Analisar capacidades dos atores: esta atividade envolve a identificação das capacidades que os agentes devem conter. As capacidades representam a habilidade de um ator em definir, escolher e executar um plano para atendimento de uma meta, mediante certas condições ambientais e na presença de um evento específico. As capacidades são identificadas analisando-se o *Diagrama de Arquitetura* e identificando os macro planos executados associados às metas. Normalmente cada associação entre macro planos e metas dará lugar a uma ou mais capacidades. A Tabela A-32 apresenta o detalhamento desta atividade.

Tabela A-32 - Detalhamento da atividade *Analisar a capacidade dos atores*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Identificar os agentes e papéis mapeados no documento de mapeamento. 2. Analisar o modelo de arquitetura, modelo ambiental e modelo de planos para identificar os macro planos dos papéis de agentes. 3. Documentar no documento de decisões de projeto.	*Produtos de Trabalho:* • Diagrama de arquitetura, diagrama de ambiente, diagrama de planos	• Decisões de projeto	• Projetista de agentes • Arquiteto de software

9 - Selecionar padrões sociais para o sistema: esta atividade envolve tanto a análise do catálogo da especificação de padrões sociais que podem ser utilizados no projeto do sistema, como a comparação das capacidades dos agentes com as capacidades de padrões sociais. As capacidades dos agentes identificadas anteriormente são agrupadas para identificar padrões de comportamento. Estes padrões são comparados com os padrões sociais para identificar características comuns. A identificação e seleção de padrões sociais são documentadas no produto de trabalho *Decisões de projeto*. A Tabela A-33 apresenta o detalhamento desta atividade.

Tabela A-33 - Detalhamento da atividade *Selecionar padrões sociais para o sistema*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Agrupar as capacidades dos agentes em grupos de comportamento semelhante. 2. Analisar as características dos padrões sociais. 3. Comparar as características dos padrões sociais com os grupos de capacidades para identificar quais padrões sociais podem ser usados no sistema. 4. Documentar no documento de decisões de projeto.	*Produtos de Trabalho:* • Decisões de projeto • Catálogo de padrões sociais	• Decisões de projeto	• Projetista de agentes • Arquiteto de software

10 - Aplicar padrões sociais ao projeto dos agentes: esta atividade envolve a aplicação dos padrões sociais aos modelos do projeto detalhado do sistema, se algum padrão for selecionado. Para executar a aplicação dos padrões sociais é necessário instanciar cada papel da estrutura do padrão aos elementos presentes no projeto arquitetural. Os *diagramas de arquitetura, comunicação, intenção, raciocínio, ambiente e plano* são atualizados com os elementos dos padrões sociais selecionados, caso seja necessário. A Tabela A-34 apresenta o detalhamento desta atividade.

Tabela A-34 - Detalhamento da atividade *Aplicar padrões sociais ao projeto de agentes*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Para cada padrão social selecionado, instanciar seus elementos em elementos dos diagramas do projeto detalhado. 2. Atualizar o modelo de arquitetura 3. Atualizar o modelo de comunicação 4. Atualizar o modelo de ambiente 5. Atualizar o modelo de raciocínio 6. Atualizar o modelo de intenção 7. Atualizar o modelo de planos	*Produtos de Trabalho:* • Decisões de projeto	• Projeto detalhado dos agentes – Modelo de arquitetura – Modelo de comunicação – Modelo de ambiente – Modelo de raciocínio – Modelo de intenção – Modelo de planos	• Projetista de agentes

11 - Validar Modelos: Após a análise do projeto de arquitetura e detalhamento do projeto de agentes, os modelos são apresentados a todas as partes interessadas no desenvolvimento do sistema para validação e refinamento da versão final do projeto do sistema. A saída é o *Projeto detalhado dos agentes* atualizado, que é a composição de todos os diagramas de projeto detalhado de agentes. A Tabela A-35 apresenta o detalhamento desta atividade.

Tabela A-35 Detalhamento da atividade *Validar Modelos*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Apresentar o modelo de projeto detalhado para todas as partes interessadas. 2. Coletar e analisar mudanças com as partes interessadas 3. Refinar os modelos acrescentando as mudanças sugeridas 4. Validar as versões finais que serão usadas como base para a implementação do sistema. 5. Concluir a documentação (refinar modelos) com as informações analisadas nesta disciplina.	*Produtos de Trabalho:* • Projeto detalhado dos agentes	• Projeto detalhado dos agentes	• Projetista de agentes • Arquiteto de software • Partes Interessadas

Disciplina Implementação

As atividades da disciplina *Implementação* (Figura A-5) são detalhadas apresentando uma breve descrição da atividade, a relação das etapas sugeridas para realização do trabalho, a identificação dos produtos de entrada e saída, os guias usados e o papel de processo responsável pela execução da atividade.

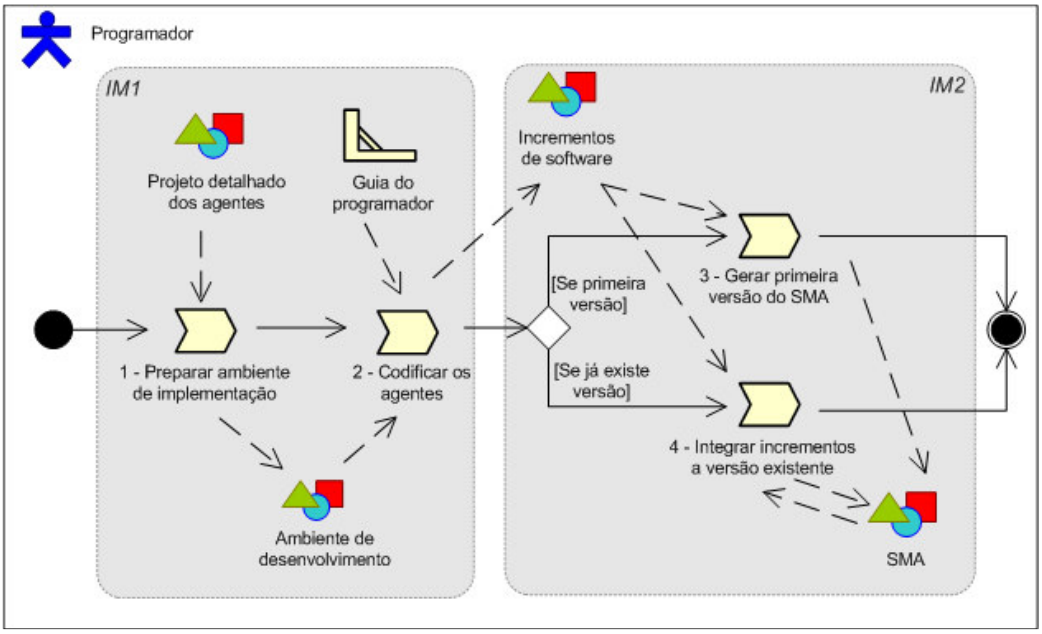


Figura A-5 – Atividades da disciplina Implementação

1 - Preparar ambiente de implementação: Esta atividade envolve a configuração do *ambiente de desenvolvimento* para iniciar o desenvolvimento, a carga dos dados iniciais e a geração do esboço dos agentes que serão desenvolvidos a partir do projeto detalhado. A Tabela A-36 apresenta o detalhamento desta atividade.

Tabela A-36 - Detalhamento da atividade *Preparar ambiente de implementação*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Configurar o ambiente de desenvolvimento. 2. Criar bases de dados e preencher com dados para dar apoio a implementação. 3. Gerar o esboço dos agentes a partir do projeto detalhado.	*Produtos de Trabalho:* • Projeto detalhado dos agentes *Guias:* • Guia do programador	• Ambiente de desenvolvimento	• Programador

2 - Codificar os agentes: Esta atividade envolve a codificação dos agentes, de seu comportamento, de suas ações, da interação com outros agentes e com o ambiente através do

framework de desenvolvimento configurado. A saída são os *Incrementos de software,* ou seja, uma nova versão do sistema que contém extensões e refinamentos sobre a versão anterior do SMA implementado. A Tabela A-37 apresenta o detalhamento desta atividade.

Tabela A-37 - Detalhamento da atividade *Codificar os agentes*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Codificar as crenças do agente (base de dados) 2. Codificar os agentes 3. Codificar o comportamento dos agentes 4. Codificar as mensagens trocadas 5. Codificar a interface com o usuário 6. Documentar o código	*Produtos de Trabalho:* • Framework de desenvolvimento *Guias:* Guia do programador	• Incrementos de software	• Programador

3 - Gerar primeira versão do SMA*:* Esta atividade envolve a geração da primeira versão executável do sistema multiagente. Só será realizada na primeira iteração que envolva a implementação do sistema. A saída é a versão inicial do SMA implementado. A Tabela A-38 apresenta o detalhamento desta atividade.

Tabela A-38 - Detalhamento da atividade *Gerar primeira versão do SMA*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Unificar o código dos agentes. 2. Gerar uma versão executável. 3. Executar testes de implementação.	*Produtos de Trabalho:* • Incrementos de software	• SMA executável	• Programador

4 - Integrar incrementos a versão existente: Esta atividade envolve a integração do código implementado à versão executável existente do sistema multiagente. Só será realizada a partir da segunda iteração que envolva a implementação do sistema. A saída é o sistema multiagente (SMA) implementado com a integração dos incrementos de software já codificados até a iteração atual. A Tabela A-39 apresenta o detalhamento desta atividade.

Tabela A-39 - Detalhamento da atividade *Integrar incrementos a versão existente*

Etapas da Atividade	*Entrada*	*Saída*	*Papel Executor*
1. Unificar o código dos agentes 2. Gerar uma versão executável 3. Executar testes de implementação	*Produtos de Trabalho:* Incrementos de software	• SMA executável	• Programador

Apêndice B

Publicações

- Silva, M. J., Maciel, P. R., Pinto, R. C., Alencar, F., Tedesco, P., Castro, J. (2007) “Extracting the Best Features of Two Tropos Approaches for the Efficient Design of MAS” X Workshop Iberoamericano de Ingenieria de Requisitos Y Ambientes de Software, (IDEAS’07) Isla de Margarita, Venezuela, May 2007.
- Pinto, R.; Castro, J.; Tedesco, P.; Silva, M. J.; Alencar, F. "A Traceability Reference Model for Agent Oriented Development". In proc. of the Third Workshop on Software Engineering for Agent-oriented Systems, (SEAS 2007). João Pessoa, Oct 2007.
- Lucena, M.; Santos, E; Silva, C.T.L.S.; Alencar, F.; Silva, M.J.; Castro, J. "Towards a Unified Metamodel for i*". In proc. of the Research Challenges in Information Science (RCIS 2008). Marrocos, Jun 2008.
- Alencar, F.; Menezes, J.; Silva, J. J. R; Silva, M. J.; Fontes,S.S.; Castro,J. "Modelagem de requisitos intencionais de um sistema de informação como suporte à gestão ambiental na construção". XII Encontro Nacional de Tecnologia do Ambiente Construído (ENTAC2008). Fortaleza, Outubro de 2008.

Dissertação de Mestrado apresentada por **Maria Jocélia Silva** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"U-TROPOS: uma Proposta de Processo Unificado para Apoiar o Desenvolvimento de Software Orientado a Agentes"**, orientada pelo **Prof. Jaelson Freire Brelaz de Castro** e aprovada pela Banca Examinadora formada pelos professores:

Profa. Patrícia Cabral de Azevedo Restelli Tedesco
Centro de Informática / UFPE

Prof. Sergio Castelo Branco Soares
Departamento de Sistemas Computacionais / UPE

Prof. Jaelson Freire Brelaz de Castro
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 27 de agosto de 2008.

Prof. FRANCISCO DE ASSIS TENÓRIO DE CARVALHO
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.